Attribute Level De-duplication Backup System for Relational Database

Than Than Sint, Ni Lar Thein
University of Computer Studies, Yangon, Myanmar
thanthansint@gmail.com, nilarthein@gmail.com

Abstract

Efficient storage utilization is very important for today's business organizations to maintain business continuity and reduce the operational costs. Most of enterprises need large amount of data to operate their daily transactions. Data availability and integrity are becoming necessary for such systems for the extension of data retention periods in disk. So, we present the disk-based backup system in which relational database files are stored by using data de-duplication technology. Our system applies attribute-level backup strategy in reducing the replicated data among records of a relation (intra-database) and also inter-database. It reduces more redundant values than record-level and file-level de-duplication and other traditional backup methods, especially in storage consumption. We only need lower space for a large amount of files including more redundant data because this system eliminates redundant data not only within a file but also among different files. Our storage mechanism also supports to the purpose of recovery from accidentally damages. We use MySQL data set of one food mart. The result depends mainly on how much data duplication in our data set used. Our result reduces at least 25% in a little amount of duplication and more for larger duplicated data volume.

1. Introduction

Traditionally, magnetic tape has been used for data backup. Today's Enterprise data volumes are exploding as organizations collect and store increasing amounts of information for their own use. The amount of backup data may not fit into a single backup device. Some of companies today use more than 10 terabytes of data storage. With the explosion in disk capacity, it is now affordable to use disk for data backup. Using large amount of data volumes lets them to buy more storage, consume more processing power and energy in handling and managing the information, utilize more network resources in transmitting the material, and spend more time on related functions such as data backup and replication. However, much of the information in storage is duplicate data. Different users within an organization often create identical files or duplicate existing files so that they can work with them independently. To overcome duplication of data, organizations are increasingly using data de-duplication technology.

Data de-duplication technology applied to disk-based backup can substantially reduce disk-based backup storage consumption and bandwidth requirement. Recently, data de-duplication technology becomes a hot topic in the storage industry [11, 12, 13].

In our system, we propose a relational database backup system which uses attribute based de-duplication strategy in avoiding duplicate data within a relation or among relations (tables). Our system is implemented in the form of client server architecture using MySQL table format. Any of the clients connected to the remote server can send their data files when they want to backup in first come first serve order, and they can also do the de-duplication process at their sides locally. When a client does the de-duplication process at its side, it has to store its non-duplicated data at its side locally and have to send them to the server. The server accepts them sequentially and stores them in its local disk.

During the de-duplication process, this system finds non-duplicated data in the received file and stores them with the corresponding index values in disk at the remote server. These indexes form as a name of De_duped_Data_Store. For the subsequences of the duplicated data, our system only replaces them with the corresponding index values of the single instance stored in disk in the form of tables in which only index values are stored. So, before doing the replacement, tables are created as the structure of the original table but not as the format (data type, size, etc.) of it, which intends to save storage need, and form as Reference_Data_Store in disk. Using such kind of structure is definitely easy for the restore process. So, our backup system will increase the storage consumption at the server effectively and also reduce the size of data to be transmitted.

The rest of this paper is organized as follows: in the next section, we provide more background. Section 3 describes an overview of our approach while Section 4 introduces main components of the system in greater detail. In Section 5, we present a filtering mechanism for efficiency of our approach and tackling approach on updating to file in section 6. And we report on other approach with real data in the next section 7. Finally, we describe our conclusions.

2. Background

There have been a number of efforts to eliminate redundancy in file system, backup system and archival system. Most of the systems have used chunking to find the near duplicates in data repositories [16], conserve network bandwidth [17], and reduce storage space requirements [7, 19].

A system which uses individual fixed-size disk blocks, Venti [7], is a disk-based write-once storage system. It consists of an append-only log of blocks and a disk resident hash table for locating blocks within the log. A unique hash (SHA1 hash), of the contents of the block, identifies the blocks in Venti and Single Instance Store [10] are archival system to eliminate redundancy data. They use fixed size chunk in partitioning a file. SAN File System [2] is a file system to share the duplicate content among files. LBFS [1] uses ABC to help build a distributed file system on low-bandwidth network.

Deep Store [18] is a large scale archival storage system that uses both delta compression [20] and chunking to reduce storage space requirements.

Variable-sized chunking has also been used in the commercial sector. Pastiche [5] adopts 'variable sized chunking' in partitioning the file into chunks. Rabin's fingerprint algorithm [6] is the most widely used algorithm for this purpose. Objective of these files systems are to reduce the storage overhead. A number of distributed file system attempts to reduce the remote I/O overhead via synchronizing cache contents between the server and the client [8]. During the past few years, a number of de-duplication based software has been introduced. They are for email attachment [9], web document [3] or generic file system [4].

Some database backup system use attribute level de-duplication strategy in storing data from one location to another as a backup for the recovery purpose. Those systems have used this strategy only within one file content. Unlike them, our proposed system can eliminate redundant data among different file contents, thus it can save more replicated data than others and it can save the transmitted data from the clients' machines to the remote server.

3. Overall approach

Our backup system is implemented in client server architecture. There are many client machines which will store their database files at the remote backup server for the recovery purpose. When a client wants to store its data, it sends the database file and its associated schema file altogether.

Clients can backup their data at any time they wish. So, the backup server has to listen to which clients want to connect for backing up their data. When multiple clients want to keep their data simultaneously, this system does the backup process with first come first serve order.

At the client sides, they do the de-duplication process before transmitting their data to the remote backup server. At the time of doing this process, clients send those de-duped data to the server simultaneously and its schema in text file format which is actually needed to restore the original file. Doing de-duplication at the client sides intends to reduce the size of the data to be transmitted over the network.

4. Main processes

Our backup system consists of two main components: 1) Backup and 2) Restore.

4.1. Backup process

In the backup process, there are four main tasks to perform:

1) Building Database Schema: Database Schema is needed to restore the original data from the de-duped data and this schema is kept in the form of a text file. This file is given the same name with original data file.
2) Identifying the repeating contents: Repeated data are identified by comparing all attribute values and storing the distinct ones to data store. If an attribute value is not in data store, it stores the data itself. Otherwise, it only keeps the index value of those already stored data.

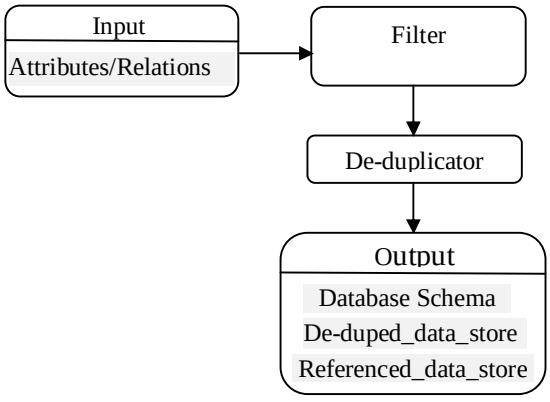

Figure 1. Block diagram of the de-duplication process

3) Keeping only the instance of the same copy in data store: Non- duplicated data are stored to disk using hash-based mapping for direct access.
4) Referencing the subsequent redundant data to the single data being saved: This step is needed to store the associated index values of the non-redundant content. It uses pointers or index values tables to a single copy already stored in "Deduped_Data_Store" are stored in

"Reference_Data_Store", which are built in an appropriate structure as shown in Table 3. These index values are created automatically for the individual non-duplicated data.

4.2. Restore process

Restore process is one of the most important steps in backup system to get the original data from the de-duped data stored in disk. This process can be done at the client's side locally and also at the server side remotely. Restore process does referencing or indexing to the non-duplicated data saved in data store and rebuilding the relation as the original one's schema.

There are three main tasks in this phase:

- Creating the table as the original file's schema by using it's database schema in text file format
- Mapping the pointer or index value in Deduped_Data_Store to the data stored in Reference_Data_Store and
- Restoring the data to the newly created relation.

5. Filtering mechanism

For the efficiency of de-duplication process, we use a filtering mechanism based on the type and length of all data fields in our MySQL data. We categorized data by their data types each first and then store non-de-duplicated data among them by their types in disk. This will affect that we do not need to check and make comparison on different data types' values. It will not only save time consuming but also be easier on restoration process.

Length filtering is used at the time of storing data to disk in order to detect duplicate data which have already been stored there. If the data's length is not the same, we do not need to compare and store it. So, this mechanism gives more benefits in less time consuming.

6. Tackling problem statement on updating to the same file

In our backup system, we don't allow to modify the file during the de-duplication process. But if some modifications are made to the same file after doing the de-duplication process, we need to handle such case in our system. So, we need to check that whether a client's file is already de-duped earlier or not. We can know that by examining that there is a text file with client's file name in Database Schema/ Catalog. If yes, that file is already stored in disk. After that, we retrieve its associated index table stored in Reference_Data_Store. And then we have to check the updated file again that which attribute values are changed by comparing their respective data values concerning with index values in table to ones stored in disk. In this case, there are two options: the new updated data is already in disk and is not there. In the first situation, we only substitute the old data's index value with the new one's index value already stored in disk. In the latter case, we need to store it with the newly index value and then do the substituting step. So, we can handle database update effectively in this way.

7. Work flow

Our system is tested with the realistic sales data of one food mart. We present the sample data in the following Table 1 as an example. Assume that product_class_id be integer type and the rest be text, respectively.

Table 1. Product class data

product_class_id	product_subcategory	product_category	product_department	product-family
1	Nuts	Specialty	Produce	Food
2	Shellfish	Seafood	Seafood	Food
3	Fruit	Fruit	Canned Products	Food
4	Spices	Baking Goods	Baking Goods	Food
5	Pasta	Starchy Foods	Starchy	Food
6	Yogurt	Dairy	Dairy	Food
7	Coffee	Dry Goods	Baking Goods	Drink
8	Deli Meats	Meat	Deli	Food
9	Ice Cream	Frozen Desserts	Frozen Foods	Food
10	TV Dinner	Frozen Entrees	Frozen Foods	Food
11	Sauces	Baking Goods	Baking Goods	Food
12	Cooking Oil'	Baking Goods	Baking Goods	Food
13	Sugar	Baking Goods	Baking Goods	Food
14	Pizza	Pizza	Frozen Foods'	Food
15	Hot Dogs	Meat	Deli	Food
16	Crackers	Snack Foods	Snack Foods	Food
17	Dips	Snack Foods	Snack Foods	Food
18	Milk	Dairy	Dairy	Drink
19	Cookies	Snack Foods'	Snack Foods'	Food
20	Hamburger	Meat	Meat	Food

After doing the de-dupe process, the results are formed in two database files. One is De-

duped_Data_Store and another is Reference_Data_Store. Non-duplicated data are stored in De-duped_Data_Store as in Table 2 and the index values to a single copy stored in Deduped_Data_Store are stored in Reference_Data_Store as in Table 3.

7.1. Test Data Analysis

We need to calculate the size of the three files stored as Database Schema, De-duped_Data_Store and Reference_Data_Store. Assume the size of data be as below: integer (2 bytes), float (4 bytes) and char (1 byte) respectively.

In Table I, there are totally 20 records of 5 attribute values. As we can see, among the 100 attribute values, 42 data values are duplicated. There will need to store only distinct 58 data values as shown in the following table.

Table 2. De-duped_Data_Store

H(K)	Key	Index Value	H(K)	Key	Index Value
0	1	1	31	Cooking Oil	32
1	2	2			
2	3	3	32	Sugar	33
3	4	4	33	Pizza	34
4	5	5	34	Hot Dogs	35
5	6	6	35	Crackers	36
6	7	7	36	Dips	37
7	8	8	37	Milk	38
8	9	9	38	Cookies	39
9	10	10	39	Hamburger	40
10	11	11			
11	12	12	40	Specialty	41
12	13	13	41	Seafood	42
13	14	14	42	Fruit	43
14	15	15	43	Baking Goods	44
15	16	16			
16	17	17	44	Starchy Foods	45
17	18	18			
18	19	19	45	Dairy	46
19	20	20	46	Dry Goods	47
20	Nuts	21	47	Meat	48
21	Shellfish	22	48	Frozen Desserts	49
22	Fruit	23			
23	Spices	24	49	Frozen Entrees	50
24	Pasta	25			
25	Yogurt	26	50	Snack Foods	51
26	Coffee	27			
27	Deli Meats	28	51	Produce	52
			52	Canned Products	53
28	Ice Cream	29			
			53	Starchy	54
29	TV Dinner	30	54	Frozen Foods	55
30	Sauces	31	55	Deli	56
			56	Drink	57
			57	Food	58

After de-duplication, Deduped_Data_Store:

Int : (20 items * 2 bytes) = 40 bytes

Char : (275 char * 1 byte) = 275 bytes

Index : (57 items * 1 byte) = 57 bytes

Total := 372 bytes

Reference_Data_Store (all data are stored in text format) is as shown in Table 3.

Index : 100 items* 1 byte= 100 bytes

Table 3. Rereference_Data_Store

product_class_id	product_subcategory	product_category	product_department	product_family
1	21	52	52	58
2	22	42	42	58
3	23	53	53	58
4	24	44	44	58
5	25	54	54	58
6	26	46	46	58
7	27	47	44	57
8	28	48	56	58
9	29	49	55	58
10	30	50	55	58
11	31	44	44	58
12	32	44	44	58
13	33	44	44	58
14	34	34	55	58
15	35	48	56	58
16	36	51	51	58
17	37	51	51	58
18	38	46	46	57
19	39	51	51	58
20	40	48	48	58

Database Schema is kept in the form of the following format as in Figure 2.

```
DB# foodmart
T# product_class
F# product_class_id # int (11)#
F# product_subcategory#varchar(30) #
F# product_category#varchar(30)#
F# product_department# varchar(30)#
F# product_family# varchar(30)#
```

Figure 2. Schema text file format

Database schema size is (179 char * 1 bytes)= 179 bytes.

So, the total required size is (372+100+179) = 651 bytes. The original data size is 977 bytes.

According to the result, we can see that our approach saves about 300 bytes out over 900 bytes even with a little amount of original data. This analysis supports at least 1/3 of the total size. So, our system will reduce the disk space more than 35% with a larger amount of data.

8. Experimental result

We test our system with "Foodmart" dataset (65.3MB) [14]. Our approach needs only about 30.1 MB. So, our system reduces about 55% on original data size. If de-duplication is not applied to that data set, it will need at least 2 times than the space needed in our system. If our approach is tested with a huge amount of data, it will definitely reduce much more than 55%. If we apply with record level de-duplication mechanism, it will be sure that we won't reduce the storage capacity as much as our approach. We can only save nearly 2% in record level, and can't get any saving by relation or table as in the following figures.

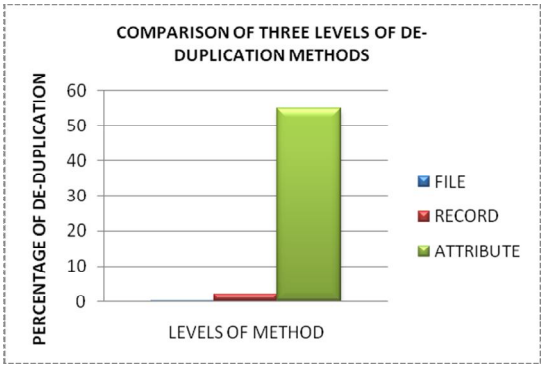

Figure 3. Results of three levels of de-duplication

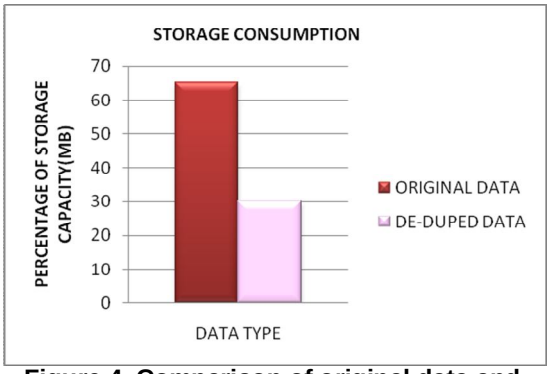

Figure 4. Comparison of original data and de-duped data size(MB)

9. Conclusion

Our backup system is intended to keep relational data for long term retention period and efficient utilization of disk storage for large amount of data value. And also our approached solution recovers the data damaged due to errors. For recovery, our backup system creates the structure of a data file as the original data's schema using it's schema in text file format stored in Database Catalog. And it uses the files stored in Reference_Data_Store for retrieving data values by index values kept in there. So, our system can do the restore process easier and effectively by the steps mentioned above.

Our attribute level backup system reduces not only disk space capacity locally at the client location but also at the remote machine in which only the de-duped data are stored. So, our system is mostly appropriate for the enterprises (sales centre with many branches, banking system) which do their job processing with daily transaction data.

10. References

[1] A. Muthitacharoen, B. Chen, and D. Mazières, "A low-bandwidth network file system," in Proceedings of the eighteenth ACM symposium on Operating systems principles Banff, Alberta, Canada: ACM Press, 2001.

[2] B. Hong, D. Plantenberg, D. D. E. Long, and Sivan-Zimet, "Duplicate data elimination in a san file system," in Proceedings of the 21st IEEE/12th NASA Goddard Conference on Mass Storage Systems and Technologies (MSST), 2004, pp. 301-314.

[3] D. Gomes, A. L. Santos, and M. J. Silva, "Managing duplicates in a web archive," in Proceedings of the 2006 ACM symposium on Applied computing Dijon, France: ACM, 2006, pp. 818-825.

[4] F. Douglis, J. LaVoie, and J. M. Tracey, "Redundancy Elimination Within Large Collections of Files," in USENIX Annual technical Conference, 2004, pp. 59-72.

[5] L. P. Cox, C. D. Murray, and B. D. Noble, "Pastiche: Making backup cheap and easy," in Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI), Boston, MA, 2002, pp. 285–298.

[6] M. O. Rabin, "Fingerprinting by Random Polynomials," in Tech. Rep., TR-15-81, Center for Research in Computing Technology, Harvard University, 1981.

[7] QUINLAN, S., AND DORWARD, S. Venti: A new approach to archival storage. In Proceedings of the FAST 2002 Conference on File and Storage Technologies (Monterey, CA, USA, January 2002), USENIX Association, pp. 89–101.

[8] R. B. Deepak, J. Suresh, and D. Cezary, "Improving duplicate elimination in storage systems," ACM Transactions on Storage vol. 2, 2006, pp. 424-448.

[9] T. Avishay, J. Nikolai, S. Josef, and Z. Erez, Using free web storage for data backup," in Proceedings of the second ACM workshop on 12Storage security and survivability Alexandria, Virginia, USA: ACM, 2006.

[10] W. J. Bolosky, S. Corbin, D. Goebel, and J. R. Douceur, "Single instance storage in Windows 2000," in Proceedings of the 4th USENIX Windows Systems Symposium, Seattle, WA, 2000, pp. 13–24.

[11] www.netapp.com/products/storage-systems/near-linestorage / asis-dedup.html

[12] www.quantum.com/Products/Disk-Based Backup/ DXi3500/Index.aspx.

[13] www.symantec.com/zh/cn/enterprise/products

[14] http://sites.google.com/a/dlpage.phi-integration.com/ pentaho/mondrian/mysql-foodmart-database

[15] M. Lillibridge , K. Eshghi , D. Bhagwat , V. Deolalikar , G. Trezise , and P. Camble , “Sparse Indexing: Large Scale, Inline Deduplication Using Sampling and Locality”, in Proceedings of the 7th USENIX Conference on File and Storage Technologies, 2009.

[16] FORMAN, G., ESHGHI, K., AND CHIOCCHETTI, S. Finding similar files in large document repositories. In Proceeding of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD) (Chicago, IL, USA, August 2005), ACM Press, pp. 394–400.

[17] MUTHITACHAROEN, A., CHEN, B., AND MAZI`ERES, D. A low-bandwidth network file system. In Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP) (Banff, Alberta, Canada, October 2001), ACM Press, pp. 174–187.

[18] YOU, L. L., POLLACK, K. T., AND LONG, D. D. E. Deep Store: An archival storage system architecture. In Proc. of the 21st International Conference on Data Engineering (ICDE ’05) (Tokyo, Japan, April 2005), IEEE, pp. 804–815.

[19] RHEA, S., COX, R., AND PESTEREV, A. Fast, inexpensive content-addressed storage in Foundation. In *Proceedings of the 2008 USENIX Annual Technical Conference* (Boston, Massachusetts, June 2008), pp. 143–156.

[20] DOUGLIS, F., AND IYENGAR, A. Application-specific delta encoding via resemblance detection. In Proceedings of the 2003 USENIX Annual Technical Conference (San Antonio, Texas, June2003), pp. 113–126.