IOSR Journal of Engineering (IOSRJEN) **www.iosrjen.org**
ISSN (e): 2250-3021, ISSN (p): 2278-8719
Vol. 05, Issue 10 (October. 2015), ||V2|| PP 30-34

Differential snap-shot algorithms for data extraction from data sources and updating a centralized data warehouse

Nirav Desai, Pravin Metkewar
Symbiosis Institute of Computer Studies and Research, Pune

Abstract: - An in-depth review of existing techniques for loading a data warehouse using differential snapshot algorithms is presented. Further, certain performance bottlenecks have been identified and improvements to the existing state of the art are suggested.

I. INTRODUCTION

Data warehousing projects involve setting up centralized data and meta-data repositories that store pooled data from various departmental databases of an organization or from various offices spread across geographically separated areas. The steps involved in setting up centralized data repositories are extraction, transformation and loading.

Extraction involves extracting data from varied sources such as Excel files, flat files, SQL databases, NoSQL databases and other legacy systems such as mainframes. This extracted data is stored in a staging area where different steps such as cleansing, validation, accuracy check and data type conversion are applied to ensure that all the data confirms to the same meta data standard and can be merged and processed together.

Transformation involves mapping the schemas of the source files which have been extracted to the destination databases of the data warehouse. Typically, dimensional modeling is carried out in the data warehouse to aid query processing and online analytical processing. Source databases are normalized and destination dimensional models are not. Hence a few steps of transformation are typically necessary for all data warehouses. The transformed data is loaded into the data warehouse relational tables.

This paper concerns the extraction step of the process of data warehousing. The central dimensional databases need to be updated at regular intervals with data from the transactional OLTP data bases. The transactional databases usually store up to 10-20 GB of data and there could be multiple databases from which data needs to be extracted. The extracted data would have to be loaded into the data warehouse. Typically, the write times of dimensional databases are very long and they are mainly optimized for read access and interactive query processing. Thus, the data warehouse might have to be taken offline at regular intervals for batch uploads of data. Such down times might not be desirable for data warehouses that serve customers for self-service BI. Also nowadays, the trend is moving towards real time BI and for this efficient updates of the data warehouses are necessary. This paper reviews 2 different techniques used for efficient updates of central data warehouses from different sources.

This paper is divided into 2 parts: the first one deals with a literature review of existing techniques for extraction and differential updates and the second part deals with suggested improvements. Here we are trying to build a procedure for differential updates of a central data warehouse with data from data sources that are heterogeneous in nature. We are assuming that all the extracted data is available to us in the staging area and is ready for the upload.

We will call the present extracted set of data from the data source as snapshot F2 and the existing set of data in the data warehouse (which needs to be updated) as F1. Each entry of the snapshot F1 is denoted by tuple <ki, bi> where ki is the key and bi the corresponding set of fields. Each entry of the snapshot F2 is denoted by the tuple <ki, bj> where ki is the key and bj the corresponding set of fields that have the present value of the tuple.
The general differential update algorithm proceeds as follows: For each <ki,bj> in F2 find a tuple with the matching key <ki, bi> in F1.

1. If we find a tuple <ki, bi> in F1 for a tuple <ki, bj> in F2, the tuple in F1 needs to be updated. The query will be: UPDATE <ki, bj> in F1
2. If there is no tuple in F1 corresponding to tuple <ki,bj> in F2, we have to insert a tuple in F1. The query will be: INSERT <ki, bj> in F1

3. If there is no tuple in F2 that has the key <ki, bj> corresponding to a tuple in F1, the tuple in F1 will have to be deleted. The query will be: DELETE <ki, bj> in F1

For large datasets, a limited working set will be loaded into working memory and this algorithm will be executed. If the corresponding entries are not present in either of the 2 working sets, there might be INSERT-DELETE or DELETE-INSERT queries on the same tuple. An INSERT-DELETE query could lead to consistency problems. When the INSERT is processed, it might not affect the data warehouse since the entry is already present. However, the DELETE would be processed and this would lead to consistency problems. A DELETE-INSERT pair is acceptable and does not create any consistency problems.

The same record could be updated a multiple number of times if we are extracting data from a transactional database. In this case, it would be advisable to aggregate all updates to one key and do a single update for all the extracted values.

The process of matching the keys in F1 and F2 is one of outer join of F1 with F2 on key ki. The records which are returned as a match are to be updated and the records of F1 which are not returned in the join are to be deleted. The record of F2 which did not join to any values in F1 are to be inserted into F1. The outer join can be carried out as hash-join on randomized sets or it can be carried out using a merge-sort algorithm which are standard techniques for carrying out outer joins in databases. Hash joins work better on randomized data sets and merge-sorts work better on sorted data sets. This has been explained in reference [1].

A windowing approach to perform the differential update is further presented in reference [1],which has a better performance than the outer join techniques. The outline of the windowing algorithm is as follows:

1. Extract data from F1 and store it in input buffer A1.
2. Extract data from F2 and store it in input buffer A2.
3. Compare entries of A1 against A2 and for all matches, update the entry in A2 as matched and the entry of A1 as free.
4. Compare entries of A1 against all entries of ageing A2 buffer (description next) and update the corresponding entries of A2 buffer. The matched A1 entries are marked as free.
5. Compare entries of ageing A1 buffer with all entries of A2 and mark the matched entries of A2 as matched and matched entries of A1 as free.
6. Compare entries of ageing A1 buffer with all entries of ageing A2 and update the matched entries of A2 as matched and matched entries of A1 as free.
7. The left over entries of A1 buffer are inserted into the ageing A1 buffer which is a first in first out queue. The entries which are popped out of ageing A1 buffer are to be deleted from the data warehouse.
8. The left over entries of A2 buffer are inserted into the ageing A2 buffer which is also a first in first out queue. The entries which are popped out of the queue are to be inserted into the data warehouse.
9. The buffers A1 and A2 are filled with next set of entries from F1 and F2 respectively and the above process is repeated.

The search in A1 is a nested loop search and the A1 ageing buffer has hashed tables with hashing based search from A1 to A2. The ageing buffer is much larger than A1 and hence hashing is preferable here.

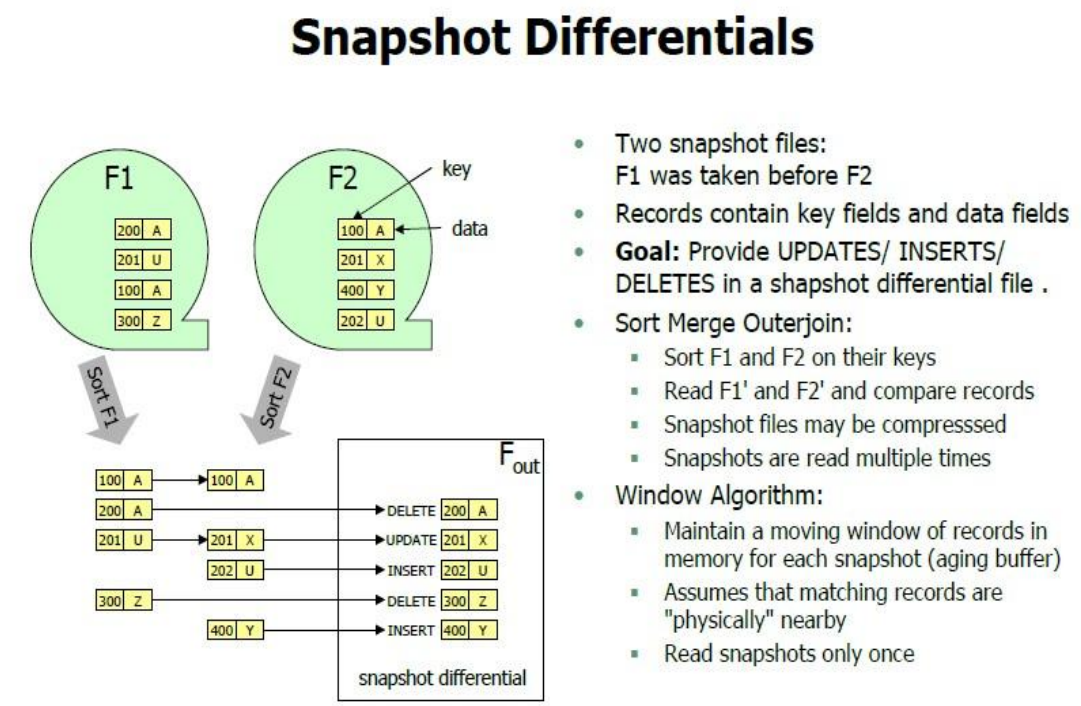
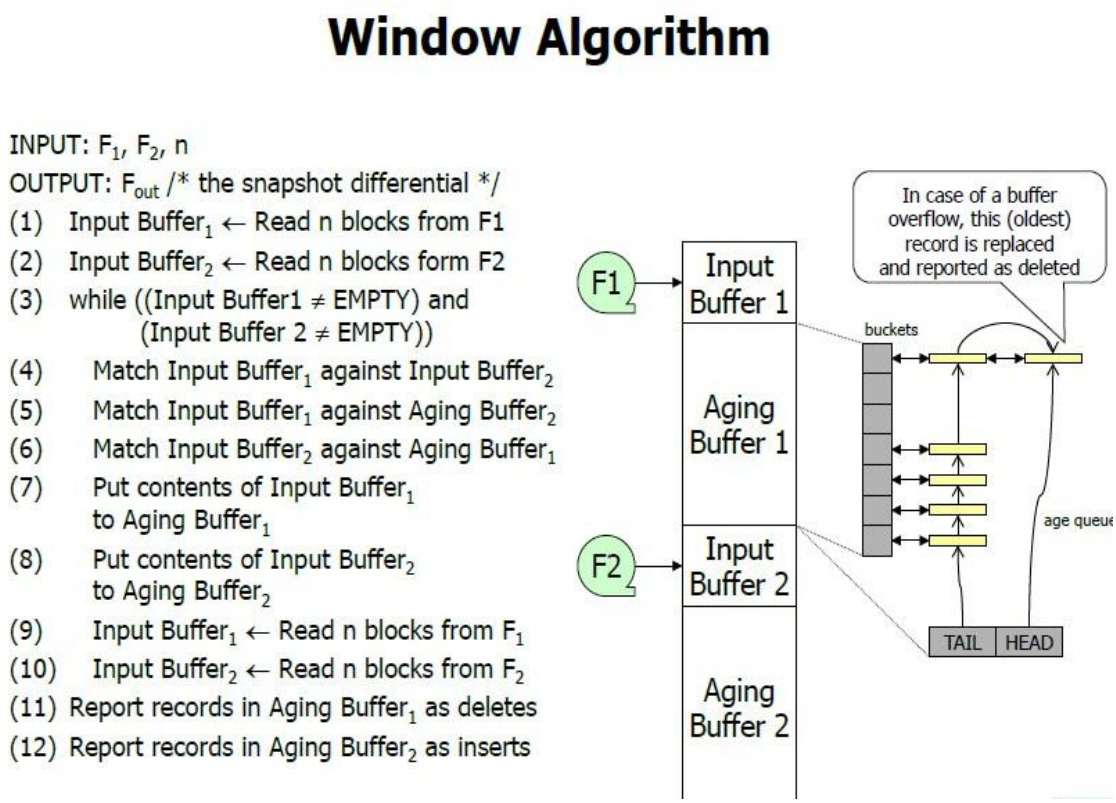


Figure 1: Graphical illustration of the window search. Reference: Data warehousing course from University of Stuttgart, Germany taught by Bernhard Mitschang.



FiguFigure 2: Visual depiction of the window algorithm [4].

The window algorithm described here works on the assumption that fields with matching keys are at close locations in F1 and F2. If this is not the case, the algorithm leads to redundant delete-insert queries which do not lead to consistency problems.

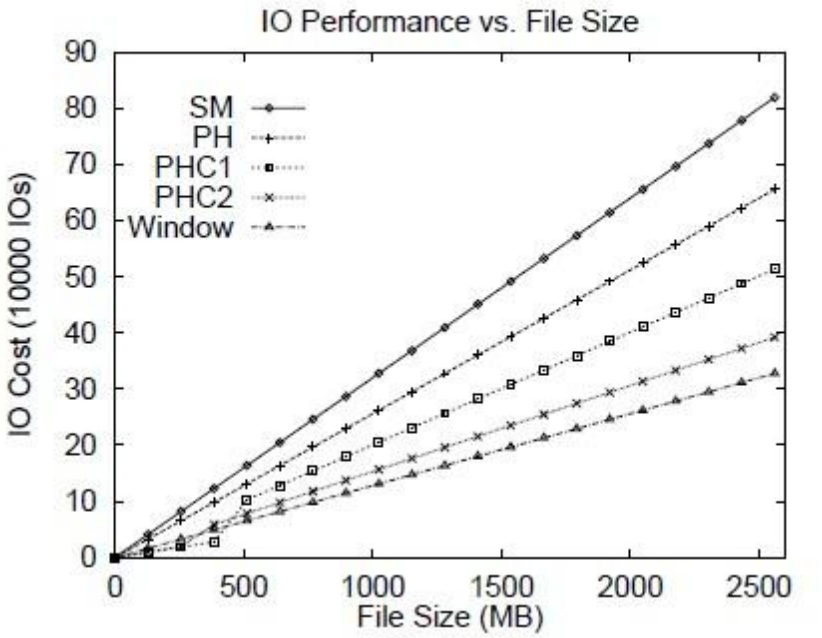


Figure 3: Performance of the windowed (Window) algorithm against sort-merge (SM), partitioned hash outer join (PH), compressive partitioned hash outer join (PHC1, PHC2). It can be seen that window algorithm is optimal in terms of number of input and output operations. Figure from reference [1].

II. PERFORMANCE BOTTLENECKS

The window algorithm does not function well when the records with matching keys are not at close locations in F1 and F2. This would happen if the records of F2 would be sorted in reverse against the records of F1. This would be worst case performance of the window algorithm. Also the buffer size of the ageing buffers would have to be optimized based on the size of the available memory. If the ageing buffers are sized too small, there would too many redundant delete-inserts that would slow down the performance and a buffer which is too large would lead to high search times. An optimal buffer size for the ageing buffer would be a function of the size of the memory and block of data we are analyzing.

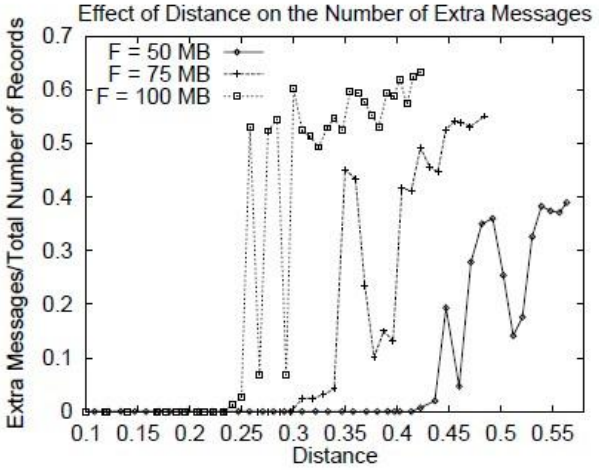


Figure 4: Impact of distance between the buffers on the extra number of messages. Distance is measured as difference in position of the key in two buffers.

III. SUGGESTED

It would be useful to carry out a simulation of the windowed algorithm using PHP to extract the data and measure the actual performance of the algorithm against the published results. Further, it would be useful to see if sorting the databases on primary keys before extraction would improve the performance of the window

algorithm. It would also be useful to try optimizing the size of ageing and primary buffers against the total memory size and block size fetched to find the most effective buffer size.

REFERENCES

1. **Efficient Snapshot Differential Algorithms for Data Warehousing**
Wilburt Labio and Hector Garcia-Molina. 1996. Efficient Snapshot Differential Algorithms for Data Warehousing. In *Proceedings of the 22nd International Conference on Very Large Data Bases* (VLDB '96), T. M. Vijayaraman, Alejandro P. Buchmann, C. Mohan, and Nandlal L. Sarda (Eds.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 63-74.

2. **Research problems in data warehousing**
Author: Jennifer Widom, Department of Computer Science, Stanford University, Stanford, CA
Proceedings of CIKM '95 Proceedings of the fourth international conference on Information and knowledge management Pages 25-30

3. **The Stanford data warehousing project**
Author: Joachim Hammer, Hector Garcia-Molina, Jennifer Widom, Wilburt Labio, and Yue Zhuge
Computer Science Department, Stanford University, Stanford, CA 94305

4. Course slides: Data warehousing course from University of Stuttgart, Germny
Author: Bernhard Mitschang. University of Stuttgart