SAME: An Extended Semantic-Aware Multi-tiered Source Deduplication Framework on Cloud Backup

Chynna Julia Cordevilla, Rafael Gerard Formes, Francis Bien Jan Viernes

***Abstract*—Goes here**

I. Introduction

CLOUD computing is catching on fast, and people are becoming more and more reliant on storing and processing their data on the cloud. In fact, in todays world, the explosion in the volume of online digital content resulted in an enormous strain on storage systems. According to the recent study of the International Data Corporation, 2.8 Zettabytes of data will have been created and replicated in 2012. By 2020, it is predicted that the digital universe will skyrocket to 40 ZB, an amount that exceeds previous forecasts. As cloud computings role grows in the management of the Big Data, the number of servers worldwide is expected to grow tenfold and the amount of information managed directly by enterprise data centers will grow by a factor of 14. [1] The digital universe lives increasingly in a computing cloud, above terra firma of vast hardware data centers linked to billions of distributed devices. The continuous barrage of data (the addition, deletion, and updating of data) on cloud servers creates a problem that database designers are familiar with: data duplication, or the presence of data duplicates.

Data duplication is a problem, most especially in the cloud, for the following reasons:

1) Wasted storage space; redundant storage means more storage space is used up. This is going to be a problem if the space being used or allotted for various users is/are limited.
2) Data inconsistency; it is highly likely that a file update will get lost among the redundant data, because various links still point to all the previous version/s of said file. This is a huge problem most especially if the cloud server is shared among different users.
3) Slow data searches and high data management costs; redundant storage would mean that the system would have to go through a lot more data than it should to be able to find a single file. This is going to be a huge problem if the cloud holds a lot of data.
4) Slow data transmission; having duplicates in a system would mean more backups, and a longer wait for data to transmit.

These are why data duplication must be addressed, and data deduplication (as the name implies), does exactly that.

A. Significance of Study

Many deduplication services have sprouted through the years as storage systems have evolved into a necessity for business and personal use alike. All these aim to solve the problems aforementioned, making the storage process more efficient than it would without deduplication.

Research on the use of deduplication methods that efficiently reduce space occupied by redundant data have already been done, beginning with simple storage systems [12] and with how data can be stored in a manner that would make it easier for future computations such as data access and retrieval [5], making its way into data streams [8], even to adding interfaces to mediate data transmissions [7], and finally, into cloud storage systems [1][2], cloud backup systems [3][4], that involve mobile systems [9] as well.

However, despite being a solution, researchers have proven that deduplication holds much room for improvement. Many researchers have been working on ways to improve the deduplication process, looking into various hashing techniques, applying various algorithms, and rearranging the architectures of their systems with the goal to optimize them further.

One such improvement we are interested in is the application of semantic-awareness into the deduplication system.

B. Scope, Limitations, and Objectives of the Study

Among the numerous applications of data deduplication, this study aims to focus on Cloud Backup due to the reason that Cloud Storage is a booming industry and innovations within this field will be helpful towards advancing said technology. There is also much room for improvement and study within this field, as the architecture is quite different from older systems.

Initially, we wanted to delve into the manipulation of semantic data for efficiency, as the use of semantic and contextual data has proven very useful to recent studies and projects. The Semantic Aware Multi-Tiered Source Deduplication Framework is a notable deduplication solution that uses file semantics (e.g. file locality, file type, file size, file timestamps), and it aims to achieve an optimal tradeoff between deduplication efficiency and deduplication overhead. We decided that its use of file semantics, its application to Cloud Backup and Cloud Storage, and its achieved ratio between efficiency and overhead makes it an appropriate framework to study and add on to. Hence, we develop SAMe: an Extended Semantic Aware Multi-Tiered Source Deduplication Framework for Cloud Backup.

Having already given the scope, the objectives of SAMe are as follows:

- To readily solve the aforementioned duplication problems: wasted storage space, data inconsistency, slow data searches, and slow data transmission.

- To improve the deduplication solution the SAM offers by adding additional semantics.
- To significantly reduce deduplication overhead, deduplication time, disk IO, and the bandwidth eaten up by deduplication.

The following sections of this paper is organized as follows. Section II will contain a review of literature related to our study, giving reasons as to why the solutions present in SAM are viable and efficient, and why our additional semantics are viable and efficient, as well. We placed our methodology in Section III, wherein we describe the specifics of the algorithms we used, test data, and other important data we used throughout the duration of experimenting and testing. In Section IV we further discuss our attained results.

II. Review of Related Literature

A. Data Deduplication

Data Deduplication is a specialized data compression technique for removing duplicate copies of repeated data. Deduplication is considered to be the most important emerging technology in the field of data storage backup research [14]. Most deduplication research focuses on the improvement of each step of its major processes, namely: data storage, data partitioning, indexing, and searching.

Existing deduplication solutions make use of various methods to make data deduplication faster and more efficient. Some known solutions make use of compression in the chunk-level and/or file-level, global and/or local deduplication [4], hashing [1] and, recently, the exploitation of file semantics [4][12][13]. This section will focus on the general considerations made when creating a deduplication system.

1) Data Storage and Partitioning:

a) File-level vs. Chunk-level: The speed and efficiency of the data deduplication process is heavily-reliant on how the data is represented and partitioned in the system, therefore numerous research has been done with regard to this topic. Two types of data partitioning have been implemented in these research, namely: a) saving data as files, and b) partitioning data into chunks. It is important to take note that speed and accuracy are factors that should be considered when determining which type will best serve the deduplication process one has in mind. Focusing on the chunk-level has been proven accurate for data deduplication, but at the cost of more lookups which demands more memory for computations. This is slower compared to focusing on the file-level which has fewer lookups, since the chunk-level has more elements that should be compared to each other. Searching within the file-level, however, is less accurate when it comes to duplicate detection.

Regarding data chunking, data used to be divided into meaningless chunks with fixed sizes until projects such as Semantic Data Deduplication (SDD) [12] came about and used file semantics in the process of chunk-partitioning, since the researchers behind SDD believed that grouping data by factors that they have in common will make the searching process faster. SDD proved to be able to reduce the storage used by 20-50 percent compared to methods preceding it. Data-writing performance also increased by 50-70 percent.

b) Local vs. Global: In addition to the types of data-partitioning, there is also another factor to be considered in deduplication, and that is: the scope of the deduplication. Deduplication can be applied locally, or within the clients filesystem. This means only having to go through files or chunks that are stored within a single computer. Deduplication can also be applied on a global level, where the process would look into either files or chunks across clients and across servers to check each for duplicates.

Recent research use hybrids of both global and local, file-level and chunk-level deduplication.

One such example is the AA-Dedupe [4], which is a deduplicating application that caters to four factors (i.e. a large percentage of storage is occupied by a very small number of large files with very low sub-file redundancy) surveyed in an experiment. The application adapts to these scenarios and switches to chunk or file level, whichever is best for the situation.

SAM [3], on the other hand, uses a hybrid of both chunk-level and file-level approaches. SAM makes use of chunk-level deduplication locally, while using file-level deduplication for a broader, global level, e.g. for comparing and for searching across multiple computers. Using chunks in the global level would mean longer searches, and using files on the local level would mean lesser accuracy, hence, SAMs idea of using both chunk-level and file-level approaches within their proper areas achieves greater speed and efficiency. In fact, one of the main goals of this particular research is to achieve an optimal tradeoff between deduplication efficiency and deduplication overhead to maintain a shorter backup window than existing solutions, which it has proven to do in the end.

c) Scope vs. Target: Aside from considering both the data-partitioning and the scope of deduplication, a third factor exists: the area upon which deduplication must be applied.

There are two classifications under this factor: a) source deduplication, and b) target deduplication. Source deduplication removes redundant data before any transmission occurs. This helps reduce the bandwidth used for transmission. Target deduplication, on the other hand, removes redundant data as the data is transmitted, e.g. between the source and backup target. Unlike with source deduplication, target deduplication does not help with reducing the bandwidth. Target deduplication, however, is a better option for larger sets of data, although it requires some hardware to be placed at remote sites.

2) Indexing and Searching Methods: Determining which indexing and searching methods should be used is very essential to the speed and efficiency of the entire deduplication process, as these functions are what run during the entire procedure.

Hashing is used in deduplication, as it provides a quick solution to data accessing. The ones most frequently used in deduplication are both MD5 and SHA-1, since these algorithms are the least likely to garner hash collisions (MD5 with the hash collision probability of $\frac{1}{2^{28}}$ and SHA-1 with $\frac{1}{2^{160}}$, which is important in deduplication, since file-uniqueness is being tested.

Since hashing is to be applied in deduplication, it becomes an additional factor to be considered when choosing how to store files (chunk vs. file) and when deduplication must be used (global vs. local).

In an experiment worth 475.2 GB of data, the deduplication size they computed was 37GB, but the final size was 38.1GB. The extra 1.1GB came from the extra data created from lookups, which is still good. This goes to show that storage isnt the only factor to be focused on when it comes to deduplication, but also efficiency on reads and writes, file integrity, and the like, and the use of good hashing and indexing techniques make such efficiencies possible.

3) Applications and Testing: Deduplication would be useless without any applications to use it with. Most studies on deduplication deals with testing the efficiency of their deduplication process using their own unique applications. The niches covered range from applying deduplication in Virtual Machine clouds [11][14] to creating a deduplicating language known as Dedupalog [13]. Other applications involve applying deduplication methods onto platforms such as the Mobile Web [9].

B. The Semantic-Aware Multi-Tiered Source De-duplication Framework for Cloud Backup (SAM)

1) Semantic-Aware Systems with Data Deduplication:

2) SAM Baselines: Two well-known and widely-used methods of deduplication are the Local Chunk Deduplication Scheme (L-CDS), which chunks files locally (leaving 40.69 percent of redundant data), and the Global Chunk Deduplication Scheme (G-CDS), which chunks files on a global scale (leaving 0 percent redundant data). Despite being extremely efficient in deduplicating, however, G-CDS leaves much room for improvement in deduplication overhead. Both baselines have been used by SAM for it to achieve its goal of achieving an optimal tradeoff between efficiency and overhead.

3) SAM Semantics: Studies have shown that similar and/or related data are likely to have unneeded file duplicates, proving that the use of file semantics as a factor could greatly improve the deduplication process.

One of the earliest examples of such research is [12], which uses semantic information such as Application Metadata (file type, file format), Application or User Tags (blog tags, multimedia tags), and Filesystem Metadata, to be able to chunk all related data together into what the researchers call Semantic Chunks or SCs. The system then obtains these SCs and stores them further into fixed-sized objects for easier data management, which are then stored in Intelligent Storage Nodes or ISNs for quick searching. The method proposed in [12] reduces the used storage by 20-50 percent compared to methods before it. Data-writing performance also increases by 50-70 percent.

SAM follows this trend and is noteworthy because of its clever utilization and addition of file semantics to effective methods of deduplication. Many file-semantics attributes can be exploited to narrow down the search space of redundant data, which can reduce the overall deduplication overhead

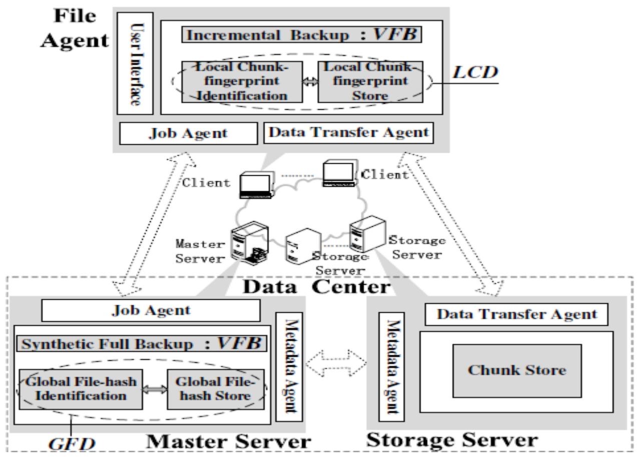


Fig. 1. SAM Phases

while maintaining high deduplication efficiency. [3] SAM utilized this fact to drastically reduce Disk IO, hence, decreasing time and overhead as well.

File semantics can be revealed at various levels, including definitional, associative, structural, environmental level or through other relative information of the files, where various hints can be obtained. [3] SAM focuses on File Locality, File Size, File Type, and File Timestamp.

- **File Locality.** SAM uses this semantic hand-in-hand with caching. Studies say that it is likely that the files in a folder with a duplicate file are also duplicates of other files found within the folder where the original file resides. With that being said, SAM caches files that have the same locality as previously checked original files so that searching is made more efficiently.
- **File Size.** SAM had concluded that files that are less than 8KB are to be ignored due to their research. It is said that 63.8 percent of the files in the storage are less than or equal to the said size. These files are checked and they deduced that these small files have only 0.53 percent chances of getting duplicated, plus it reduces the overhead of the process.
- **File Type.** When it comes to chunking, SAM filters out multimedia files and compressed files, as these files are represented differently in binary form, thus, updates arent detected via chunking. Also, multimedia files are less likely to be updated, and are the least likely to have similarities in chunks that will determine overall similarity, so SAM usually leaves these types solely to file deduplication.
- **File Timestamp.** SAM also exploits the file timestamps to easily determine what files could possibly be updated / changed. A file is unchanged simply if the timestamp is also unchanged, removing it from the deduplication process, lowering the lookups.

4) SAM Phases: SAMs architecture consists of three parts: the File Agent, the Master Server, and the Storage Server. The File Agent resides in each client computer, and is in charge of all local processes to do with the clients filesystem.

Such processes include file hashing, sending file hash lists to the Master Server, chunking files, and sending chunks to the Storage Server. The Master Server, on the other hand, is in charge of gathering lists of file hashes from different clients and checking them for duplicates. Only lists are stored in the Master Server, as all its processing power is used for comparison and indexing. The Storage Server is where all the file chunks are stored and backed up.

Having each parts main goals arranged linearly in order, SAMs entire system can be arranged into three phases: Virtual Full Backup (VFB), Global File-level Deduplication (GFD), Local Chunk-level Deduplication (LCD), all of which will be discussed in the methodology section of this paper.

III. Methodology

We used SAM as our baseline and from there, we would compare our work, from optimizations to different/additional algorithms.

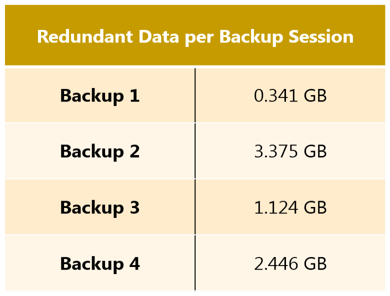
A. Implementation of SAM

There are 3 major stages that divides the whole SAM process. And these are the following:

1) **Virtual Full Backup**: This is the initial filter of duplicates. Files are checked locally within each of the clients where they would send the only ones that are unique. Each files are checked through its attributes and hashes to help determine more easily if the file is changed or not. After checking, they are then hashed and are sent the master server.
2) **Global File-Level Deduplication**: The sent hashes, from all of its clients, are received by the master server and are then to be further filtered. If there are duplicate files with the other clients, they are then notified back to the clients to not send that/those specific file/s. Multimedia files wont be checked anymore, immediately chunking it and sending directly to the master server; this is because a change to a single pixel (multimedia, for example .jpeg file) would give a very different hash. After these, the clients will now proceed to chunking.
3) **Local Chunk-Level Deduplication**: The clients will now chunk the files that are labeled unique by the master server and will send these chunks to the storage server to be saved.

B. Operating System

We used Ubuntu 14.01 as the environment where we ran the baseline. Soon, we would be porting part of our code to Windows 8 to build the software for the clients side; the server side code will still be ran in Ubuntu. This is because we initially planned this program for Microsoft Windows users, and soon, would also open up to the other common operating systems.



Redundant Data per Backup Session	
Backup 1	0.341 GB
Backup 2	3.375 GB
Backup 3	1.124 GB
Backup 4	2.446 GB

Fig. 2. Redundant data per session.

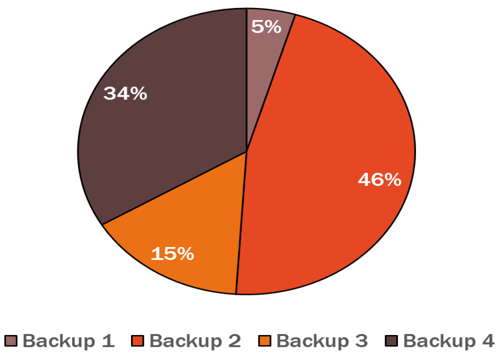


Fig. 3. Redundant data per session (in percentage).

C. Dataset

The dataset that we first used were .txt, .png, .jpg, .doc, .ppt, .mp3, .html, .scl and .c to give an approximate to the files that a student usually saves via cloud. We got samples of these files of roughly 2.4GB worth of unique files and distributed them randomly, with repetitions, to a certain number of folders (our temporary clients) resulting to 11.46GB of files to be tested (Fig 4). We created the datasets we used such that they mirror how real-world student filesystems look like, adding mostly study material and organizing them in ways as how we would likewise organize them.

D. Assessment

1) Total percentage of duplicates: For our initial testing, we pre-determined the size (2.4GB as stated previously) then we randomly distribute those files throughout different folders and determine the whole size. The percentage of the size of duplicates would be determined by the excess size from the 2.4GB.

2) Deduplication Efficiency:

- **Time and Memory**. Time efficiency would be determined based on the baseline which is SAM[3]. If the length of the process is comparable to the baseline (or even better, which is our target), then we can say that it is acceptable enough. But another thing that should be considered is the amount of memory or lookups it takes up. There should be a trade-off between the both and that will be cleared off during the tests and runs.
- **Size and integrity**. Efficiency would also be determined by its resulting size. Since the size of our unique files

would be 2.4GB, then the resulting size of the 11.46GB test data must also be 2.4GB; that is if the deduplication is dependent on file comparison. Since we are deduplicating files at the chunk level, we may see results that are of size 2.4GB (for this test data) or even less, depending on how many duplicate chunks we can find. Of course the chunks must be tested if they can be reverted back to its full file form to confirm its consistency and integrity.

3) Deduplication Overhead: Deduplication overhead is described in terms of latency and reduced throughput. This would depend on how fast the server reacts to the clients and vice versa; also, data exchange from both sides should be successful and should avoid any penalties (packet losses, delays, etc.).

4) Backup Window: The backup window represents the time spent sending a particular dataset to a backup destination. The backup window consists of two parts: data de-duplication time and data transfer time. To obtain a fixed dataset, the deduplication time is generally fixed, whereas the data transfer time varies with network bandwidth.

5) Future Work: A good suggestion for future work regarding this project would be to create larger datasets, for a more thorough and accurate testing of our system. Perhaps, to obtain a few real-world datasets from willing participants will be sufficient. Related to testing, the frequency of each wave or each run of the program must also be considered in order to get a more accurate understanding of the systems backup window.

On a more methodological level, to add additional file semantics that would help boost deduplication efficiency and reduce overhead is another good thing to be added. Such a semantic that can be used is File Age. Delta Encoding can also be added for version control.

Another improvement that can be applied is to be able to port this system to various other platforms (Linux and Mac OS for the File Agent, Windows and Mac OS for the servers).

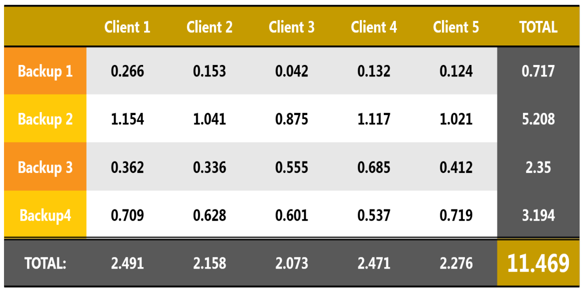
IV. Results and Discussion

We have created a simulator for SAM and fed the preliminary dataset specified in the previous section to evaluate SAM'S performance.

Given the percentage of duplicates as seen in figure 5, we managed to generate results (fig 6) that gives us SAM's average dedup efficiency of 94.8375 percent, which leaves about 5.1625 percent of redundancy intact. As one can see in fig 7 and fig 8, SAM's results are very close to 100 percent efficiency.

We ran our application and we got results that were slightly comparable to SAM, but what we didnt account for yet was the time it took for the whole process. We tried to see first if our deduplication process, at its basic form, would give considerable results.

We've also simulated the data transmission of the to-be-backed-up files using varying bandwidth and transmission speed (fig 9). Again, the results that we have obtained are almost similar to SAM's, the difference accounts to the diversity of the original paper's dataset.



	Client 1	Client 2	Client 3	Client 4	Client 5	TOTAL
Backup 1	0.266	0.153	0.042	0.132	0.124	0.717
Backup 2	1.154	1.041	0.875	1.117	1.021	5.208
Backup 3	0.362	0.336	0.555	0.685	0.412	2.35
Backup4	0.709	0.628	0.601	0.537	0.719	3.194
TOTAL:	2.491	2.158	2.073	2.471	2.276	11.469

Fig. 4. Dataset across five clients in four backup sessions. All numbers are in Gb.

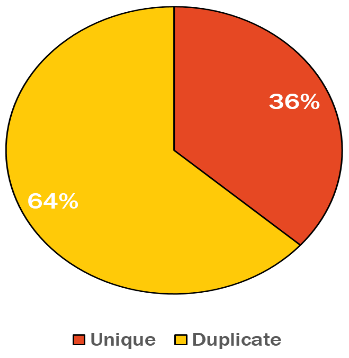
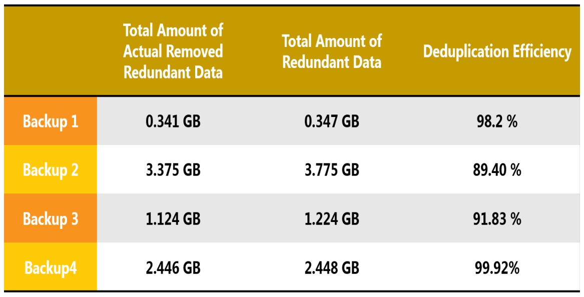


Fig. 5. Unique vs duplicated data in percentage in our dataset.



	Total Amount of Actual Removed Redundant Data	Total Amount of Redundant Data	Deduplication Efficiency
Backup 1	0.341 GB	0.347 GB	98.2 %
Backup 2	3.375 GB	3.775 GB	89.40 %
Backup 3	1.124 GB	1.224 GB	91.83 %
Backup4	2.446 GB	2.448 GB	99.92%

Fig. 6. Dataset across five clients in four backup sessions. All numbers are in Gb.

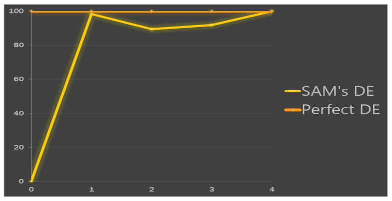
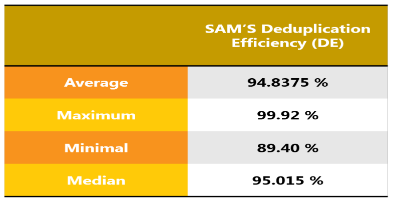


Fig. 7. Deduplication Efficiency of SAM vs Maximum.



	SAM'S Deduplication Efficiency (DE)
Average	94.8375 %
Maximum	99.92 %
Minimal	89.40 %
Median	95.015 %

Fig. 8. Deduplication Efficiency of SAM

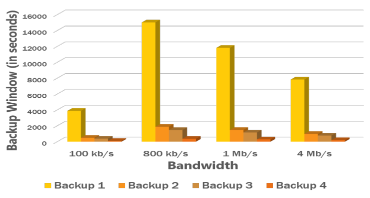


Fig. 9. Backup Window of SAM

References

[1] DeDu: Building a Deduplication Storage System over Cloud Computing
[2] SAR: SSD Assisted Restore Optimization for Deduplication-based Storage Systems in the Cloud
[3] SAM: A Semantic-Aware Multi-Tiered Source Deduplication Framework for Cloud Backup
[4] AA-Dedupe: An Application-Aware Source Deduplication Approach for Cloud Backup Services in the Personal Computing Environment
[5] Analysis of Data Fragments in Deduplication System
[6] Reducing the De-Linearization of Data Placement to Improve Deduplication Performance
[7] Chunk and Object Level Deduplication for Web Optimization: A Hybrid Approach
[8] Detecting Duplicates over Sliding Windows with RAM-Efficient Detached Counting Bloom Filter Arrays
[9] End-to-end Data Deduplication for the Mobile Web
[10] Estimation of Deduplication Ratios in Large Data Sets
[11] The Case for Content Search of VM Clouds
[12] Semantic Data De-duplication for Archival Storage Systems
[13] Large-Scale Deduplication with Constraints using Dedupalog
[14] IM-Dedup: An Image Management System Based on Deduplication Applied in DWSNs