CloudDT: Efficient Tape Resource Management using Deduplication in Cloud Backup and Archival Services

Abdullah Gharaibeh*, Cornel Constantinescu, Maohua Lu, Anurag Sharma, Ramani R Routray, Prasenjit Sarkar, David Pease, Matei Ripeanu*
abdullah@ece.ubc.ca, cornel@us.ibm.com, lum@us.ibm.com, anushar@us.ibm.com, routrayr@us.ibm.com, psarkar@us.ibm.com, pease@us.ibm.com, matei@ece.ubc.ca
IBM Research – Almaden, *The University of British Columbia

***Abstract*—Cloud-based backup and archival services use large tape libraries as a cost-effective cold tier in their online storage hierarchy today. These services leverage deduplication to reduce the disk storage capacity required by their customer data sets, but they usually re-duplicate the data when moving it from disk to tape.**

Deduplication does not add significant I/O overhead when performed on disk storage pools. However, when deduplicated data is naively placed on tape storage, the high degree of data fragmentation caused by deduplication--combined with the high seek and mount times of today's tape technology--leads to high retrieval times. This negatively impacts the recovery time objectives (RTO) that the service provider has to meet as a part of the service level agreement (SLA).

This work proposes CloudDT, an extension to Cloud backup and archival services to efficiently support deduplication on tape pools. This paper (i) details the main challenges to enable efficient deduplication on tape libraries, (ii) introduces a class of solutions based on graph-modeling of similarity between data items that enables efficient placement on tapes, and (iii) presents the design and initial evaluation of algorithms that alleviate tape mount time overhead and reduce on-tape data fragmentation.

Using 4.5 TB of real-world workloads, our initial evaluations show that our algorithms retain at least 95% of the deduplication storage efficiency, and offer up-to 40% faster restore performance compared to the case of restoring non-deduplicated data. Therefore, our techniques allow the backup service provider to increase tape resource utilization using deduplication, while also improving the restore time performance for the end-user.

I. Introduction

Data backup and archival in the cloud is becoming increasingly popular due to the agility and ease of management it provides for customers. Cloud backup providers need to offer their services at a compelling price point, realize strong margins, and still offer competitive Service Level Agreements (SLAs). Cloud backup providers typically use a combination of disk and tape in their backend data stores [16, 18, 19, 3]. While tape has a high mount and seek time overhead when compared to disk, it is significantly cheaper, has a much longer shelf life, lower bit error rate, and is more energy efficient for long term archiving [15, 17, 1]. Furthermore, technologies like the Linear Tape File System (LTFS) make large tape library farms increasingly easier to use in an online fashion [8].

However, disk-based data reduction techniques like deduplication cannot be applied easily to tapes. This is because deduplication results in high data-fragmentation, and tape's high mount and seek times make restores of highly fragmented data prohibitively time consuming. Therefore, backup systems usually re-duplicate the data when moving it to the tape pool [10, 11]. Backup and archival data generally consists of desktop and server images, virtual machine images, database snapshots, and email; these types of data are highly deduplicable [4, 5, 6, 20]. Our own analysis confirms this (§IV.A.4): for workloads captured from a commercial enterprise backup system, deduplication reduced the storage footprint by 32%.

We propose a system called CloudDT (Cloud-based Deduplicated Tape) that allows Cloud backup service providers to preserve deduplication when moving data to tape. To the best of our knowledge, this is the first work to explore solutions for data deduplication on tape libraries: (i) we identify and address the main challenges for efficient data deduplication on tapes, (ii) we present an innovative graph model for representing deduplicated data with sparse, low memory footprint graphs, a key enabler for efficient graph processing, and (iii) our initial evaluations show that our cross- and on-tape data placement algorithms are able to preserve up to 95% of the deduplication efficiency, while significantly improving the perfomance of both migrating data to tape, and restoring data from tape (up to 40% faster restore times).

The rest of this paper presents related work (§II), the operating environment (§III), the solution and early evaluation for chunk placement algorithms (§IV), and (§V) summarizes our work with future outlook.

II. Related Work

We are not aware of any refereed publication presenting a solution for storing and restoring deduplicated data on tape pools. Data protection solutions from Symantec (Backup Exec [9]) and EMC (Avamar [10]), which include tape pools as a back-end disaster recovery storage have a deduplication option. Still, it is not clear whether the data written on tape is

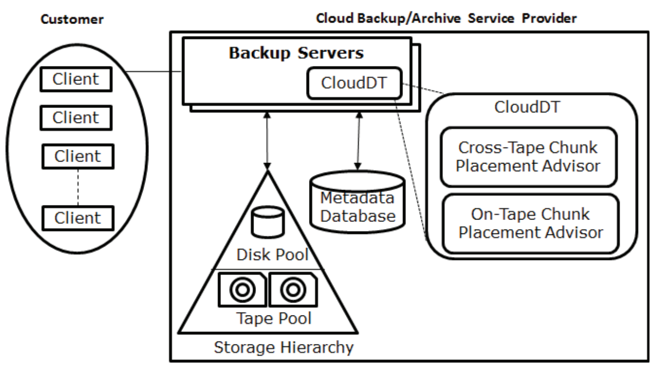

Figure 1 Design of a typical Cloud backup service

deduplicated or not (i.e. re-duplicated to original file format similar to what IBM Tivoli [11] does), and if deduplicated, how the chunks are placed on the tape pool.

Burns et al. [21] present a solution for managing delta backups for file versions on tapes. Lillibridge et al. [22] present a technique for efficient chunk-lookup for large-scale deduplicated tape images, but do not address the problem of chunk placement. Related literature on deduplication technology has mostly focused on the efficiency of duplicate detection [23-25], design [26] and scalability [27, 28] aspects on disk storage.

III. Operating Environment

Our proposed framework CloudDT (Figure 1) is based on the design of typical enterprise-class data-protection and recovery services [11], which are hosted inside the Cloud service provider's facility. These data-protection services provide a storage-medium agnostic interface to their clients, which are usually enterprise or web application servers. The clients use these interfaces to perform backup of customer-data based on a defined schedule, and restore it on demand. Clients specifically communicate with a component called the backup service, which performs data protection operations. Internally, the backup service orchestrates data placement—this includes staging data from non-deduplicated to deduplicated disk pools and eventually to tape storage.

In this context, the role of CloudDT is to determine on which tape and where within that tape a data-chunk will be placed when migrating already deduplicated data from disk pools to tape pools. We refer to these operations as cross- and on-tape chunk placement, respectively.

CloudDT constructs the placement plan by examining the disk pool's deduplication metadata (typically stored in a scalable database) which offers information about data-chunk dependencies between files.

IV. Solution

A. Cross-Tape Chunk Placement

The cross-tape chunk placement algorithm has two main inputs: (i) the deduplication metadata (the chunk-maps of files),

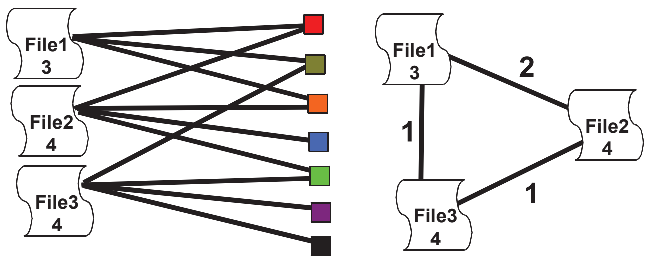

Figure 2 Left: Chunk-centric model. Right: File-centric model. (This example assumes fixed-sized chunks and weights represent number of chunks).

and (ii) the tape size(s). The output is the list of chunks that will be placed on each tape.

Cross-tape chunk placement has the following requirements: (i) all chunks of a file must be available on a single tape, to minimize tape mounts, and (ii) the solution must be scalable and fast. The competence of the solution is measured by the following metrics: (a) Deduplication loss – how much data is re-duplicated across tapes, due to the constraint that all the chunks of a file must be assigned to a single tape, (b) transfer-to-tape performance, and (c) restore performance.

Our cross-tape chunk placement algorithm uses a graph representation to model similarity between files. The motivation behind this approach is to leverage graph algorithms to identify clusters of files that share significant amount of data. The idea is to place these clusters together on the same tape to reduce the cost of replicating chunks across tapes.

1) Graph Models

We identify two main ways to expose data similarity through a graph representation: chunk-centric and file-centric (Figure 2). Note that either representation supports both fixed- and variable-sized chunks.

The ***chunk-centric*** model represents both chunks and files as vertices. Edges exist between files and chunks only: an edge exists between a file and a chunk, if the chunk exists in the file; hence forming a bipartite graph (Figure 2, left).

Most practical graph processing algorithms assume that the graph data-structure fits in memory. Although the chunk-centric model includes detailed, chunk-level sharing information, it may consume prohibitively large amounts of memory when constructed for Cloud-based data repositories. E.g. the size of the chunk-centric graph can be proportional to $O(\textit{number of files} * \textit{number of unique chunks})$, if all files contain all chunks. Furthermore, since chunks are usually much smaller in size relative to files, the number of unique chunks may be orders of magnitude higher than the number of files. Therefore, we considered a sparser graph model, dubbed file-centric, which offers a much lower memory footprint.

The ***file-centric*** model represents only files as vertices. Edges are placed between files that share chunks. Edge weights represent the amount of sharing (Figure 2, right). This model summarizes the detailed information offered by the chunk-centric model: it includes the amount of sharing rather than which chunks are shared. This model produces a relatively

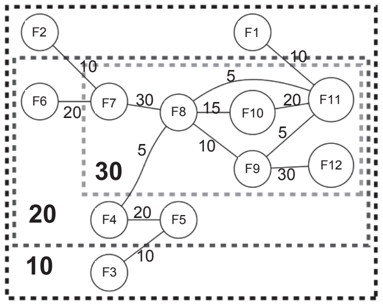

Figure 3 p-core algorithm in a hypothetical scenario. The files inside the inner clusters share more data with each other than the files outside. Numbers in bold indicate corness.

small graph that is generally proportional in size to the order of number of files.

The vertices in our file-centric graph are connected with a minimal set of edges in the following sense. Let's say that n files share a chunk, then the number of edges between this set of files is *n – 1*. The idea is to not have duplicate edges that model the sharing of the same content between files.

This enables an important property of our file-centric graph model: the ability to approximate --as an upper bound-- the deduplicated size of a set of files (or its corresponding graph or sub-graph) through a simple graph traversal. In particular, size can be computed by a breadth-first search (BFS) traversal that sums the vertices' weights (which represent files sizes) and subtracts the edges' weights (which represent the duplicates size). We note that for a non-partitioned connected component of the graph this is not an approximation but always produces the correct result.

2) Building the Graph

To efficiently build the file-centric graph, given the deduplication metadata (file-chunk map), we perform the following steps. (I) Group the files by chunk hash: essentially sort the file-chunk map by the chunk hash. (II) Create edges: create an edge between any two files that share a chunk, this is done by doing a linear scan of the list sorted in the previous step; to satisfy the minimal set of edges requirement we use a star-like connection topology (STAR): for a set of files that contain a given chunk, one of the files is declared a 'master node' and is linked with every other file. This step may produce more than one edge between any two vertices – consider two files sharing two different chunks. (III) Group edges: group together edges between similar vertices (files). As in the first step, this is done via sorting. (IV) Reduce edge weights: perform a sum reduction over the weights of similar edges. This is done via a single scan over the ordered edge-list that resulted from the previous step.

The graph generation process does not impose excessive memory requirements. The performance of the two major operations, hash and edge grouping, can benefit from additional physical memory, but these steps can also be processed efficiently on disk.

3) Partitioning the Graph

The goal for partitioning is to divide the graph into a number of partitions such that each partition can fit into a given tape size, while reducing the number of chunks replicated across the resulting partitions (i.e., reducing the edge cuts).

Table 1 Workload summary

Workload	*WL1*	*WL2*
Total size	3,052GB	1,532GB
Duplicates (size)	978GB	460GB
Duplicates (%)	32%	30%
Num. of files	289,295	201,406
Avg. file size	10MB	7.79MB
Median file size	82KB	18KB
Num. of chunks	17,509,025	12,021,126
Avg. chunk size	182KB	102KB
Median chunk size	71KB	52KB

One solution to this problem is to recursively bisect the graph into two partitions of about equal size, while minimizing the number of edges that span the partitions. This is an NP-complete problem [12], but a number of heuristics exist to address it. The best available heuristic has *O(ve)* complexity [12], where v is the number of vertices and e is the number of edges. Such expensive algorithms can potentially be a bottleneck, especially when considering large petabyte-scale storage systems.

However, in our case, partitioning is not meant to identify balanced sized partitions; rather its goal is to identify clusters of files that share significant amounts of data. To this end, we propose a simple partitioning heuristic that is based on two linear-time graph processing algorithms: breadth-first search (BFS) and p-core [13, 14].

Our algorithm first identifies connected components in linear time using BFS. Since separate components do not share data, a component that fits within a tape can be placed on it without deduplication loss. If the size of a component is larger than the tape-size, then we partition it further using the p-core clustering algorithm.

A p-core is a maximal subgraph in which the sum of the edge weights of any vertex is at least p. The p-cores decompose a graph into nested subgraphs, shown as "circles" in Figure 3, where core *(p+1)* is a subgraph of core p. The "coreness" of a vertex is defined as the maximum core it belongs to. Note that the inner circles contain vertices with higher coreness. The p-core finding algorithm runs in *O(e)* time, where e is the number of edges.

The core-based partitioning algorithm divides the large-sized components into smaller partitions such that each partition spans a range of circles. For example, the first partition includes files with coreness between [0,x), where x is chosen such that the partition size fills a tape, the second one between [x,y) and so on. This partitioning heuristic leads to grouping together files that share a specific amount of data, hence reducing the chance of splitting clusters of files that share significant amount of data.

4) Cross-Tape Chunk Placement Evaluation

We evaluated our algorithm using two real workload traces taken from Tivoli Storage Manager (TSM), a commercial backup solution from IBM. Deduplication in the traces is based

Table 2 Graph characteristics

Workload	*WL1*	*WL2*
Time to generate	5.6min	3.8min
# of nodes	289,295	201,406
# of edges	327,472	246,244
Graph density	8e^{-6}	12e^{-6}
# of components	166,089	149,083
Largest component	695GB	987GB

on variable size chunking that used SHA1 to produce chunk fingerprints. Table 1 shows the characteristics of the workloads.

Table 2 shows the characteristics of the generated graphs. The low graph density and edge degree of both workloads highlights the sparsity of our file-centric graph model. The graphs were generated using a relational database on a commodity machine (quad-core Intel processor and 8GB of memory). The Linux sort command was used for the grouping steps described in §4.1.2. These steps accounted for over 90% of the processing time.

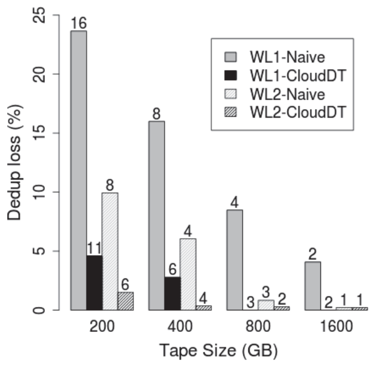

Figure 4 CloudDT vs naive placement. Numbers on the bars are the resulting partitions. Missing bars represent zero dedup loss.

We compare CloudDT with a naive placement heuristic. The naive algorithm simply places the files one by one, as long as there is enough space left, and then the files on the same tape are deduplicated.

Figure 4 shows the results. Small tape sizes force the algorithms to cut sharing dependencies between files to create corresponding small partitions. Still, CloudDT maintains a lower than 5% dedup loss, and offers 5 times improvement over the naive algorithm for the smallest two tape sizes. As the tape size increases, the deduplication loss decreases as larger clusters of files can fit in a tape. For the largest two tapes, CloudDT is able to place the whole WL1 without loss in deduplication as even the biggest component can fit in a tape without partitioning.

B. On-Tape Chunk Placement

Tapes have high seek times in comparison to disks [8], this factor combined with the data fragmentation resulting from deduplication can lead to high file restore times, if chunks are not carefully placed on tape. As an initial attempt to address this issue, we implemented and evaluated a simple yet effective on-tape chunk placement algorithm -- using a method proposed by Knuth [29], we assumed that all files have an equal probability of access, and we placed their unique chunks on tape in increasing file-size order.

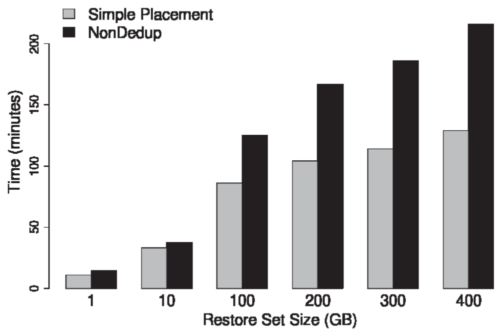

Figure 5 On-tape chunk placement -- restore performance while varying the restore set size. (LTO-5 FC Tape Drive, 1.5 TB Cartridge).

We experimentally compared the performance of our deduplicated chunk placement algorithm, with that of a non-deduplicated data placement algorithm, which also placed the files in increasing size order, but without the data fragmentation of deduplication. Our experiment randomly picked files from WL1 (Table 1) and defined "restore sets" ranging from 1 GB to 400 GB in size.

Figure 5 shows that while reading these "restore sets", our simple dedup placement algorithm consistently outperformed the non-dedup on-tape placement, with restore times that were up to 40% faster (for 400 GB). The simple dedup placement algorithm was effective for two reasons: (i) duplicate data has enough spatial locality that the impact of data fragementation is mitigated, (ii) our dedup data placement has shorter seeks than the non-deduplicated data placement, and is also reading less actual data from tape.

V. Summary and Future Work

The amount of archived data is predicted to grow at massive rates: doubling every two years and up to 50 times by 2020 [2]. In this paper we propose a system called CloudDT, a deduplication system for tape that allows Cloud-based backup and archive service providers to get increased efficiency from their tape storage resources.

Our initial evaluation shows that CloudDT's chunk placement algorithms preserve up to 95% of the deduplication efficiency, while significantly improving restore time performance. Next, we plan to study sparser and hierarchical graphs to model larger Cloud archives, address incremental backups, and evaluate new algorithms to further increase the efficiency of on-tape chunk placement.

Finally, in addition to tape deduplication, we believe that our graph model, and associated clustering algorithm, can be applied to other use-cases such as chunk placement for deduplicated disk-based pools in large shared-nothing commodity server clusters, where servers have asymmetric access to disks, or to reduce failure propagation on disk pools (if a disk fails, data loss does not spread to other disks).

References

[1] D. Reine and M. Kahn, "In Search of the Long-Term Archiving Solution — Tape Delivers Significant TCO Advantage over Disk," Clipper Notes, 2010.

[2] R. Amatruda, "Worldwide Tape Drive 2011–2015 Forecast Update and 2010 Vendor Shares," IDC Report, 2011, .

[3] Google, "http://gmailblog.blogspot.com/2011/02/gmail- back-soon-for-everyone.html," Official Gamil Blog, 2010, .

[4] D. Bhagwat, K. Eshghi, D. D. E. Long and M. Lillibridge, "Extreme binning: Scalable, parallel deduplication for chunk-based file backup," in Modeling, Analysis & Simulation of Computer and Telecommunication Systems, 2009. MASCOTS'09. IEEE International Symposium on, 2009, pp. 1-9.

[5] K. Jin and E. L. Miller, "The effectiveness of deduplication on virtual machine disk images," in Proceedings of SYSTOR 2009: The Israeli Experimental Systems Conference, 2009, pp. 7.

[6] D. Meister and A. Brinkmann, "Multi-level comparison of data deduplication in a backup scenario," in Proceedings of SYSTOR 2009: The Israeli Experimental Systems Conference, 2009, pp. 8.

[7] Ultrium – LTO Technology, "http://www.lto- technology.com, 2011," .

[8] D. Pease, A. Amir, L. Villa Real, B. Biskeborn, M. Richmond and A. Abe, "The linear tape file system," in Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on, pp. 1-8.

[9] CA ARCserve Backup, "http://www.arcserve.com/us/products/ca-arcserve- backup.aspx, 2011," .

[10] EMC Avamar, "http://www.emc.com/collateral/software/data-sheet/h2568-emc-avamar-ds.pdf, 2011," .

[11] IBM Tivoli, "http://www.ibm.com/software/tivoli/, 2011," .

[12] M. Newman, Networks: An Introduction. Oxford Univ Pr, 2010.

[13] V. Batagelj and M. Zaversnik, "An O(m) algorithm for cores decomposition of networks," Arxiv Preprint cs/0310049, 2003.

[14] V. Batagelj and M. Zaversnik, "Generalized cores," Arxiv Preprint cs/0202039, 2002

[15] Disk and Tape Storage Cost Models. Richard L. Moore, Jim D'Aoust, Robert H. McDonald and David Minor; San Diego Supercomputer Center, University of California San Diego; La Jolla, CA, USA

[16] Fujifilm Permivault Intelligent Online Archive, "http://www.permivault.com/technology, 2012".

[17] A. Osuna, R. Sharma, M. Silvestri and S. Wiedemann, "IBM System Storage Tape Library Guide for Open Systems," IBM Redbooks, pp 59, 84, 2011.

[18] Crossroads Systems Tape for Cloud Storage, "http://www.crossroads.com/Company/PressRoom/PressReleases.asp?id=477, 2012".

[19] Quantum LTFS NAS Appliance, "http://www.computerworld.com/s/article/9226247/Quantum_releases_LTFS_appliance_that_makes_tape_like_NAS, 2012".

[20] M. A. Smith, J. Pieper, D. Gruhl and L. V. Real, "IZO: Applications of Large-Window Compression to Virtual Machine Management," in Proceedings of the 22nd Large Installation System Administration Conference (LISA '08), 2008.

[21] R. C. Burns and D. D. E. Long, "Efficient distributed backup with delta compression," in Proceedings of the Fifth Workshop on I/O in Parallel and Distributed Systems, 1997.

[22] M. Lillibridge, K. Eshghi, D. Bhagwat, V. Deolalikar, G. Trezise and P. Camble, "Sparse indexing: Large scale, inline deduplication using sampling and locality," in Proccedings of the 7th Conference on File and Storage Technologies, 2009.

[23] C. Policroniades and I. Pratt, "Alternatives for detecting redundancy in storage systems data," in Proceedings of the Annual Conference on USENIX Annual Technical Conference, 2004.

[24] S. Quinlan and S. Dorward, "Venti: A new approach to archival storage," in Proceedings of the Conference on File and Storage Technologies, 2002.

[25] D. T. Meyer and W. J. Bolosky, "A study of practical deduplication," in FAST'11: Proceedings of the 9th Conference on File and Storage Technologies, 2011.

[26] F. Guo and P. Efstathopoulos, "Building a high- performance deduplication system," in Proceedings of the 2011 USENIX Conference on USENIX Annual Technical Conference, 2011.

[27] B. Zhu, K. Li and H. Patterson, "Avoiding the disk bottleneck in the data domain deduplication file system," in Proceedings of the 6th USENIX Conference on File and Storage Technologies, 2008.

[28] C. Dubnicki, L. Gryz, L. Heldt, M. Kaczmarczyk, W. Kilian, P. Strzelczak, J. Szczepkowski, C. Ungureanu and M. Welnicki, "Hydrastor: A scalable secondary storage," in Proccedings of the 7th Conference on File and Storage Technologies, 2009.

[29] Knuth, Donald, "The Art of Computer Programming, Second Edition", pp 403-404,1998.