Metadata Considered Harmful ... to Deduplication

Xing Lin, *University of Utah*
Fred Douglis, *EMC Corp.*
Jim Li, *EMC Corp.*
Xudong Li, *Nankai University*
Robert Ricci, *University of Utah*
Stephen Smaldone, *EMC Corp.*
Grant Wallace, *EMC Corp.*

Abstract

Deduplication is widely used to improve space efficiency in storage systems. While much attention has been paid to making the process of deduplication fast and scalable, the effectiveness of deduplication can vary dramatically depending on the data stored. We show that many file formats suffer from a fundamental design property that is incompatible with deduplication: they intersperse metadata with data in ways that result in otherwise identical data being different. We examine three models for improving deduplication in the presence of embedded metadata: deduplication-friendly data formats, application-level post-processing, and format-aware deduplication. Working with real-world file formats and datasets, we find that by separating metadata from data, deduplication ratios are improved significantly—in some cases as dramatically as 5.6×.

1 Introduction

The amount of digital data continues to grow rapidly. Data deduplication has been shown to be effective in improving space efficiency for backup/archive storage systems [3, 9, 20] and there is an increasing interest in applying deduplication to general-purpose file systems [16,19]. The effectiveness of deduplication is therefore crucial to the efficiency of such storage systems.

Generally, there are three types of deduplication: whole-file (also known as single-instance store [1]), fixed-size blocks [14], and variable-size content-defined chunks [12, 20]. Whole-file and fixed-block deduplication work well in some environments [7,11,15], but using the content itself to determine deduplication unit boundaries is popular for two reasons. First, within a file, a small edit that shifts the remaining content would cause fixed-size blocks to align differently such that they would not deduplicate. Second, even small unmodified files may be written to the backup system by applications such as EMC *NetWorker* or Symantec *NetBackup* as part of a larger *aggregate* file to amortize overheads [18]; these aggregate files resemble UNIX *tar* files.

In this position paper, we show that many file formats suffer from a fundamental design property that is incompatible with deduplication: they **intersperse metadata with data** in ways that result in otherwise identical data being different. Metadata is changed more frequently, sometimes in trivial ways, leading to poor deduplication.

We observe that there are at least three ways to adapt ill-behaved data to deduplicating storage:

Deduplication-friendly formats The best solution is to design file formats that separate metadata and data in a way that preserves potential deduplication. We provide a case study of EMC *NetWorker*, which has migrated to a new deduplication-friendly data format for backup data.

Application-level post-processing When it is hard to change the file format for an established application, it is often possible to post-process files to produce a new format that is better suited to deduplication. We describe *mtar*, which transforms *tar* files into a more deduplication-friendly format.

Format-aware deduplication Sometimes, neither of the previous approaches is feasible, and special logic is required within the deduplicating system. We describe how EMC *Data Domain* appliances handle two cases of file formats that use special *markers* interspersed with data: tape markers for virtual tape libraries and block headers within the Oracle RMAN backup format [13].

This paper makes three contributions. 1) It is the first detailed study that identifies the impact of metadata on deduplication. 2) It proposes two new formats (Common Data Format and *mtar*) that improve deduplication. 3) It evaluates the new formats with real-world datasets and shows quantitative improvements to deduplication ratios. We hope that this study contributes to an increased understanding of the role of metadata in deduplication, and thus improved storage efficiency in future deduplicating systems and file formats.

2 Deduplication-friendly Formats

In this section, we discuss our experiences with EMC *NetWorker*, a commercial backup software system. *NetWorker* was first developed in the age of tape-based backups and has since evolved. It uses application-specific data formats that describe data in different formats for different types of backup devices: disk backups and tape

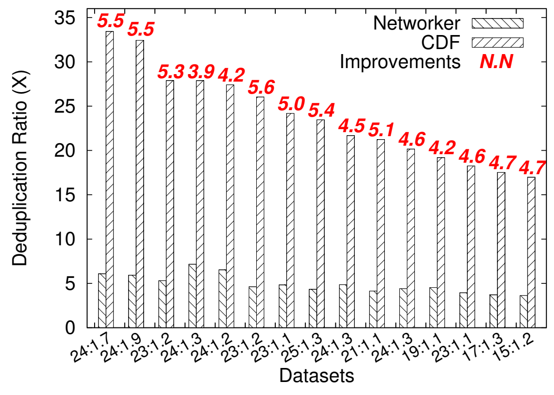

Figure 1: Deduplication ratios for workstation backups, using the original *NetWorker* and deduplication-friendly CDF formats. X-axis labels are X:Y, where X is the number of full backup generations analyzed and Y is the average internal deduplication ratio. Hosts were sorted by the CDF deduplication ratio. The number above each bar shows the improvement due to CDF.

backups use different formats. The format is proprietary, but we discuss it here in general terms.

For disk-based file systems, the *NetWorker* save format includes these fields, among others: internal file identifier, file name, offset and size, and file attributes. These metadata fields precede the data of each saved file, and some of these fields are unfriendly to deduplication. In particular, the internal file identifier is a monotonically increasing sequence number, so adding a file to a directory shifts the sequence number of every file that follows, and the blocks are no longer identical. Attributes such as timestamps can thwart deduplication; we discuss this further in the context of *tar* in Section 3.

We have designed a new Common Data Format (CDF), which separates data from metadata. The metadata of all files is grouped together and stored in one section, where it references file data stored in another section. This separation has a substantial impact on deduplication, and as a result, EMC's backup software products are migrating to this new format.

We evaluated CDF by estimating the deduplication across the backups of 15 hosts using content-defined chunks (8 KB average, 4 KB min, 12 KB max). Fingerprints are checked using Bloom filters. This dataset is a subset of the *workstations* dataset used by Douglis, et al., in an earlier study [2]; here we have fewer workstations and evaluate deduplication ratios (defined as $\frac{original\_size}{deduplicated\_size}$) only for the full backups. There were 15–25 backups per workstation, totalling up to about 420 GB. The more backups there are for a host, the higher its deduplication ratio is likely to be, since the same data may appear more times. In addition, some hosts have remarkably high *internal* deduplication: the fraction of data within even a single backup that is eliminated by deduplication with other data in the same backup.

Figure 1 shows the deduplication within each of the datasets using the original *NetWorker* data format and the deduplication-friendly CDF format. Since both the number of backups and the internal deduplication affect the overall deduplication ratio, these are shown (colon-separated) as the x-axis labels for each host. (We ignore traditional LZ compression, which is applied after deduplication.) The datasets are sorted in decreasing order of deduplication.

The backups in the original format deduplicated rather poorly, with deduplication ratios of 3.6–6.1$\times$ even with over 20 backups stored. Moving to the CDF format produces deduplication ratios from 17.0–33.4$\times$, with an average improvement (shown in red) of 4.9$\times$. Many systems had aggregate deduplication better than simply finding the same data once in each backup. We attribute this to these workstations being engineering workstations containing multiple copies of certain data such as source code, leading to internal deduplication as high as 1.9$\times$.

In addition we found that inter-host duplication reported in the earlier study [2], using the original *NetWorker* format, understated the available deduplication. For instance, the most content in common across two hosts reported in that study was 74% of one host's chunks, but when considering only the data without the impact of metadata, it rose to 93%.

3 Application-level Post-processing

Next, we look at *tar* as an example of an application that has a well-defined data format that is 1) unfriendly to deduplication; and 2) in wide use for decades, and is thus hard to change for compatibility reasons. For this class of applications, we propose post-processing as a way to de-interleave data and metadata, improving deduplication.

3.1 *tar*

tar [6] was initially designed for archival storage on magnetic tapes, and the format was optimized for sequential IO. A *tar* file is a sequence of entries, one per file, each containing a file header and data blocks. The file header includes metadata for that file, including its path, ownership and modification time. The *tar* program works in 512-byte blocks. Thus, each file entry consists of one header block[1] and many data blocks. File headers are placed immediately before the corresponding data blocks to avoid multiple "seeks" when extracting a single file. An illustration of the *tar* format is presented in the top half of Figure 2.

While the *tar* format works well for tape backups, it is now also commonly used to store and distribute source

[1]Multiple header blocks are used when the file path is too long to fit in a single header block.

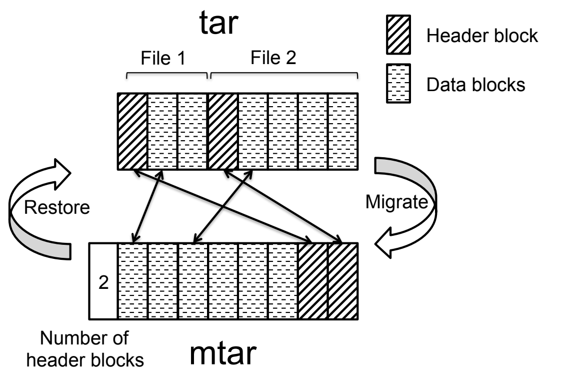

Figure 2: The *tar* and *mtar* formats and transformations between them.

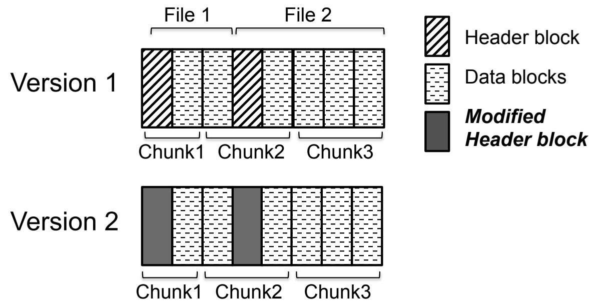

Figure 3: In *tar*, changes in header blocks lead to many new unique chunks. Chunk1 and chunk2 in version 2 are different from those in version 1 because of changes in header blocks.

code, binaries, and disk-based backups. *tar*'s decision to place metadata and data together in its file format, however, interacts poorly with deduplicating storage. Specifically, we found that when storing *tar* files of multiple releases of source code, we were only able to achieve deduplication ratios of about 2×; if we simply concatenated the files (thus throwing out the metadata), we were able to achieve deduplication ratios of up to 18× (with an average chunk size of 8 KB). The *tar* format clearly interferes with deduplication.

The underlying reason is that metadata changes more frequently than data blocks: we found that the modification time for the same file in two consecutive releases of Linux source code distributions is different, even when the file's content remains the same. By mixing more frequently changing metadata with data blocks, the *tar* format unnecessarily introduces many more unique chunks. An illustration of this problem is presented in Figure 3.

3.2 Migratory *tar*

We propose a new *Migratory tar* format (*mtar* for short), in which we separate metadata from data blocks by co-locating metadata blocks at the end of the *mtar* file. Changes in metadata are localized and isolated from data blocks, enabling better deduplication of the data.

An *mtar* file can be created by *migrating* a *tar* file. Specifically, we scan a *tar* file, output all data blocks to the *mtar* file and all header blocks to a temporary file, and then concatenate the two.[2] We store the offset of the metadata block in the first block of a *mtar* file for efficient access. To get back the original *tar* file, a *restore* operation reads the first block, finds the first header block, reads all data blocks for that file starting from the second block and outputs it. This process is repeated for every file, resulting in re-creation of the original *tar* file. This dynamic reorganization of the *tar* file is similar to migratory compression (MC) [10]. An illustration of the *mtar* format and the migrate and restore operations are shown in Figure 2.

mtar works best when a *tar* file includes many small files, because metadata interleaves with data more frequently. This is generally true for source code distributions, which we evaluate. For *tar* files that include mostly large files, we expect less benefit from *mtar*. We implemented *mtar* by extending GNU *tar* version 1.27.1 (the extension is available at `https://github.com/xinglin/mtar`).

Software	Versions	Size (MiB)
automake	64	304.72
bash	23	276.69
coreutils	37	1284.49
fdisk	13	21.61
gcc	68	20315.45
gdb	32	4004.77
glibc	43	3811.48
smalltalk	33	685.39
tar	21	219.86
linux	308	98444.58

Table 1: Datasets for evaluating *mtar*

3.3 Evaluation

To evaluate *mtar*, we use source code distributions of the Linux kernel and 9 GNU software packages from `ftp://ftp.gnu.org/gnu/`. For each package, we examine deduplication for multiple released versions. The software packages are shown in Table 1.

For each dataset, we download compressed *tar* files and decompress them. We remove padded blocks at the end of each *tar* file,[3] then use our modified *tar* program to convert each *tar* file into an *mtar* file. We compared deduplication for *tar* and *mtar* using the *fs-hasher* tool, released by Stony Brook University [4]. We use variable-size chunking, with 8 KB as the average chunk

[2]Putting metadata at the end of the *mtar* file allows us to make a single pass over the input *tar* file: the amount of metadata and data cannot be known without reading the entire file, and appending the smaller metadata to the larger data is more efficient than the reverse.

[3]*tar* does IO in fixed multiples of blocks, called records. It pads the last few blocks to be a full record.

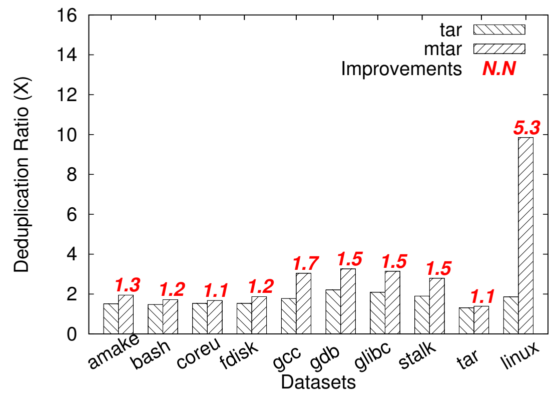

Figure 4: Deduplication ratios for *tar* and *mtar*. The number above each bar shows the improvement due to *mtar*.

size (4 KB min, 16 KB max). An MD5 hash value is generated for each chunk; note that this is not collision-resistant but for our analysis an occasional error is unimportant. We compare bytes in unique chunks to bytes in all chunks.

Figure 4 shows that *mtar* improves deduplication ratios over *tar* by 1.1–5.3×. Using a 2 KB average chunk size, with the same ratio for the minimal and maximal chunk sizes, *mtar* achieves 1.1–3.3× improvements. For a 32 KB average chunk size, the improvements range from 1.1–6.2×. These results show *mtar* improves deduplication significantly.

Next, we examined deduplication ratios for metadata blocks and data blocks separately. We found that metadata blocks have no duplication, while data blocks show high deduplication ratios: up to 16.62× for Linux. Deduplication ratios for data blocks are close to our earlier experiment which simply concatenates source files, showing that *mtar*'s improvement over *tar* comes from increased deduplication among the data blocks. It is interesting to note the effect on stored (post-deduplication) blocks: as *mtar* improves the deduplication ratios for data blocks, the fraction of stored blocks that consist of undeduplicatable metadata blocks increases significantly. For the Linux kernel, before deduplication, 95.61% are data blocks and only 4.39% are metadata blocks. After deduplication, the unique data blocks become 5.75% (16.62× deduplication ratio) while we still have the same metadata blocks. While the percentage of metadata blocks is small in the original data, the weight becomes much more significant after deduplication: here, 43.3% of post-deduplication storage comes from metadata($\frac{4.39}{(4.39+5.75)}$). Future work should also study how to store metadata efficiently.

4 Format-aware Deduplication

Formats that are not designed with deduplication in mind may needlessly degrade deduplication effectiveness. If it is not possible to change the data format or post-process prior to writing it to storage, then the storage system needs to understand and address the effects on the fly. Here we describe two examples of format-aware deduplication in EMC *Data Domain* appliances [20].

One of the earliest data types requiring special handling was the existence of "block markers" intended for magnetic tapes. When using a disk-based system to emulate tape, the incoming stream for a "virtual tape library" (VTL) device continues to periodically include block markers with a special bit pattern. Since these block markers appear at fixed intervals, a shift in content results in the marker appearing at a different point within a chunk, and the chunk does not deduplicate. Worse, the block markers can also contain variable metadata, even preventing deduplication of unmodified data. Data Domain addressed this by allowing the system to scan for block markers while performing chunking and fingerprinting. If one is identified, then it is removed from the content and stored in a separate location, thus the fingerprints of the remaining data are unaffected. Upon a read, the marker is inserted at the specified offset to restore the original data. Marker handling can have significant impact: for example, one customer saw deduplication improve from 9.9× to 16.8× with proper treatment of the interspersed metadata (a 70% improvement).

Another data type requiring special handling arises in Oracle RMAN backups [13]. RMAN writes fixed-sized blocks, configurable between 2KB and 64KB, each containing a block header and footer. Portions of the header and footer can change from backup to backup even when the block data remains unchanged, due to internal Oracle data formats. Additionally, RMAN may multiplex multiple data files together into a backup and the ordering of multiplexing can change; this affects content-defined chunk creation. Without special handling, therefore, deduplication would be degraded due to the modified metadata in the header and footer and the possibility of a change to the order in which files were multiplexed.

To solve the above problem, Data Domain modified the system to use block headers to delineate Oracle blocks as the (fixed-size) unit of deduplication and to remove portions of the block header and footer similar to tape marker handling: the variable portion is stripped out and stored as a small inlined data unit within the file system metadata, and the remaining content is fingerprinted. By doing this deduplication becomes impervious to multiplexing order or changes isolated to the header.

In experiments doing 6 backups of an Oracle database with a 5% change rate between backups, we observed a 1.2–2× improvement in deduplication (4.5→5.28× without multiplexing and 2.48→5.07× with multiplexing). Thus removing block headers restores the deduplication to that expected from the underlying data.

While we see promising results by making deduplication systems aware of data formats, this approach has a significant drawback, requiring deduplicating appliance manufacturers to track a moving target. Changes to the format cause deduplication to drop, requiring engineers to address the change. This results in a cycle of analysis, implementation, testing, and finally a patch release—a process that can take substantial time and effort.

5 Related Work

Most work in the deduplication space has focused on improving write throughput (e.g., Data Domain [20] and Sparse Indexing [9]), with a significant recent effort on improving restore performance [5, 8]. However, little work has been done to examine the impact of input data in deduplication. The closest work to both RMAN block special handling and *mtar* is a poster proposing a *tar*-format aware chunking algorithm [17]. To prevent the interference from metadata blocks, Sung, et al., partition every header block as a deduplication chunk while in our *mtar* approach, we group header blocks together and separate header blocks from data blocks. Their approach requires changes in the chunking algorithm for existing deduplication systems, to make them aware of the tar format. *mtar* does not require any changes. Their chunking algorithm produces small chunks in the tar block size (512 bytes). This could break minimal chunk size requirements; in addition, small chunks dramatically increase deduplication system metadata overhead [18].

MC [10] has similarities to *mtar*: both reorganize data to improve space efficiency, but *mtar* uses knowledge of the *tar* format to improve deduplication while MC rewrites generic data to improve traditional compression.

6 Conclusion

We have examined the effect of metadata on deduplication effectiveness. When metadata changes frequently over time, it is essential to separate it from data that stays more stable and would otherwise deduplicate. This separation can occur within the deduplication process, but that leads to complexity as well as dependencies on both data formats and deduplication environments. It can be done as a post-processing step, which makes the benefits more generic: any deduplication back-end can benefit from the conversion process, but the post-processor still must closely track the input format. Designing a data format to be deduplication-friendly has the best benefits of all, as the application improves deduplication in a platform-independent manner while isolating the storage system from the data format.

Acknowledgments

We thank Yamini Allu, Abhinav Duggal, Kai Li, Vladimir Mandic, Philip Shilane, and others in Data Domain engineering for comments and for their work reported here. We thank our shepherd, Dan Ports, and the anonymous reviewers for their insightful feedback.

References

[1] Bolosky, W. J., Corbin, S., Goebel, D., and Douceur, J. R. Single instance storage in windows 2000. In *Proc. of USENIX Windows Systems Symposium* (2000).

[2] Douglis, F., Bhardwaj, D., Qian, H., and Shilane, P. Content-aware load balancing for distributed backup. In *Proc. of LISA* (2011).

[3] Dubnicki, C., Gryz, L., Heldt, L., Kaczmarczyk, M., Kilian, W., Strzelczak, P., Szczepkowski, J., Ungureanu, C., and Welnicki, M. Hydrastor: A scalable secondary storage. In *Proc. of FAST* (2009).

[4] File systems and Storage Lab from Stony Brook University. fs-hasher. `http://tracer.filesystems.org/`. Retrieved March 9, 2015.

[5] Fu, M., Feng, D., Hua, Y., He, X., Chen, Z., Xia, W., Huang, F., and Liu, Q. Accelerating restore and garbage collection in deduplication-based backup systems via exploiting historical information. In *Proc. of USENIX ATC* (2014).

[6] gnu tar. Basic tar format. `http://www.gnu.org/software/tar/manual/html_node/Standard.html`.

[7] Jin, K., and Miller, E. L. The effectiveness of deduplication on virtual machine disk images. In *Proc. of SYSTOR* (2009).

[8] Lillibridge, M., Eshghi, K., and Bhagwat, D. Improving restore speed for backup systems that use inline chunk-based deduplication. In *Proc. of FAST* (2013).

[9] Lillibridge, M., Eshghi, K., Bhagwat, D., Deolalikar, V., Trezise, G., and Camble, P. Sparse indexing: Large scale, inline deduplication using sampling and locality. In *Proc. of FAST* (2009).

[10] Lin, X., Lu, G., Douglis, F., Shilane, P., and Wallace, G. Migratory compression: Coarse-grained data reordering to improve compressibility. In *Proc. of FAST* (2014).

[11] Meyer, D., and Bolosky, W. A study of practical deduplication. In *Proc. of FAST* (2011).

[12] Muthitacharoen, A., Chen, B., and Mazières, D. A low-bandwidth network file system. *SIGOPS Oper. Syst. Rev.* (2001).

[13] Oracle Corp. Database backup and recovery user's guide. `http://docs.oracle.com/cd/E11882_01/backup.112/e10642/`. Retrieved March 9, 2015.

[14] Quinlan, S., and Dorward, S. Venti: A new approach to archival data storage. In *Proc. of FAST* (2002).

[15] Smaldone, S., Wallace, G., and Hsu, W. Efficiently storing virtual machine backups. In *Proc. of HotStorage* (2009).

[16] Srinivasan, K., Bisson, T., Goodson, G., and Voruganti, K. idedup: Latency-aware, inline data deduplication for primary storage. In *Proc. of FAST* (2012).

[17] Sung, B., Park, S., Oh, Y., Ma, J., Lee, U., and Park, C. An efficient data deduplication based on tar-format awareness in backup applications. In *FAST* (2013). Poster.

[18] Wallace, G., Douglis, F., Qian, H., Shilane, P., Smaldone, S., Chamness, M., and Hsu, W. Characteristics of backup workloads in production systems. In *Proc. of FAST* (2012).

[19] ZFS. `http://en.wikipedia.org/wiki/ZFS`.

[20] Zhu, B., Li, K., and Patterson, H. Avoiding the disk bottleneck in the data domain deduplication file system. In *Proc. of FAST* (2008).