Evaluation of Efficient Archival Storage Techniques

Lawrence L. You
University of California, Santa Cruz
Jack Baskin School of Engineering
1156 High Street
Santa Cruz, California 95064
Tel: +1-831-459-4458
you@cs.ucsc.edu

Christos Karamanolis
Storage Systems Group
Hewlett-Packard Labs
1501 Page Mill Road, MS 1134
Palo Alto, CA 94304
Tel: +1-650-857-6956
Fax: +1-650-857-5548
christos@hpl.hp.com

Abstract

The ever-increasing volume of archival data that need to be retained for long periods of time has motivated the design of low-cost, high-efficiency storage systems. Inter-file compression has been proposed as a technique to improve storage efficiency by exploiting the high degree of similarity among archival data. We evaluate the two main inter-file compression techniques, data chunking and delta encoding, and compare them with traditional intra-file compression. We report on experimental results from a range of representative archival data sets.

1 Introduction

Over the last several years, we have witnessed an unprecedented growth of the volume of stored digital data. A recent study estimated the amount of original digital data generated in 2002 alone to be close to 5 exabytes, approximately double the volume of data created in 1999 [4]. An increasing fraction of this corpus is archival data: immutable data retained for long periods of time for legal or archival purposes. Examples of archival data include rich media such as audio, images and video, documents, email, and instant messages.

The high rate of archival data generation has motivated a number of research projects to look into ways of improving the space efficiency of disk-based archival storage systems. Researchers have observed that they can take advantage of content overlapping, which is common in archival data, to improve storage efficiency [9, 3]. There are two main techniques proposed for this purpose. The first technique divides each data object into a number of non-overlapping *chunks* and stores only *unique* chunks in the archival storage system. Chunks may be of fixed or variable size. The second technique is based on resemblance detection between data objects and uses *delta encoding* to store only deltas instead of entire data objects.

These two approaches have been developed and used in very different contexts, with different goals and data sets. For example, variable-size chunking was proposed for improving the bandwidth consumption of network file systems [7]. Delta encoding [1] has been used for data compression in HTTP [6] as well as in version-control systems [11]. However, there has been no attempt to compare the two approaches side-by-side, evaluating the storage efficiency they achieve and their applicability to different archival data sets.

The goals of this work can be summarized as follows: (i) evaluate the applicability of the approaches on different data types that exhibit different degrees

of inter-file similarities; (ii) identify the key parameters for each technique and provide rules of thumb for their settings for different data types; (iii) compare inter-file compression techniques with traditional lossless intra-file techniques and explore the potential benefits of hybrid approaches. Further, we provide a performance analysis of the different approaches, and discuss system design and engineering considerations.

2 Overview

Archival data, by its nature, often exhibits strong inter-file resemblance. This paper examines techniques that take advantage of such inter-file resemblance to avoid storing redundant data and thus improve storage efficiency. Such techniques may be combined, if necessary, with lossless intra-file compression, such as sliding-window compression techniques (e.g. *zip* variants).

Several systems that exploit data redundancy at different levels of granularity have been developed in order to improve storage efficiency. One class of systems detects redundant chunks of data at granularities that range from entire file (EMC's Centera) down to individual fixed-size disk blocks (Venti [9]) and variable-size data chunks (LBFS [7]). We focus on the use of variable-sized chunks, which have been reported to exhibit better efficiency over the special case of fixed-size blocks [8]. Typically, such techniques are used in content-addressable storage (CAS) infrastructures. The second class of systems detects and stores only differences (deltas) between similar files, at the granularity of bytes [3].

2.1 Chunking

Data chunking involves two problems. First, a data stream, such as a file, needs to be divided into chunks in a deterministic way. We consider the general case of variable-sized chunks, which works for any type of data, including binary formats. Chunk boundaries are defined by calculating some feature (a digital signature) over a sliding window of fixed size. In our prototype, we use Rabin fingerprints [10], for their computational efficiency in the above scenario. Boundaries are set where the value of the feature meets certain criteria, such as when the value, modulo some specified integer divisor, is zero; the divisor affects the average chunk size. Such deterministic algorithms do not require any knowledge of other files in the system. Moreover, chunking can be performed in a decentralized fashion, even on the clients of the system.

The second problem is to uniquely identify chunks. An algorithm is required that computes a digest over a variable-length block of data. Currently in the prototype, we reuse the Rabin fingerprinting code for deriving chunk identifiers. In practice and for very large data sets, one would need an algorithm that guarantees low probability for collisions, such as MD5 and SHA variants. Exactly because chunks are content-addressable, chunking is suitable for CAS systems. Only unique chunks are stored for any file. The original files can be reconstructed from their constituent chunks. To do that, the system needs to maintain metadata that maps file identifiers to a list of chunk identifiers. Any evaluation of storage efficiency must take into account the overhead due to the metadata. Figure 1 illustrates the main parameters of a chunking technique.

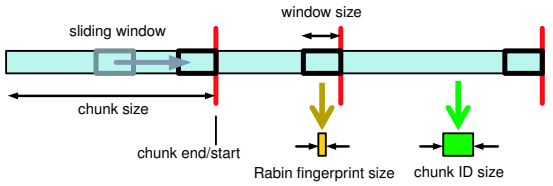

Figure 1: Chunking parameters

We developed a prototype program, named *chc*, to evaluate the efficiency and performance characteristics of chunking without having to build a complete storage system. The input to *chc* is an archive (*tar* file) of a number of files that form the target data set; *chc* produces an output archive that includes unique chunks derived from the original files. Optionally, we can compress the individual chunks in the archive using the *zlib* compression library. *chc* captures a list of chunk identifiers for each file, as well as the identifier and size for each chunk. This metadata is stored also in the output archive and provides an estimate of storage overhead due to chunking. In addition, *chc* can reconstruct the original data set from the final chunk archive.

2.2 Delta encoding

Delta compression is used to compute a delta encoding between a new file and a reference file already stored in the system. When resemblance is above some threshold, a delta is calculated and only that is stored in the system. There are three key problems that need to be addressed in delta encoding.

First, resemblance has to be detected in a content-independent and efficient way. We use the shingling technique proposed by the DERD project [3]. It calculates Rabin fingerprints using a sliding window along an entire file (the window size is a configurable

parameter). The number of fingerprints produced is proportional to the file size. A deterministic *feature selection* algorithm selects a subset of those fingerprints, called a *sketch*, which is retained and later used to compute an estimate of the *resemblance* between two files by comparing two sketches using the *approximate min-wise independent permutations* [2]. This estimate computes similarity between two files by counting the number of matching pairs of features between two sketches. It has been shown that even small sketches, e.g. sets of 20 features, capture sufficient degrees of resemblance.

Thus, when new data needs to be stored, one has to find an appropriate reference file in the system: a file exhibiting a high degree of resemblance with the new data. In general, this is a computationally intensive task (especially given the expected size of archival data repositories). In the prototype, we use an exhaustive search over all stored files. We are currently investigating the use of hierarchical clustering of sketches to reduce the search.

The third problem is to calculate the delta encoding once a reference file has been found. Delta compression is a well-explored area, and in the prototype we used the *xdelta* tool [5], which computes the output using the *zlib* (*gzip*) library. Pointers to reference files are stored with every delta. These identifiers (e.g. SHA digests), along with sketch data, contribute to accounted storage overhead. Our prototype consists of three programs, one for each of the three above problems: feature extraction, resemblance detection, and delta generation.

3 Evaluation

We explain the experimental methodology we used to measure the efficiency, describe the data sets, and analyze the storage efficiency and differences in performance of the two archival storage techniques.

As noted above, storage efficiency is the determining factor for the applicability of an inter-file compression approach. We report on the storage space required as a percentage of the original, uncompressed data set size. For example, stored data that is 20% the size of the original represents an efficiency ratio of 5:1. The functionality and performance of each approach depends on the settings of a number of parameters. As expected, experimental results indicate that no single parameter setting provides optimal results for all data sets. Thus, we first report on parameter tuning for each approach and different data sets. Then, using optimal parameters for each data set, we compare the overall storage efficiency achieved by each approach. The required storage includes the overhead due to the metadata that needs to be stored. Last, we discuss the performance cost and the design issues of applying the two techniques to an archival storage system.

3.1 Data Sets

We chose a range of data sets that we believe to be representative of archival data. Email messages often contain headers (and sometimes attachments) that show great resemblance. Source code and web content are typically versioned. Non-textual content such as presentations and imagery are often similar and require lots of storage space. Finally, computer-generated data such as logs are generated in high volumes and can contain repeated content such as field descriptors. The following is the list of data sets we use.

- HP Support Unix logs (two sets of different total volume)
- Linux kernel 2.2 source code (four versions)
- Email (single user)
- Mailing list archive (BLU)
- HP ITRC Support web site
- Microsoft PowerPoint presentations
- Digital raster graphics (California DRG 37122 7.5 minute untrimmed TIFF)

3.2 Parameter Tuning

In the case of chunking, the *expected chunk size* is a key configuration parameter. It is implicitly set by setting the fingerprint divisor as well as the minimum and maximum allowed chunk size. In general, the smaller it is, the higher the probability of detecting common chunks among files. For data with very high inter-file similarity (such as log files), small chunk sizes result in greater storage efficiency. However, for most data this is not the case, because smaller chunks also mean higher metadata overhead. Often, because of this overhead the storage space required may be greater than the size of the original corpus. As Figure 2 shows, the optimal expected chunk size depends on the type of data; using 128-bit identifiers, the best efficiencies range from 256 to 512 bytes.

The main configurable parameter in the case of delta encoding is the size of the sketches—i.e. the number of features used for resemblance detection. Our experimental results are consistent with what was reported by Douglis et al.: a sketch size of 20 to 30 features is sufficient to capture resemblance among files. Another parameter is the *resemblance threshold*, the

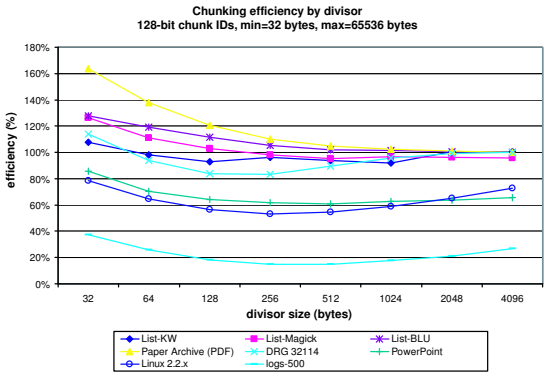

Figure 2: Chunking efficiency by divisor size

number of features that must correspond between two files to consider that sufficient resemblance exists to justify calculating the delta instead of storing the entire new file. For the evaluation of delta encoding, we traverse the target data set one file at a time in a random order. For a file, a delta is created against the file with the highest resemblance that is already in the output archive, as long the resemblance is above a threshold of one corresponding feature. Otherwise, the entire file is stored as a new reference file.

3.3 Storage Efficiency

Table 1 shows the achieved storage efficiency by the two approaches. We do that with and without additional *zlib* compression of chunks and deltas respectively. To establish a baseline for each data set, we create a single *tar* file from the data set, and then compress it with an intra-file compression program, *gzip*. As expected (and as shown by the two first rows of the table), inter-file compression improves with larger corpus sizes. This is not the case with *gzip*.

The HP Unix Logs (8,000 files) show very high similarity. Chunk-based compression on this similar data was reduced to 11% of the original data size, and when each chunk is compressed using the *zlib* (similar to *gzip*) compression, it is just 7.7% of the original size. Even more impressive are the reductions in size when using delta compression. When delta compression is used alone, the data set is reduced to 4% of the original size, but when combined with *zlib* compression, the compressed data is less than 1% of the original size.

Textual content, such as web pages, can be highly similar. However, in the case of the HP ITRC content, *gzip* compression is more efficient than chunking or delta. More surprisingly, *gzip* is better even when we do additional compression of chunks and deltas. The reason is that *gzip*'s dictionary is more efficient across entire files than within the smaller individual chunks, and chunk IDs appear as random (essentially non-compressible) data. But in the context of an archival storage system, *gzip*'s advantage is not likely to be as effective in practice; this is discussed below.

Non-textual data, such as the PowerPoint files with chunking and delta (especially with *gzip*) achieve better efficiency than *gzip* alone. However, the achieved compression rates are less impressive than those for the log data. For raster graphics, delta encoding with *gzip* achieves modest improvement over *gzip* alone.

A single user's email directory and a mailing list archive show little improvement when using delta. Chunking is less effective than *gzip*, although we would expect it to reduce redundancy found across multiple users' data.

In most cases, inter-file compression outperforms intra-file compression, especially when individual chunks and deltas are internally compressed. Chunking achieves impressive efficiency for large volumes of very similar data. On the other hand, delta encoding seems better for less similar data. We believe that this is due to the lower storage overhead required for delta metadata. Typical sketch sizes of 80 to 120 bytes (20 to 30 features $\times$ 4 bytes) for a file of any size are significantly smaller than the overhead of chunk-based storage, which is linear with the size of the file.

Although compressing a set of files into a single *gzip* file to establish a baseline measurement helps illustrate how much redundancy might exist within a data set, it is not likely that an archival storage system would reach those levels of efficiency for several reasons. Most important is that files would be added to an archival system over time and files would be retrieved individually. If a new file were added to the archival store, it would not be stored as efficiently unless the file could be incorporated into an existing compressed file collection, i.e. the new file would need to be added to an existing *tar*/*gzip* file. Likewise, retrieving a file would require first extracting it from a compressed collection and this would require additional time and resources over a chunk or delta-based file retrieval method.

Our experiments measured the size of an entire corpus, in the form of a *tar* file after it has been compressed with *gzip*. Had we compressed each file with

Data Set	Size	# Files	tar + gzip	Chunk	Chunk + zlib	Delta	Delta + zlib
HP Unix Logs	824 MB	500	15%	13%	5.0%	3.0%	1.0%
HP Unix Logs	13,664 MB	8,000	14%	11%	7.7%	4.0%	0.94%
Linux 2.2 source (4 vers.)	255 MB	20,400	23%	57%	22%	44%	24%
Email (single user)	549 MB	544	52%	98%	62%	84%	50%
Mailing List (BLU)	45 MB	46	22%	98%	53%	67%	21%
HP ITRC Web Pages	71 MB	4,751	16%	86%	33%	50%	26%
PowerPoint	14 MB	19	67%	55%	46%	38%	31%
Digital raster graphics	430 MB	83	42%	102%	55%	99%	42%

Table 1: Storage efficiency comparison (64-bit chunk IDs)

gzip first and then computed the aggregate size of all compressed files, the sizes for *gzip*-compressed files would have been much larger. For example, in the case of the HP ITRC web pages, *gzip* efficiency would have been 30% of the original size, much larger than the 16% shown in table 1, and larger than the 26% that can be achieved by using delta compression with *zlib*. When delta compression (or to a lesser extent, chunking) is applied across files first and then an intra-file compression method second, it is more effective than compressing large collections of data because additional redundancy can be eliminated.

3.4 Performance

In practice, space efficiency is not the only factor used to choose a compression technique; we briefly discuss some important systems issues such as computation and I/O performance.

The chunking approach requires less computation than delta encoding. It requires two hashing operations per byte in the input file: one fingerprint calculation and one digest calculation. In contrast, delta encoding requires $s+1$ fingerprint calculations per byte, where s is the sketch size. It also requires calculating the deltas, even though this can be performed efficiently, in linear time with respect to the size of the inputs. Additional issues with delta encoding include efficient file reconstruction and resemblance detection in large repositories.

The two techniques exhibit different I/O patterns. Chunks can be stored on the basis of their identifiers using a (potentially distributed) hash table. There is no need for maintaining placement metadata and hashing may work well in distributed environments. However, reconstructing files may involve random I/O. In contrast, delta-encoded objects are whole reference files or smaller delta files, which can be stored and accessed efficiently in a sequential manner. But, placement in a distributed infrastructure is more involved.

4 Conclusions

Inter-file compression is emerging as a technique to improve space efficiency in archival storage systems. This paper provides the first direct comparison of the two main techniques proposed in the literature, namely chunking and delta encoding, and compares them against traditional intra-file compression. In general, both chunking and delta encoding outperform *gzip*, especially when they are combined with compression of individual chunks and deltas. Chunking is computationally cheap and can be easily used in distributed systems. It works well for data with very high similarity. Thus, it is applicable to applications where there are multiple versions of the same data, such as version control systems, and log files. On the other hand, delta encoding is more computationally expensive, but more efficient with less similar data and thus, it is potentially applicable to a wider range of data sets.

Acknowledgments

Lawrence You was supported by a grant from Hewlett-Packard Laboratories (via CITRIS), Microsoft Research, and supported in part by National Science Foundation Grant CCR-0310888. We thank Kave Eshghi and George Forman of Hewlett-Packard Laboratories for their help and insight into the behavior of file chunking. We are also grateful to members of the Storage Systems Research Center at the University of California, Santa Cruz for their help preparing this paper.

References

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

[2] A. Z. Broder, M. Charikar, A. M. Frieze, and M. Mitzenmacher. Min-wise independent permutations. *Journal of Computer and Systems Sciences*, 60(3):630–659, 2000.

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

[4] P. Lyman, H. R. Varian, K. Searingen, P. Charles, N. Good, L. L. Jordan, and J. Pal. How much information? 2003. `http://www.sims.berkeley.edu/research/projects/how-much-info-2003/`, Oct. 2003.

[5] J. P. MacDonald. File system support for delta compression. Master's thesis, University of California at Berkeley, 2000.

[6] J. Mogul, F. Douglis, A. Feldmann, and B. Krishnamurthy. Potential benefits of delta-encoding and data compression for HTTP. In *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM '97)*, Sept. 1997.

[7] A. Muthitacharoen, B. Chen, and D. Mazières. A low-bandwidth network file system. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP '01)*, pages 174–187, Lake Louise, Alberta, Canada, Oct. 2001.

[8] C. Policroniades and I. Pratt. Feasibility of data compression by eliminating repeated data in practical file systems. First Year Report.

[9] S. Quinlan and S. Dorward. Venti: A new approach to archival storage. In D. D. E. Long, editor, *Proceedings of the 2002 Conference on File and Storage Technologies (FAST)*, pages 89–101, Monterey, California, USA, 2002. USENIX.

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

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