USENIX Association

Proceedings of the General Track: 2003 USENIX Annual Technical Conference

San Antonio, Texas, USA
June 9-14, 2003

Application-specific Delta-encoding via Resemblance Detection

Fred Douglis
IBM T. J. Watson Research Center
Hawthorne, NY 10532
douglis@acm.org

Arun Iyengar
IBM T. J. Watson Research Center
Hawthorne, NY 10532
aruni@us.ibm.com

Abstract

Many objects, such as Þles, electronic messages, and web pages, contain overlapping content. Numerous past research projects have observed that one can compress one object relative to another one by computing the differences between the two, but these *delta-encoding* systems have almost invariably required knowledge of a speciÞc relationship between themÑ most commonly, two versions using the same name at different points in time. We consider cases in which this relationship is determined dynamically, by efÞciently determining when a sufÞcient resemblance exists between two objects in a relatively large collection. We look at speciÞc examples of this technique, namely web pages, email, and Þles in a Þle system, and evaluate the potential data reduction and the factors that inßuence this reduction. We Þnd that delta-encoding using this resemblance detection technique can improve on simple compression by up to a factor of two, depending on workload, and that a small fraction of objects can potentially account for a large portion of these savings.

1 Introduction

Delta-encoding is the act of compressing a data object, such as a Þle or web page, relative to another object [1, 13]. Usually there is a *temporal* relationship between the two objects: the latter object exists, and when it is subsequently modiÞed, the changes can be represented in a small fraction of the size of the entire object. There is often also a *naming* relationship between the objects, since a modiÞed Þle can have the same name as the original copy. In these cases, identifying the base version against which to compute a delta is straightforward.

Delta-encoding is particularly attractive for situations where information is being updated across a network with limited bandwidth. For example, web sites are often replicated both for higher performance and availability. The bandwidth between the replicas can be limited. Another example would be replicated mail systems. Electronic mail systems often allow clients to replicate copies of mail messages locally. Clients may be connected to the network via phone lines with limited bandwidth. For an email client connected to a mail server via a slow link, techniques which minimize bandwidth required for updates are highly desirable. However, in each of these environments, it is not always possible to identify an appropriate base version to take advantage of delta-encoding.

Our work therefore addresses a domain in which there are very many objects with arbitrary overlap among different pairs of objects, and the relationships between these pairs are not known *a priori*. If one can identify which pairs are suitable candidates, delta-encoding can reduce the size of one relative to another, thereby reducing storage or transmission costs in exchange for computation. We consider several application domains for this technique, which we refer to as *delta-encoding via resemblance detection*, or DERD: web trafÞc, email, and Þles in a Þle system.

We defer additional discussion of our research until after a more detailed discussion of delta-encoding and resemblance detection, which appears in the following subsection. After that, the next section describes the framework of our analysis in greater detail, including the metrics we consider. Section 3 presents the various datasets we used. Section 4 describes the experiments, and Section 5 provides the results of these experiments. Section 6 discusses the resource usage issues that would arise in a practical implementation of DERD. Section 7 surveys related work, and Section 8 summarizes and describes possible future work.

1.1 Background

It is difÞcult to describe our approach without providing a general overview of both delta-encoding and resemblance detection. We cover enough of each of these areas here to set the stage for combining the two, then return to a more comprehensive comparison with related work toward the end of the paper.

Deltas are useful for reducing resource requirements, and existing applications of deltas generally fall into two categories: storage and networking. For storage, when one already stores a base version of a Þle, subsequent versions can be represented by changes. This lowers storage demands within Þle systems (the Revision Control System (RCS) [25] is a longstanding example of this), backup-restore systems [1], and similar environments.

Over a network, transmitting data that are already known to the recipient can be avoided. The most common approach in this case is to work from a common base version known to the sender and recipient, compute the delta, and transmit it. This technique has been applied to web traffic [16], IP-level network communication [24], and other domains. An extension to the traditional web delta-encoding approach is to select the base version by finding similar, rather than identical, URLs [7].

What if one wishes to find a similar file based on *content* rather than *name*, among a large collection of files? Manber devised a method for extracting **features** of files based on their contents, in order to find files with overlapping content efficiently [14]. He computed hashes of overlapping sequences of bytes (also known as "shingles"), then looked for how many of these hashes were shared by different files. Manber indicated that clustering similar files for improved compression would be an application of this technique. Broder used a similar approach but used a deterministic sampling of the hash values to dramatically reduce the amount of data needed for each file [5, 6]. With his approach, a subset of features of a file is used to represent the file, and if two files share many of those features in common, there is a high probability of significant content in common as well. A common use for this technique is to suppress near-duplicates in search engine results [6], and variations of the technique have been used in link-level duplicate suppression [24] and file systems [8, 17, 20].

Because the shingling technique has seen so much use in the systems community of late, we refrain from providing a detailed description of it. Briefly, it uses Rabin fingerprints [21] to compute a hash of consecutive bytes; the key properties of Rabin fingerprints are that they are efficient to compute over a sliding window, and they are uniformly distributed over all possible values. Thus, Broder's approach of selecting the N fingerprints with the smallest values effectively selects N "random" features in a deterministic fashion, and two documents with many features in common overall would hopefully have many of these N features in common.

1.2 Goals

As Manber suggested, one can use the features of documents to identify when files overlap and then delta-encode pairs of overlapping files to save space or bandwidth. One goal of this work was to assess whether this technique is generally applicable, and if not, to identify some specific instances in which it is applicable. A second goal was to evaluate a number of the parameters used in this process, such as:

- the size of a shingle,
- the amount of overlap among features necessary to get a sufficiently small delta,
- the number of files with similar overlap necessary to get close to the "best" delta,
- selection of delta-encoding algorithms and parameters to those algorithms,
- whether delta-encoding the contents of specially formatted files such as Zip files in an application-specific method is beneficial,
- and other metrics.

1.3 Summary of Results

We have found that the benefits of application-specific deltas vary depending on the mix of content types. For example, HTML and email messages display a great deal of redundancy across large datasets, resulting in deltas that are significantly smaller than simply compressing the data, while mail attachments are often dominated by non-textual data that do not lend themselves to the technique. A few large files can contribute much of the total savings if they are particularly amenable to delta-encoding. Application-specific techniques, such as delta-encoding an unzipped version of a zip or gzip file and then zipping the result, can significantly improve results for a particular file, but unless an entire dataset consists of such files, overall results improve by just a couple of percent.

Numerous parameters can be varied in assessing the benefits of deltas in this context, and we have evaluated several. The results do not appear to be sensitive to the size of shingles or the delta-encoding algorithm, within reason. The extent of the match of the number of features is a good predictor of the delta size. Perhaps most importantly, when multiple files match the same number of features, there is minimal difference between the *best* delta—the smallest delta obtained across all the files—and the *average* delta. The latter two results suggest that while it is beneficial to determine the file(s) with the maximal number of matching features, only one delta need be computed. This is crucial because finding matching features, given a precomputed database of the features of other files and the dynamically computed feature set of the file being delta-encoded, is far more efficient than computing an actual delta.

2 Framework

This section describes our approach to the problem of delta-encoding with resemblance detection in greater detail. We discuss the types of data we considered and the way in which we evaluate the potential benefits of DERD.

2.1 Types of Data

In the past, delta-encoding has been used for many types of data in numerous environments. Our interest has focused on data that are located Òtogether,Ó meaning that they belong to a single user, or they reside on a single server. Earlier work has demonstrated the potential beneÞts of deltas when the same object is modiÞed over time, whereas we consider different objects that exist at the same time. Thus far, we have analyzed web data (primarily HTML), email, and a Þle system.

In a Research Report [10] coauthored with Kiem-Phong Vo of AT&T Labs, we previously argued that one could use BroderÕs technique for efÞciently selecting features of objects to determine dynamically a suitable candidate to serve as the base for HTTP delta-encoding. This would be an extension to the proposed standard described in a recent RFC [15]. The report described a possible protocol but gave no statistics to support the utility of the idea in practice. In the case of individual web clients, objects must be large enough to justify the added overheads of transmitting their features, comparing the features on a client, possibly computing a new delta-encoding on the ßy in response to the clientÕs request, and reconstructing the page on the client. Beyond that proposal, similarity among different web pages could be used for efÞcient distribution of new pages to caches in a content distribution network (CDN), or other replicas; in this case, by transmitting many pages at once, overheads could be minimized. We have estimated the best-case beneÞts for a web-based DERD system, by downloading numerous pages from several sites at a single point in time, and then comparing each page against the others. In practice, not all the other pages would be cached by an individual client, though they might be cached by a CDN if they are not completely dynamic.

In parallel with assessing the overlap of content on real web sites, we identiÞed the overlap of content in email and other local Þle system content as an appropriate application domain. At any instant, all the Þles are available, so in theory any Þle could be represented as a delta from one or more other Þles. As new Þles are created, they could be encoded against all earlier stored Þles, especially a previous version of the same Þle should it exist. If a ÒliveÓ Þle system uses this approach, it must use techniques such as copy-on-write and reference counting to ensure that the base version against which a delta was computed is not modiÞed or deleted until the delta itself is no longer needed. The same approach could be used to efÞciently back up a Þle system: rather than delta-encoding updates in an incremental backup, the entire Þle system would be compressed by identifying where similarity exists.

None of these techniques would be useful without signiÞcant reduction in Þle sizes, so the primary focus of this study is to evaluate those reductions. Like the earlier study of deltas in HTTP [16], we consider regular compression as a basis for comparison, since compressing each object to remove internal redundancy is trivial. We analyzed several datasets: the contents of `/usr` on a Redhat Linux 7.1 PC, totaling nearly 2 Gbytes of data; the contents of a userÕs `MH` mail repository, with each message stored in a separate Þle (possibly including one or more MIME attachments) totaling 566 Mbytes of data; and the contents of several usersÕ Lotus Notes mail, with message bodies and attachments separated into distinct Þles. Section 3 describes the datasets in detail.

2.2 Evaluation Metrics and Practical Considerations

As noted above, size reduction is the crucial determining factor for the success of our proposal. This reduction must be considered not only relative to the original content, but relative to the size of the content using traditional compression tools such as *gzip*. Considering that reconstructing the original requires the reference Þle to be available, one might favor a compressed version over a delta-encoded version if the former is marginally larger.

Furthermore, the effect of the reduction is dependent on the environment:

- If an individual Þle is encoded, either as a delta or simple compression, and then stored on disk or some other block-based medium, the gain is not exactly the number of bytes by which the Þle is reduced. Instead, it is a function of the number of blocks taken up by the Þle before and after encoding. For instance, if every Þle is rounded to the nearest 4-Kbyte boundary, then shrinking a Þle from 4097 bytes to 4095 bytes actually saves 1 block, i.e. 4096 bytes. More typically, a Þle might be encoded but still use the same number of blocks on disk.
- Similarly, reducing trafÞc over a network has low marginal beneÞts if the same number of packets is used; however, if the number of round-trips in communication can be decreased, the improvement in response time is more signiÞcant.
- If many Þles are encoded together, such as a full backup or web server replication, then the beneÞts are more directly related to the actual per-Þle gains, since rounding effects are amortized over the entire dataset.

There are other evaluation metrics of interest, including:

Computation There are overheads due to computing the features for each Þle, comparing the features of the candidate and stored Þles, and encoding a delta once a base version is selected. Since there has been extensive research in making both delta-encoding [1] and resemblance detection [5, 6] efficient even in enormous datasets such as Internet search engines, and because our prototype is geared toward assessing space reduction benefits rather than speed, we do not report timings in this paper. However, we discuss performance issues in general terms in Section 6.

Space overheads A system that is selecting a base version given a set of features must be able to compare those features to a large set of existing files. The overhead per file may be from 50-800 bytes depending on how much information is stored, which in turn affects the quality of the comparison [6].

Execution parameters There are a number of run-time parameters that can affect the performance and/or effectiveness of the system. We consider the following:

Size and number of features Shingling a file creates an enormous number of fingerprints, or features, representing sequences of data. Broder's technique selects a "small" number of them, where "small" is parameterizable [5]. We evaluated the sensitivity of the results to this parameter. We also can require a minimal fraction of features to match before computing a delta, to see if the poorer matches still demonstrate benefits. Finally, the number of bytes used to create a single feature can vary.

Best matches If multiple files match the same number of features, an exhaustive computation could determine which base file produces the smallest delta. In fact, a file matching fewer features could produce a smaller delta than one matching more features. However, in practice, one would want to consider as few base versions as possible. While it was not possible to perform an exhaustive search within large datasets, we sampled several files with an equal number of matching features to determine whether there is a significant variance among candidate base files.

There is also an interaction between the number of features and the quality of the match. If more features are compared, then different base files can be distinguished more finely, possibly resulting in a smaller delta.

Lastly, some files may produce particularly large savings relative to an entire dataset, while others may contribute relatively little. Assuming files are sorted by the savings from encoding them, we analyze how many files need be delta-encoded to produce a given fraction of the total benefit.

Unzip-Rezip A small change to a file can result in significant differences in a compressed version of the file. For example, we made a copy of the Redhat 7.1 `/usr/share/dict/words` (409,276 bytes, 45,424 one-word lines) and changed line six from `abandon` to `xyzzy`. We call the copy `words1`. Both `words` and `words1` generated gzipped files of about 131 Kbytes, with a difference of just four bytes in size. Encoding the differences between the uncompressed `words1` and `words`, using vcdiff, represented the differences in just 79 bytes. In stark contrast, delta-encoding `words1.gz` against `words.gz` generated about 93 **K**bytes.

Therefore, delta-encoding two compressed files by encoding their uncompressed versions and compressing the result (if needed) has the potential for significant gains. Since zip can store an arbitrarily large number of files and directories as a single compressed file, comparing its contents individually and zip-ing the results into a single zip file can have similar benefits. One might assume that `tar` need not be handled specially, since it concatenates its input without compression. We find below that this hypothesis is incorrect for the three delta-encoding programs we tried. For all these datatypes, however, the overall effects depend on the mix of data: in practice, the number and size of compressed files that can benefit from this approach may be dwarfed by all the other data.

Delta-encoding algorithm and parameters There are a few possible delta-encoding programs. We did not find significant differences in output sizes among the available programs; therefore, following the approach of delta-encoding in HTTP [16], we report numbers using Korn and Vo's *vcdiff* [13].

Delta-encoding versus compression We vary a parameter that specifies how much smaller a delta must be than simply compressing a file before the delta is used. If no delta is small enough, of the files used as potential base versions, the compressed version is used instead. We use *vcdiff* for compression (delta-encoding a file against `/dev/null`), due to historical reasons. Its data reduction is comparable to *gzip*, though typically slightly worse.

Identical files When an identical file appears multiple times in a dataset, it can be trivially encoded against another instance through the use of hash functions such as MD5. Past studies have investigated the prevalence of mirrors on the web [4] and techniques for suppressing duplicate payloads [12]. We chose to suppress duplicates from consideration in our analysis, since they are trivially handled through other means, except when a file contained in a zip archive is duplicated (since two zip files may have many identical files and some changed content, and our unzip-rezip procedure would match up the identical files).

3 Datasets

We separate our analyses into two types of data: web pages and files in a file system. We lump email into the latter category, since in general we expect the benefits to be greater for static encoding (space reduction) than network transmission. Note that not all the datasets we analyzed are discussed further in this paper, but we include them in the tables to give a sense of the variability of the results.

3.1 Web Data

Ideally, to analyze the benefits of DERD for the web, one would study a live implementation over an extended time, and/or use full content traces to simulate an implementation. The latter approach was used effectively to study delta-encoding based on identical URLs [16], but such traces are difficult to obtain.

Instead, we used the *w3get* program to download a small set of root web pages, and recursively the pages linked from them, up to two levels. We specifically excluded file suffixes that suggested image data, such as JPG and GIF, focusing instead on the base pages. This is partly because delta-encoding has already been demonstrated to be ineffective across two different image files, even having the same name [16], and partly because images change more slowly than HTML [9] and are more likely to be cached in the first place.

While periodic downloads of specific web pages have been used in the past to evaluate delta-encoding [13], cross-page comparisons require a single snapshot of a large number of pages. We believe these pages, and the results obtained from them, demonstrate a high degree of overlap in content between pages on the same site; this has been observed in other research due to the high use of "templates" for creating dynamic pages [3, 23].

Table 1 lists the sites accessed, all between 24-26 July 2002, with the number of pages and total size. Note that in the case of Yahoo!, the download was aborted after about 27 Mbytes were downloaded, as that offered sufficient data to perform an analysis, and it was unclear how much additional data would be retrieved if left unchecked.

3.2 File Data

We used two types of file data, which are summarized in Table 2. First, we scanned the entire */usr* directory in a nearly unmodified Redhat Linux 7.1 distribution, totaling just under 2 Gbytes of data in over 100K files. Second, we examined email from several users and in several formats. Much of our analysis used over 500 Mbytes of one user's UNIX-based email, which is stored individually in separate files by the `MH` mail system. The remaining data came from Lotus Notes, which stores message bodies and attachments as separate objects in a flat-file document database. We studied the attachments of five users and the message bodies of two.

4 Experiments

As described in Section 2.2, we varied a number of parameters in the delta-encoding and resemblance detection process. Our general goals were to determine how much more data could be eliminated by using deltas rather than just compression, and how sensitive that result would be to this set of parameters. In particular, we wanted to estimate the minimal work a system might do to get a reasonable benefit (i.e., the point of diminishing returns).

In general, we fixed the parameters to a common set. We then varied each parameter to evaluate its effect. Table 3 lists these parameters, with a brief description of each one, the default value in **boldface**, and other tested parameters. The parameters are clustered into two sets: the first controls the pass over the data to compute the features, and the second controls the comparison of those features and computation of the deltas.

In some cases, due to space constraints, we do not present additional details about variations in parameters that did not significantly affect results; these are denoted by *italic text*. Additional descriptions of many of the parameters were given above in Section 2.2. Note that `min_features_ratio` is special, in that it is possible to compute the savings for each number of matching features and then compute a cumulative benefit for each number of matches in a later stage, as demonstrated in Section 5.1.

4.1 Implementation Details

Most of the work to encode differences based on similarity is performed by a pair of Perl scripts. One of these recursively descends over a set of directories and invokes a Java program to compute the features. Each computation is a separate invocation of Java, though that could be optimized. Once a file's features have been computed, they are cached in a separate file.

The other script takes the precomputed set of filenames and features, and for each file determines which

Name	Files From...	Files	Size (Mbytes)	Delta%	Comp%
Yahoo	`yahoo.com`	3,755	27.55	8	34
IBM	`ibm.com`	177	3.21	19	36
Masters	`masters.com`	192	3.19	9	35
CNN	`cnn.com`	73	2.53	15	29
Wimbledon	`wimbledon.com`	190	2.40	10	35

Table 1: Web datasets evaluated. Delta and compression percentages refer to the size of the encoded dataset relative to the original.

Name	Files From...	Files		Size (Mbytes)	Delta%	Comp%
		Included	Excluded			
`/usr`	`/usr`	102,932	1,250	1,964.16	36	45
`MH`	one userÕ `MH` directory	87,005		565.69	34	54
User1_Bod	User 1Õ Notes mail bodies	3,097		5.97	29	60
User1_Att	User 1Õ Notes mail attach.	189		81.29	71	75
User2_Bod	User 2Õ Notes mail bodies	445		1.18	42	56
User2_Att	User 2Õ Notes mail attach.	1,078		417.35	32	37
User3_Att	User 3Õ Notes mail attach.	140		36.18	52	61
User4_Att	User 4Õ Notes mail attach.	1,982		991.45	53	66

Table 2: File datasets evaluated. Excluded Þles are explained in the text. Delta and compression percentages refer to the size of the encoded dataset relative to the original.

other Þles have the maximum number of matching features. Currently this is done by identifying which features a Þle has, and incrementing counters for all other Þles with a given feature in common, using the value of the feature as a hash key. This records the most features in common at any point, F. After all features are processed, any Þles that have at least one feature in common are sorted by the number of matching features. Typically, only the Þles that match exactly F features are considered as base versions, up to the `max_comparisons` parameter, but if the best matches fail to produce a small enough delta, poorer matches are considered until the maximum is reached. There are methods to optimize this comparison by precomputing the overlap of Þles, as well as through estimation [22], which we intend to integrate at a later date.

Delta-encoding is performed by one of a set of programs, all written in C. Once a pair of Þles has been so encoded, the size of the output is cached. Occasionally, the delta-encoding program might generate a delta that is larger than the compressed Þle, or even larger than the original Þle. In those cases, the minimum of the other values is used.

For a given dataset, the results are reported by listing how many Þles have a maximum features match for a given number of features, with statistics aggregated over those Þles: the original size, the size of the delta-encoded output, and the size of the output using *vcdiff* compression (delta-encoding against `/dev/null`, comparable to *gzip*). Table 4 is an example of this output. The rows at the top show dissimilar Þles, where deltas made no difference, while the rows at the bottom had the greatest similarity and the smallest deltas. The BestDelta and AvgDelta columns show that, in general, there was at most a 1% difference in size (relative to the original Þle) between the best of up to ten matching Þles and the average of all ten. This characteristic was common to all the datasets. Correspondingly, in all the Þgures, the curves for the savings for delta-encoding depict the average cases.

There are two apparent anomalies in Table 4 worth noting. First, there is a substantial jump in size at the complete 30/30 features match, despite a consistent number of Þles, showing a much higher average Þle size. This is skewed by a large number of nearly identical Þles, resulting from form letters attaching manuscripts for review; if each manuscript was sent to three persons and the features in the large common data were all selected by the minimization process, they all match in every feature. (This is a desirable behavior, but may not be typical of all datasets.) Second, the Þles with 0–2 out of 30 features matching have a dramatically worse compression ratio than the other data. We believe these are attributable to types of data that neither match other Þles to a great extent nor exhibit particularly good compressibility from internally repeated text

Processing Stage	Parameter	Description	Values
Preprocessing	`shingle_size`	Number of bytes in a Þngerprinted shingle	**20**, 30
	`num_features`	Number of features compared	**30**, 100
	`min_size`	Minimum size of an individual Þle to include in statistics	**128**, 512 bytes
	`unzip`	Should `zip` Þles be unzipped before comparison	**yes**, no
	`gunzip`	Should `gz` Þles be unzipped before comparison	**yes**, no
Encoding	`static_files`	Whether encoding A against B precludes encoding B against A	web=**no**, Þles=**yes**
	`program`	Program to perform delta-encoding	**vcdiff**
	`exhaustive_search`	Whether to compare against all Þles, or just best matches	**no**, *yes*
	`max_comparisons`	Maximum number of Þles to compare against, with equal maximal matching features	**10**, *1, 5*
	`min_features_ratio`	What fraction of features must match to compute a delta?	**0**-1 (cumulative distribution)
	`improvement_threshold`	What is the maximum size of a delta, relative to simple compression, for it to be used?	25%, 50%, 75%, **100%**

Table 3: Parameters evaluated. **Boldface** represents defaults, and *italics* represent evaluated cases not reported here.

Matches	Files	Size (Mbytes)	BestDelta (%)	AvgDelta (%)	Compressed (%)
0	230	4.37	65	65	**65**
1	2634	95.09	64	65	**65**
2	3308	63.87	58	58	**60**
3	3927	30.86	39	40	45
4	4284	32.53	31	32	39
5	4710	22.86	35	36	46
			...		
27	294	2.85	4	4	46
28	227	3.09	2	2	44
29	174	9.39	0	0	43
30	224	**91.38**	0	0	48
All	87005	565.69	34	34	54

Table 4: Delta-encoding and compression results for the `MH` directory. Percentages are relative to original size, e.g. 34% means deltas save about two-thirds of the original size. Boldfaced numbers are explained in the text. This table corresponds to the graphical results in Figure 1.

strings. MIME-encoded compressed data would have this attribute, when the same compressed Þle does not appear in multiple messages.

To analyze the beneÞts of unzipping Þles, encoding them, and zipping the results, we take two approaches. Zip Þles can contain entire directory hierarchies, while gzip Þles compress just one Þle. Therefore, for `zip` Þles, we create a special *ZIPDIR* directory, into which the contents are *unzip*ped before features are calculated. We assume there are no additional beneÞts to compression, since zip has already taken care of that. For deltas, we delta-encode each Þle in this directory, storing the results in a second temporary directory, and then zip the results. For `gzip` Þles, we *gunzip* the Þles, compute the features, and discard the uncompressed output. Each time we delta-encode a gzipped file, either as the reference or the version, we uncompress it on the fly (the most recent uncompressed version file is then cached and reused for each encoding). Section 5.4 discusses the added benefits of these two approaches.

In some cases, the features for all the files in a single dataset, with other run-time state, resulted in a virtual memory image that exceeded the 512 Mbytes of physical memory on the machine performing the comparisons— this is an artifact of our Perl-based prototype, and not inherent to the methodology, as evidenced by the scale of the search engines that use resemblance detection to suppress duplicates [6]. For the `usr` and `MH` datasets, we preprocessed the data to separate them into manageable subdirectories, then merged the results. This would result in files in different partitions not being compared: for example, a file in `Mail/conferences` would not be compared against a file in `Mail/projects`. In general, spatial locality would suggest that the best matches for a file in `Mail/conferences` would be found in `Mail/conferences`. (We subsequently validated this theory by rerunning the script on all `MH` directories at once, using a more capable machine, with no significant difference in the overall benefits.) Also, since partitions were based on subdirectories of a single root such as `/usr`, it also would result in some partitions having too few files to perform meaningful comparisons; we skipped any subdirectories with fewer than 100 files, resulting in a small fraction of files being omitted (listed in Table 2).

5 Results

Here we present our analyses. We start with overall benefits for different types of data, then describe how varying certain parameters impacts the results.

5.1 Overall Benefits

Our overall goal is to reduce file sizes and to evaluate how sensitive this reduction is to different data types, the amount of effort expended, and other considerations. Table 4 gives a sense of these results, in tabular form, for a dataset that is particularly conducive to this approach; Figure 1 shows the same data graphically. Figure 1(a) plots compressed sizes and delta-encoded sizes, as well as the original total file sizes, against the number of matching features. For each possible number of matching features from 0–30, we plot the total data of files having that number of matching features as their maximum match. As we expected, the more features match, the smaller the delta size. The cumulative effect is shown in Figure 1(b). In this graph (as well as several subsequent ones with the same label on the X-axis), a point (X,Y) shows that the total data size obtained using a particular technique such as compression or delta-encoding is Y if all files with at least X maximal matching features are encoded. For instance, the Y-value of the point on the *Compressed* curve with X-value 15 is the percent of the total data size obtained if all files matching at least one other file in at least 15 features are compressed. Figure 1(b) shows that the most benefit is derived from including all files, even with zero matches, although in those cases these benefits come from compression rather than deltas— recall that the size of a delta is never larger than delta-encoding it against the empty file, i.e., compressing it.

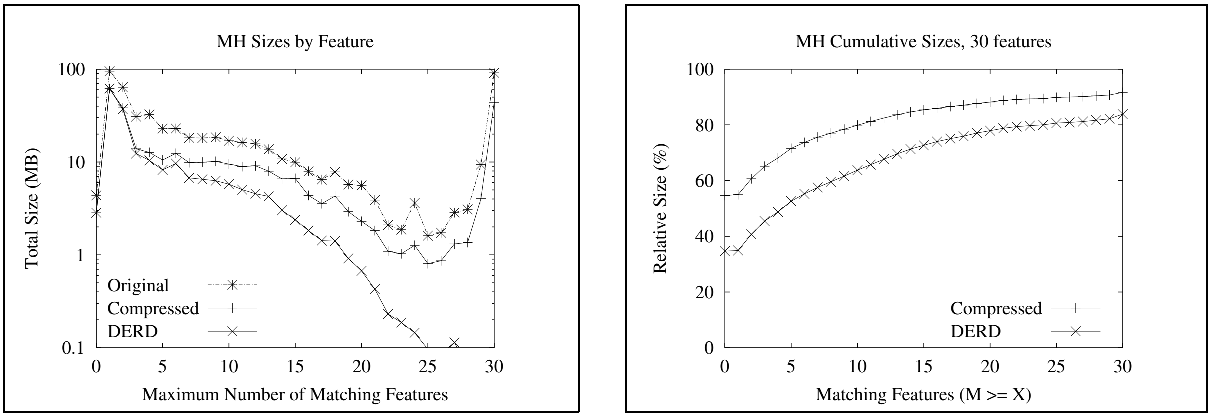
Figure 2(a) shows the cumulative benefits of deltas and compression for two of the static datasets: `usr`, and the `MH` data. Figure 2(b) does the same for two of the web datasets, IBM and Yahoo. Both graphs are limited to two datasets in order to avoid cluttering them with many overlapping lines, but the bottom-line savings for the other datasets were reported in Table 2 and Table 1, respectively. In each, the different datasets show different benefits, due to the amount of data being compared and the nature of the contents. Specifically, the graphs have very different shapes because many more files in the web datasets have high degrees of overlap.

5.2 Contributions of Large Files

The graphs presented thus far have emphasized the effect of statistics such as the number of features that match. Another consideration is the skew in the savings: *do a small number of files contribute most of the benefits of delta-encoding*? In the case of the `MH` dataset, such a skew was suggested by the statistics in Table 4, which showed 91 of the 566 Mbytes matching in all 30 features and delta-encoding to virtually nothing.

We visualize an answer to this question by considering every file in a particular dataset, sorting by the most bytes saved for any delta obtained for it, and plotting the cumulative distribution of the savings as a function of the original files. Figure 3(a) plots the cumulative savings of the `MH` dataset (as a fraction of the original data) against the fraction of *files* used to produce those savings or the fraction of *bytes* in those files. In each case the savings for DERD and strict compression are shown as separate curves. Finally, points are plotted on a log-log scale to emphasize the differences at small values, and note that the `Comp by byte%` curve starts at just over 2% on the X-axis.

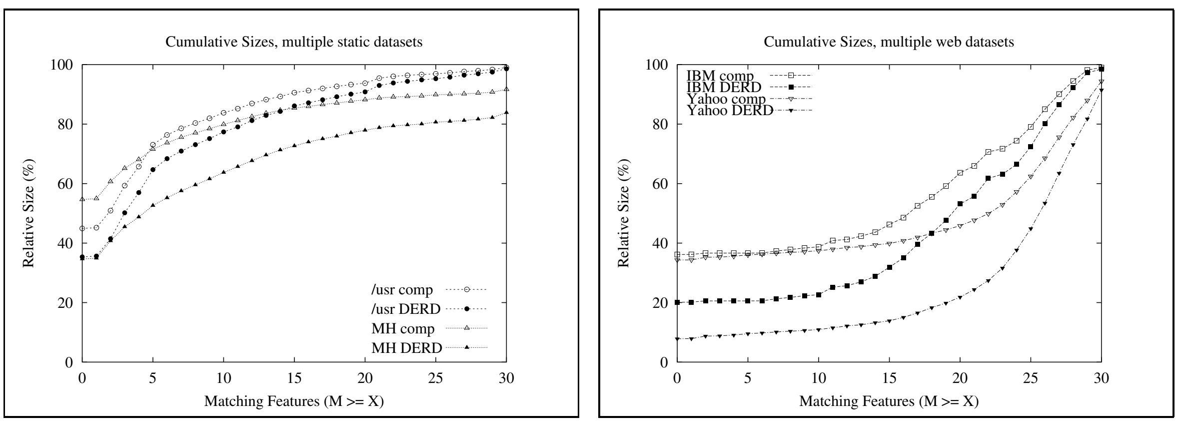
The results for this dataset clearly show significant skew. For example, for deltas, 1% of the files account for 38% of the total 65% saved; encoding 25% of the bytes will save 22% of the data. Compression also shows some skew, since some files are extremely compressible. If one compressed the best files containing 25% of the bytes, one would save 17% of the data. This degree of



(a) Total data sizes for the original dataset, using compression, and using DERD, for individual numbers of matching features. Most of the data match very few features in any other Þle, or match all the features. The y-axis is on a log scale.

(b) Cumulative beneÞts. The y-axis shows the relative size, in percent, of compressing or delta-encoding each Þle. A point on the x-axis shows the beneÞt from performing this on all Þles that match at least that many features.

Figure 1: Effect of matching features, for the MH data. These Þgures graphically depict the the data in Table 4.



(a) Static datasets.

(b) Web datasets.

Figure 2: Effect of matching features, cumulative, for several datasets.

skew suggests that heuristics for intelligently selecting a subset of potential delta-encoded pairs, or compressed Þles, could be quite beneÞcial.

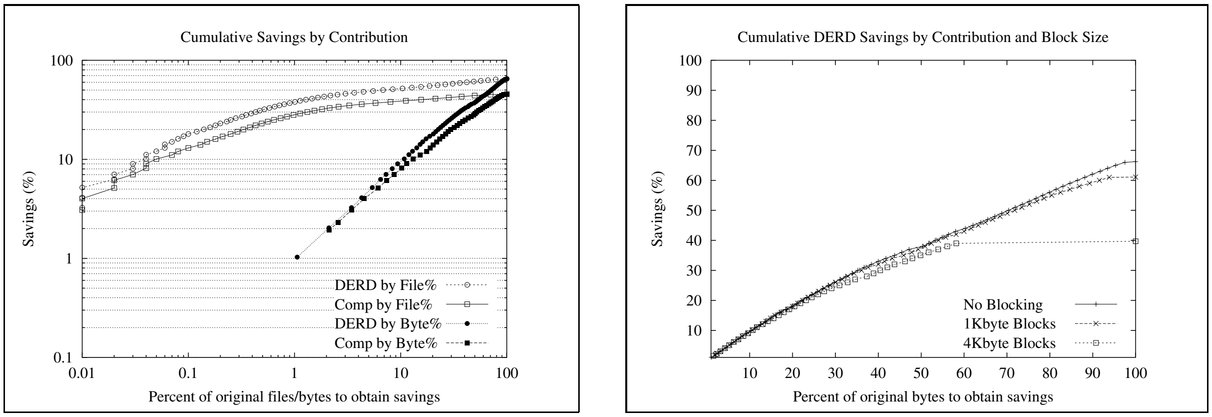
5.3 Effects of File Blocking

Section 2.2 referred to an impact on size reduction from rounding to Þxed block sizes. In some workloads, such as Þle backups, this is a non-issue, but in others it can have a moderate impact for small blocks and a substantial impact for large ones.

Figure 3(b) shows how varying the blocksize affects overall savings for the MH dataset. Like Figure 3(a), it plots the cumulative savings sorted by contribution, but it accounts for block rounding effects. A 1-Kbyte minimum blocksize, typical for many UNIX systems with fragmented Þle blocks, reduces the total possible beneÞt of delta-encoding from around 66% (assuming no rounding) to 61%, but a 4-Kbyte blocksize brings the beneÞt down to 40% since so many messages are smaller than 4 Kbytes.

5.4 Handling Compressed and Tarred Files

Section 2.2 provided a justiÞcation for comparing the uncompressed versions of zip and gzip Þles, as well as a hypothesis that tar Þles would not need special treatment. For some workloads this is irrelevant, since for example the MH repository stored all messages with full



(a) Relative savings as a function of cumulative Þles, by count or by bytes. Plotted on a log-log scale.

(b) Relative savings assuming no Þle blocking, or rounding to 1-Kbyte or 4-Kbyte units.

Figure 3: Cumulative savings from `MH` Þles, sorted in order of contribution to total savings.

bodies, uncompressed. An attachment might contain MIME-encoded compressed Þles, but these would be part of the single Þle being examined, and one would have to be more sophisticated about extracting these attachments. In fact, there was no single workload in our study with large numbers of both zip and gzip Þles, and overall beneÞts from including this feature were only 1-2% of the original data size in any dataset. For example, the `User4_Attach` workload, which had the most zip Þles, only saved an additional 2% over the case without special handling. Even though the zip Þles themselves were reduced by about a third, overall storage was dominated by other Þle types.

We expected directly delta-encoding one tar Þle against a similar tar Þle to generate a small delta if individual Þles had much overlap, but this was not the case in some limited experiments. Vcdiff generated a delta about the size of the original gzipped tar Þle, and two other delta programs used within IBM performed similarly. We tried a sample test, using two email tar Þle attachments unpacked into two directories, and then using `DERD` to encode all Þles in the two directories. We selected the delta-encoded and compressed sizes of the individual Þles in the smaller of the tar Þles, and found delta-encoding saved 85% of the bytes, compared to 71% for simple compression of individual Þles and 79% when the entire tar Þle was compressed as a whole. Depending on how this extends to an entire workload, just as with zip and gzip, these savings may not justify the added effort.

5.5 Deltas versus Compression

By default, our experiments assumed that if a delta is at all smaller than just using compression, the delta is used. There are reasons why this might not be desirable, such as a web server using a cached compressed version rather than computing a specialized delta for a given request. As another example, consider a Þle system backup that would require both a base Þle and a delta to be retrieved before producing a saved Þle: if the compressed version were 25% larger than the delta, it would consume that extra storage, but restoring the Þle would involve retrieving 125% of the deltaÕs size rather than the delta and a base version that would undoubtedly be much larger than that 25%.

We varied the threshold for using a delta to be 25—100% of the compressed size, in increments of 25%. Figure 4 shows the result of this experiment on the `MH` dataset. There is a dramatic increase in the relative size of the delta-encoded data at higher numbers of matching features, because in some cases, there is no longer a usable match at a given level. The most interesting metric is the overall savings if all Þles are included, since that no longer suffers from this shift; the relative size increases from about 35% to about 45% as the threshold is reduced.

5.6 Shingle Size

Unlike some of the other parameters, the choice of shingle sizeÑ within reasonÑ seems to have minimal effect on overall performance. As an example, Figure 5 shows how the size reduction varies when using shingle sizes of 20 versus 30 bytes. If all Þles are encoded, even for minimal matches, the total size reduction is about the same. If a higher value of `min_features_ratio` is used, the 20-byte shingles produce smaller deltas for the same threshold within a reasonable range (10-15 of 30 features matching).

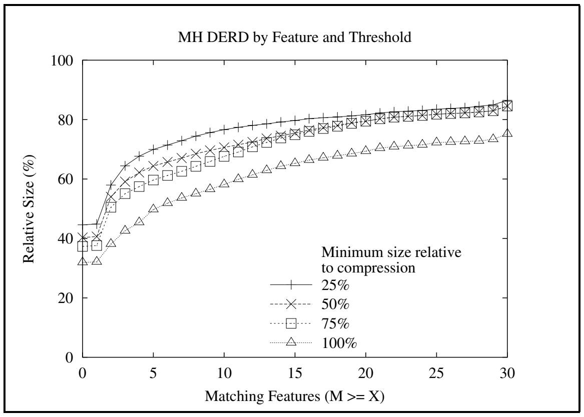


Figure 4: Effect of limiting the use of deltas to a fraction of the compressed Þle, for the MH dataset.

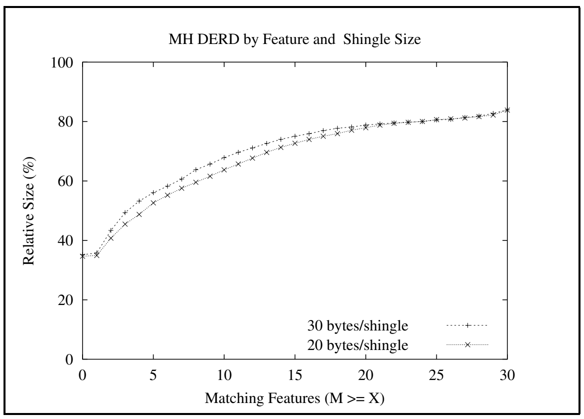


Figure 5: Effect of varying the shingle size between 20 and 30 bytes, for the MH dataset.

5.7 Number of Features

The number of features used for comparisons represents a tradeoff between accuracy of resemblance detection and computation and storage overheads. In the extreme case, one could use ManberÕs approach of computing and comparing every feature, and have an excellent estimate of the overlap between any two Þles. The other extreme is to use no resemblance detection at all or have just a handful of features. Since we have found a fair amount of discrimination using our default of 30 features, we have not considered fewer features than that, but we did compute the savings for the MH dataset from using 100 features instead of 30. The results were virtually indistinguishable in the two casesÑ leading to the conclusion that 30 features are preferable, due to the lower costs of storing and comparing a given number of features.

Broder has described a way to store the features even more compactly, such as 48 bytes per Þle, by treating the features as aggregates of multiple features computed in the ÒtraditionalÓ method [6]. For one such meta-feature to match, all of some subset of the regular features must match exactly, suggesting a higher degree of overlap than we felt would be appropriate for DERD.

6 Resource Usage

A system using our techniques to efÞciently delta encode Þles and web documents could compute features for objects when it Þrst becomes aware of them. The cost for determining features is not that high, and it could be amortized over time. The system could also be tuned to perform delta-encoding when space is the critical resource and to store things in a conventional manner when CPU resources are the bottleneck.

Using 30 features of 4 bytes apiece, the space overhead per Þle is around 120 bytes. For large Þles, this is insigniÞcant. Once the features for a Þle have been determined, it requires $O(n)$ operations to determine the maximum number of matching features with existing Þles where n is the total number of Þles. However, to get a reasonably good number of matching features, it is not always necessary to examine features for all of the existing Þles. A reasonable number of matching features can often be determined by only examining a fraction of the objects when the number of objects is large. That way, the number of comparisons needed for performing efÞcient delta-encoding can be bounded.

Delta-encoding itself has been made extremely efÞcient [1], and it should not usually be a bottleneck except in extremely high-bandwidth environments. Early work demonstrated its feasibility on wireless networks [11] and showed that processors an order of magnitude slower than current machines could support deltas over HTTP over network speeds up to about T3 speeds [16]. More recent systems like *rsync* [26] and LBFS [17], and the inclusion of the Ajtai delta-encoding work in a commercial backup system, also support the argument that DERD will not be limited by the delta-encoding bandwidth.

7 Related Work

Mogul, et al., analyzed the potential beneÞts of compression and delta-encoding in the context of HTTP [16]. They found that delta-encoding could dramatically reduce network trafÞc in cases where a client and server shared a past version of a web page, termed a Òdelta-eligibleÓ response. When a delta was available, it reduced network bandwidth requirements by about an order of magnitude. However, in the traces evaluated in that study, responses were delta-eligible only a small fraction of the time: 10% in one trace and 30% in the other, but the one with 30% excluded binary data such as images. On the other hand, most resources were compressible, and they estimated that compressing those resources dynamically would still offer significant savings in bandwidth and end-to-end transfer times—factors of 2-3 improvement in size were typical.

Later, Chan and Woo devised a method to increase the frequency of delta-eligible responses by comparing resources to other cached resources with similar URLs [7]. Their assumption was that resources "near" each other on a server would have pieces in common, something they then validated experimentally. They also described an algorithm for comparing a file against several other files, rather than the one-on-one comparison typically performed in this context. However, they did not explain how a server would select the particular related resources in practice, assuming that it has no specific knowledge of a client's cache. We believe there is an implicit assumption that this approach is in fact limited to "personal proxies" with exact knowledge of the client's cache [11, 2], in which case it has limited applicability.

Ouyang, et al., similarly clustered related web pages by URL, and tried to select the best base version for a given cluster by computing deltas from a small sample [18]. While they were not focused on a caching context, and are more similar to the general applications described herein, they did not initially use the more efficient resemblance detection methods of Manber and Broder to best select the base versions. Subsequently, they applied resemblance detection techniques to scale the technique to larger collections [19]. This work, roughly concurrent with our own, is similar in its general approach. However, the largest dataset they analyzed was just over 20,000 web pages, and they did not consider other types of data such as email. Another possibly significant distinction is that they used shingle sizes of only 4 bytes, whereas we used 20-30 bytes. (We did not obtain this paper in time to repeat our analyses with such a small shingle size.)

Spring and Weatherall [24] essentially generalized Chan and Woo's work by applying it to all data sent over a specific communication channel, and using resemblance detection to detect duplicate sequences in a collection of data. This was done by computing fingerprints of shingles, selecting those with a predetermined number of zeroes in the low-order bits (deterministically selecting a fraction of features), and scanning before and after the matching shingle to find the longest duplicate data sequence. Like Chan and Woo's work, this system worked only with a close coupling between clients and servers, so both sides would know what redundant data existed in the client. In addition, the communication channel approach requires a separate cache of packets exchanged in the past, which may compete with the browser cache and other applications for resources.

In some cases, the suppression of redundancy is at a very coarse level, for instance identifying when an entire payload is identical to an earlier payload [12], or when a particular region of a file has not changed. Examples of system taking this approach include *rsync* [26], a popular protocol for remote file copying, and the Low-bandwidth File System (LBFS) [17]. However, there are applications for which identifying an appropriate base version is difficult and the available redundancy is ignored. For instance, LBFS exploits similarities not only between different versions of the same file but across files. To identify similar files, it hashes the contents of blocks of data, where a block boundary is (usually) defined by a subset of features—like the Spring & Wetherall approach, except that the features determine block boundaries rather than indices for the data being compared. Variable block boundaries allow a change within one block not to affect neighboring blocks. (The Venti archival system [20] and the Pastiche peer-to-peer backup system [8] are two more recent examples of the use of content-defined blocks to identify duplicate content; we use LBFS here as the "canonical" example of the technique.)

Similarly, it is not always possible to ensure that both sides of a network connection share a single common base version. *Rsync* allows the two communicating parties to ascertain dynamically which blocks of a file are already contained in a version of the file on the receiving side.

LBFS and *rsync* are well suited to compressing large files with long sequences of unchanged bytes, but if the granularity of change is finer than their block boundaries, they get no benefit. Most delta-encoding algorithms remove redundancy if it is large enough to amortize the overhead of the pointers and other meta-data that identify the redundancy. A resemblance detection procedure should therefore be suited to the delta-encoding algorithm, and the size and contents of the data. Our work demonstrates that fine-grained deltas work well in a variety of environments, but a head-to-head comparison with LBFS and *rsync* in these environments will help determine which approach is best in which context.

8 Conclusions and Future Work

Delta-encoding has been used in a number of applications, but it has been limited to two general contexts: encoding a file against an earlier version of the same file, or encoding against other files (or data blocks) where both sides of a communication channel have a consistent view of the cached data. We have generalized this approach in the web context to use features of web content to identify appropriate base versions, and quantified the potential reductions in transfer sizes of such a system. We have also extended Manber's use of this technique on a single server [14], and quantified potential benefits in a general file system and specific to email.

For web content, we have found substantial overlap among pages on a single site. This is consistent with Chan and Woo [7], Ouyang, et al. [19], and recent work on automatic detection of common fragments within pages [23]. For the five web datasets we considered, deltas reduced the total size of the dataset to 8–19% of the original data, compared to 29–36% using compression. For files and email, there was much more variability, and the overall benefits are not as dramatic, but they are significant: two of the largest datasets reduced the overall storage needs by 10–20% beyond compression. There was significant skew in at least one dataset, with a small fraction of files accounting for a large portion of the savings. Factors such as shingle size and the number of features compared do not dramatically affect these results. Given a particular number of maximal matching features, there is not a wide variation across base files in the size of the resulting deltas.

A new file will often be created by making a small number of changes to an older file; the new file may even have the same name as the old file. In these cases, the new file can often be delta-encoded from the old file with minimal overhead. For the most part, our datasets did not consider these scenarios. For situations where this type of update is prevalent, the benefits from delta-encoding are likely to be higher.

Now that we have demonstrated the potential savings of DERD, in the abstract, we would like to implement underlying systems using this technology. The smaller deltas for web data suggest that an obvious approach is to integrate DERD into a web server and/or cache, and then use a live system over time. However, supporting resemblance-based deltas in HTTP involves extra overheads and protocol support [10] that do not affect other applications such as backups. We are also interested in methods to reduce storage and network costs in email systems, and hope to implement our approach in commonly used mail platforms. As the system scales to larger datasets, we can add heuristics for more efficient resemblance detection and feature computation. We can also evaluate additional application-specific methods, such as encoding individual elements of tar files, and compare the various delta-based approaches against other systems such as LBFS and *rsync* in greater depth.

Acknowledgments

Kiem-Phong Vo jointly developed the idea of web-based DERD, resulting in a research report [10] from which a small amount of the text in this manuscript has been taken. Andrei Broder has been extremely helpful in understanding the intricacies of resemblance detection, Randal Burns and Kiem-Phong Vo have similarly been helpful in providing and helping us to understand their delta-encoding software packages, and Laurence Marks is a LotusScript guru extraordinaire. Ziv Bar-Yossef, Sridhar Rajagopalan, and Lakshmish Ramaswamy provided code for computing features. Several people have permitted us to analyze their data, including Lisa Amini, Frank Eskesen and Andy Walter. Ramesh Agarwal, Andrei Broder, Ron Fagin, Chris Howson, Ray Jennings, Jason LaVoie, Srini Seshan, John Tracey, and Andrew Tridgell have provided helpful comments on some of the ideas presented in this paper and/or earlier drafts of this paper. Finally, we thank the anonymous reviewers and our shepherd, Darrell Long, for their advice and feedback.

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] Gaurav Banga, Fred Douglis, and Michael Rabinovich. Optimistic deltas for WWW latency reduction. In *Proceedings of 1997 USENIX Technical Conference*, pages 289–303, January 1997.

[3] Ziv Bar-Yossef and Sridhar Rajagopalan. Template detection via data mining and its applications. In *Proceedings of the Eleventh International Conference on World Wide Web*, pages 580–591. ACM Press, 2002.

[4] K. Bharat and A. Broder. Mirror, mirror on the web: A study of host pairs with replicated content. In *Proceedings of the 8th International World Wide Web Conference*, pages 501–512, May 1999.

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

[6] Andrei Z. Broder. Identifying and filtering near-duplicate documents. In *Combinatorial Pattern Matching, 11th Annual Symposium*, pages 1–10, June 2000.

[7] 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.

[8] 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.

[9] Fred Douglis, Anja Feldmann, Balachander Krishnamurthy, and Jeffrey Mogul. Rate of change and other metrics: a live study of the World Wide Web.

In *Proceedings of the Symposium on Internet Technologies and Systems*, pages 147–158. USENIX, December 1997.

[10] Fred Douglis, Arun K. Iyengar, and Kiem-Phong Vo. Dynamic suppression of similarity in the web: a case for deployable detection mechanisms. Technical Report RC22514, IBM Research, July 2002.

[11] Barron C. Housel and David B. Lindquist. WebExpress: A system for optimizing Web browsing in a wireless environment. In *Proceedings of the Second Annual International Conference on Mobile Computing and Networking*, pages 108–116. ACM, November 1996.

[12] Terence Kelly and Jeffrey Mogul. Aliasing on the World Wide Web: Prevalence and Performance Implications. In *Proceedings of the 11th International World Wide Web Conference*, May 2002.

[13] 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.

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

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

[16] 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.

[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, and Torsten Suel. Using delta encoding for compressing related web pages. In *Data Compression Conference*, page 507, March 2001. Poster.

[19] 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.

[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] Sridhar Rajagopalan, 2002. Personal Communication.

[23] Lakshmish Ramaswamy, Arun Iyengar, Ling Liu, and Fred Douglis. Techniques for efficient detection of fragments in web pages. Manuscript, November 2002.

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

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

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