Delta Extraction Optimization for View Maintenance in a Limited Collaborative Environment

Zheng Lu
School of IT and Computer Science
University of Wollongong, Australia
Email: lu@uow.edu.au

Haijun Liu
Department of Mathematics
Zhengzhou University, Zhengzhou
People's Republic of China, 450052
Email: liuhaijun@zzu.edu.cn

Peter Hyland
School of IT and Computer Science
University of Wollongong, Australia
Email: phyland@uow.edu.au

***Abstract*—This work considers maintaining materialized view in a distributed environment where the collaboration from highly autonomous operational database management systems is limited to granting read only access on a set of selected relational tables. In addition, due to possibly large volumes of remote source data involved in the view maintenance process, and consequently enormous overhead associated with data transfer, efficiency issues must been taken into the consideration. This proposal is based on the observation that usually, there is a large amount of static data in relational tables. It is proposed to use some statistical techniques at data warehouse system to helps us understand the essential characteristics of raw data at remote sites. Based on the characteristics of raw data, some optimization techniques are presented.**

***Index Terms*—Delta Extraction, View Maintenance, Limited Collaborative Environment.**

I. Introduction

Large and multinational businesses often distribute their operational database management systems over wide area networks. Implementation of a data warehouse system that integrates the distributed and highly autonomous operational database systems must overcome many technical and organizational problems. Firstly, distribution and heterogeneity of the operational database systems contribute to the transmission problems and to syntax and semantic related data integration problems. Additionally, the high level of autonomy exercised by local business units of a large organization causes the problems with processing priorities, because the data warehouse related applications are typically run at the lowest possible priority level in order to maintain the highest throughput of the local systems. Optimization of data transmission, i.e. the lower transmission rates, can only be achieved by doing more processing at the remote sites. On the other land, the lower processing load at the remote sites requires transmission of all needed data to an integrated data warehouse.

A *materialized view* is defined as a permanently stored relational table which contains integrated data from other data sources. It is an important technique in data warehousing. When large volumes of data from multiple data sources are involved in query processing, materialized view can significantly accelerate the query processing. However, data sources change over time. To keep the data up to date, a materialized view must reflect these changes and this process is called *materialized view maintenance*. In a distributed environment, view maintenance by recomputing the view definition can incur an enormous overhead associated with data transportation and loading. To avoid this, we need to be able to detect changes from remote data sources. However, the problem of detecting and extracting these changes from highly autonomous systems in a distributed environment is neither a straight forward process nor an efficient one [5], [9].

At present, many proposed solutions to the problem of distributed materialized view maintenance have some similarities. They all assume that remote sites involved are fully engaged in the view maintenance process. Our work aims to highlight an area important to the view maintenance problem that has only received very limited research attention, namely, view maintenance with a limited degree of collaboration supported by a distributed environment.

We define a taxonomy of collaboration levels of view maintenance that is based on the functionalities, resources, distribution and control provided by the remote sites to the data warehouse system. At the highest level, individual remote sites take an active part in the view maintenance by implementing the functions to detect the modifications of their source relations, to validate whether the modifications are applicable to the current view, to compute the minimum change to the current view and to send the result to the data warehouse system. The control over the interactions among the functions mentioned above and resources are also handled by the remote sites. The remote sites prepare all information required to maintain a materialized view, but hide the low level implementation details. The remote sites are a single image thus transparent as viewed from the data warehouse system.

The next level is characterized by the loosely coupled structure of remote sites. From the data warehouse system's point of view, in such setting, although remote sites are logically interrelated over a computer network and participate in a federation to make their local data sharable, they operate

COLLABORATECOM 2012, October 14-17, Pittsburgh, United States
Copyright © 2012 ICST
DOI 10.4108/icst.collaboratecom.2012.250533

independently and are in fact separate systems. Individual remote sites are engaged to the view maintenance process by detecting the modification of their local data sources, notifying the modifications to the data warehouse system or exchanging information among other remote sites if required. The implementation techniques may include deployment of database trigger or writing the transaction file at a remote data sources, etc. The functions and resources owned by remote sites for view maintenance are all made accessible to the data warehouse system. However, the data warehouse system itself is responsible for coordinating the interactions among all remote sites, computing the final modification and applying the modification to the current state of view.

The lowest level is characterized by the isolation of remote sites, where individual remote sites are stand alone DBMS, which know neither the existence of others nor how to communicate with them, so can only exchange information with the data warehouse system, but not with other remote sites. Each remote site restricts its cooperation to granting read-only access rights to the selected relational tables and imposes limitations on the computational resource available to the data warehouse related applications. Furthermore, individual remote sites do not provide any function to support the view maintenance process, for example, no triggers or transaction files upon the data sources to detect and notify of the change of data sources. With this limited degree of collaboration, the data warehouse system not only has to coordinate the view maintenance process, but also has to undertake the tasks such as detecting modification upon data sources, validating the applicability of modification to the current view and generating final changes to the view.

When the collaboration is at the lowest level, many easy options to detect and extract date changes are unavailable. One possible way is to ship all data of interest from all remote sites to a data warehouse system. Here the data from each remote site is referred to as a *data fragment*. After all data fragments are received, based on view definition, the data warehouse assembles these data fragments and reload them into materialized view. In the case of isolation among the remote sites, there is no join operation performed between remote sites, thus the fragments shipped to a data warehouse may have a large amount of redundancy. Secondly, the amount of data transmitted can be quite large and thus costly. The large amount of data transportation may make this solution impractical or too expensive

Our solution is to compare the data in the current view with the data in remote source relations, derive the modification, and generate the change to the view if applicable. Clearly, both precessing costs and communication costs may incurred during this process. Processing costs usually are evaluated in terms of the number of disk accesses and CPU processing time, while communication costs are expressed in terms of total amount of data transmitted. The challenges are to reduce the communication costs over the computer networks and to optimize processing costs at remote systems. In this paper, we are mostly concerned with geographically distributed computer networks and propose a method to reduce the communication costs at the expense of some additional data processing. With the advent of more powerful processors and cheaper memory, we believe the issue of data transportation of a large volume of data will be highlighted.

The paper is structured as follows: In Section 2, we briefly review some previous related works. In Section 3, we present the preliminaries and system overview. In Section 4, we formulate the our research problem. In Section 5, we define a statistic model and present the optimization techniques. In Section 6, we explain the process for delta extraction in details. Finally, in Section 7, we summarize the proposed work and discuss some future works.

II. Previous works

To maintain a materialized view in a distributed environment, one of the critical issues is how to efficiently extract the delta from the remote source systems, particularly if the data volumes are large. The previous works in [4], proposed their data structures and algorithms for the distributed view maintenance, which is based on the assumption that the delta can be accumulated from the transaction logs of the remote source systems. We argue that this approach has its difficulties. Firstly, because of system security and data privacy, the administrators of remote source systems may not be willing to provide such accesses. Secondly, the structure of transaction log file may change over time, which in turn requires the frequent changes of the view maintaining algorithm and data structure used. Recently, many view maintaining algorithms [8], [7], [2] all have the assumption that every remote data source system is able and willing to send a message to notify of the data changes. After receiving the notification message, the data warehouse system issues a set of queries to all other remote source relations involved. The result of queries will be used to compute the update to the current materialized view. This approach requires *all* remote sites to implement database triggers upon the source relations to notify data changes. Certainly, using database trigger is an ideal way to capture delta from remote source relations. However, some legacy source systems may not have such trigger facilities at all. Even if there is one, because of extra costs on application development and maintenance, for remote systems, this may not always be an acceptable solution. The environment of view maintenance with the assumptions listed above, namely, using database trigger and access to transaction log, is referred as a higher degree collaborative environment. A higher degree of collaborations in a distributed environment usually is preferred, however, sometimes it is unachievable in reality.

Extracting delta by comparing a current data snapshot with an earlier one is called the snapshot differential algorithms. To address the problem of delta extraction for a data warehouse in which some autonomous source system may be involved, the works in [5] formally defined the snapshot differential problem and proposed the algorithm by which snapshot differentials can be computed. Basically, a snapshot denotes the a replica of a selected portion of one source relation. However, many materialized views contain integrated data created by relational join or set operations over multiple data sources, thus the snapshot differential algorithm cannot be directly applied to the problem of materialized view delta extraction. In addition, to deal efficiently with large volume of data, [5] also recommended the use of compressed data to perform the data comparison. The work [6] proposed a method of delta extraction in a limited collaboration environment by using a dictionary based data compression technique. Data compression certainly can reduce the volume of data transmission over long distance networks, however, without distinguishing the different characteristics of huge volume of data, compression techniques alone can only achieve a very limited degree of optimization.

III. Preliminaries

A. Data Warehouse Environment

A wide range of transaction scenarios have been considered, however, in this paper, we focus on the PSJ materialized view which is defined by using project Π, select σ and join $\bowtie$, i.e. PSJ expressions over source relations. We assume that either unique attributes or primary key attributes of source relation are included in the view definition. Also, the read-only access to the remote source relational table is granted, and a standard hash function is available at the remote database systems. This assumption is realistic because of ready availability of source code implementations. Finally, multiple interactions between data warehouse system and remote sites are allowed.

A distributed data warehouse environment $E = (\mathcal{V}, \mathcal{S}, \mathcal{L})$, where each $\mathcal{V}_i \in \mathcal{V}$ is a materialized view to be maintained, $\mathcal{S} = \{\mathcal{S}_1, \mathcal{S}_2, ...\mathcal{S}_n\}$ is a set of remote sites involved and $\mathcal{L} = \{l_1, l_2, ...l_n\}$ is a set of communication links between individual remote sites and the data warehouse system. Each link $l_i \in \mathcal{L}$ is described by the pair of nodes it connects, its capacity and unit cost. At the lowest level of collaboration environment, the link only exists between data warehouse system and individual remote site, but not among remote sites. Materialized view contains the integrated data from the set of source relations $R = \{R_1, R_2, ..., R_n\}$, which spread over the remote sites. Each source relation $R_j \in R$ is located at one remote site $\mathcal{S}_i \in \mathcal{S}$ and each $\mathcal{S}_i$ consists of at least one, possibly more source relations.

B. Motivation

It is common that in a relational table, some portion of data may not be changed once or may only infrequently be changed after being created. This portion of data is called *static data*. For example, a table called student-personal-details in an university, when a person is enrolled in any course provided by the university, a data entry is created in the table, which records this student's first name, middle name, last name, title, date of birth, country of origin, contact phone number, physical address, mailing address..., etc. The records in this table are updated for many reasons. Some records in the table are frequently updated because students change their addresses or contact phone numbers. On the other hand, some records are relatively static after being created. Furthermore, year after year, many students complete their degree, but their records are still kept in this table for historical-reference purposes. These records will remain unchanged unless students register a new course. The proportion of this kind of data in the table will be incremented gradually, in other words, this table may include a significant amount of static data. If there is a large amount of static data in a source relation, it is inefficient to treat static and frequently changed data in the same way. In this work, a simple statistical model is used at the data warehouse system to monitor the frequency of records updating in the remote source relation. In order to optimize the data transmission over long distance networks, based on the statistics gathered, we also propose a hash function based approaches to deal with different categories of data, namely, static data and frequently changed data.

IV. Problem of Delta Extraction

A. View Decomposition

As we mentioned before, the data modifications of remote source relations are identified by comparing the current view with the data held by the current state of remote source relations. Thus two sets of data have to be established to compare against one another. As far as the data in remote site is concerned, according to the view maintenance environment specified above, we consider to decompose the view definition into a set of subqueries $\varphi = \{\varphi_1, \varphi_2, ..., \varphi_n\}$ as a preliminary step. Unlike the works in [12], that was designed for central database query processing and the query was proposed to be decomposed into in a sequence of irreducible queries, i.e. single variable queries. The view definition decomposition in this work aims to find the data that is relevant to the current materialized view at each remote site, thus the granularity of the decomposition is at the individual remote site, rather than each single source relation. The process of view definition decomposition is firstly, detaching the source relations of each single remote site from view definition, then push a select restriction and local join conditions q into the detached source relations, whenever possible. Finally, assigning the projection operator to ensure that only attributes α either in the list of view definition or involved in some global join condition are included in the projected attribute list. After the decomposition, each decomposed subquery φ_i can be directly evaluated by a single remote site. Also, each decomposed subquery φ_i consists of either a single source relation or multiple joinable source relations. Regarding to this, a simple rule is that at each remote site, a decomposed subquery φ_i consists of multiple source relations if they are joinable, or φ_i only consists of a single source relation if it is not. Let $R_i...R_n$ be a set of the source relations located at the remote site $\mathcal{S}_i$. A decomposed subquery φ_i from the view definition is of the form:

$$\varphi_i = \begin{cases} \Pi[\alpha_i](\sigma_q(R_i)) & R_i \text{ not joinable} \\ \Pi[\alpha_i, ..., \alpha_m](\sigma_q(R_i... \bowtie ...R_m)) & R_i...R_m \text{ joinable} \end{cases}$$

Apparently, another set of data is from the current materialized view. For each decomposed subquery φ_i, its counterpart φ'_i at the data warehouse system can be acquired by assigning a projection operator to the materialized view and making the projected attribute list identical to one that the subquery φ_i holds, that is $\varphi'_i = \Pi[\alpha_i](\mathcal{V})$. Each φ'_i represents the data fragment being included as the part of current materialized view and has the identical schema as φ_i. In the following sections, φ_i denote a decomposed query and φ'_i denotes the *snapshot* of φ_i. *Snapshot projection* is to generate a set of snapshots corresponding to each of decomposed subqueries.

B. Operations that Cause Data Inconsistency

In relational model, three basic types operations on source relations cause the data inconsistency between source relations and their snapshots. Based on the key constraints, these three types of inconsistency are interpreted as follows: let φ_i be a decomposed query and φ'_i be its corresponding snapshot:
- insertion, denoted by t^+_j, that is $\exists t_j \in \varphi_i, \forall t'_j \in \varphi'_i$ such that $t_j.k \neq t'_j.k$.
- deletion, denoted by t^-_j, that is $\exists t'_j \in \varphi'_i, \forall t'_j \in \varphi_i$ such that $t_j.k \neq t'_j.k$.
- update, denoted by $t^\pm_j$, that is $\exists t_j \in \varphi_i, \exists t'_j \in \varphi'_i$ such that $t_j.k = t'_j.k$ and $t_j.v \neq t'_j.v$.

C. Differential File

A differential file $\Delta(R_i, R'_i)$ refers to the differences from relation R'_i to relation R_i. A differential file $\Delta(R_i, R'_i)$ has following properties: 1). both R_i and R'_i are query expressions that have identical relation schema; 2). a differential file consists of a pair of relations

$$\Delta(R_i, R'_i) = (\Delta^+(R_i, R'_i), \Delta^-(R_i, R'_i))$$

where $\Delta^+(R_i, R'_i) = R_i - R'_i$ and $\Delta^-(R_i, R'_i) = R'_i - R_i$. 3). the the query R_i can be computed as follows:

$$R_i = (R'_i \cup \Delta^+(R_i, R'_i)) - \Delta^-(R_i, R'_i)$$

This process is referred to as *data synchronization* of R_i and R'_i. Regarding materialized view maintenance, the expression $\Delta(\varphi_i, \varphi'_i)$ represents a differential file of a query result φ_i at remote system and its corresponding snapshot φ'_i projected from materialized view. A differential file does not only reflect the inconsistency between the current state of materialized view and the data residing in operational databases, but also can be used to refresh the materialized view. When no handy options are available, differential files are inferred by comparing each query result φ_i to its corresponding snapshot φ'_i. We call this problem as delta extraction.

V. Optimization Techniques

A. Overview

In this work, a group hash method is proposed, which requires less data transmission to generate a differential file. By using this method, the delta is extracted by examining the differences between groups of tuples, rather than comparing each individual tuple. Let $\eta_j \subseteq \varphi_i$, and κ_j be all keys of η_j. Also let $\eta'_j \subseteq \varphi'_i$, and κ'_j be all keys of η'_j. If η_j and η'_j are about to compare against each other, then η_j and η'_j are referred to as *peer groups*. If η_j and η'_j have a set of identical keys, i.e. $\kappa_j = \kappa'_j$, then η_j and η'_j are referred to as *matched groups*, represented by $\{\eta_j : \eta'_j\}$. Group η_j and η'_j are the *matched group* of each other. Being matched groups implies that η_j and η'_j have the same cardinality, i.e. $|\eta_j| = |\eta'_j|$, and there is a one to one correspondence between κ_j and κ'_j.

The basic idea of delta extraction is as follows: the hash value of η'_j, denoted by $h(\eta'_j)$, along with its key set κ'_j are sent to the remote site where φ_i is located. Provided that the matched groups $\{\eta_j : \eta'_j\}$ can be established, and in case of $h(\eta_j) = h(\eta'_j)$, the matched groups are referred to as an *identical groups*, represented by $\{\eta_j : *\eta'_j\}$. To explain the benefits of using group hash method, assuming that a group η'_j contains 20 tuples, the average tuple size is 100 bytes, i.e. 800 bits, the average key size is 10 bytes, i.e. 80 bits, and Φ_h is 160 bits. For the delta extraction, in case of $h(\eta_j) = h(\eta'_j)$, i.e. the identical groups can be established, we can expect $800 \times 20 - (80 \times 20 + 160) = 14240$ bits saving on data transmission, and the saving ratio is $\frac{14240}{800 \times 20} = 89\%$.

Using group hash method, the benefits result from the successful identification of the static data existing in remote source relations or say decomposed queries. The greater the amount of static data can be identified, the smaller the amount of data is required to complete the data synchronization. To gain the benefits as the example above, the necessary and sufficient condition is that the matched groups are also the identical groups, i.e. $h(\eta_j) = h(\eta'_j)$. This condition implies that the matched groups must be able to be established in the first place. However, deletion to a decomposed query causes the failure of establishing the matched groups. Hence, at the data warehouse system, a group selection strategy must be developed to ensure that any selected group from the snapshot has a better chance to find its matched group in the decomposed query. Furthermore, even if the matched groups could be established, any update occurred in a group also makes the matched groups no longer an identical group. In short, deletion causes the failure of establishing matched groups, and update causes the failure of establishing the identical group even if the matched groups have been established. Deletion and update to a decomposed query are two important factors that should be addressed by the group selection strategy.

B. Statistical Tables

The *group selection* means which tuples should be combined together as a group and which tuples should not. The group selection strategy is a part of the overall optimization, which generates a grouping plan that contains a set of disjoint groups, that is $\Lambda = \{\eta'_1, \eta'_2, ..., \eta'_n\}$ where $\eta'_i \cap \eta'_j = \emptyset$ and $\varphi_i = \eta'_1 \cup \eta'_2, ..., \cup \eta'_n$. We have listed the two factors that the group selection strategy should consider of, i.e. deletion and update to a decomposed query, while in this section, we answer questions: what information is required by the strategy and how to acquire that information.

The deletion frequency and the update frequency of remote data are two important parameters for the group selection decision making process. Certainly, having more information at hand about the remote data may lead to a more efficient way of differential extraction. However, in a limited collaboration environment, we cannot expect that all relevant information is ready to be used. To be able to obtain the necessary information, for each decomposed query φ_i, the data warehouse system maintains a pair of statistical tables: $\omega_i^-, \omega_i^\pm$, which monitor the data pattern of the remote source relations that contain a large volume of data.

A statistical table $\omega_i^\pm$ is to monitor the tuples' update frequency in a the decomposed query φ_i. $\omega_i^\pm$ is described by (*tid, tno, uno*), where tid represents an identifier of a tuple. Note that the tid can consist of either single or multiple attributes. tno represents the total number of times the view has been maintained since this tuple was included in the materialized view, and uno represents the number of times that this tuple has been updated so far.

The maintenance of the statistical table $\omega_i^\pm$ is simple. For example, when a tuple t_i in the decomposed query is first included into the materialized view, a new entry is created in $\omega_i^\pm$ as follows: $tid_i = t_i.k$, $tno_i = 1$ and $uno_i = 1$. After that, it will incremented tno by one every time the data synchronization is completed until t_i is excluded from the materialized view. Whether t_i was updated or not there are always two possible outcomes: $\{upd(t_i), \neg upd(t_i)\}$, where $upd(t_i)$ means the update has occurred on t_i, and $\neg upd(t_i)$ means no update has occurred on t_i. Only if $upd(t_i)$, then it increments uno_i by one.

For the next time, whether or not the tuple $t_i \in \varphi_i$ will be updated, there are two possible outcomes: $\{\pm t_i, *t_i\}$, here $\{\pm t_i\}$ denotes t_i will be updated, and $\{*t_i\}$ denotes the contrary to $\{\pm t_i\}$, i.e. t_i will not be updated. Clearly, the actual outcome is not known until the synchronization is complete. If the sampling dataset is large enough, then the following formula:

$$P(\{\pm t_i\}) = \frac{uno_i}{tno_i}$$

not only represents the frequency of $upd(t_i)$ in the past, but also represents the probability of $\{\pm t_i\}$ in the future. By their definition, it holds that

$$P(\{\pm t_i, *t_i\}) = P(\{\pm t_i\}) + P(\{*t_i\}) = 1$$

Another statistical table ω_i^- aims to monitor the deletion pattern of a decomposed query φ_i. Unlike $\omega_i^\pm$ which keeps track of the individual tuples, ω_i^- only records the average deletion ratio of every single snapshot synchronization. ω_i^- is described by tno, dno, where tno represents the total number of tuples in a snapshot before the synchronization, and dno represents the number of tuples being deleted after the synchronization. For a particular synchronization j, $a_j = dno_j/tno_j$ represents its deletion ratio. The deletion ratio by considering the overall synchronizations rather than a single synchronization can also be calculated

$$P(\{-t_j\}) = \frac{1}{|\omega_i^-|}\sum_{j=1}^{|\omega_i^-|} a_j$$

The value $P(\{-t_j\})$ is interpreted as the tuple's deletion probability. In addition, $P(\{: t_j\})$ is used to represent the contrary of $P(\{-t_j\})$, i.e. the probability of a tuple not being deleted, then it has

$$P(\{-t_j\}, \{: t_j\}) = P(\{-t_j\}) + P(\{: t_j\}) = 1$$

Symbols	Description
$1 : \omega_i^\pm$	monitor update frequency
$2 : \omega_i^-$	monitor deletion frequency
$3 : upd(t_i)$	tuple t_i was updated
$4 : \neg upd(t_i)$	tuple t_i was not updated
$5 : \{\pm t_i\}$	tuple t_i will be updated
$6 : \{*t_i\}$	tuple t_i will not be updated
$7 : \{-t_i\}$	tuple t_i will be deleted
$8 : \{: t_i\}$	tuple t_i will not be deleted
$9 : \{\pm\eta_j\}$	at least one tuple in η_j will be updated
$10 : \{*\eta_j\}$	no tuple in η_j will be updated
$11 : \{-\eta_j\}$	at least one tuple in η_j will be deleted
$12 : \{: \eta_j\}$	no tuple in η_j will be deleted
$13 : \{\eta_j : \eta_j'\}$	matched pair
$14 : \{\eta_j : *\eta_j'\}$	identical group

TABLE I
STATISTICAL TABLES AND EVENTS

C. Properties of Group

For the next synchronization, at least one $t_i \in \eta_j$ *will be updated* has two possible outcomes $\{\pm\eta_j, *\eta_j\}$, where:
1). $\{\pm\eta_j\}$ is understood as: at least one tuple in this group will be updated, that is $\exists t_i \in \eta_j$, such that $\{\pm t_i\}$;
2). $\{*\eta_j\}$ is understood as: no tuple in this group will be updated, that is $\neg\exists t_i \in \eta_j$, such that $\{\pm t_i\}$.
We are more interested in the second one, because it helps to better identify the static portion of the remote data. The outcome is not known in advance, i.e. before the data synchronization is complete. However, what we can know is its probability, denoted by $P(\{*\eta_j\})$.

The derivation of update probability at the group level is based on each individual tuple's update probability in this group. A group's update probability is understood as the likelihood that: $\forall t_i \in \eta_j$, such that $\{*t_i\}$. Since the each tuple's update probability can be obtained from the statistical table, i.e. $P(\{\pm t_i\})$, its complement is:

$$P(\{*t_i\}) = 1 - P(\{\pm t_i\})$$

Based on the understanding of the group update probability, it has

$$P(\{*\eta_j\}) = P(\{*t_1\}) \cap P(\{*t_2\}) \cap ... \cap P(\{*t_n\})$$

In a relational database, the fact that update occurred on a tuple t_i gave us no information about whether the update occurred on a tuple t_j where $i \neq j$, therefore updating a tuple is an *independent event* and $P(\{*\eta_j\})$ can be equally transformed as:

$$P(\{*\eta_j\}) = P(\{*t_1\}) \times P(\{*t_2\}) \times ... \times P(\{*t_n\})$$

Regarding deletion, for each individual tuple, it also has two possible outcomes for the next synchronization:
1). $\{-\eta_j\}$: at least one tuple will be deleted from this group, that is $\exists t_i \in \eta_j$, such that $\{-t_i\}$;
2). $\{: \eta_j\}$: no tuple will be deleted from this group, that is $\neg\exists t_i \in \eta_j$, such that $\{-t_i\}$.
From the value of deletion probability, it is easy to get value of its complement, i.e. $P(\{: t_i\}) = 1 - P(\{-t_i\})$, then

$$P(\{: \eta_j\}) = P(\{: t_1\})^n$$

In the process of snapshot synchronization, for any peer groups, only if there was no deletion occurred in the group at the remote site, this peer groups can be classified as the matched groups. For any matched groups, only if there is no update occurred in the group at the remote site, this matched groups can be classified as identical groups. In other words, if peer groups η_j and η'_j are identical groups, i.e. $\{\eta_j : *\eta'_j\}$, then both $\{: \eta_j\}$ and $\{*\eta_j\}$ must be true, and its probability is

$$P(\{\eta_j : *\eta'_j\}) = P(\{: \eta_j\}) \cap P(\{*\eta_j\})$$

Symbols	Description
$1 : P(\{\pm t_i\}$	The probability of updating tuple t_i
$2 : P(\{*t_i\})$	The probability of not updating tuple t_i
$3 : P(\{-t_j\})$	The probability of tuple t_i being deleted
$4 : P(\{: t_j\})$	The probability of tuple t_i not being deleted
$5 : P(\{\pm\eta_j\})$	The probability of having update in η_j
$6 : P(\{*\eta_j\})$	The probability of no updating in η_j
$7 : P(\{-\eta_j\})$	The probability of having deletion in η_j
$8 : P(\{: \eta_j\})$	The probability of having no deletion in η_j
$9 : P(\{\eta_j : *\eta'_j\}$	The probability of η_j remains unchanged

TABLE II
PROBABILITIES

D. Cost Model

In this subsection, we will formulate the basic costs and benefits for our method. Let Φ_t be the average tuple size, Φ_k be the average key size, $|\eta_j|$ be the cardinality of a group η_j, i.e. how many tuples in η_j. If the synchronization method is simply to transfer all data, then the data transmission for the group η_j is:

$$sz(\eta_j) = \Phi_t \times |\eta_j|$$

When using the group hash method, in order to compare the groups η_j and η'_j, the amount of data required to be transfered to the remote site is:

$$hz(\eta_j) = \Phi_k \times |\eta_j| + \Phi_h$$

In the case that identical groups could be established, i.e. $\{\eta_j : *\eta'_j\}$, by using the group hash method, the expected benefit for synchronizing groups η_j and η'_j is formulated as:

$$\mu(\eta_j) = sz(\eta_j) - hz(\eta_j) = (\Phi_t - \Phi_k) \times |\eta_j| - \Phi_h$$

By considering the probability of establishing the identical groups, i.e. $P(\{\eta_j : *\eta'_j\})$, the expected benefit can be formulated as:

$$ES(\eta_j) = \mu_j \times P(\{\eta_j : *\eta'_j\})$$

In addition, update or deletion all causes the failure of establishing identical groups. In this case, $hz(\eta_j)$ is considered to be the data losses, represented by a negative number, i.e. $-hz_j$. The chance of data losses is the complement of the chance that identical groups can be established, that is: $1 - P(\{\eta_j : *\eta'_j\})$. When the data losses are taken into account, then the formulated expected benefits would be:

$$EB(\eta_j) = ES(\eta_j) - hz(\eta_j) \times (1 - P(\{\eta_j : *\eta'_j\}))$$

The group selection process for a snapshot is to generate a set of selected groups $\Lambda_i = \{\eta'_1, ..., \eta'_n\}$. By using group hash method, the overall expected benefits gained from the data synchronization is formulated as:

$$EB(\varphi_i) = \sum_{j=1}^{n} EB(\eta_j)$$

The value of $EB(\varphi_i)$ relies on the frequency interpretation of probabilities, which is not interpreted as the reduction in data transmitted from a particular data synchronization, but rather as the average outcome over the long run.

E. Group Selection Strategy

Different group selection strategies generate different group selection plans, and different group selection plans yield different results of expected benefits. Among all the possible plans, let $max(EB)$ represent the maximum expected benefits that can be achieved. The group selection strategy manages to find the best possible plan, such that the expected benefits is approximately equal to $max(EB)$, that is:

$$EB(\varphi_i) \approx max(EB)$$

The characteristics that each individual group has made group selection an interesting problem. For an individual group, to decide its proper size, one option is to use the group cardinality as the selection criteria. If using larger cardinality, i.e. having more tuples in this group, then it would lead to a larger value of expected benefits in the case the identical groups could be established. However, at the same time, the increased cardinality also implies the less chance of establishing identical groups and greater chance of data losses. Another option is to use the probability of identical group as the selection criteria. If we decide to choose a greater probability of identical groups, it would increase the chance of data saving, but it would also leads to a smaller cardinality of group thus the smaller value of expected benefits in case the identical groups could be established. The cardinality of group and the probability of identical groups are two correlated variables in the group selection problem. To resolve its inherent dilemma, we propose an algorithm that provides an optimized solution to the group selection problem.

Before presenting the details of the algorithm, we define two functions. Firstly, $f_{\uplus} : \eta_j \rightarrow \eta_{j+1}$, referred to as the group inclusion function, η_{j+1} is produced by adding one more tuple into a group η_j. Secondly, the sorting function is defined as: $f_{\Downarrow\tau} : (R_i) \rightarrow \ell_i$, where τ is an attribute or a derived value of R_i, ℓ_i is a list of tuples sorted by τ in descending order. In the algorithm, a snapshot φ'_i is sorted as follows: $f_{\Downarrow P(\{\pm t\})}(\Pi[A_1, ..., A_n](\varphi'_i \bowtie \omega_i^{\pm}))$, where $P(\{\pm t\})$ is a tuple's update frequency, and $A_1, ..., A_n$ is the set of all attributes in φ'_i. The input to the algorithm includes: the snapshot φ'_i, and the statistical tables $\omega_i^{\pm}$, ω_i^{-}. The output from the algorithm is an optimized grouping plan $\Lambda_i = \{\eta'_1, \eta'_2, ..., \eta'_n\}$. The guideline for group selection is to always find the group with the maximum expected benefits from the current set of tuples. The algorithm details is listed in algorithm 1;

Algorithm 1 Group Selection Algorithm

```
1: procedure GroupSelection($\varphi'_i, \omega_i^{\pm}, \omega_i^{-}$)
2:     $j \leftarrow 1$
3:     $i \leftarrow 1$
4:     $\eta_{tmp1} \leftarrow \emptyset$
5:     $\eta_{tmp2} \leftarrow \emptyset$
6:     $\ell_i \leftarrow f_{\Downarrow p(\{\pm t\})}(\Pi[A_1, ..., A_n](\varphi'_i \bowtie \omega_i^{\pm}))$
7:     while $j < |\ell_i|$ do
8:         $\eta_{tmp1} \leftarrow f_{\uplus}(\eta_{tmp1}, t_j \in \ell_i)$
9:         if $EB(\eta_{tmp1}) \leq 0$ then
10:            exit                          ▷ Terminates the process
11:        end if
12:        $\eta_{tmp2} \leftarrow f_{\uplus}(\eta_{tmp1}, t_{j+1} \in \ell_i)$
13:        if $EB(\eta_{tmp1}) > EB(\eta_{tmp2})$ then
14:            $\eta_i \leftarrow \eta_{tmp1}$               ▷ Output: group selected
15:            $i \leftarrow i + 1$
16:            $\eta_{tmp1} \leftarrow \emptyset$
17:        else
18:            $\eta_{tmp2} \leftarrow \emptyset$
19:        end if
20:        $j \leftarrow j + 1$
21:    end while
22: end procedure
```

There are two circumstances that cause the algorithm to terminate. Firstly, when the expected benefits of the new selected group is negative, the algorithm terminates. If the current selected group has negative value for expected benefit, then all following groups can only have negative values for expected benefit. Secondly, if all tuples are processed and are allocated into different groups.

VI. Process of Delta Extraction

Delta extraction is a process to generate a differential file $\Delta(\varphi_i, \varphi'_i)$, where φ_i denotes a query result at a remote site, and φ'_i denotes the corresponding snapshot projected from a materialized view. A differential file $\Delta(\varphi_i, \varphi'_i)$ consists of two relations: $\Delta^+(\varphi_i, \varphi'_i)$ and $\Delta^-(\varphi_i, \varphi'_i)$. To generate a differential file, the process includes three steps:
1). data preparation at the data warehouse system;
2). identification of the insertion and deletion at remote sites;
3). identification of the update at remote sites;

At the data warehouse system, step 1 takes the group selection plan $\Lambda_i = \{\eta_1, \eta_2, ..., \eta_n\}$ as its input. The function $f_{\mathsf{T}} : \eta_i \rightarrow u_i$ transforms each $\eta_i \in \Lambda_i$ into a data unit u_i. Each u_i is a 3-tuple $\langle uid_i, \eta_i.k, h(\eta_i)\rangle$, where uid_i is the identifier of group η_i, $\eta_i.k$ is a set of all keys of group η_i, and $h(\eta_i)$ is the group hash value. The output of this step are all transformed data units $\{u_1, ..., u_n\}$, which will be sent to the remote site where the φ_i is located.
Step 2 aims to identify all tuples that were added into the decomposed query, i.e. t^+, and all tuples that were remove from the decomposed query, ie. t^-, since the last synchronization. Let $\varphi_i.k$ and $\varphi'_i.k$ represent all keys in the current state of the decomposed query and the snapshot respectively. The operation to identify the insertion at remote site is: $\varphi_i.k - \varphi'_i.k$. As the result of this operation, it has:

$$t^+ = \{t \,|\, t.k \in \varphi_i.k \wedge t.k \notin \varphi'_i.k\}$$

The operation to identify the deletion is: $\varphi'_i.k - \varphi_i.k$. As the result of this operation, it has:

$$t^- = \{t \,|\, t.k \in \varphi'_i.k \wedge t.k \notin \varphi_i.k\}$$

Unlike the identified insertion t^+, a full set of values of each tuple are required to send to data warehouse system to complete the synchronization. For the identified deletion t^-, only the key value itself is required. However, the deleted tuple from the decomposed query makes the received data units that contains these tuples are no longer able to find its matched group. In this case, for the affected data units, uid together the key values of deleted tuples are sent to the data warehouse to be regrouped. Regroup here means the deleted tuple is removed from the original data unit and then the remaining tuples are used to compute a new data unit.

At this step, the major operation at the remote site is limited to the key comparison. The identified insertion is a subset of the differential file, i.e. $t^+ \subseteq \Delta^+(\varphi_i, \varphi'_i)$. Similarly, the identified deletion is a subset of the other part of the differential file, i.e. $t^- \subseteq \Delta^-(\varphi_i, \varphi'_i)$.

Step 3 aims to identify the update of tuples $t^{\pm}$ in the decomposed query since the last synchronization, by using the group hash approach. After step 2, all the remaining data units plus the regrouped data units have resolved the problem of group mismatch and they are all qualified to have their matched groups at the remote site. The function to locate the matched group of a data unit is: $\eta_j \leftarrow \sigma_{\varphi.k \in u_j.k}(\varphi_i)$. By using this function, the algorithm to identify the update is represented in algorithm 2.

Algorithm 2 Group Comparison Algorithm

```
    procedure IDENTIFYUPDATE($\varphi_i, u$)
2:      $i \leftarrow 1$
        $\eta_{tmp} \leftarrow \emptyset$
4:      $t^{\pm} \leftarrow \emptyset$
        while $i < |u|$ do          ▷ the number of all data units
6:          $\eta_{tmp} \leftarrow \sigma_{\varphi.k \in u_i.k}(\varphi_i)$
            if $h(\eta_{tmp}) \neq u_i.h(\eta)$ then
8:              $t^{\pm} \leftarrow t^{\pm} \cup \eta_{tmp}$
            end if
10:         $i \leftarrow i + 1$
        end while
12: end procedure
```

The outcome of the algorithm above is the identified update that is the subset of a differential file, i.e. $t^{\pm} \subseteq \Delta^{+}(\varphi_i, \varphi_i')$. In addition, when all steps of deleta extraction are compelte, a differential file is generated and represented as:

$$\Delta^{+}(\varphi_i, \varphi_i') = t^{+} \subseteq \varphi_i \cup t^{\pm} \subseteq \varphi_i$$

$$\Delta^{-}(\varphi_i, \varphi_i') = t^{-} \subseteq \varphi_i' \cup t^{\pm} \subseteq \varphi_i'$$

VII. Summary and Future Works

In this paper, we defined the taxonomy of collaboration levels of view maintenance in a distributed environment, in which the differences in terms of functionalities, resources, distribution and control are highlighted. We claim that the problem of maintaining a materialized view in a distributed environment only with limited degree of collaborations is fundamentally different from the conventional one over distributed databases. Thus the existing assumptions and optimization techniques must be re-examined.

In a limited collaboration environment, the differential files are inferred by comparing decomposed queries to its corresponding snapshots. The decomposed queries represent the current state of data at remote sites. The snapshots that have identical schema as their corresponding decomposed queries represent the current state of data in the materialized view. Our optimization objectives is to reduce the communication costs over the computer networks when extracting the differential files.

Based on the observation that usually there is a large amount of static data in relational tables, the statistic tables were proposed to be created at the data warehouse system, which represent the history of data change frequency at remote sites. Certainly, maintaining statistical tables incurs some extra costs at the data warehouse system, but it provides a simple statistic model that helps us to gain some knowledge about remote data. The essential characteristics of data reflected by the statistical tables in turn creates some possibilities of optimization, which underpin the objectives we are trying to achieve. By using the statistical tables, we proposed an approach for the delta extraction, which includes a cost-driven group selection strategy and the group hash method. An example was given to show that by our approach over long run, an expected saving on data transmission could be achieved. We also explained in details that how delta extraction process is completed and how the differential files are generated.

We are not going to claim that this proposed work is the best solution to the distributed view maintenance problem. However, we believe that it is complementary to some other approaches, because it creates a possibility to maintain a materialized view when the higher level of collaborations are not easily available.

References

[1] B. Babcock, C. Olston. Distributed top-k monitoring. *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 28-39, June 2003.

[2] S. Chen, B. Liu and E. Rundensteiner. Multiversion Based View Maintenance over Distributed Data Sources. *ACM Transactions on Database Systems (TODS)*, 29(4), pages 675-709, 2004.

[3] G. GRAEFE. Query Evaluation Techniques for Large Databases, *ACM Computing Surveys* (CSUR), 25(2), pages 73-169, 1993.

[4] A. Kawaguchi, D. F. Lieuwen, I. S. Mumick and K. A. Ross. Implementing incremental view maintenance in nested data models. In *Proceedings of the 6th International Workshop on Database Programming Languages*, 1998.

[5] W. J. Labio and H. Garcia-Molina. Efficient snapshot differential algorithms for data warehousing. In *Proceedings of VLDB Conference*, pages 63-74, 1996.

[6] Z. Lu and J. Getta. Identify and extract delta of materialized view with limited collaborations. In *Proceedings of Parallel and Distributed Processing Techniques and Applications*, page 645-651, 2008.

[7] G. Moro and C. Sartori. Incremental maintenance of multi-source views. In *Proceedings of the 12th Australasian conference on Database technologies* (ADC), pages 13-20, 2001.

[8] N. Roussopoulos. An incremental access method for ViewCache: concept, algorithms, and cost analysis. *Transactions on Database Systems* (TODS), 16(3), 1991.

[9] P. Ram and L. Do. Extracting delta for incremental data warehouse maintenance. In *proceedings of the 16th International Conference on Data Engineering*, 2000.

[10] K. Stocker, D. Kossmann, et al. Integrating semijoin reducers into state-of-the-art query processors. In Proceedings of the 17th International Conference on Data Engineering, 2001.

[11] T. Westmann, D. Kossmann, S. Helmer and G. Moerkotte. The Implementation and performance of compressed databases. *SIGMOD Record* 29(3), pages 55-67, 2000.

[12] E. Wong and K. Youssefi. Decomposition - a strategy for query processing. *ACM Transactions on Database Systems*, 1(3), 1976.