2010 24th IEEE International Conference on Advanced Information Networking and Applications

Towards Near Real-Time Data Warehousing

Li Chen and Wenny Rahayu
Department of Computer Science and Computer Engineering
Latrobe University
Bundoora, Victoria 3086, Australia

David Taniar
Clayton School of Information Technology
Monash University
Clayton,Victoria 3800, Australia

***Abstract*—A data warehouse is built as a layer on top of existing operational database systems. Once built, it has to be regularly updated (refreshed). Currently, most data warehouse approaches employ static refresh mechanisms whereby updates are based on a static timestamp, eg. once every day/week/quarter only. Whilst for some systems this might be adequate, others require a more rigorous approach ensuring that analysis is always 'up-to-date'. Static time interval for refreshing data warehouse is not adequate enough for systems with high update frequency. A real-time data warehouse incorporates operational data changes in real time. However, sometimes, it is often unnecessary or even inefficient to immediately refresh and send updates from the operational database into a data warehouse. In this paper, we propose a near real-time refresh mechanism that takes into consideration a number of measures: (i) Impact from record, (ii) Number of records affected, and (iii) Frequency Request Measure. The combination of these measures can accurately identify when the data warehouse needs to be strictly real-time, or near real-time (ie. right-time). Our experimentation shows that the proposed approach offers a significant benefit in terms of refresh operation cost in comparison to real-time warehousing, while at the same time still maintaining a high freshness level of the data warehouse.**

I. INTRODUCTION

The concept of the data warehouse was first introduced in [1]. A data warehouse (DW) is used to store a massive amount of data integrated from multiple, independent data sources, which can reflect the reality of the real world, and its tools can discover trends or potential directions of development from data. On the one hand, the dynamic business environment of many organizations requires high freshness of requested data. This process includes extracting the new transaction data from the database, transforming, aggregating and then updating it into DW. On the other hand, users usually expect short response times for their queries. Meeting these fundamental requirements is challenging to DW due to the high loads and update complexity of DW refresh mechanisms. In general, DW refresh mechanisms have slowly shifted from traditional DW to real-time DW mechanisms, as outlined in the following three categories [2]:

- The first category is a traditional DW refresh mechanism which consists of **traditional batch-based** updates. The whole DW is rebuilt from scratch by running the schema, repopulating all the data from the current operational database with a timestamp. Based on the schedule of the business at that time, DW can be refreshed monthly, quarterly, even yearly. However, the full refresh mechanism can not possibly meet users' requirements when the size of the DW is greatly increasing, the data is required to be refreshed more frequently and the batch windows for data acquisition need to be smaller and smaller.
- The second category is **incremental batch refresh** based on a timestamp, which means that instead of redoing the entire old and new data, it recalculates only the data updated since the last refresh, and it will push only these new data into the DW. This is much more efficient. However, when a DW deals not only with strategic decision support but also tactical decision support, timestamp refresh may potentially cause a considerable delay between the information in the DW and the reality in data sources.
- The third is the **continuous feed refresh** mechanism of a real-time DW. This is the most ideal refresh mechanism for an advanced DW. As the real-time DW is always refreshed [3], people can always get up-to-date information. This mechanism seems to be very tempting for the business world; however, a significant burden is imposed on the operational database side. One transaction may take a long time to convert and load data into the DW, which will seriously affect the daily transaction performance. Second, it may be unnecessary to refresh immediately after each change in operational database.

Therefore, our paper concentrates on investigating a combination of the second and third approaches, an **incremental continuous refresh** mechanism. This mechanism only aggregates the new insertions and updates from the operational database, when the system accurately identifies that the DW needs to be continuously refreshed or just-in-time.

II. RELATED WORK

Recent works reflect the increased interest in the utilization of real-time DW, and research has mostly dealt with the problem of refreshing the DW efficiently and accurately to meet the requirements of the real world [4]. Research topics include update of DW, and dimensions efficiently [3][5][6][7]. In [7], the author describes a real-time loading methodology by duplicating the schema of DW to store the temporary up-to-date information without defining any type of index, primary key, or constraints. This approach enables DW to obtain continuous data integration with minimum OLAP response time. All the continuous integration data will be merged with

1550-445X/10 $26.00 © 2010 IEEE
DOI 10.1109/AINA.2010.54

original DW when the system performance is not acceptable, then all index and other data structures will be rebuilt.

In the business world, many companies adopt data marts in the management of a DW. In order to increase the system performance, numerous summary tables represented as *materialized views (MV)* are stored in the data mart to answer queries. Several works on maintaining MV efficiently when contents change [8], or contents and schema both change in the data source [9][10], have been widely studied. In [11], the authors addressed content-based filtering for efficient online MV maintenance. The paper describes a way to maintain MV based on the content to detect the non-relative changes from the operational database side to MV. This will catch the operations that do not affect the real-time DW, and save all the unnecessary expensive checking by considering only the 'where' clause condition in MV's definition.

A temporary DW is able to provide up-to-date data for daily tactical management. A temporary DW is a temporary view which holds temporary aggregated data before the data is refreshed in the DW. Based on the timestamp on the view, when the query relates to the old data in the DW, the system will collect the data from DW, and if the query relates to the up-to-date data, the system will join temporary DW with the old data from DW. A related challenge is supporting the large cost of temporal aggregation operations by joining both DWs [12].

All the above papers focus on the issue of maintaining a DW in an efficient way, and are not concerned with the fundamental aspects of the refresh time interval of DW, which is the main issue that we are going to focus on in this paper. In [4], the authors argue that not all transactional data need to be immediately dealt with even in a real-time decision making requirement. In the same DW, we should allow different time priorities, such as urgent, which should be in real-time, or just be in time based on the business characteristics. For example, in a company, the sales department requires real-time data acquisition and consolidation, but the finance department, may require only monthly data loading for payment of employees. DW data freshness should always be driven by the business requirements, not the technology itself [2]. It is a challenging and critical goal to enable DW to provide multiple levels of freshness time intervals.

Early work in [13], focuses on the refresh mechanism for the XML DW. A waiting queue is built to discretely refresh XML DW, which consists of three variables to define the right time to refresh XML DW. This method provides a very good basis for a continuous DW update mechanism. However, the methodology of the experiment is built on a prototype system. No real data is applied and the comparison with other types of DWs is not sufficient.

To the best of our knowledge, none of the existing approaches have a specific method to define the criteria to measure data changes in order to maintain just-in-time data warehousing. In this paper, our main focus is to build a near real-time DW mechanism, which can reflect the real world requirements by making the interval of the timestamp 'dynamic' or flexible by depending on factors such as impact from one update, the number of records affected, and the frequency of requests.

III. NEAR REAL-TIME REFRESH MECHANISM

Our proposed near real-time refresh mechanism consists of three aspects to ascertain the level of importance of the updates: (A) Impact from record Measure, (B) Number of records affected Measure, and (C) Frequency of Request Measure. To illustrate our methodology, we will use a simple operational database (see Figure 1).

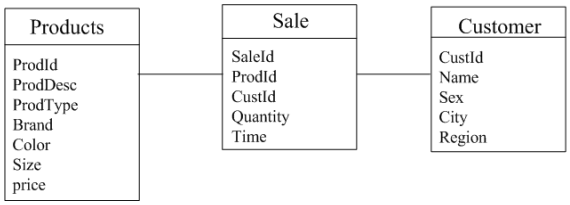


Fig. 1. A sample of operational database

A. Impact from one update

Impact from one update implies the impact of a particular update aggregated from an operational database. The aim is to determine the impact of the update on the data in the DW. If an update (e.g. the sale information of products in a particular color and season) has a significant impact, analysis of this information is very valuable in decision making (e.g. the stock of products for next week/month). And at the opposite side, if the update has a negligible impact on the decision making, it is not necessary to push the data into DW, thereby wasting the process cost.

It is very likely that one update from one transaction in the operational database will not have a very significant impact on the DW, but one update accumulated from small transactions will generate a significant effect on the DW. So we will build a queue to store the negligible updates. These updates will be continuously aggregated from the later transactions, until any one update has a significant impact on the DW, the DW will be refreshed from the queue, and all the records are sent to the DW.

Consequently, calculating the impact from an update and defining the criteria to measure whether an update is 'significant' or 'negligible' are difficult tasks since different types of businesses have different requirements regarding the sensitivity of data changes. Here we offer two possible solutions to the problem: (i) if the user has a specific focus on the analysis, they can define a specific data and percentage, and if the aggregate data reaches that percentage or higher compared with the data they pre-defined, the update is significant. (ii) the DW itself could calculate, after every update, to compare the new aggregation data with the old data in the DW, and if the new data is twice that of the old data, or the user defines this difference from DW, the updates are significant; otherwise it is negligible.

TABLE I
UPDATE (QUEUE)

Sex	Season	Time	max(Price)	sum(Quantity)
F	Summer	02-01-08	50	20
F	Autumn	15-04-08	20	150
M	Autumn	13-04-08	30	100
M	Winter	13-07-08	20	130
M	Spring	10-10-08	40	12
F	Spring	12-10-08	600	4500

However, the second solution may give rise to two problems. First of all, each update is compared with the corresponding aggregated data from the DW. But usually the size of DW is large, therefore a search for the exact data from the DW will be expensive and sometimes it is unnecessary. So we can have a table that records the average aggregated data from the DW, and compare the new data with this average data to calculate the difference. Every time the DW is refreshed, this table will be updated as well. Secondly, in the DW, there may be more than one aggregate data. Therefore, one way we can average all aggregation data, or assign a weight to each aggregation data to calculate the impact for users who may have different focuses on the aggregation data:

I : one update impact

s_i: the i_{th} average aggregate data summarized from the DW

n : the amount of aggregation attributes

Agr_i: the i_{th} aggregation data

$Agr_i\%$: the wight of the i_{th} aggregation attribute

$$I = \frac{\sum_{i=1}^{n} \mid \frac{agr_i}{s_i} \mid}{n} \times 100\% \quad (1)$$

or

$$I = \sum_{i=1}^{n} (\mid \frac{Agr_i}{s_i} \mid \times Agr_i\%) \quad (2)$$

Here we give a very simple example of the consumer trend for each gender at each season. When there is a new update retrieved, we calculate the update impact. If it is not significant enough, it will be kept in the queue, until one update which has a significant impact arrives, and then all updates will be updated into the DW. Here we use table I to represent the queue. From Table I, all updates before *sex* = 'F', *season* = 'Spring' are all kept, so when the new update *sex* = 'F', *season* = 'Spring' arrives, we can see, 'F' in 'spring' purchases 4500 items. And then we compare this figure with the average data (3780 items per season), and it is higher than the average data. At the same time, in all items 'F' purchased, the highest price they paid is 600 dollars, which is two times of the average max price in the DW 300 dollars. This single update has an impact on the DW. To calculate the impact of an update, we compare the difference from the summary of the DW see Table III. This summary table can be the summary of the DW for last year or recent one month or even last ten years.

If we use equation (1),

$$I = \frac{\mid \frac{600}{300} \mid + \mid \frac{4500}{3780} \mid}{2} \approx 1.60$$

TABLE II
SALESFACT

Sex	Season	max(Price)	sum(Quantity)
F	Spring	200	2000
F	Summer	250	4500
F	Autumn	70	3900
F	Winter	90	5000
M	Spring	60	2800
M	Summer	80	4580
M	Autumn	90	2760
M	Winter	300	4700

TABLE III
SUMMARY FROM SALESFACT

Max(maxPrice)	300
Avg(totalQuantity)	3780
rowCount	8
totalQuantity	30240

However, some users might not calculate the impact equally for each aggregation data; therefore, we use the second equation. Max% and Sum% are the weights for maxPrice and totalQuantity; these weights are flexible. If some users do not want to consider max price, then they can assign Max% =0%, and Sum% = 100%, so the sum aggregation will be the only indicator affecting the impact.

$$I = \mid \frac{600}{300} \mid \times 0\% + \mid \frac{4500}{3780} \mid \times 100\% = 1.19$$

The example above shows how to calculate the impact from the update. How to convert the operations from operational database to the aggregated update is discussed later.

B. Number of Records Affected

The number of records affected indicates how much data have not been updated or are no longer correct. We use **RAM (Records affected measure)** to represent the impact caused by this. We can see the impact from one update measure is used to detect when a single update has a significant influence on the DW affecting decision making. But it is possible that many new data transactions are made in the operational database and each single update of these transactions does not have a significant influence on our decision analysis. All these data will be stored in the waiting queue. If there is no transaction taking place in the operational database or we still continue to have negligible updates added into queue, the data of the DW will remain incorrect for a long time. This is not desirable for our requirement of near real-time DW.

For example, if there are 50 records in the DW, and there are 30 aggregated updates in the waiting queue, then there are around 30 out of 50 pieces of information that are outdated in the DW. Even though some data changes may be quite small, certain businesses require that their DW is constantly maintained up-to-date. This indicator allows an overview of changes made to the DW and allows us to ascertain how much data correction has not yet been made to the DW.

AT.rowsCount: the number of records kept in the queue

AV.rowsCount: the number of records in the DW

TABLE IV
QUEUE(UPDATE)

Sex	Season	Time	max(Price)	sum(Quantity)
F	Spring	12-10-08	50	1
F	Summer	23-12-08	200	1
F	Autumn	12-04-08	70	1
F	Winter	23-07-08	50	1
M	Summer	30-12-08	40	1
M	Autumn	24-04-08	30	1
M	Winter	15-07-08	80	1

$$RAM = \frac{AT.rowsCount}{AV.rowsCount} \tag{3}$$

From table IV, we can see that in the queue, in each dimension, the sum(Quantity) is just 1, so each update has a very small impact on salesDW, but we can see that almost all the data of the salesDW has changed, $\frac{7}{8}$ of the data is no longer correct data. This is not desirable for the users who want the DW to remain up-to-date.

It is noteworthy that this figure is dynamic, changing with the size of DW. The size of DW is increasing, so the bigger the size of the DW, the smaller is the influence of the amount of refreshed data from the queue. For example, at the beginning, if there are 8 rows in the DW, one update takes 1 out of 8, but after a certain period, the DW may have 100 rows, so one update takes only 1 out of 100, so the effect is less significant.

C. Frequency Request Measure

Frequency Request Measure (FRM) is the frequency of DW being requested in the warehousing reports. The previous two measures do not reflect the users' feedback about the DW usage; it is just about the data itself. Conversely, FRM considers the usage of the data, but not the data itself. If FRM is low, it implies the DW is not usually used as an analysis tool to make decisions, so maintaining a constantly refreshed DW is useless and wasteful in terms of cost. So when FRM is high, we need the DW to keep up-to-date more, and when FRM is low, we might not need to refresh the DW all the time.

However, we should not consider only the influence of FRM alone, because it may produce a vicious circle of DW refresh. If FRM is low, the DW would be refreshed even later, and the DW will not be up-to-date longer, the users would access the DW less since they would know the DW would not have been refreshed yet, then the FRM would be even lower. But if we combine another two aspects, we will achieve an appropriate updating adjustment interval.

FRM works as follows: At the beginning, users should have some idea about how frequently they might access the DW or just how many times they batch DW within a fixed time. For example, they may assume that in one month, they batch DW 4 times; we call it **Planfrequency**. Then the DW maintains a file that stores the usage frequency of each DW and appears in the warehousing reports, and that is **actualfrequency**. Therefore,

$$FRM = \frac{actualFrequency}{PlanFrequency} \tag{4}$$

we can see that this measure can make the system self-adjusting, since the actualfrequency is a dynamic figure. If in one month, the usage of DW is actually more than what was expected, FRM will increase, which will cause the DW to be refreshed more frequently. And actualfrequency will be reset to 0 to record how many times DW has actually been requested in a new month. If there are not many requests to DW, the following month FRM will slow down the frequency of refreshing DW, and save the unnecessary cost of an extra refresh process since the DW has not been used that much.

D. Summary

Based on the descriptions of the three measures above, each time the impact from one update is high, the number of records that needs to be refreshed is low. If we consider only one measure to determine when to refresh DW, it will produce different consequences for the DW.

1) Only consider the impact from one update: Each time the impact from one update is high, DW is always refreshed, but if the usage of DW is quite low, update becomes unnecessary and it will cost a lot. Or each time the update is low, numerous of updates will be stored in a waiting queue, and the information of the DW may be out of date for a long time.

2) Only take the number of records affected into consideration: Sometimes even one or a few updates can have a serious effect on the decision making, but because this affects only very few records in the DW, the system may ignore them, which causes important information to be held up.

3) Only consider the frequency request on DW: As we have already discussed, it will produce a vicious circle.

Therefore, by combining all three, each indicator will cause the others to adjust, thereby an appropriate call will be made to refresh the DW without incurring any unnecessary cost. So here we assign each measure a weight to meet the necessary requirements. **Agr%**, **RAM%** and **FRM%** respectively represent each indicator I, RAM and FRM's weight. The user can adjust these weights according to preference. For example, in the sales department, decision making may be based on analysis of data which is more sensitive to daily transactions and therefore they are more concerned with the impact of the updates than with any other indicator. The HR department might instead be more concerned with request frequency.

IV. NEAR REAL-TIME DW MODEL

The DW contains a large amount of information aggregated from diverse and independent data sources. One of the major challenges facing a DW is to minimize the query response time during the maintenance of DW. Materialized views (MVs) are used to greatly improve the performance for the query in DW. In order to be able to manage the near real-time DW, a DW with materialized view and summary tables of the MV, a model of Delta view with auxiliary table and an operational database model are applied, (see example in Figure 3). Some definitions will be introduced in each model. There are three main processes (see Figure 2). (1) is about the maintenance of an auxiliary table (AT) when there are changes in the

operational database model. (2) is each time after we maintain the AT, we calculate the delta view (ΔV). (3) is when the ΔV meets the threshold, we load the data from AT into MV.

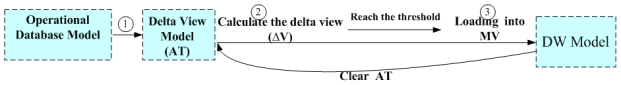


Fig. 2. The three models for near real-time DW

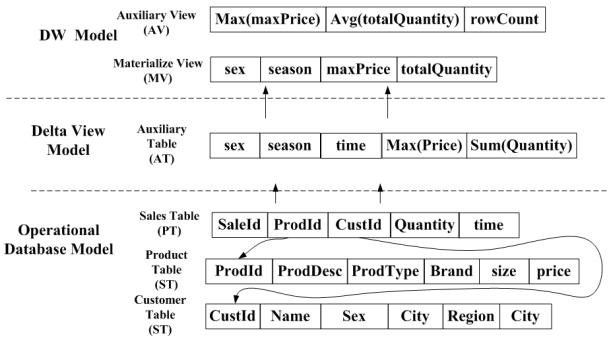


Fig. 3. Example of the three models

A. Operational Database Model

In the *operational database model*, we need to identify the primary table and the secondary tables. Once these are identified, we can update the Delta View Model when there are changes in the operational database model.

DEFINITION 1. Primary table (PT) is defined as a table in the operational data base, whose change (insert/update/delete) always has a direct influence on the fact table in the DW.

PT contains the main transaction items that the Fact table considers as the fact. For example sales table will be the PT for Sales Fact, login table will be the PT for Login Fact.

DEFINITION 2. Secondary table (ST) is defined as the table in the operational data base, whose change (insert/update/delete) may have an indirect influence on the fact table in the DW. In any single DW domain, there will be one PT and a number of STs around it.

So in the operational database model, ST will be used to classify various changes in the operational database, and calculate the aggregating data transfer into the Delta view model. There are two types of changes in the data source: one is data content changes, and in a dynamic environment the schema of the data source might change too. So, depending on the different type of changes, we propose different methods to deal with each of them. This is the first step in the process mentioned in Figure 2.

B. DW Model

DW model contains materialized views (MVs) and summary tables of MVs called Auxiliary Views (AVs). MVs in DW model will store all the historical data and current data from the operational database, and AV is used to keep the general information about each aggregation column in MV, which is used for comparing with the changes in operational database and the number of rows of MV. DW Model manages the normal inquiries to DW, and at the same time, it will track the general data information about each aggregate column in each MV. Because this paper focuses on the maintenance of DW, we assume that DW has already been built.

TABLE V
SALESMV

Season	Color	max(Price)	sum(Quantity)
Spring	Yellow	100	3000
Spring	White	80	5000
Summer	Red	150	4500
Autumn	Yellow	90	2000

TABLE VI
SALESAT

Season	Color	max(Price)	sum(Quantity)	Time
Spring	Yellow	90	3000	12-10-08
Autumn	Yellow	90	2000	22-04-08

DEFINITION 3. Auxiliary view (AV) is defined as a table that stores the data of each aggregates column in the MV. It is a summary table for each MV, MV.rowCount represents the number of rows in MV. Each MV has a corresponding AV, and this is how we store the data for each corresponding column:

$$AVMax[atr] = Max(MVMax[atr])$$
$$AVMin[atr] = Min(MVMin[atr])$$
$$AVSum[atr] = \frac{Sum(MVSum[atr])}{MV.rowCount}$$
$$AVCount[atr] = \frac{Sum(MVCount[atr])}{MV.rowCount}$$

C. Delta View Model

Delta View Model is used for aggregating data from the Operational Database Model and finding the right time to upload data into the DW model. It contains **auxiliary tables (ATs)** and **delta views** (ΔV). Each MV has a corresponding AT in the Delta View Model. AT is used to store the temporary aggregating data from operational database model and prepares the data before uploading into the MV Model when the right time trigger is fired. ΔV is for deciding the right time to fire the trigger in order to load the data into MV. Each time we update the MV, the data in AT must be cleaned.

DEFINITION 4. Delta View (ΔV) ΔV is used to define the right time for loading data from AT to MV. Each AT requires one ΔV, in the paper, we let

$$\Delta V = I \times Agr\% + RAM \times RAM\% + FRM \times FRM\% \quad (5)$$

Agr%, RAM% and FRM% can be pre-defined by the system or adjusted any time by the user.

Here is a small example about how to calculate the ΔV. We assume that there are already some data in table V and table VI, the *planFrequency* is 4, *actualFrequency* is 5, *Agr%* is 40%, *RAM%* is 30% and *FRM%* is 30%.

$$FRM = \frac{ActualFrequency}{PlanFrequency} = \frac{5}{4} = 1.25$$

TABLE VII
SALESAV

Max(price)	150
avg(totalQuantity)	3625
sum(Quantity)	14500
rowCount	4

$RAM = \frac{saleAT.rowCount}{saleAV.rowCount} = \frac{2}{4} = 0.5$
$\Delta V = I \times Agr\% + IDM \times IDM\% + FRM \times FRM\%$
$= 0.39 \times 40\% + 1.25 \times 30\% + 0.5 \times 30\%$
$= 0.156 + 0.375 + 0.15$
$= 0.681$

From the above example, the weights are stored in the Weights table.

DEFINITION 5. Right Time Trigger (RTT) is an event trigger which will be fired when the $\Delta V >= 1$. After RTT is fired, data in AT will be loaded into MV. We assume we already know the schema of MV, so based on the schema, we can define the schema of the Auxiliary Table (AT).

Every time the Delta View Model is updated, ΔV is calculated to find the right time to refresh the DW. Here we will discuss the third step in our process depicted in Figure 2, Loading into DW Model. When the ΔV reaches the threshold, we will load the data from Delta View Model into DW Model. Since in Delta View Model, all the data have already been aggregated, we do not need to have extra calculation. For each record in the AT, if we can find the corresponding dimensions in the MV, we can update it in the MV; otherwise we just insert a new record into the MV. Moreover, when we load data into the MV, we will update the AV as well. After we refresh the MV, the last operation is to clear the data in the AT, and the actual frequency will be recorded. Every time we load new data into the MV, we can put a timestamp on the MV and have a backup copy of the old MV, in case people receive a different analysis because they retrieve data at a different timestamp. It is like building a sequence of snapshot of the old MVs, which is indexed by timestamp. For example, if one report result based on the MV was made in September, then the MV is refreshed in October. If we make the same report in October, the result might be different for the MV has already been refreshed. But it does not mean the previous report is not correct. Our system should be able to retrieve the same result from the old MVs if we provide the timestamp (September).

V. EXPERIMENTS AND RESULTS

In our experimental study, we compared the performances of our proposed update mechanism against the traditional DW and real-time DW mechanisms. In the experiment we used TPC-H Benchmark [14], and the size of DW is varied from 50MB to 1 GB.

A. Cost comparison among different DWs

We compare the cost of traditional DW, real-time DW and our near real-time DW by counting the number of times that updating occurs in the DW when there are transactions in the operational database. Because the traditional DW refreshing process is typically updated in a discrete manner which is based on a pre-defined timestamp (eg. daily, weekly, monthly), we assume the experiment data will be updated based on a certain timestamp for traditional DW. For real-time DW, each insertion will directly incur an update to the DW.

We set Agr% = 60%, RAM% = 50% and FRM% = 30%. From figure 4, we can see the costs for the three models are significantly different. In figure 4, at the end of each timestamp, the DW is refreshed only at once, no matter how many transactions we have during that time interval.

However, from figure 5, in timestamp 1, there are 3 important updates; in timestamp 2, there are 10 important updates; in timestamp 3, 8 updates; and in timestamp 4, there are 2 updates. These important updates have to wait until the end of the time interval in order to be refreshed, which causes a considerable delay between the information in DW and the reality in data sources. On the other hand, if each time that there is one transaction on the database side, the DW is refreshed immediately, from the experiment result, we can see it is not always necessary, especially in the timestamp 1 and timestamp 4, where there are only 3 updates out of 10, 2 out of 20 which need to be immediately updated into DW. Other updates are not very strictly required to be in real-time. The experiment performed in 1 GB DW had a similar pattern. See Figure 6 and Figure 7 .

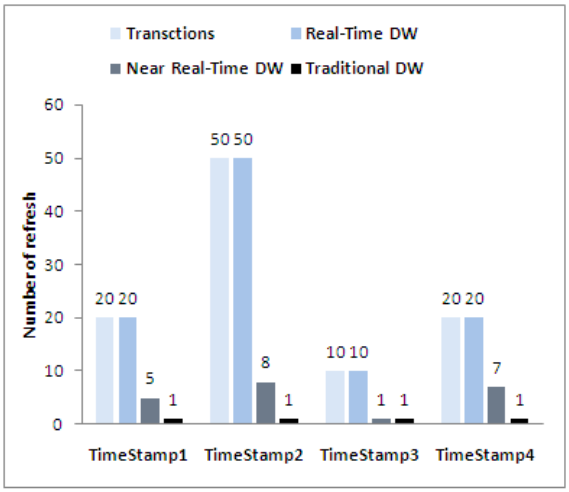


Fig. 6. Comparison among 1 GB DWs

B. The effect of weights

This experiment is used to demonstrate the difference when there are changes to the three aspects of our experiment. We insert 100 transactions based on exactly the same environment with 1 GB DW. We record the number of refreshes of near real-time DW by changing only one of the measures. From Figure 8, we can see that each measure has an effect on the frequency of refreshes of DW. When we change the update impact measure from 0.5 to 2, the number of refreshes of DW increases from 0 to 12. So in the same transactions, the

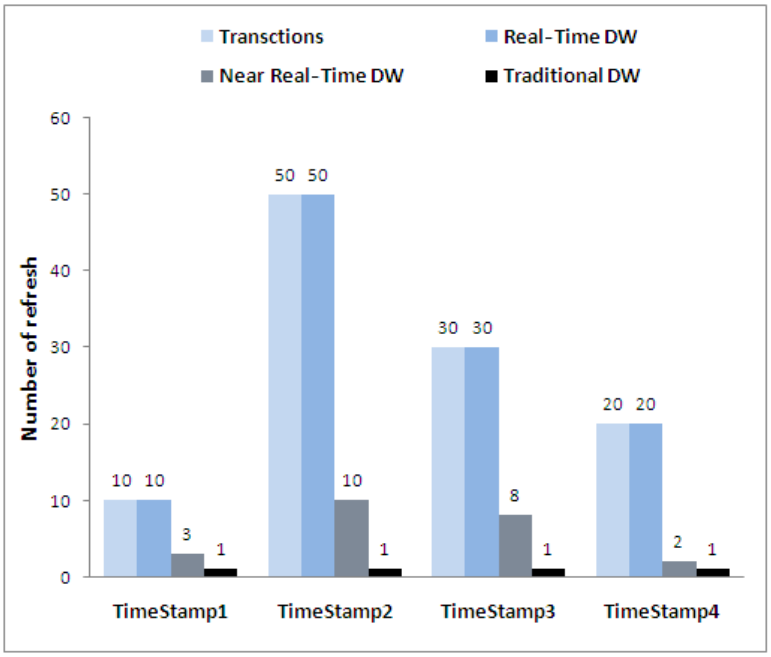


Fig. 4. Comparison among 50 MB DWs

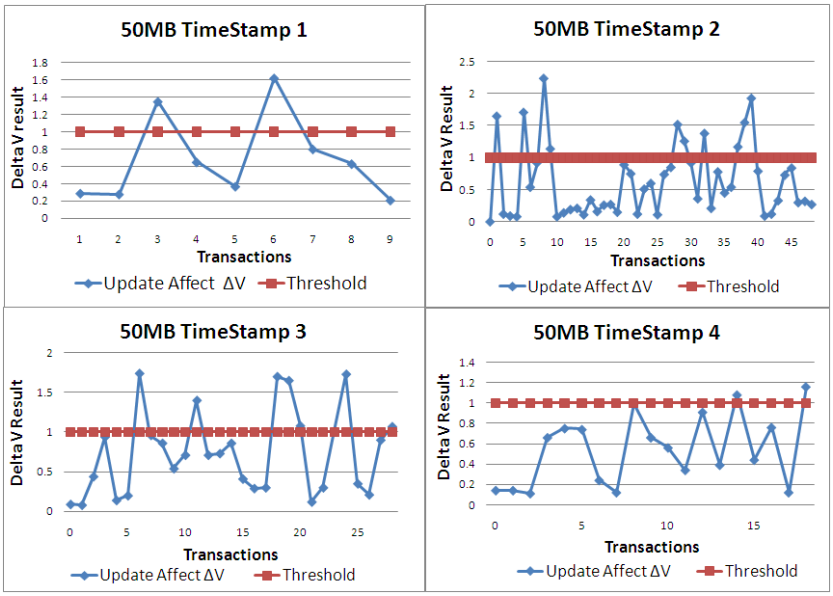


Fig. 5. Number of times refreshed in 50 MB near real-time DW

frequency of refreshing will be totally different due to the different value of measures. From the frequency aspect, by observing the usage of DW, we can see the system can be self-adjusted. If DW is more frequently required, DW will be refreshed more frequently. Moreover, we also can see that one single update measure will affect more than other measures in the frequency of refreshing DW, and the number of records measure has less impact, especially with large DW.

C. Query difference with real-time DW

We use query 11 from TPC-H Benchmark [14] to query our near real-time DW just before we update it. This query result is compared with the result from real-time DW. We run the query 480 times with different parameters, and there are only 20 query results which have different answers. From Figure 9, we can see the query result difference is less than 0.1% with our near real-time DW, the information delayed has a minor

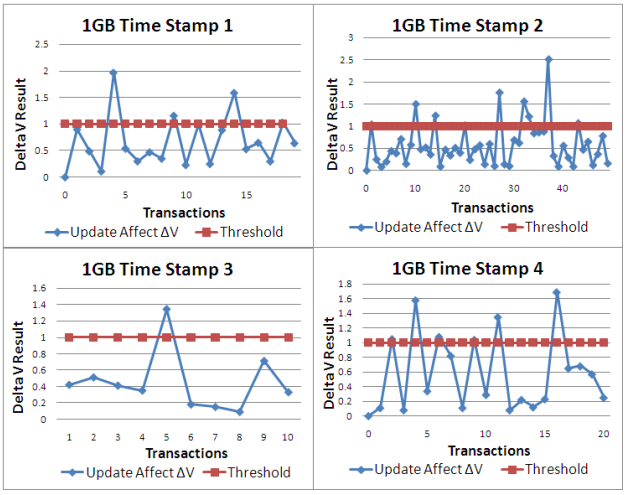
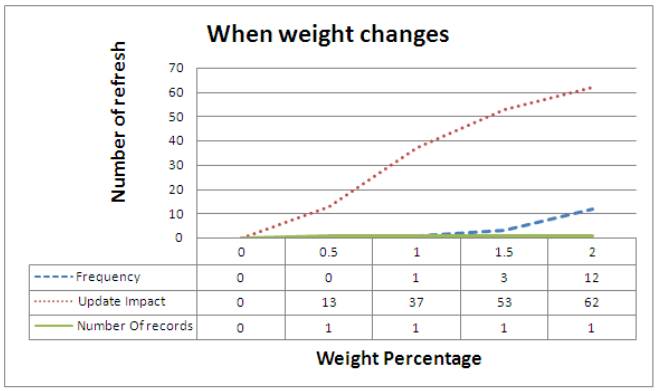


Fig. 7. Number of times refreshed in 1 GB near real-time DW



	0	0.5	1	1.5	2
Frequency	0	0	1	3	12
Update Impact	0	13	37	53	62
Number Of records	0	1	1	1	1

Fig. 8. The effect from weights

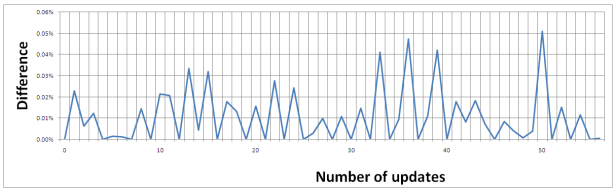


Fig. 9. Difference between Real-Time DW and Near Real-Time DW with query

effect on the delay of information.

VI. CONCLUSION AND FUTURE WORK

In this paper we have presented a methodology for near real-time DW refresh which has been proven to be a beneficial extension to the existing DW environment. This paper presents the necessary requirements for near real-time DW and shows that near real-time DW could save significant operational costs without having important information delayed.

The experiment is built on the real data from TPC-H benchmark [14]. During the experimentations, we particularly focus on cost comparison measured by the number of updates being pushed into DWs. We observe the effect caused by our proposed three indicators.

In the future, we plan to extend this work to cover XML data warehousing, whereby we will concentrate on near real-time updates of XML warehousing [15][16][17] as well as a versioning technique to maintain update queues [18][19][20].

REFERENCES

[1] S. Rizzi and E. Saltarelli, "View materialization vs. indexing: Balancing space constraints in data warehouse design," in *CAiSE*, vol. 2681 of *Lecture Notes in Computer Science*, pp. 502–519, Springer, 2003.

[2] S. Brobst, "Delivery of extreme data freshness with active data warehousing," 2009. Available at Business Intelligence Journal, http://www.tdwi.org/research/display.aspx?ID=6373, Accessed 04 October 2009.

[3] E. Truban, R. Sharda, and J. E.AronsoDavid, *Business Intelligence: A Managerial Approach*. Upper Saddle River, N.J., 2008.

[4] I. C. Italiano and J. E. Ferreira, "Synchronization options for data warehouse designs," *Computer*, vol. 39, no. 3, pp. 53–57, 2006.

[5] B. Liu, S. Chen, and E. A. Rundensteiner, "Batch data warehouse maintenance in dynamic environments," in *In CIKM02*, pp. 68–75, 2002.

[6] M. Thiele, U. Fischer, and W. Lehner, "Partition-based workload scheduling in living data warehouse environments," in *DOLAP '07: Proceedings of the ACM tenth international workshop on Data warehousing and OLAP*, (New York, NY, USA), pp. 57–64, ACM, 2007.

[7] R. J. Santos and J. Bernardino, "Real-time data warehouse loading methodology," in *IDEAS*, pp. 49–58, ACM, 2008.

[8] R. Kimball, *The Data Warehouse Toolkit: Practical Techniques for Building Dimensional Data Warehouses*. John Wiley, 1996.

[9] S. Samtani, V. Kumar, and M. K. Mohania, "Self maintenance of multiple views in data warehousing," in *CIKM*, pp. 292–299, 1999.

[10] B. Bebel, J. Eder, C. Koncilia, T. Morzy, and R. Wrembel, "Creation and management of versions in multiversion data warehouse," in *SAC*, pp. 717–723, 2004.

[11] G. Luo and P. S. Yu, "Content-based filtering for efficient online materialized view maintenance," in *CIKM '08: Proceeding of the 17th ACM conference on Information and knowledge management*, (New York, NY, USA), pp. 163–172, ACM, 2008.

[12] J. Yang and J. Widom, "Incremental computation and maintenance of temporal aggregates," *The VLDB Journal*, vol. 12, no. 3, pp. 262–283, 2003.

[13] D. Maurer, J. W. Rahayu, L. I. Rusu, and D. Taniar, "A right-time refresh for xml data warehouses," in *DASFAA*, vol. 5463 of *Lecture Notes in Computer Science*, pp. 745–749, Springer, 2009.

[14] "Tpc-h decision support benchmark." Website. Transaction Processing Council, Available at http://www.tpc.com.

[15] L. I. Rusu, J. W. Rahayu, and D. Taniar, "Warehousing dynamic xml documents," in *DaWaK*, vol. 4081 of *Lecture Notes in Computer Science*, pp. 175–184, 2006.

[16] L. I. Rusu, J. W. Rahayu, and D. Taniar, "On building xml data warehouses," in *IDEAL*, vol. 3177 of *Lecture Notes in Computer Science*, pp. 293–299, 2004.

[17] E. Pardede, J. W. Rahayu, and D. Taniar, "Object-relational complex structures for xml storage," *Information & Software Technology*, vol. 48, no. 6, pp. 370–384, 2006.

[18] L. I. Rusu, J. W. Rahayu, and D. Taniar, "Mining changes from versions of dynamic xml documents," in *KDXD*, vol. 3915 of *Lecture Notes in Computer Science*, pp. 3–12, 2006.

[19] L. I. Rusu, J. W. Rahayu, and D. Taniar, "Storage techniques for multi-versioned xml documents," in *DASFAA*, vol. 4947 of *Lecture Notes in Computer Science*, pp. 538–545, 2008.

[20] E. Pardede, J. W. Rahayu, and D. Taniar, "Xml data update management in xml-enabled database," *J. Comput. Syst. Sci.*, vol. 74, no. 2, pp. 170–195, 2008.