Progressive Deduplication Technology

Whitepaper

arkeia® SOFTWARE

Contents

Executive Summary

Managers of network backup solutions seek to minimize the time required for backups and restores, as well as the cost of maintaining many restore points (i.e. materialized snapshots). Optimum data protection solutions have evolved significantly in the last ten years as tape backup targets have given way to disk backup targets, and file system source data have given way to volume-image backups. These latter were made important by the growth of virtualized environments. The growth of cloud storage brings new challenges for storage managers.

This white paper reviews the evolution of deduplication technology and compares traditional fixed- and variable-block technologies with Arkeia Software's newer Progressive Deduplication™ technology. Progressive Deduplication is superior to fixed-block and variable-block deduplication for reasons of speed and compression:

Higher Compression

- Progressive Deduplication identifies known blocks anywhere in a file. Unlike variable block technology, promoted by Quantum and Data Domain, Progressive Deduplication does not depend on file contents to define block boundaries.
- Progressive Deduplication uses different block sizes for different types of files. Optimal block sizes deliver maximum compression for each type of file. For example, the optimal block size for Microsoft® PowerPoint® files is 2,048 bytes; for essentially random data, like MPEG files, the use of sub-file-size blocks does not improve compression results.

Faster Compression

- Progressive Deduplication delivers the same performance as fixed-block deduplication for known files and for sections of unknown files that have been previously scanned. These two conditions account for the vast majority of data encountered during backup.
- Progressive Deduplication accelerates comparisons of new blocks to known blocks. Light-weight algorithms detect probably matches and heavy-weight hashes confirm that blocks are identical. This two-phase approach is called "progressive matching".

Arkeia's Progressive Deduplication was developed by mathematicians expressly for network backup. Rapid compression of backup sets is important to network backup for three primary reasons. First, source-side deduplication accelerates backups by reducing the amount of data that must traverse a LAN or SAN. Second, reduced storage requirements for backup sets allow more restore points to be maintained with the same volume of storage. Third, smaller backup sets make cost-effective the replication of backup sets to cloud storage.

As organizations adopt disk backup to replace tape backup, public and private clouds permit secure off-site protection of data without the cumbersome and expensive transport of physical media. The key enabling technology for cloud backup is data deduplication because the high cost of moving data over a WAN.

Progressive Deduplication technology was developed by Kadena Systems of Santa Clara, California. The first patent for Progressive Deduplication was filed in 2004 and granted in 2008. Kadena was acquired in November 2009 by Arkeia Software.

What Is Data Deduplication?

Data deduplication is a kind of "global compression" that looks for commonality across multiple files. By only having to store content once, content that is common across multiple files can be dramatically compressed to reduce storage requirements. Global compression is distinct from "local compression" and its more familiar algorithms. Local compression strategies target the compression of individual files, and use algorithms like DEFLATE (i.e. "zip"), JPEG, or MPEG.

Global compression is especially relevant to backup because

1. Multiple computers within an enterprise often contain similar data (similar documents, similar operating system files, similar databases), and

2. Successive backups of the same computer often contain data that differ only slightly from the previous version of the data.

The strategies that underlie local compression and deduplication are often similar. Generally, both identify byte sequences that are repeated. Rather than store the byte sequence more than once, the second and subsequent instances are recorded by a reference to the first instance.

Compression Type	Compression Target	Examples
Local	Files and Blocks (files and smaller)	DEFLATE (LZ77, LZ78) JPEG MPEG
Global	Files and File Systems (files and larger)	Deduplication, Byte-differencing

Figure 1 – Comparison of local and global compression characteristics

"Data deduplication" is a data compression technique made possible by the invention in the 1980s of message digest hashes that create a "signature" for a block or file. If two signatures are equal, their corresponding blocks are considered equal (with a probability that exceeds random errors in semiconductor memory).

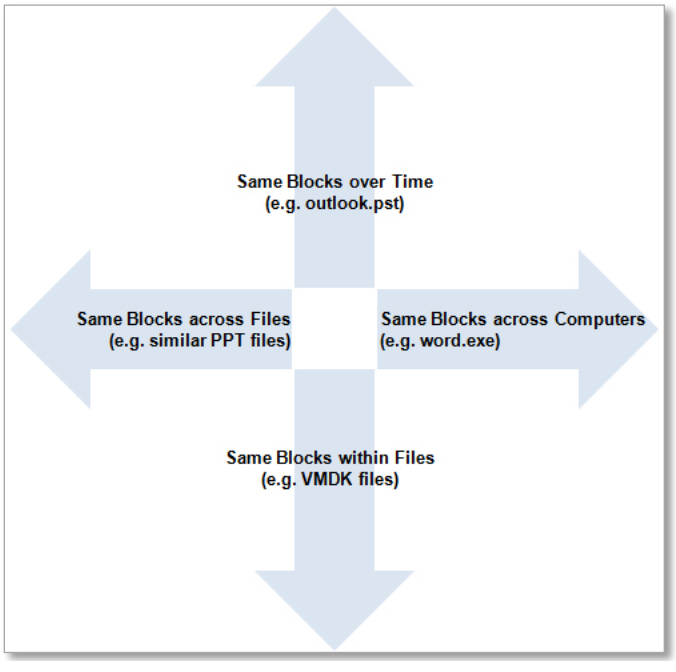


Figure 2 - Factors that improve deduplication ratio

The basic benefit of deduplication over local compression derives from scope. *The larger the scope of the deduplication, the better the aggregate compression ratio.* Deduplication of content on a single Windows 2008 Server can never deliver compression ratios as high as the deduplication of content across multiple Windows 2008 Servers.

Further, deduplication of a snapshot at one moment in time can never deliver compression ratios as high as the deduplication of multiple snapshots over time of the same disk. The commonality of content across similar computers and snapshots causes compression ratios to rise with the scope of the data deduplication.

The benefits of deduplication grow as the factors that improve the deduplication ratio grow. (See Figure 2.) While the benefit of the "Same Block over Time" is special to backup, the other three factors are relevant to both primary and secondary data storage.

Traditional Data Deduplication Strategies

Data deduplication has been used to reduce data volumes (both for storage and transmission) since the 1990s. The open source utility, rsync, employs data deduplication to reduce the bandwidth needed to synchronize a file across a network. The application of data deduplication was first popularized in data protection by Data Domain and in WAN performance optimization by Riverbed.

The first element of data deduplication is the use of message digest hashes as a substitute for byte-by-byte comparisons of data. If the hashes are equal, the data is considered to be equal. The probability of a hash collision drops as the size of the hash increases. Several popular hashes are listed to the right.

Hash Algorithm	Hash Length (bytes)
MD-5	16
SHA-1	20
SHA-2	32
SHA-2	64

Figure 3 – Examples of hash algorithms

The second element of data deduplication is the grain of deduplication and the strategy for breaking large data sets (e.g. streams, files) into smaller chunks. New strategies for dividing a file into smaller chunks has been the focus of innovation in data deduplication over the past decade.

Deduplication Challenges. Another perspective on the challenges of deduplication is shown below in Figure 4. The table presents four progressively difficult deduplication scenarios. The technologies developed to attack the four scenarios are discussed in detail in the following subsections.

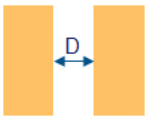
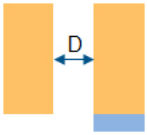
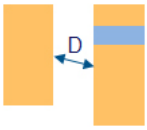
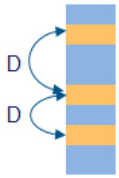
Scope of deduplication	Enabling innovation	Example	Duplication
Identical files across file systems	File-grain deduplication: Single Instance Storage	Operating system files	
Above + Similar content across file versions (appended data)	Block-grain deduplication: Fixed-block	Outlook files (.PST)	
Above + Similar content across file versions (inserted data)	Block-grain deduplication: Variable-block	Powerpoint files (.PPT)	
Above + repeated content within a disk image	Block-grain deduplication: Progressive Deduplication	Volume images (VMDK)	

Figure 4 –Deduplication scenarios

Single-Instance Storage. Deduplication began with what was called “single-instance storage” (SIS). If multiple files shared identical content, that content would only be recorded once. All subsequent “copies” would simply reference the first copy of the content. Files can have the same content even when file names, dates, permissions, and other metadata differ. SIS was the first technology to use hashes to compare file content, making byte-by-byte comparisons unnecessary. If the hashes of two files are equal, the files are considered to be the same.

SIS was especially useful for managing file duplicated due to email distribution or for managing multiple instances of the same operating system on a network of computers. The shortcoming of SIS for backup is that, if a single byte in the file is modified, the entire file must be recorded again separately.

Fixed-Block Deduplication. Fixed-block deduplication divides a file into any number of blocks of a fixed size, where blocks are typically between 4kB and 64kB. Rather than comparing the hashes of entire files, each non-overlapping block is hashed separately.

Block-grain deduplication requires more processing than file-grain deduplication, but delivers large benefits when large files grow by having data appended to the file, or when clusters of bytes are modified between backups or across computers. Examples of large datasets undergoing small changes are email repositories (e.g. Microsoft Outlook .PST files), log files, and some kinds of databases.

The major shortcoming of fixed-block deduplication is its inability to manage files when data are pre-pended to or inserted into a known file. The boundaries of known blocks following the insertion don't occur at the fixed block boundaries and fixed-block deduplication won't identify them. See Figure 10.

Variable-Block Deduplication. Variable-block deduplication uses pre-determined byte sequences in a file to define block boundaries. As a result, variable-block dedupe, unlike fixed block dedupe, can identify file changes that are modified, appended, and inserted. When data are inserted, the byte sequences that define block boundaries shift as the file contents shift.

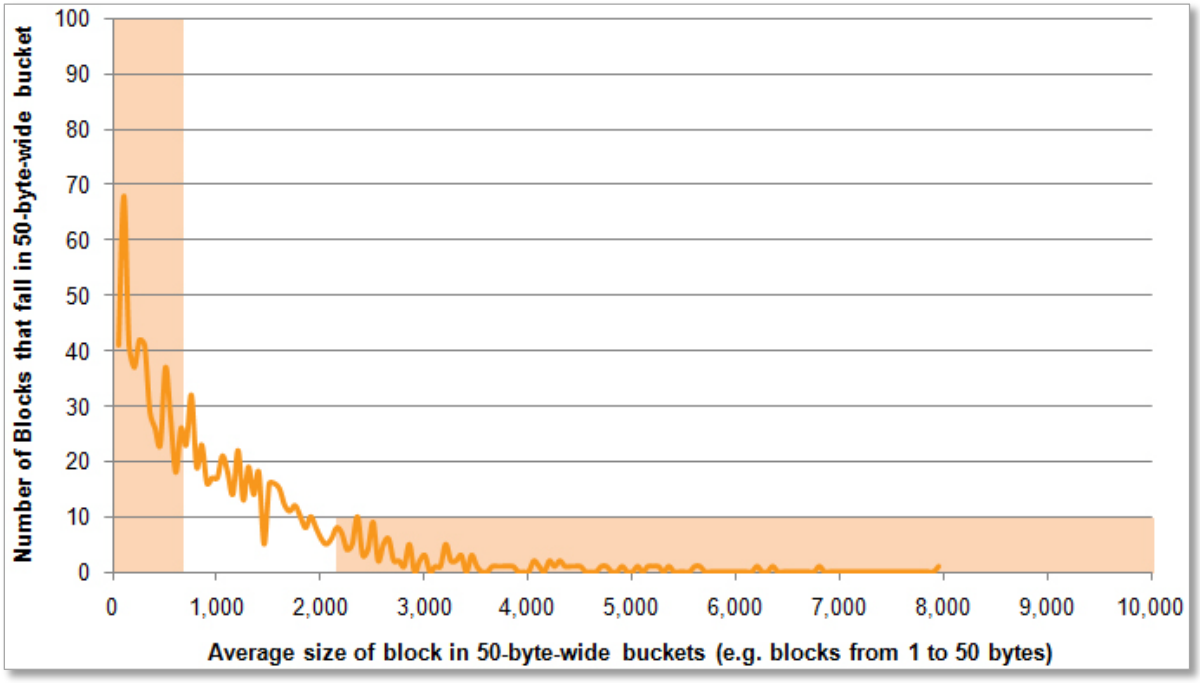


Figure 5 – Example of distribution of variable block sizes for a 1MB file with an average block size of 1kB

Variable-block deduplication always requires more processing than fixed-block deduplication because the whole file must be scanned, one byte at a time, to identify block boundaries. If file data is randomized before the scan for block boundaries, still more processing resources are required.

This performance problem is exacerbated because variable-block deduplication does not allow simple control of block sizes. While *average* block sizes can be defined, the distribution of block sizes is very broad, as shown by an example in Figure 5. Block sizes can range from one byte to the length of the entire file, complicating block storage and requiring special handling of mal-sized blocks.

How broadly are block sizes distributed? For an average block size of “n”, the probability distribution of block size “x” in variable-block deduplication is:

$$\rho(x) = \frac{\left(1-\frac{1}{n}\right)^{x}}{n}$$

A representative case for a 1MB file with a target “average block size” of 1kB is shown in Figure 5. Note that:

- 53% of blocks by number are **shorter than half** or **longer than twice** the average block size
- 50% of blocks by volume are **shorter than half** or **longer than twice** the average block size

Very small blocks are impractical for dedupe because the cost of maintaining references to the very small blocks outweighs the benefits of omitting the duplicated data. Very large blocks are impractical because their large size limits their usefulness to deduplicate similar files—approaching the inefficiency of file-grain deduplication. As a result, special accommodations are needed to deliver a useful result and this increasing complexity slows performance.

Progressive Deduplication. Progressive Deduplication combines the best features of fixed-block and variable-block deduplication. Like fixed-block dedupe, Progressive Deduplication is fast because fixed-size blocks simplify data management. And, as will be discussed later, fixed-size blocks make practical content-aware dedupe.

Like variable-block dedupe, Progressive Deduplication delivers high compression because it can tolerate “data inserts” into files, as well as “data appends” and “data modifies”. Block boundaries can be present anywhere in a file. However, unlike variable-block dedupe, known content is processed at the speed of fixed-block dedupe.

The following sections describe Progressive Deduplication in greater detail.

Progressive Deduplication Overview

Progressive Deduplication is an alternative to traditional deduplication technologies. The technology can be deployed in a variety of combinations. These options, and those selected for Arkeia Network Backup, are outlined below:

Deduplication Characteristic	Available Options	Options adopted in Arkeia Network Backup
Timing	In-line or Post-processed	In-line
Location	Source-side or Target-side	Either
Scope	Per-machine or Across-machines	Across-machines

Figure 6 – Deduplication Characteristics

Some of these characteristics impact others. For a backup application where agents must be installed (either ephemerally or statically) on a client computer, deduplication on that client (i.e. source-side deduplication) is clearly a better alternative to target-side deduplication. First, source-side dedupe accelerates backups by compressing backup sets before sending them across a local network to the backup server or media server. Second, source-side dedupe distributes the deduplication load across all the clients being backed up. This scale-out architecture is much more cost-effective than a scale-up architecture which concentrates the deduplication load on one or a handful of media servers.

For backup applications, source-side deduplication implies in-line processing. If the client computer will perform the deduplication, there is no reason to touch the data more than once.

Setting Block Boundaries. Progressive Deduplication is different, first, because its strategy for choosing block boundaries is different. Fixed-block deduplication postulates a block size for a deduplication repository, typically 16kB or 32kB, and segments all files into non-overlapping blocks of this size. The fixed-size blocks make it impossible to accommodate data inserts. Variable-block deduplication scans a file, looking for pre-defined byte sequences. The scanning process and broad range of block sizes slows the deduplication process.

Like fixed-block dedupe, Progressive Dedupe divides a file into blocks, where every block that makes up the file is the same size. Unlike fixed block dedupe, Progressive Dedupe allows blocks to overlap—which in turn allows block boundaries to occur at any location. This behavior permits Progressive Dedupe to tolerate byte inserts into files, recognizing blocks that have been shifted down in a file as a result. The “negative compression” of overlapped blocks is greatly outweighed by the advantage of tolerating inserted bytes.

Optimum Block Size for Each File Type. Progressive Deduplication is different, second, because it assigns block sized based on file type. Unlike fixed-block dedupe which uses the same block size across all types of files, Progressive Deduplication assigns different block sizes (e.g. 1k, 2k, 4k, 8k, 16k, and 32k bytes) to different types of files. This strategy has a large impact on compression rates because different file types are compressed to different degrees with different block sizes. Read about block size optimization in the next section.

Sliding Windows With Progressive Matching. Progressive Dedupe uses a sliding window to evaluate the potential of duplicate blocks at *every* byte location in a file. If a known block appears anywhere in a file, Progressive Dedupe will find that duplication. The challenge of this approach is the computational load it imposes if naïvely implemented.

A naïve implementation might use a hash to establish equality of blocks. For a block size of B bytes and a file size of F bytes, a naïve implementation means that the application must compute as many as $\mathrm{F} - \mathrm{B} + 1$ hashes. This compares unfavorably to $\frac{F}{B}$ hashes for a fixed-block strategy. Because hash calculations are computationally expensive, calculating $\mathrm{F} - \mathrm{B} + 1$ hashes for a file of F bytes is impractical for all but the smallest files. As a result, a naïve implementation of a sliding-window strategy is unlikely to enjoy good performance.

Arkeia's implementation is different. Arkeia uses a speedy, light-weight algorithm to determine if data under the sliding window is a probable match to blocks in the known-block-pool. Probable matches are scrutinized with a heavy-weight hash algorithm. Because over 99% of probable matches prove to be exact matches, progressive matching is extremely efficient. Arkeia's patented "progressive matching" technology inspired the name "Progressive Deduplication."

This two-phase matching strategy is illustrated in Figure 7. Critical to the success of this strategy is a light-weight algorithm that has a very low rate of false positives. The throughput of this strategy is high for two reasons. First, because source-side deduplication distributes the processing load across all clients, CPU is light and per-client hash caches are optimal for each client, accelerating hash lookups.

Second, the "sliding window" only slides when files (or streams) are unknown. If a known file is encountered, the performance of the sliding window algorithm is comparable to the performance of fixed-block deduplication. Sliding is required only if unknown data falls under the sliding window.

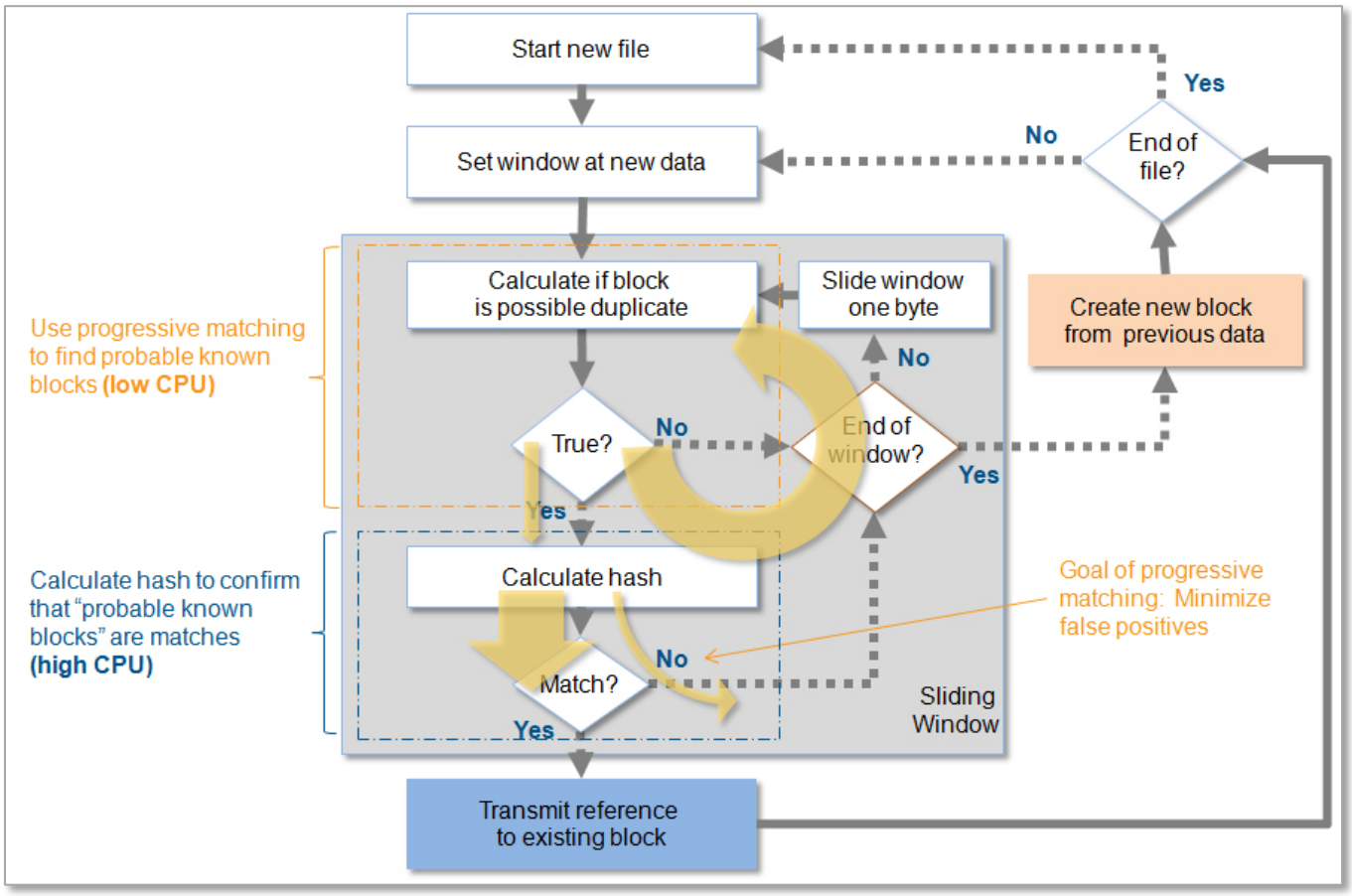


Figure 7 – Progressive matching strategy for determining block equivalence

Choosing Block Sizes Based on File Type

Traditional fixed-block data deduplication, where all the blocks in a group of files are the same size, offers one compelling advantage: managing blocks of fixed size simplifies block storage, indexing, and replication. This is the same management efficiency reason for which file systems divide files into fixed-size blocks.

The missed opportunity of this approach is that different types of data are more efficiently deduplicated with different sized blocks. Using the optimum deduplication block size of block for every type of file improves compression rates.

Empirical Evidence. The following chart illustrates the missed opportunity. If hundreds of files of the same file type are deduplicated at different block sizes, different compression rates result. A representative example is the case of 1,322 real-world Microsoft Powerpoint (PPT) files on a single computer. Compression peaks at 3.5x at 1k-byte blocks, but drops to 2.4x at a block size of 8k bytes. Using the optimum block size improves compression by almost 50%. The compression rates of many types of files follow this peaking pattern.

For the case of image files compressed with MPEG (Motion Picture Experts Group) algorithms, compression rates improve as block sizes grow. This pattern is true of all files with largely random data: the bigger the block size the better. Single-instance storage, rather than block-grain dedupe, is generally the best strategy for files of these types.

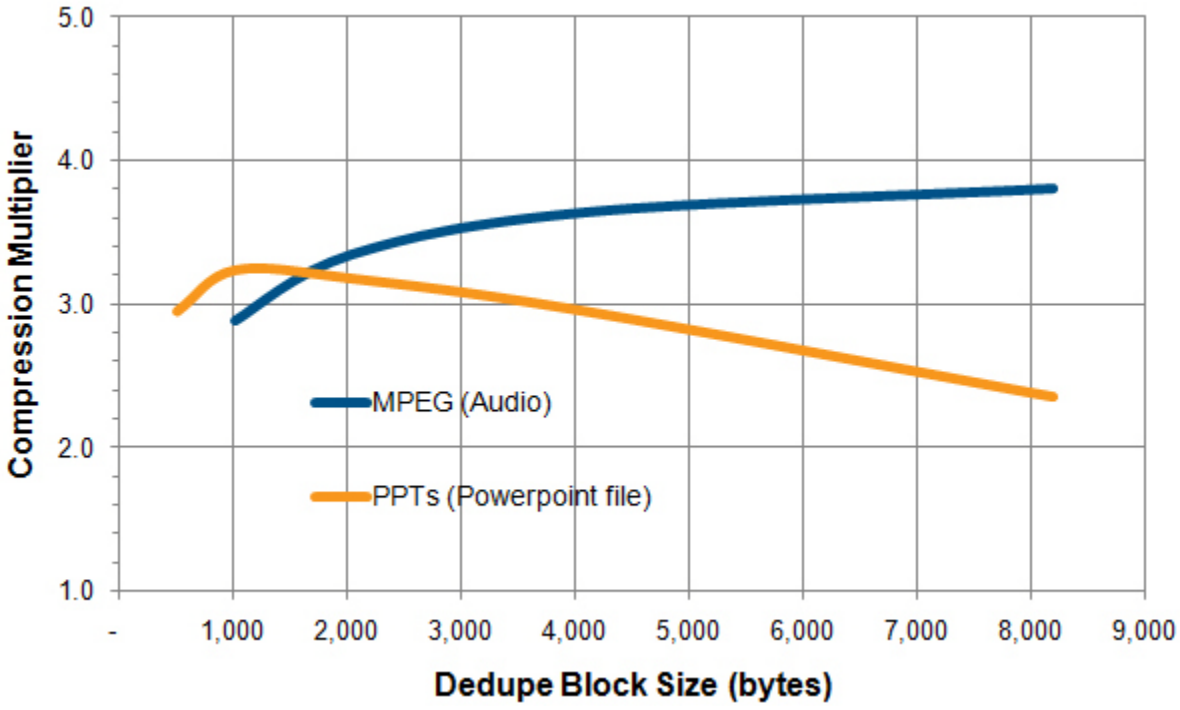


Figure 8 – Examples of varying compression multipliers at various dedupe block sizes

Rationale. What accounts for these curve shapes? Three primary factors explain these empirical results.

1. Clearly, the smaller the deduplication block size, the more likely that block is to be repeated in a collection of files. One is more likely to find multiple instances of “the” than “The quick brown fox jumps over the lazy dog”.
2. However, when blocks become too small, the metadata overhead to keep track of every block instance dominates data storage. The dedupe metadata outweighs the benefits of deduplication.

3. Finally, the efficiency of "local compression" (e.g. DEFLATE) of individual blocks improves as block size grows. A text block of 32k bytes will enjoy better local compression than a text block of 1k bytes for the same reason that larger sets of files will generally enjoy better deduplication than smaller sets of files.

This behavior is schematized below in Figure 9.

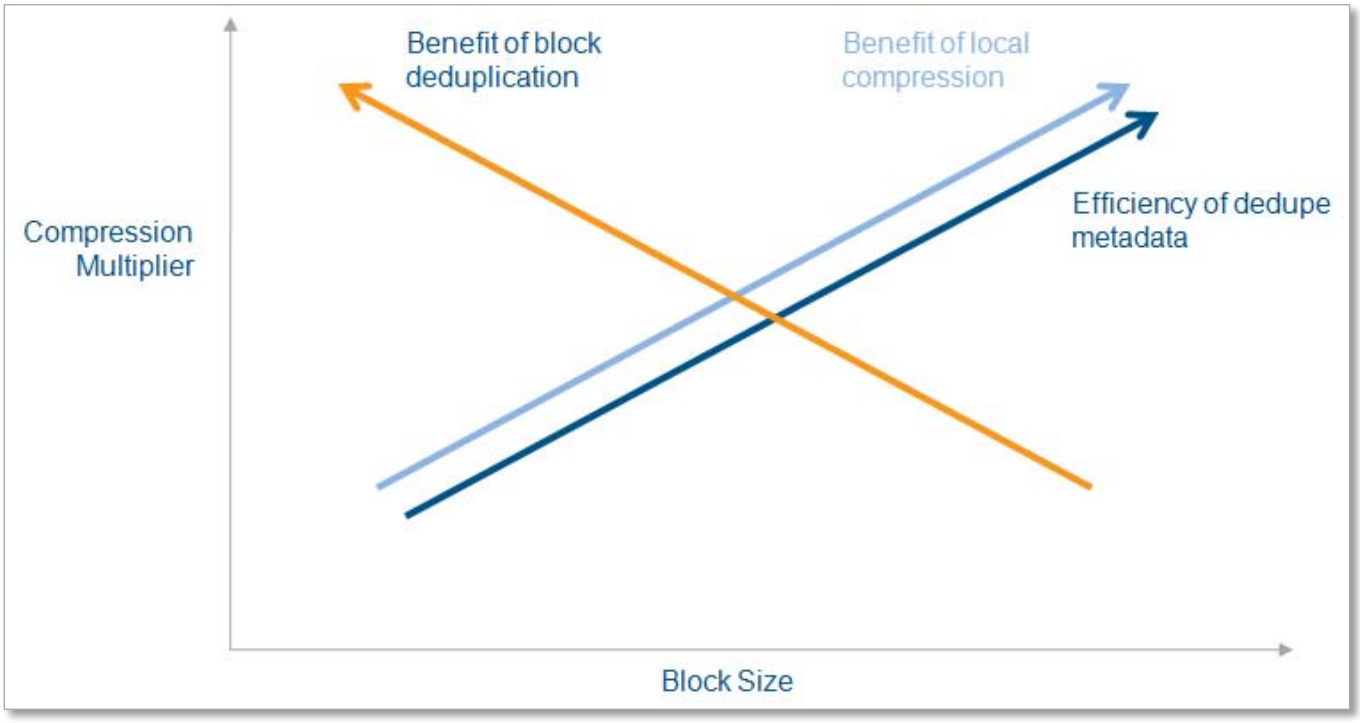


Figure 9 – Impact on compression multipliers of various dedupe block sizes

Impact. Different types of data enjoy optimal data compression at different block sizes. Based on analyses by Arkeia Software of customer backup sets, most optimal block sizes in "peaked" distributions occur between 1kB and 32kB. Files composed of random data do not benefit from block-grain deduplication and are best managed as indivisible entities. Examples of files composed of random data are:

1. compressed files, such as files already compressed with DEFLATE, JPEG, MPEG or other algorithms; the more complete the local compression of the file, the more random the resulting file;
2. encrypted files
3. scientific data whose patterns are hidden behind a random façade.

The Importance of Data Insertion

The time dimension is of special importance to backup, where a basic requirement is to be able to rewind the state of an individual file—or an entire hard drive—to a specific point in the past. As a result, the same files are backed up repeatedly as they change, either programmatically or by human editing. Understanding how files change is important to understanding how a series of file backups, or disk image snapshots, can be efficiently compressed.

Generally, files change in three ways. Each application changes files in some combination of these three manners:

1. Bytes in a file are modified. The size of the file is constant, but byte values change. Practically, this type of change is rare.
2. Bytes are inserted or deleted from the middle of a file. These changes result in a file of a changed size.
3. Bytes are appended to a file or deleted from the end of a file. This is a special case of "inserting" or "deleting" bytes, but a very common case.

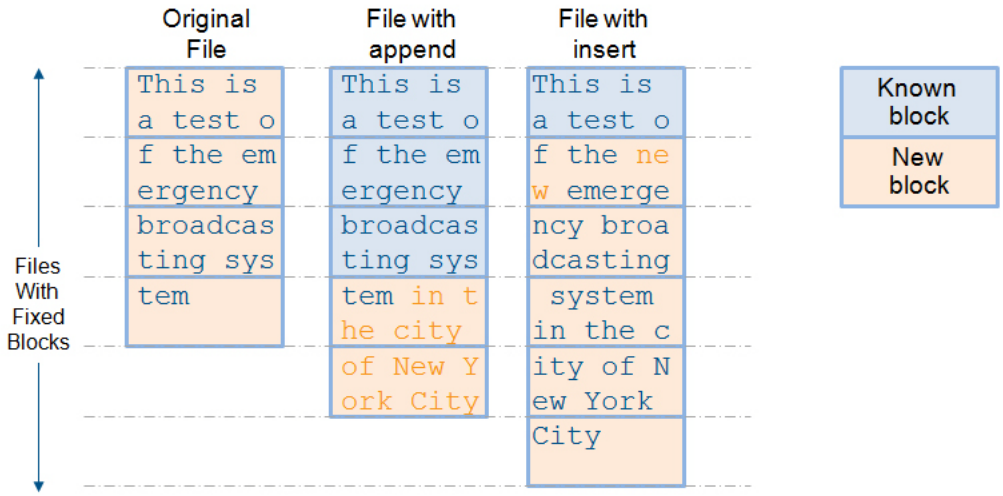


Figure 10 – Comparison of fixed-blocks needed to capture (1) a file, (2) the same file after bytes are appended, and (3) that file after bytes are inserted

If file updates were restricted to modifying or appending bytes in a file, fixed-block deduplication would deliver compression multipliers comparable to variable-block deduplication. However, when data is inserted or deleted from a file, the fixed-block boundaries shift relative to the file contents and all the blocks following the insert/delete point are different from those before the change.

Adoption of variable-block dedupe algorithms has grown because inserts are common in many types of software. Examples include databases, documents, and disk images. Variable-block is most effective when applied across individual files, rather than within large files or to byte streams, because the start-of-file serves as a known block boundary. By comparison, Progressive Deduplication is equally effective when applied within large files (e.g. VMDK images) or to byte streams.

	Single-instance	Fixed-block	Variable-block	Progressive
Tolerates "Append" or "Modify"	No	Yes	Yes	Yes
Tolerates "Insert"	No	No	Yes	Yes
Tolerates "Fine-grain intra-file"	No	No	No	Yes

Figure 11 – Comparison of the treatment of file modifications by various deduplication strategies

Compression Multipliers

[Note: A compression multiplier is the ratio of the volume of data before and after deduplication. The value "after" deduplication is the sum of the storage necessary to record (1) the references to the blocks that make up a file and (2) the unique blocks themselves. This paper uses this nomenclature as a clearer alternative to percentage growth or percentage savings.]

The effect of data duplication depends greatly on source data. Just as the DEFLATE algorithm, commonly available as part of various "zip" utilities, will sometimes fail to compress a file or even make it larger, the effect of data deduplication on a collections of files depends enormously on the profile of the source data.

As illustrated in Figure 2, compression multipliers increase as more, similar files are collected into a single repository. Virtually any extravagant claim of "high compression" can be achieved with a suitable collection of files. To achieve 100-to-1 compression, simply replicate the same file more than 100-times. Need a higher dedupe ratio, replicate more.

Early implementations of dedupe solutions treated deduplication as a post-process—something performed on backup sets, after the backup process completes. Still in 2011, several major vendors compare a series of 100 daily "full backups" to the same 100 backup sets after deduplication—as though the benefits of file-grain incremental backups were unknown.

The best backup deduplication approaches are tightly integrated into the backup process. Deduplication must be a complement to file-grain incremental backups and to block-grain incremental image backups. Collecting replicated data before dedupe may improve compression multipliers, but only serves to lengthen backup windows.

In practice, deduplication multipliers should be calculated against a base-line of incremental backups ("incrementals forever") to disk. In this scenario, deduplication delivers its greatest benefit when

- Large files are incrementally changed (because most blocks in most files do not change between snapshots), or
- Similar, but not identical, files are encountered across computers, or
- Disk images include abandoned blocks (as for virtual machines disk images)

While actual compression multipliers depend heavily on the source data, for purposes of backups, improvements over incremental backups typically range from 2x to 10x.

Finally, note that the natural redundancy of full backups is lost upon deduplication. If an administrator performs weekly full and daily incremental backups, most files on the weekly full backups will be present week after week. If media for one week's full backups are destroyed, other media may contain the desired files. Because this natural redundancy is squeezed out of backup sets, it is essential to protect your backup sets against media failure. Arkeia recommends the use of RAID-6 (rather than RAID-5) for storage of deduplicated backup sets. RAID-6 use two parity drives in place of RAID-5's single parity drive.

Summary & Conclusion

Deduplication constitutes a proven strategy for compression of large disk images or sets of files.

Single-instance storage (SIS), the first application of deduplication technology to storage, has been largely replaced by fixed-size block-grain (*fixed-block*) deduplication because the latter delivers superior compression multipliers. Fixed-block dedupe, however, because it fails to tolerate byte insertion into files, has been significantly displaced by variable-block deduplication. Variable-block dedupe delivers better compression multipliers, but is CPU-intensive and cannot be application-aware because control of block sizes is poor.

Progressive Deduplication tolerates byte insertion while precisely controlling block sizes. Control of block size makes possible application-aware dedupe and results in higher compression multipliers. Because deduplication block sizes in a file are fixed, known data is processed as quickly with Progressive Deduplication as with fixed-block deduplication.

The benefits of superior deduplication for backup are increased backup speed and reduced costs of storage and network infrastructure. While there is a tradeoff between time (less computation) and space (better compression), Progressive Deduplication successfully pushes out the curve by applying new and better algorithms.