

Data Deduplication

February 2011

What is Data Deduplication?

Data deduplication is a technology that can reduce disk storage-capacity requirements and replication bandwidth requirements by a factor of 20:1.

This is certainly a very powerful marketing statement, and it is generally accurate. However, data deduplication comes with a lot of ifs, ands, and buts. In this article, we explore what data deduplication is and its many strengths, along with some caveats.

In simple terms, data deduplication is a method in which a specific block of data in a large database is stored only once. If it appears again, only a pointer to the first occurrence of the block is stored. Since pointers are very small compared to the data blocks, significant reductions in the amount of data stored can be achieved.

The amount of storage savings is very dependent upon the characteristics of the data. For instance, full database backups contain mostly data that is present in earlier backups. Email systems often contain many copies of the same email message and/or attachments that have been sent to several people. These examples can benefit significantly from data deduplication. However, incremental backups contain mostly unique data and will not benefit very much.

Vendors claim storage-reduction ratios from 10:1 to 50:1, depending upon the particular data characteristics. A ratio of 20:1 seems to be the average quoted reduction ratio.

The reduction of storage capacity, along with the associated costs for hardware, power, cooling, and space, is only one of the advantages of deduplication. Another is bandwidth reduction requirements for replicating data to a disaster-recovery (DR) site. Replicating a deduplicated file that has been reduced to 1/20th of its original size will likewise only require 1/20th of the bandwidth to replicate it to a remote site.

Driving the need for data deduplication is the information explosion that is leading to the generation of a massive amount of data. All of this data must not only be stored online, but restoration snapshots and archives of the data must also be kept. It is estimated that the worldwide amount of digital data in 2009 reached 800 billion gigabytes (8×10^{20} bytes) and that it will grow another 44% by 2020 – an exponential growth.¹ Currently, 23% of all companies are using deduplication; and another 61% are either deploying it or are evaluating it.²

¹ HP StoreOnce: reinventing data deduplication, HP white paper, July 2010.

² 2010 Storage Priorities Survey, searchstorage.com; March 2010.

Uses for Deduplication

Perhaps the original motivator for data deduplication was the advent of virtual-tape systems. With virtual tape, periodic backups of a database are made to disk rather than to magnetic tape. This is typically transparent to the applications. Applications think they are writing to magnetic tape. However, the virtual tape facility writes the magnetic-tape images to disk rather than to tape.

Virtual tape backups have many advantages over magnetic-tape backups. Backups to disk can be much faster, and they eliminate the need for physical media handling. Backup tapes do not have to be moved offsite to a secure storage vault; rather, tape images are transmitted to a remote data center. Most importantly, the recovery of lost or corrupted files or of an entire database is much faster and eliminates the need to recover tapes from a vault and to deal with tapes that cannot be read. Recovery of a large database that used to take days from magnetic tape can often now be done in hours from disk.

The problem with virtual tape is that the amount of disk storage required to store archival backups soars over a short period of time. To facilitate recoveries to a point-in-time and to satisfy regulatory requirements, each daily or weekly backup has to be stored and retained. A petabyte database can soon consume hundreds of petabytes of disk storage. Therefore, data has to be rolled off to magnetic tape after only a few backup iterations.

Enter data deduplication. It is based on the recognition that most of the data in a backup is an exact copy of data in the preceding backup. Why not simply store the changed data and provide a way to reconstitute a file or database when needed? This technique has been found to reduce the amount of additional storage required after the first backup by a factor of 20 or more. Backups can now be economically retained on disk for years rather than for months before they have to be rolled off to tape for long-term archival storage.

Data deduplication is also of value to disaster-recovery sites. A disaster-recovery site must contain a current copy of the database in order to ensure fast recovery and to minimize the amount of lost data should the primary data center fail. Data replication techniques work fine for relational databases, in which only changed records are replicated. However, a good bit of data is kept in files that are a single logical entity. If a few bytes in a file change, the entire file must be replicated.

Data deduplication can reduce the amount of file data that has to be replicated to the DR site, thus effecting significant savings in the communication bandwidth required. To achieve this, data to be replicated is deduplicated at the primary site. The deduplicated data is then replicated to the remote site.

Companies are now looking at deduplicating certain primary databases. Relational databases are effectively deduplicated since the third normal form requires that no data be repeated. However, large files may contain repeated data and are candidates for storing in deduplicated form. The restoration of a deduplicated file, as we shall see, requires very little overhead and supports online use in many applications.

Data deduplication is certainly not a cure-all for all data storage needs:

- Deduplication requires significant processing overhead. Therefore, it is not suitable for data that is changing frequently and to which frequent access is required. This precludes its use for most online data in primary databases.
- Deduplication is not suitable for storing data on magnetic tape because of the way it fragments data.

Therefore, deduplication's role is a complementary one with other classic backup technologies:

- If data changes rapidly and is needed for only a few days, full storage on disk is best.
- If data changes moderately and is needed only for weeks or months, deduplicated data on disk is preferred.
- If data must be archived after months and saved for years, tape is the choice.

How Data Deduplication Works

Deduplication requires that an initial full copy of the data be present. Thereafter, if further updated copies of that data must be made, only the changes are stored.

Deduplication proceeds as follows. The file to be copied is broken up into *chunks*. A *hash value* is then calculated for each chunk. The hash values for previously stored chunks (including those in the original copy) are stored in a hash table with pointers to the data chunks they represent.

If the hash value for an incoming chunk is found in the hash table, the incoming chunk of data is replaced with a pointer to the previously stored chunk. If the hash value for the incoming chunk is not found in the hash table, the chunk is preserved in the file copy; and its hash value with pointer is stored in the hash table. The result is that a copied file is a series of pointers to pre-existing chunks interspersed with new unique chunks.

As deduplicated files are deleted, chunks that are pointed to by hash keys cannot be deleted. However, when there are no more pointers to a chunk, it may be deleted and its hash value removed from the hash table.

The calculation of hash values is processing-intensive. Therefore, the deduplication of a file requires significant processing resources. Depending upon the implementation, deduplication may be done either as *inline processing* or as *post processing*. An inline-processing implementation deduplicates as the file is being received. Post processing stores the file first and then deduplicates it. Inline processing may slow down the file transfer. Post-processing is faster but requires additional storage to hold the full file before it deduplicates it.

When a file is to be restored, it is read a chunk at a time and is delivered to the requesting application. Whenever a pointer is encountered, the chunk to which it is pointing is delivered instead. Unlike deduplication, file restoration imposes very little overhead and is almost as fast as full-file reading. Both deduplication and restoration are transparent to the end users and applications.

Note that deduplication is not performed just within a file. The hash table can point to chunks anywhere in the entire database. Therefore, if one file contains data that is identical to that contained in other files, the deduplicated file contains pointers to the common data no matter where that data is found.

Chunks and Hashes

Chunk Selection

The selection of a chunk size is the most important characteristic with respect to the effectiveness of deduplication. As the chunk size grows, the probability of finding matching chunks decreases. As the chunk size gets smaller, processing overhead increases.

Experience has shown that the chunk size should be less than ten kilobytes. A 4KB chunk size is common.

There are two fundamental methods for breaking up a data stream into chunks – fixed chunk size and variable chunk size.

Fixed Chunks

A fixed chunk size is usually based on a physical characteristic of the disk subsystem. For instance, if the disks use a 4 KB block size, the chunk size might be 4 KB. Data stored in a block generally remains in that block. Room is left in the block so that if data is to be inserted, there is room for it. If the block overflows, it is split into two blocks; and each block now has room for additional data.

Therefore, if data is inserted into or deleted from the file, only the disk blocks containing that data are affected. The rest of the blocks remain unchanged and are replaced with pointers to the original data.

Arbitrarily breaking up a file into fixed-size blocks will not work. If data is inserted, all following blocks are now changed and may not be deduplicated.

Variable Chunks

With variable chunks, a chunk is defined by some repetitive sequence found in the data. A simple example is a text file. A chunk may be a paragraph. Whenever a new paragraph is found, the current chunk is terminated and deduplicated. A new chunk is then begun.

Variable chunks are sensitive only to the characteristics of the data and not to the underlying characteristics of the data store.

Hashing Algorithms

Another choice important to the effectiveness of deduplication is the selection of an appropriate hashing algorithm. The algorithm should have the following characteristics:

- It should generate a hash key that is sufficiently large to guarantee a very sparse hash space (i.e., only a small proportion of hash keys are in use by chunks). This is important to minimize data collisions, in which two different chunks generate the same hash key.
- It should not be sensitive to similar data patterns in such a way that it generates similar hash keys. For instance, if one character is changed in a paragraph, the resulting hash key should be a great distance from the original hash key in hash space. This will prevent data with similar characteristics from bunching up in hash space, increasing the probability of data collisions.
- It should be reasonably efficient in its requirements for processing resources.

Most deduplication products today use the MD4/MD5 (Message Digest algorithm 4 or 5)³ or the SHA-1 (Secure Hash Algorithm 1)⁴ hashing algorithms. The MD algorithms generate a 128-bit hash key. The SHA-1 algorithm generates a 160-bit key.

³ MD4, *Wikipedia*.

⁴ SHA-1, *Wikipedia*.

SHA-1 seems to be the more predominant hash key in use today. It was designed by the U.S. National Security Agency and was published as a federal standard for security applications and protocols.⁵

To improve hashing efficiency, some deduplication products use a weak hash key to detect potential duplicate chunks. If one is found, the more secure hash key is then used to determine whether there is a match or not. This can reduce processing requirements significantly, as the more secure hash key only has to be calculated for a subset of chunks rather than for every chunk.

Server Deduplication

Deduplication can occur at the receiving end (server deduplication), at the sending end (client deduplication), or as a cooperative effort between the sender and the receiver (client/server deduplication).

With server deduplication, the entire file is sent to the receiving end. There it is deduplicated either via inline processing or post processing.

Server deduplication is typically done via a deduplication appliance, which is a self-contained hardware system containing the deduplication and restoration logic and the disk storage for deduplicated files. Use of such an appliance offloads the deduplication processing load from existing systems and makes deduplication totally transparent to the applications.

Many deduplication appliances accept data streams from many sources so that deduplicated data from several systems coexists in one place. Many appliances also support replication of the deduplicated data to remote DR sites. This approach requires a fraction of the bandwidth that full replication would require.

Client Deduplication

With client deduplication, the deduplication is done on the system hosting the primary database. The deduplicated data can then be directly replicated to a remote DR site. If performance permits, the deduplicated data can also replace the original full copy of the data for application-processing purposes.

Client deduplication is necessarily software-based. It has the advantage of eliminating the need for a deduplication appliance, which represents a single point of failure in the data stream. Also, licensing costs for the deduplication software may be less expensive than the cost of an appliance.

However, deduplication agents on all sending systems must now be licensed and managed.

Client/Server Deduplication

With client/server deduplication, the sending site and the receiving site cooperate to bring both processing efficiency and network efficiency to deduplication. This method generally involves deduplication appliances at both the sending and the receiving sites.

Rather than sending the entire file to the receiving site, with client/server deduplication the sending site first calculates the hashes on each chunk of the data stream and sends the file as a sequence of hash keys to the receiving site. The receiving site compares these hash keys to its

⁵ The newer SHA-2 generates hash keys up to 512 bits in length.

hash key table and responds with a list of hash keys for which it has no record. The sending site then needs only to send the unique chunks to the receiving site.

Processing efficiency is achieved since the hash computations are offloaded from the application systems. Furthermore, the network is utilized efficiently since only changed data must be sent (plus the relatively small overhead of sending hash keys).

Another advantage of this approach is that data from many remote branch locations can use the same deduplicated central storage site for backup and disaster recovery. Each location takes advantage of the totality of data across the enterprise for deduplication.

Data Collisions

A concern with data deduplication is that of data collisions. A data collision occurs if two different chunks generate the same hash key. In this case, the database is corrupted and may not be easily recoverable without going back to a previous backup. The problem is that the corruption may not be detected until the data is restored, which could happen a long time after the corruption occurred.

The bigger the database and the higher the rate of change, the more likely data collisions become. The key to minimizing data collisions is a very large hash space. The more hash keys there are, the less likely are data collisions.

A feel for the current state of this technology can be obtained by considering the 160-bit key generated by the SHA-1 algorithm. This generates $2^{160} = 1.5 \times 10^{48}$ unique hash keys. This is a pretty big number.

In Appendix A, we look at the collision rate for a petabyte database that is being changed at a rate of one-million updates per second. We find that we can expect a collision about once every billion trillion years. This is a pretty long time; the age of the universe is only about 14 billion years.

We can expect a virtually uncountable number of total disk-system failures before we ever see a deduplication data collision. However, that being said, the probability of a data collision is still not zero.

Compression and Encryption

Both data compression and data encryption can be used with data deduplication. However, the relationship is critical.

Compression tends to eliminate similar sequences in the data. Therefore, a compressed file may not fare very well under deduplication. For this reason, compression should be applied to the deduplicated file, not the original file. Typical compression ratios for a deduplicated file tend to run in the 2:1 range.

Similarly, the purpose of encryption is to completely randomize a file. Therefore, deduplication is not effective on an encrypted file. Encryption should be made on the deduplicated file, not the original file before deduplication.

Cost

Of course, the cost of deduplication appliances varies with the vendor. However, in 2008, the typical cost for a deduplication appliance was reported to be about \$1 per deduped gigabyte of

storage.⁶ Twenty terabytes of deduped storage cost about \$20,000. At a 20:1 compression ratio, this is equivalent to 400 gigabytes of original unduplicated data.

A Little Deduplication History

The concept, and indeed the technology, of data deduplication is not new. An early form of deduplication is the data compression methods that have been around for decades. Deduplication by duplicating only files that have changed (single-instance storage, or SIS) has been around for a while.

Probably one of the first instances of data deduplication as we know it today is the Unix utility *rsync*. Introduced in 1996, *rsync* minimizes transmission bandwidth requirements. It uses a 128-bit hash key provided by the MD4/MD5 hash algorithm and what has been described above as client/server deduplication. Though it only acts on a file-by-file basis (not the entire database), it sends the hash keys of a file to be transferred to the remote server. The remote server responds with the list of hash keys that it does not have in its hash table, and then only the new chunks are sent.

Implementations

Many deduplication products exist in the market today – just Google “data deduplication.” Some examples are:

- HP’s StoreOnce
- IBM’s ProtecTIER
- EMC’s Data Domain
- Quantum’s DXi series
- FalconStor’s Virtual Tape Library

This is just a small sampling of available products. White papers from these vendors provided the information for this article, along with the Wikipedia subjects previously referenced.

Summary

Data deduplication is an old technology that is just now going mainstream. Used with appropriate data stores, deduplication can significantly reduce the amount of disk storage needed for disaster-recovery databases and for near-term archiving of database backups. It can also reduce the network capacity required for replicating nonrelational databases.

Deduplication does not replace disk storage for online data that is needed to support applications, nor does it replace magnetic tape for long-term archival of point-in-time backups. However, used properly, it can reduce the data-center footprint of data-storage subsystems with the resultant reduction in the costs of hardware, power, cooling, and data-center space.

⁶ Data De-duplication for Dummies; Wiley Publishing, Inc. (for Quantum Corp.); 2008.

Appendix A – Data-Collision Rates

A concern with data deduplication is data collisions, in which two different chunks resolve into the same hash key. The minimization of data collisions depends upon a sparsely populated hash space in which only a small proportion of all hash keys actually reference a chunk. The more sparse the hash space, the less likely it is that two chunks will resolve to the same hash key.

The popular hashing algorithm for data deduplication is SHA-1, which generates a 160-bit hash key. There are $2^{160} = 1.5 \times 10^{48}$ possible hash-key values for this algorithm.

Consider deduplicating a one-petabyte (10^{15} bytes) database that is being updated one-million times per second. Furthermore, assume that backups that are one-year old are rolled to tape and deleted from disk. What is the data-collision rate that occurs during deduplication?

Assume that the chunk size is 4,000 bytes. The number of unique chunks in the database is then $10^{15}/4,000 = 2.5 \times 10^{11}$ chunks.

Furthermore, new chunks are being generated at a rate of 1,000,000 (10^6) chunks per second. Taking the worst case in which each new chunk will be referenced and therefore must be stored, there will be (10^6 chunks per second)(3.2×10^7 seconds per year) = 3.2×10^{13} new chunks created per year which must be stored before they are deleted following a tape backup.⁷ Thus, the total number of chunks stored is approximately $2.5 \times 10^{11} + 3.2 \times 10^{13} = 3.2 \times 10^{13}$.

The probability that a hash key will be used is

$$\begin{aligned} \text{probability that hash key is used} &= (\text{number of chunks}) / (\text{number of hash keys}) \\ &= 3.2 \times 10^{13} / 1.5 \times 10^{48} = 2.1 \times 10^{-35} \end{aligned}$$

This is also the probability that a new chunk will cause a data collision. If chunks are changing at a rate of 1,000,000 (10^6) chunks per second, the rate of collisions is

$$\begin{aligned} \text{collision rate} &= (\text{chunk change rate}) (\text{probability of a data collision}) \\ &= (10^6)(2.1 \times 10^{-35}) = 2.1 \times 10^{-29} \text{ collisions/second.} \end{aligned}$$

The yearly collision rate is

$$\begin{aligned} \text{collision rate} &= (2.1 \times 10^{-29} \text{ collisions/second}) (3.2 \times 10^7 \text{ seconds/year}) \\ &= 6.7 \times 10^{-22} \text{ collisions per year.} \end{aligned}$$

The time between collisions is then

$$\text{time between collisions} = 1 / 6.7 \times 10^{-22} = 1.5 \times 10^{21} \text{ years.}$$

This is a collision about every billion trillion years.

⁷ Note that this is 100 times the size of the basic database. This is, of course, a strange example. If the deduped data over a year is 100 times the size of the basic database, it may be better to simply store weekly full backups. This would require only fifty times the storage space of the original database. However, this example serves to illustrate the point that data collisions will hardly ever occur.