Demystifying Data Deduplication

Nagapramod Mandagere
University of Minnesota
npramod@cs.umn.edu

Pin Zhou, Mark A Smith, Sandeep Uttamchandani
IBM Almaden Research Center
{pinzhou,mark1smi,sandeepu}@us.ibm.com

ABSTRACT

Effectiveness and tradeoffs of deduplication technologies are not well understood – vendors tout Deduplication as a "silver bullet" that can help any enterprise optimize its deployed storage capacity. This paper aims to provide a comprehensive taxonomy and experimental evaluation using real-world data. While the rate of change of data on a day-to-day basis has the greatest influence on the duplication in backup data, we investigate the duplication inherent in this data, independent of rate of change of data or backup schedule or backup algorithm used. Our experimental results show that between different deduplication techniques the space savings varies by about 30%, the CPU usage differs by almost 6 times and the time to reconstruct a deduplicated file can vary by more than 15 times.

Categories and Subject Descriptors

A.1 [**Introductory and Survey**]; D.4.8 [**Performance**]: Operational analysis

General Terms

Design, Performance

Keywords

deduplication, compression

1. INTRODUCTION

Data deduplication is becoming increasingly popular as a technology that helps enterprises reduce the data footprint by eliminating redundant copies of data – copies that are created during complete system backups, e-mail attachments distruted to multiple users,shared documents, music and projects,etc. Data compression tools such as gzip reduce intra-file data redundancy, while deduplication reduces both intra-file and inter-file data redundancy. We refer to the reduction in data footprint size due to deduplication the *fold factor*.

Middleware '08 Companion December 1-5,'08 Leuven, Belgium

The effectiveness of data deduplication depends on several factors and varies widely across different deduplication algorithms. Deduplication is a data intesive application and comes at a cost of higher resource overheads on the existing storage infrastructure, increased backup time window, and higher latency to access deduplicated data due to reconstruction overhead. Given the plethora of deduplication offerings from various vendors, administrators and small enterprise users are finding it increasingly difficult to select a solution that works best for their enterprise: a) What are the typical sources of duplication and which deduplication algorithm is the most suitable?; b) Should deduplication be done on the client or on the server?; c) What is a reasonable reconstruction time? There is no single answer for these questions – the fact is it "depends."

The goal of this paper is to characterize the taxonomy of available deduplication technologies and experimentally evaluate their properties on a real dataset. To ensure an "apples-to-apples" comparison of fold factors, resource utilization, ingestion rates, and reconstruction time of multiple technologies, we implemented a deduplication middleware appliance that was deployed on enterprise-class servers and storage systems. The middleware allows plug-in for various deduplication algorithms and helps characterize popular deduplication architectures. The experimental evaluation was using data from a real-world backup server used by enterprise users to backup official e-mails and other business-critical data.

The key contributions of this paper are - Developing a **taxonomy** to characterize and classify the increasing number of available deduplication technologies, and **Experimental evaluation** of fold factor, resource requirements, reconstruction time of deduplication algorithms on a real world dataset. This paper complements existing survey and evaluation of deduplication techniques and systems [7, 6, 4, 8, 3] by experimenting on large representative dataset, evaluating not only the space saving but also the deduplication performance, resource consumption, metadata overhead and reconstruction bandwidth, and examining the duplication inherent in backup data independent of backup algorithm or schedule.

2. DEDUPLICATION TAXONOMY

Deduplication solutions differ along three key dimensions (as shown in figure 1), namely - **Placement** of the deduplication functionality, **Timing** of deduplication w.r.t. to the foreground IO operations and **Algorithm** used to find and reduce redundancies in the data

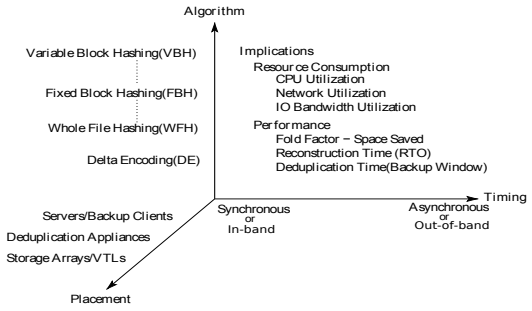


Figure 1: Deduplication Design Space

2.1 Placement

Data deduplication can be done on the *client*, *storage array/VTL* or on an *appliance*.

Client Based

Deduplication in the client follows a client-server approach wherein a deduplication client communicates with its server counterpart installed on either a deduplication appliance or the Storage Array itself. This type of deduplication is often referred to as *transmission deduplication* where in, the deduplication client processes the data, extracts deduplication metadata and exchanges it with its server counterpart. The server counterpart would use this metadata along with its own bookkeeping data to determine if duplication exists and communicates back the corresponding information to the client. The deduplication client would transmit only the unique data over fibre channel or the IP network.

Deduplication at the *Client* provides huge savings in network bandwidth. However, these savings come at a cost. Firstly, client side CPU and IO resources are used for detection of duplicates. Secondly, it has a wide range of security implications, since any client can query the Storage Array with deduplication metadata, one could attempt to reverse engineer the system to determine the contents stored by other backup clients on the same storage array. Lastly, client-side deduplication may affect the choice of *deduplication timing* at the server.

Deduplication Appliance

A deduplication appliance is a special-purpose system that implements and performs deduplication. Typical deduplication appliances operate either in-band or out-of-band, connected to both the clients and the Storage Arrays. In-band deduplication appliances examine all of the incoming data as it arrives to find duplication before writing data to the storage array, whereas out-of-band appliances perform the deduplication functionality after the data has been written to disk. One main concern with in-band appliances is that the appliance could become a bottleneck in the IO path due to the extra processing required by the system before writing the data to disk.

Storage Arrays/VTL

Disk Array controllers and Virtual Tape Library controllers provide a very good platform for performing deduplication. Modern disk array controllers often have large computational resources and their proximity to data makes them a very interesting place to perform deduplication. VTLs though primarily meant for backup/offline operation typically have powerful controllers too. However, placing deduplication functionality within a storage array rules out the option of using any type of content-aware deduplication algorithms that operate by understanding the details of the data; this is because existing Storage Arrays do not have the capability to understand file system metadata, thereby limiting the choice of deduplication algorithms to block-based.

In summary, **the choice of placement in addition to affecting the resource utilization can potentially limit the timing/deduplication algorithm selection**.

2.2 Timing

Deduplication can be performed as *Synchronous/In-Band* or as *Asynchronous/Out-of-Band* operation.

Synchronous/In-Band Deduplication involves performing the deduplication operation as part of regular data requests. Every write request to the storage system involves an attempt to perform deduplication (finding the match, if any, and then updating metadata to reflect the deduplication operation) before actually writing data to disk. In-band deduplication is amenable to client-side placement because the data store metadata synchronously reflects its contents and can be queried immediately by clients, eliminating duplicate network traffic. Additionally, it facilitates thin provisioning of storage since no staging of data is required. In-band deduplication can add a significant amount of latency to the system, e.g., in a backup environment, performing in-band deduplication would mean increasing the size of the backup window.

Asynchronous/Out-of-Band Deduplication involves performing deduplication at regular intervals or when the system reaches a high-water mark.It can be desirable in systems where the ingestion speed of data is the primary consideration. Main drawback is that it results in a large number of additional IO operations solely for the purpose of deduplication – data must be re-read from disk, and duplicate bytes must be reclaimed, often resulting in additional write IOs. Though out-of-band deduplication could be intelligently scheduled to minimize interference with regular IO operations, it can have a significant impact on power management schemes and load balancing windows. Out-of-band deduplication limits the option of placement – deduplication at the client is less beneficial due to lack of up-to-date deduplication metadata during run-time. Hence, many of the benefits (like reduction in network bandwidth utilized for deduplication) that one could achieve by placing deduplication functionality at the client is lost. Finally, out-of-band deduplication necessitates provisioning for a larger footprint since deduplication is performed at a later point in time, space to save the complete dataset is still required for staging purposes until deduplication can be performed.

In summary, **choice of timing may be restricted by placement, but the main driver is the performance requirements or Service Level Objectives (SLOs) of the storage subsystem.**

2.3 Deduplication Algorithm

Based on the granularity of deduplication, algorithms are categorized into three main categories: *Whole File Hashing*, *Sub File Hashing* and *Delta Encoding*.

Whole File Hashing (WFH): A SHA1 and/or MD5 hash function is applied to the content of each file to obtain their hash signatures. Files with matching signatures are collected and optionally a byte-to-byte comparison of them is performed as a safeguard from hash collisions.

Sub File/Chunk Hashing: schemes use blocks/chunks as the granularity. Based on the strategy used for this division of files into blocks, two different approaches have been developed, namely -
Fixed Block Hashing (FBH): Every file is divided into fixed-sized blocks starting from the beginning of the file. A hash function like SHA1 or MD5 is used to compute the signature of each chunk. A hash table is used to find exact match chunks and then optionally a byte-to-byte compare is used as safeguard against hash collisions.
Variable Block Hashing (VBH): uses Rabin Fingerpriting [5] [1], a sliding window rolling hash based technique to subdivide byte-streams into chunks with a high probability of matching other chunks generated likewise. If a signature of the chuck/block matches one of the pre-computed or prespecified delimiters, the algorithm designates the end of this window as a chunk boundary. Once the boundary has been identified, all bytes starting from the previous known chunk boundary to the end of the current window is designated a chunk. A hash of this new chunk is computed and compared against the signatures of all pre-existing chunks in the system. This approach has been proved to be very successful in identifying similar chunks irrespective of their offsets within their corresponding files [6]. In practice, this method makes use of four tuning parameters, namely - the *minimum chunk size*, the *maximum chunk size*, the *average chunk size* and the *window size*.

Delta Encoding/Compression (DE): Delta encoding [2] is a mechanism used to generate a delta (usually called patch file) between two files. Given a reference file and a new file, delta encoding produces a delta or diff between the two. Just like inter-file compression techniques, this method simply uses copy/insert commands and outputs a patch file. A stream matching algorithm is used to locate matching offsets in the source and target versions, emitting a sequence of copy instructions for each matching range and insert instructions to cover the unmatched regions. For an incoming file to a system using DE, the most important implementation detail is how to select the file against which to produce the delta. Fingerprint matching can be used for detection of resemblance between a given file and a set of files already in the system.

Algorithmic choices have several implications. **Fold Factor** to a large extent depends on the dataset itself. In some instances, the difference/improvement between algorithms might be minimal. However, in general, VBH-based products are known to provide the best fold factor. The selection of parameters for VBH has been found to be a tricky issue. One must balance the metadata overhead with the duplication detection in order to achieve best fold factors.

Reconstruction time is an important metric and mostly dependent on the deduplication algorithm and the degree of locality of the constituent chunks comprising the filesystem. Most of the algorithms that work at the granularity of fixed- or variable-sized logical blocks have a high impact on reconstruction time. This impact on reconstruction time is more evident in systems that work on top of file systems. Typically these algorithms store each of these individual blocks as separate files. Hence, retrieval of a single deduplicated file could potentially result in retrieval of all its constituent blocks or files leading to long reconstruction times. If block sizes chosen are very small compared to size of the original file, a single file reconstruction could lead to reading a large number of constituent files.

In the case of in-band deduplication, the **rate of deduplication** is very important as it directly impacts the data ingestion rate. Deduplication times are very closely related to choice of deduplication algorithm. If using VBH, the data must be both fingerprinted and hashed. With all algorithms, a lookup in a large table of data identifiers must occur. FBH and WFH algorithms both only require hashing of the data to uniquely identify it, and would have the least impact on data ingestion rates due to deduplication time. Out-of-band deduplication times are generally longer than in-band because of the additional IO required to perform the deduplication. Whereas in-band systems operate on the data while still in memory and before it is written to disk, out-of-band systems must re-read and reorganize the data to reclaim space during the deduplication process.

Resource Consumption, which is dependent on the computational complexity of these algorithms, varies widely. In general, algorithms that work at a larger granularity consume less resources at the expense of fold factor. In most cases client CPU resources are either limited or expensive, hence algorithm choices might be restricted. In some cases, network bandwidth might be considered more expensive (typically if geographically distributed). In summary, resource consumption could play a very critical role in selection of the deduplication technology.

In summary, **the choice of Deduplication Algorithm is driven by folding factor, expected reconstruction time, desired impact on ingestion rate, network speed, and availibility of CPU cycles**. Table 1 summarizes the popular deduplication products based on the above taxonomy.

3. EVALUATION METHODOLOGY

3.1 System Configuration

In our test bed, the Deduplication Appliance performs out-of-band deduplication. It is a 2-way 2.4GHz Intel Pentium Xeon server with 4GBs of DDR2 RAM with a SAN attached volume hosted by an IBM DX4000 series Disk Array with 20 10K RPM SCSI disks configured as a RAID 0 volume. Prior to the experimental runs, by trial and error we found that using 32 threads achieves the best utilization of compute and storage resources for our setup. Hence, we configure the system to use 32 instances of the Deduplication Process for the this comparison study.

Experimental results that follow are for deduplication of same dataset/workload with variation of different deduplication algorithms and their tuning parameters.

3.2 Dataset

The dataset used in our evaluations consists of base Windows backups of 16 users totaling upto 450 GBs and 2,074,810 files. No incremental backups are included in this analysis because we want to study the duplication inherent in the

Products	Placement			Timing	Algorithm
	Backup Clients	Deduplication Appliances	Storage Array/VTL		
Data Domain		X	VTL	In-Band	VBH
Diligent		X	VTL	In-Band	VBH
Avamar(EMC)	X	X		In-Band	VBH
FalconStor			VTL	Out-of-Band	Sub File Hashing
Network Appliances			Storage Array	Out-of-Band	FBH
Sepaton			VTL	Out-of-Band	Content-aware
Quantum		X	VTL	Unknown	VBH
ExaGrid Systems		X		Out-of-Band	Content-aware
Symantec PureDisk	X	X		In-Band	Unknown

Table 1: Classification of Deduplication Products.

File Type	Total Size(GB)	% of Dataset	Avg. File Size(KB)	File Count
AVI	83.3	20.1	131515.4	664
DLL	35.0	8.4	276.7	132492
EXE	24.6	5.9	602.1	42989
NSF	20.7	5.0	46228.7	469
MOV	20.5	4.9	67283.5	319
JAR	13.7	3.3	409.8	35162
MP3	12.8	3.1	4165.7	3231
NOEXT*	11.1	2.7	43.6	267121
ZIP	10.2	2.5	1111.4	9618
RDG	7.0	1.7	192061.7	45

Table 2: Statistics of the Top Ten File Types

backup data, not due to multiple backups of the same or similar data. Very high aggregate fold factors can be achieved by performing daily full backups, but we focus on the duplication in the dataset itself to eliminate the influence of backup policies and schedules on the results.

This dataset represents the backup and archival data of employees in the enterprise environment. Figure 2 shows the cumulative distribution of the file sizes for this dataset. Table 2 provides the information on the file types whose total sizes are among the top ten in the dataset.

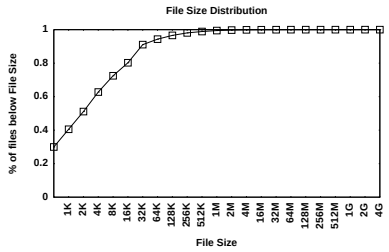


Figure 2: Cummulative Distribution of File Sizes

4. PERFORMANCE EVALUATION

Choice of Deduplication Algorithm typically dictates Fold Factor, Reconstruction Bandwidth, Metadata Overhead, and Resource Usage. Typical deduplication systems apply the same deduplication algorithm across all data types. In this section we apply different deduplicaiton alogorithms, i.e., VBH,FBH,WFH to the entire dataset to better understand the differences between them. Table 3 and Figure 3 show the comparison results for different algorithms and parameters.The minimum block size(min_block_size) is set to $\frac{1}{2} * avg\_block\_size$ and maximum block size (max_block_size) is set to to $2 * avg\_block\_size$. We use fixed window size (64 Bytes) for different VBH algorithms as our prior experiments showed that the window size has a negligible effect on the deduplication behavior.

4.1 Fold Factor Comparison

Algorithm	Fold Factor	Metadata Overhead(MBs)	Reconstruction Bandwidth(MBps)
WFH	1.310	307.64	18.873
FBH(16KB)	1.525	2656.26	4.768
VBH(16KB)	1.607	2248.71	7.002
FBH(32KB)	1.516	1474.3	14.556
VBH(32KB)	1.583	1302.59	12.777
FBH(64KB)	1.508	884.23	22.292
VBH(64KB)	1.565	818.52	15.909
FBH(128KB)	1.501	590.84	24.4
VBH(128KB)	1.547	581.26	15.711
FBH(256KB)	1.494	445.63	20.577
VBH(256KB)	1.529	447.51	15.363
FBH(512KB)	1.486	374.20	17.925
VBH(512KB)	1.513	379.99	17.748

Table 3: Deduplication Comparison

Clearly for fold factor, VBH algorithms outperform both FBH and WFH over all block sizes considered. Within VBH, smaller block sizes yield better fold factors. Both of the above observations are consistent with the earlier discussion on the working of Variable Block Hashing schemes. Even in case of FBH, smaller block sizes yield higher fold factors. In general, for sub-file chunking algorithms, the smaller the block size, the higher the probability of finding identical chunks.

The difference in fold factor between comparable instances (similar block size parameters) of VBH and FBH decreases as block size increases. At their optimal operating points - VBH(16K) and FBH(16K) - space reduction achieved using VBH is about 3.9% higher than that achieved using FBH. At the maximum tested block size of 512KB, this difference narrows down to about 1.2%. Possible reasons for this behavior are - Firstly, by using larger blocks the probability that a single different byte between blocks increases, and because this difference is contained in a larger block, the entire block including the identical portions of the block are stored independently and Secondly, the number of files that are contained in single such blocks is high, as illustrated in Figure 2. A point worth noting is that even with simple WFH-based schemes, an appreciable amount of space reduction is achievable. In our dataset use of WFH amounts to a space reduction of about 23%.

As we can see, *deduplication inherent in backup data is considerable (as high as 1.6 of fold factor with VBH(16K)), but should not be confused with the large fold factors associated with multi-day cumulative deduplication. Fold factors of 1.5 to 2.0 seem reasonable to expect out of a single day's backup data. Achieving much larger fold factors in the range of 20 or 30 requires multi-day repeated backups and is heavily dependent upon the backup algorithm and schedule.*

4.2 Reconstruction Bandwidth

Column 3 in Table 3 shows the comparison of read or reconstruction bandwidth for different algorithms. Read speeds on deduplicated datasets are a direct function of the

block size parameter of deduplication algorithms. Our prototype system currently does not implement any optimizations in read or reconstruction path between the appliance and the underlying data store. In such a system, using experiments with whole file hashing as an approximate baseline and seeing the relative differences of it yields generalizable results. For WFH, the read bandwidth was about 19MBps. Compared to WFH, the sub-file chunking based schemes are able to provide far less read throughput. In addition, this throughput reduces with the block size parameter. This is because each extent is stored as a separate file due to the fact that typical deduplication schemes work on top of file systems. Reconstructing any original file therefore involves reading multiple files. For deduplication systems that work on top of standard file systems, controlling on disk layout of these constituent files is possible by implementing contiguous segment containers. However, this technique is limited by the fragmentation inherent in deduplicated data. As a result, reconstructing a single file involves multiple random reads (as many as one per constituent extent) resulting in decreased read throughputs.

Contrasting this reconstruction performance with the fold factor analysis given before yields a valuable insight: *trying to achieve best space efficiency by choosing minimal block sizes has an adverse impact on reconstruction time. Further, for many datatypes, fold factors do not improve much by using smaller block sizes. In such cases, blindly using smaller block sizes will yield very little benefit in terms of space, while leading to further degradation in reconstruction time.*

4.3 Metadata Overhead

As mentioned in the taxonomy section, the block sizes used for deduplication dictate the amount of extra metadata that one needs to maintain. Metadata here includes the bookkeeping structure that stores hash signatures of unique blocks and the object to extent(s) mapping information. The exact size of these types of information can vary across different implementations. Column 2 in table 3 quantifies the metadata overhead in our implementation which tries to optimize such overhead. Clearly, the difference in metadata sizes between the sub-file chunking algorithms is minimal. However, the actual size of the metadata is significant. This is because because if the metadata cannot be stored completely in memory, then the part of metadata stored on disk will cause disk reads for hash lookups. For instance, in our implementation the metadata size is about 2.2GB for running FBH(16K) on our archive/backup dataset leading to virtual paging. This causes an abnormaly long deduplication time for FBH(16K) as shown in Table 3. Recent work has increased the in-memory hit-rate of these deduplication hash tables to as much as 99% by leveraging Bloom Filters and locality-aware caching, reducing this requirement[8]. However, such approaches have only been proven to work in certain backup domains and hence the initial planning or provisioning of deduplication hardware still needs to account for metadata needs as the fragmentation of the system increases, and the locality-aware caching becomes less effective.

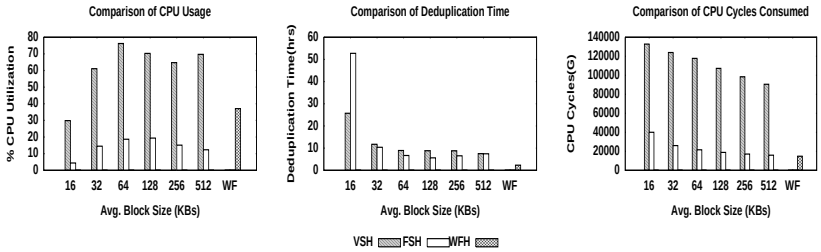
Therefore, *compared to the size of the dataset, deduplication metadata is small, but significant in terms of system performance impact.*

4.4 Resource Impact

Given that deduplication is typically a bulk data manipulation operation, an understanding of the necessary resources required to achieve or support this operation is very important in making system-wide provisioning decisions. As highlighted in Section 2, resource usage is dependent not just on the algorithm used, but also the placement and timing of deduplication. Figure 3 shows a comparison of CPU usage characteristics for the three algorithms and their variations. Figures 3(a) and 3(b) show the utilization of the CPU by the deduplication process and the time taken for deduplication process to complete respectively. As the block size decreases, the number of hash lookups and hash updates increase, consequently consuming more CPU cycles. In addition, the rate of decrease of computation time of SHA1 hash operation does not decrease linearly with block size. Hence in general, one would expect higher CPU utilization with smaller block sizes for both VBH and FBH. However, our results show that CPU utilization for a 16KB block size is significantly lower than any block size greater than 16KB. On further investigation we found two main reasons for this behavior, - Firstly, we found that with a 16KB block size, the available memory during the deduplication process dropped down to almost zero. As a result of this unavailability of physical memory, our bookkeeping data structures (2.7GB) used by the deduplication process could no longer be maintained in the physical memory in its entirety resulting in paging to disk. Hence, hash table lookups slow down by an order of magnitude. Secondly, with a 16KB blocksize, the filesystem makes IO requests of a size far smaller than the Logical Block Size on the Disk Array resulting in inefficient access. In our system, the Logical Block Size was configured as 64KB. The CPU utilization for block size other than 16KB is greatly dependent on the IO subsystem. Since the offline deduplication process performs random IO on the disk subsystem, even with a large number of concurrent deduplication threads, we found that the CPU utilization was affected by the IO bottleneck.

Both the CPU Utilization and Deduplication Times are impacted by certain hidden system factors due to which general conclusions should not and cannot be drawn from them. For instance, CPU and IO bottlenecks vary widely across systems. Hence, both these metrics need to be considered together to understand and draw generalizations about the CPU characteristics of different algorithms. This leads us to a derived metric - CPU Cycles consumed - a product of the CPU utilization and the total duration for which the CPU was used. Figure 3(c) shows the variation of CPU Cycles consumed with with different algorithms. This metric clearly shows the difference in CPU usage of these algorithms. VBH and its variations consume a large number of CPU cycles while finding the most duplicates. For instance, at a block size of 16KB, VBH consumes about 230% more CPU cycles than a comparable instance of FBH. VBH is 3.3 times as CPU intensive as FBH given that IO subsystem is not a bottleneck. An interesting point to note here is that though CPU cycles consumed by both WFH and FBH with block size 512KB are almost equal, FBH delivers a fold factor of 1.486 as against 1.31 for WFH. In essence, with similar CPU consumption, FBH can potentially deliver better space savings.

Compute resources consumed by deduplication processes could potentially make or break placement decisions. For in-



(a) CPU Utilization (b) Deduplication Time (c) CPU Cycles

Figure 3: Comparison of CPU Consumptions

stance, compute resources on servers in general are far more expensive than on dedicated appliances. Further, usage of CPU is interleaved in manners in which it could potentially have a drastic impact on performance of other applications. Hence, *the tradeoff between deduplication performance and resource usage needs to be considered on a case by case basis before making deployment decisions. For instance, in deployments where a network bandwidth between servers and storage is expensive, and CPU cycles on servers are cheap, the best choice might be VSH with smallest block size.*

5. DISCUSSION & LESSONS LEARNED

Based on extensive review of various deduplication technologies/vendor offerings and evaluation of a prototype deduplication system, in this section, we coalesce our findings into practical recommendations for storage systems administrators. When contemplating the deployment of a data deduplication system into their infrastructure, Administrators/ Decision makers need to keep two primary factors in mind - **Nature/Intent of Data Access and Resource Availibility and Tolerance of the system.**

Typical system usage varies across Enterprise. If a storage system is used primarily to serve database accesses in an online manner and is not asked to store backups of the same database, deduplication may not be warranted at all. On the other hand, if a storage system houses multiple virtual machines for different users, a deduplication system should provide significant storage savings. In cases where the system is asked to be a backup target for data, be it database data or otherwise, multiple backups of the same or similar data will almost always yield impressive deduplication rates.

Based on algorithms used for fingerprinting and their granularity of comparison, resource usage of deduplication process and its consequent impact on rest of the system varies. If storage efficiency is the primary concern, a variable-block deduplication scheme at the expense on large amount of CPU and IO resource usage. IO performance is paramount or CPU cycles are limited and the tolerance of the rest of the system w.r.to resource usage is low, a whole-file scheme or a large fixed-block scheme can provide a significant level of storage efficiency with very little resource impact to the system. In cases where network bandwidth is limited, careful tradeoffs between data transmission and client CPU usage need to be evaluated.

The order of importance of these factors varies across Enterprises and often a weighted combination needs to be considered and tradeoffs have to be evaluated before making the decision.

6. CONCLUSION AND FUTURE WORK

Data deduplication is a promising middleware appliance being increasingly adopted in enterprise storage environments. Better fold factors typically comes at the expense of increased resource overheads, but the converse is not always true. The experimental evaluation results and lessons described in this paper aim to demystify the available deduplication techniques by comparing them using our homegrown deduplication appliance prototype.

As ongoing and future work, we are exploring architectures that automatically and transparently adapt the deduplication algorithm for different data types as well as the varying CPU, network and storage bandwidth availability of the enterprise.

7. REFERENCES

[1] A. Z. Broder. Identifying and filtering near duplicate documents. In *Combinatorial Pattern Matching: 11th Annual Symposium*, 2000.

[2] J. J. Hunt, K.-P. Vo, and W. F. Tichy. An empirical study of delta algorithms. In *ICSE '96: Proceedings of the SCM-6 Workshop on System Configuration Management*, pages 49–66, London, UK, 1996. Springer-Verlag.

[3] P. Kulkarni, F. Douglis, J. LaVoie, and J. M. Tracey. Redundancy elimination within large collections of files. In *ATEC '04: Proceedings of the annual conference on USENIX Annual Technical Conference*, pages 5–5, Berkeley, CA, USA, 2004. USENIX Association.

[4] C. Policroniades and I. Pratt. Alternatives for detecting redundancy in storage systems data. In *USENIX Annual Technical Conference*, 2004.

[5] M. O. Rabin. Fingerprinting by random polynomials. In *Center for Research in Computing Technology, Harvard University. Tech Report TRCSE- 03-01, 2006*, 1981.

[6] L. You and C.Karamanolis. Evaluation of efficient archival storage techniques. In *21st IEEE/12th NASA Goddard Conference on Mass Storage systems and Technologies*, 2004.

[7] L. You, K. Pollack, and D. Long. Deep store: An archival storage system architecture. In *21st International Conference on Data Engineering*, 2005.

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