Extracting Delta for Incremental Data Warehouse Maintenance

Prabhu Ram and Lyman Do
Phantom Works Mathematics and Computing Technology
P.O. Box 3707, M/S 7L-40
The Boeing Company
Seattle, WA 98124-2207
{prabhu.ram, lyman.s.do}@boeing.com

Abstract

This paper seeks to highlight an area important to commercial data warehouse deployments that has received limited research attention, namely, the extraction of changes to the data at the source systems. We refer to these changes as deltas. Extracting deltas from source systems is the first step in the incremental maintenance of data warehouses. A common assumption among current incremental maintenance methods is that deltas are somehow made available - normally in the form of differential files. Extraction of deltas from source systems is often not a straight forward process nor an efficient one. In this paper, we analyze how deltas can be extracted from large systems. We analyze delta extraction methods that are currently available, namely, time stamps, differential snapshots, triggers, and archive logs. We point out the strengths and weaknesses of each method through analysis and when appropriate through experimentation. We have been investigating the method called Op-Delta at Boeing that better suits delta extraction from large integrated systems. We discuss the benefits of Op-Delta, discuss how it could be implemented, and present comparative results from our experimentation.

1 Introduction

Enterprises want to analyze data stored in their operational systems for reasons including process improvements, cycle time reduction, forecasting, reporting, etc. Ideally, such analysis should be performed on the operational systems to obtain timely results. However, since this is impossible for a number of reasons including autonomy and performance, data warehouses came about. One primary function of a data warehouse is to take the decision support system (DSS) load off the operational systems. Data warehouses tend to be optimized for DSS queries while operational systems are usually optimized for production workloads such as OLTP.

To incrementally maintain a data warehouse, changes to the data (henceforth referred to as deltas) in the source systems have to be extracted first and that is the focus of this paper. The delta extraction process is often difficult and almost always time consuming. Research literature has focused on the importance of how the changes in data at the source systems can be incorporated into data warehouses [1, 5, 11, 13, 14, 17, 21, 26, 27, 36, 37, 38]. There is little recognition of the intricacies associated with how delta can be extracted in a manner that supports its efficient integration into a data warehouse. Labio *et. al* [19] recognize the update window for the data warehouse needs to be shrunk to have the data warehouse be more available or the maintenance activity competes directly against the concurrent OLAP queries for data warehouse resources. The work in [19] focuses only on how aggregate views may be maintained more efficiently to reduce the outage of a data warehouse but ignores the aspects of delta capture and its integration into the data warehouse without causing an outage at the data warehouse. The assumption in most of the aforementioned studies is that differential relations are somehow available for data warehouse maintenance purposes. Labio and Garcia-Molina [18] imply that delta extraction from non-legacy systems is a routine matter and discuss ways by which snapshot differentials from legacy systems may be computed. However, delta extraction problems exist in non-legacy systems as well and we discuss the requirements faced by such delta extraction methods in Section 2. Popular data warehouse products make the same assumption and rely upon the data warehouse system designer to somehow produce deltas. Products, such as Essbase, Informix MetaCube, Oracle Express, Red Brick Data Warehouse, and SAS Data Warehouse Administrator, require differential relations or delta fact tables to be in place already [2, 23, 28, 32]. Extraction products such as Informatica PowerCapture and ETI Extract Toolkit [10, 15] support delta extraction from different types of source systems with varying granules of control but largely do not address the challenges and do not meet the requirements we have listed in Section 2.

Though only delta extraction is the focus of this paper, for completeness we briefly follow the rest of the end-to-end process of incremental maintenance. Once deltas are extracted, cleansed and scrubbed, they now need to be

transported to the data warehouse (or a staging area). Several techniques such as ftp, persistent queues, and fault tolerant logs [4] all apply and the choice of technique depends on the requirement of transaction guarantees. The deltas now need to be transformed into a format suitable to the data warehouse schema and be integrated into the data warehouse. Listings of research in the transformation and integration areas can be found in [6, 22]. The challenges of integrating delta into data warehouses include a) correct and efficient integration of deltas into the data warehouse, b) avoiding the shutdown of data warehouses during the integration process, and c) not hindering the execution of long-lived user queries. Finally, the end-to-end process - the extraction, transportation, transformation, and integration - must work quickly enough (defined by the enterprises' needs) for a data warehouse to reflect the "current" state of source systems.

2 Extracting Deltas from Source Systems

To frame our discussion we present a reference architecture. Figure 1 shows the end-to-end incremental maintenance process consisting of delta extraction at the sources, transportation of the deltas to the data warehouse, and, transformation and integration of the deltas to the data warehouse. We begin by discussing characteristics of source systems that have impact on the delta extraction processes.

2.1 Characteristics of Source Systems

In Figure 1, the architecture of typical enterprise systems consists of multiple tiers of software. A trend among large enterprises is to construct their operational systems by integrating commercial off-the-shelf (COTS) software. COTS software are inter-connected through distributed communication technology such as CORBA, DCE, and DCOM to form integrated business systems. These COTS software encapsulate DBMSs and the databases are unaware of each other's existence across multiple instances of COTS software. The wide availability of COTS software and the increasing number of them being used to integrate into large business systems, and, the architecture of distributed environments make the task of delta extraction complex. Following are some of the challenges to delta extraction from COTS software based integrated systems:

- Global serializability [33] is often not enforced in the COTS software systems for incompatibility and performance reasons [9, 30, 31]. However, research and commercial products implicitly assume that deltas are a result of serializable executions at the sources. The mismatch between this assumption and the reality of how vertical applications are integrated in an enterprise poses significant challenges to delta extraction from the sources.
- The COTS software often encapsulate their underlying databases and they only expose APIs through which to access the encapsulated data. The database encapsulation by itself poses significant challenges to the delta extraction process. For example, as required by a few delta extraction methods (Section 3), the logs and other internal database structures may not be available externally for delta extraction. Even when such structures are accessible the semantics of what is stored in them is only known by the COTS software.

2.2 Architectural Challenges to Delta Extraction

Architectural features of the source systems also make the delta extraction process complex. These features are not an issue in the source systems because they do not require a consolidated view of information as does a data warehouse. Current delta extraction products such as Informatica PowerCapture and ETI Extract Toolkit [10, 15] are unable to handle these challenges and have to be heavily customized to be applicable. Following are some of the features that contribute complexity to delta extraction methods.

- *Dynamic Replication*: When multiple representations exist for the same information in source systems, an extraction method should be able to extract an authoritative value as the delta that needs to be integrated into a data warehouse. Additionally, these multiple representations may not exist as exact replica in the system. Reconciliation of multiple representations can be performed during the later cleansing and scrubbing phases. However, the farther away from the data sources, the less knowledge there is about the semantics of replications, and more challenging the reconciliation process becomes. Solutions based on database replication products [16, 24, 25] often do not apply in large enterprise systems because the COTS software control the replication logic and the DBMSs are essentially unaware of the replication.
- *Distribution*: Data is often partitioned for performance, locality and other reasons. Geographical separation between related data in the source systems requires that the delta extraction methods be able to handle distribution and keep related deltas coherent.
- *Heterogeneity*: Variations between database products making up the source systems may require different extraction methods to be used and may require collaboration among these extraction methods.

2.3 Extraction Mechanism Requirements

Keeping in mind a reason for creation of data warehouses, all the extraction challenges should be overcome with minimal performance impact to the source systems. Here are the requirements a practical delta extraction method must address adequately:

1. *Performance Impact on Sources*: One of the primary purposes of a data warehouse is to take processing load off operational systems. If an extraction method imposes an excessive performance penalty on the operational source system, the method quickly becomes unattractive. To further lessen the performance impact on the source systems, the extraction method must capture sufficient

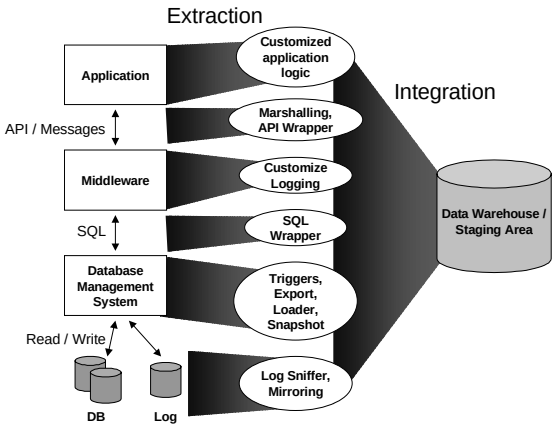

Figure 1: Reference architecture

information so that the integration process will not require referring back to the source systems (in others words, the extraction method should support a self-maintainable [11] integration process).

2. *Ease of Capture of Reconciled Delta*: An extraction process should be able to capture authoritative deltas at the source systems by masking the architectural features of replication, distribution, heterogeneity, etc.

3. *No Modification to Existing Applications*: Most data warehouses are created well after the operational systems have been developed. An extraction process should be able to get deltas from the operational systems without requiring modifications to existing applications.

Current off-the-shelf extraction products cannot address all these business requirements. Some extraction products view source systems as logical collections of data and expect the source systems to provide interfaces through which authoritative values can be extracted. Many COTS products do not support such interfaces. In cases where such interfaces are supported, performance consideration is a limiting factor. Additionally, if the schema of the source systems and the data warehouse vary significantly, utilities built using current extraction products require considerable customization and maintenance work.

2.4 Reference Architecture for Delta Extraction

Considering the multi-tiered source systems of Figure 1, there are several levels in the architecture from which deltas can be extracted - below the database level (by the DBMS vendor), between the database and the COTS software (by the COTS software vendor), between the multiple COTS software (by the integration technology architects), between the applications and the COTS software (by the application developers), and between the users and applications (by application interface developers). Extraction at the graphical user interface level using techniques like screen scraping are impractical and hence are not discussed. Though highly efficient in some cases, we do not discuss hardware based methods such as disk mirroring.

Deltas can be extracted by the DBMS vendor using a variety of techniques. The extraction mechanism at this level will not be aware of data distribution and hence sophisticated reconciliation techniques to remove redundancy (due to replication) have to be implemented. One popular method here is to the extract deltas from database archive logs using log APIs. Deltas may also be extracted inside the database using event based mechanisms such as database triggers. Other methods include querying the database to compute the deltas. Most commercial implementations of delta extractions use such techniques. We will discuss delta extraction mechanisms based on these methods in Section 3.

Deltas can be extracted in the interface between the COTS software and the DBMS. This can be implemented either by the COTS software which actively maintains a delta log or by a third-party that provides a wrapper[1] like middleware to trap and record the deltas. Deltas captured by this mechanism may be in form of SQL statements. If COTS software vendors implement the capture mechanism, it is likely that the extraction mechanism will be aware of the data distribution and hence reconciliation will not be needed. The main disadvantage with this approach is that it requires the COTS software be modified to enable the delta capture. However, if the wrapper approach is adopted, no modifications to the COTS software are required. From our experience, we have found that COTS software vendors do not resist modification to their software because it widens the appeal of their product and opens up their product to new market areas (data warehouse). We discuss one such method called Op-Delta in Section 4.

Deltas can also be captured in the integration infrastructure (CORBA, DCE, and DCOM) between the COTS software. The message channel exit points can be tapped to capture the deltas. Deltas here will be (most likely) in the form of high-level object method calls, instead of SQL statements. Since data distribution is transparent to applications, reconciliation for redundancy removal is not needed. If implemented at this level, no changes to existing applications are required. However, this implementation assumes that all business transactions cross the integration layer and the existence of global transaction management technology in the architecture to guarantee the atomicity of inter-COTS software transactions.

Finally, deltas can be captured at the interface between the user applications and the COTS software, by having the applications capture them. Deltas here are likely to be in the form of high-level object methods. A customized mapping mechanism is now required to map each object's methods (including semantics) into an equivalent method applicable to the data warehouse - something that may not be always feasible.

[1] The wrapper approach is also advocated in [35].

Table 1: Database deltas dump and load techniques

Method \ Delta Size	100M	200M	400M	600M	800M	1000M
Export	3min	13min	23min	37min	56min	1 hr 32 min
Import	28min	1 hr 7 min	3 hr 11 min	5 hr 21 min	6 hr 11 min	9 hr 59 min
DBMS Loader	20min	34min	1 hr 8 min	1 hr 40 min	2 hr 28 min	2 hr 58 min

3 Delta Extraction

As mentioned in Section 2, most commercial products extract deltas directly from the databases. We discuss four such commonly used methods and for a more detailed narration we refer the reader to [29].

1. Time stamps: If a source system supports time stamps naturally, queries can be executed to obtain deltas within a time period (*SELECT * from PARTS where last_modified_date > 12/5/99*). The results can be stored in a file and shipped to a data warehouse for incremental maintenance.
2. Differential Snapshots [20]: In some systems, snapshots [3] of source databases may be the only allowed operation. In such systems, deltas can be computed by obtaining a dump of the current state and comparing it with a previously stored snapshot of the source systems. Labio and Garcia-Molina [18] analytically analyze efficient ways by which snapshot differentials may be computed.
3. Triggers: If a source system permits, database trigger is a way by which deltas can be captured. The triggered data can be written out to a table and the table can be dumped to obtain deltas.
4. Log Extraction: The redo logs of DBMSs contain data affected by committed transactions since the last checkpoint. If "archiving" is turned on in the database, the redo logs are not recycled at checkpoint time and continue to accumulate the redo data. These logs contain deltas and can be shipped to another similar database and applied using tools based on the DBMS recovery managers.

With the exception of log based extraction, each method requires an additional step to move the deltas out of a source system. In general, the output of an extraction process can be to an operating system file or to a local database table.

Output to File: If a method captures deltas in an operating system file, then no additional step is required.

Output to Table: If a method writes captured deltas to a table then an additional step of extracting out the delta table is required. The table is most likely to be within the source system because writing to an external database is prohibitively expensive (as we will discuss in Section 3.1.3). Other reasons could be because some DBMSs do not allow such an operation and due to heterogeneity between the databases. If deltas are stored in a local table, then DBMSs provided Export utilities may be used to extract the table efficiently. The Export utilities will dump files in a proprietary format which can only be imported using the DBMS' Import utility into the same DBMS product. Alternatively, an approach similar to the time stamp based method can be used to get an ASCII dump file of the delta table that can subsequently be loaded into a data warehouse using ASCII load utilities.

We show in Table 1 the results of our experimentation with the Export utility of a commercial DBMS. Though it is not a part of the delta extraction process, the use of the Export utility requires the use of the corresponding Import utility which we also include in Table 1. For extraction methods that output to an operating system file, we show in Table 1 the time associated with using a DBMS ASCII Loader utility. Except for the fact that these dump and load techniques may need to be used after deltas are captured, by themselves they are not central to our delta extraction discussion. The sizes (of deltas) and associated dump/load time shown in Table 1 are presented for comparison purposes only and the intention is not for the results to be extrapolated to production strength systems. The DBMS Loader technique loads ASCII data directly into database blocks. The Import utility on the other hand fills its own internal pages and when the pages overflow they write the data into the database. The extra I/O is evident from the times taken by the Import utility in comparison with the DBMS Loader in Table 1. We refer readers to [29] for a detailed narration of the experiments of Table 1.

3.1 Analysis

We conducted experiments to study the behaviour of the extraction methods. Beyond requiring the use of database dump and load techniques, the differential snapshot method needs further computation to obtain deltas and is prohibitively resource intensive. The log based extraction technique is not tested since it has no direct impact on user transactions - since redo logs are being captured anyway and shipping of a log does not directly impact user transactions on the source systems. Hence numbers are not provided for differential snapshot and log based extraction methods. The time stamp based experiments write out the results (complete record) of the extraction to a file and to a local table. The trigger experiments use row-level triggers and assume that the complete record is written to a delta table in the same database where the trigger executes. The experiments were conducted on a 300MHz NT server with 128MRAM hosting a commercial DBMS. All the records in our experiments were 100 bytes long. The purpose of our experiments and the subsequent

Table 2: Time stamp based delta extraction

Method \ Delta Size	100M	200M	400M	600M	800M	1G
File output	17min	26min	43min	59min	1hr 19min	1hr 36min
Table output	29 min	55 min	1hr 45min	2hr 40min	3hr 29min	4hr 24min
Table output + Export	32 min	1hr 8min	2hr 8min	3hr 17min	4hr 25min	5hr 56min

Table 3: Total time taken to extract and load deltas (excluding network, cleanup, and integration time)

Method \ Delta Size	100M	200M	400M	600M	800M	1G
Time Stamp file output + DBMS Loader	37min	1hr	1hr 51min	2hr 39min	3hr 47min	4hr 34min
Time Stamp table output + Export + Import	1hr	2hr 15min	5hr 19min	8hr 38min	10hr 36min	15hr 55min

analyses are to show relative scale of the delta extraction methods and it is not our intention to extrapolate the results to production strength systems. In this section, we discuss the performance effects and other implications of extracting deltas from the database level of the architecture in Figure 1.

3.1.1 Time Stamp Based Extraction

The time stamp based methods require table scans unless an index is defined on the time stamp attribute. Additionally, indices may not be used by the query optimizer if the deltas form a significant portion of the table. Table 2 shows the performance cost of extracting various sizes of delta using this method from a 1G size table consisting of 10 million 100-byte records. If the output is written to a database table then the delta tables have to be exported to make the deltas available outside the database. The total time for extraction and the export action is also shown in Table 2. Since the exported data will require that the corresponding Import utility be used at the data warehouse or at a staging area we added the time taken by the import action and compare it against the total time taken by outputting to a file and using a DBMS Loader utility on it in Table 3. As mentioned earlier, use of the Export/Import utilities require that the same database product exist in the source and in the data warehouse (or a staging area) - a very restrictive constraint. Additionally, writing out the deltas to file offers other opportunities for off-line processing of the deltas.

The time stamp based extraction technique in general is handicapped by not being able to capture state changes in the source systems (only detectable changes are the final changes in the database just prior to the extraction process). The time stamp based method is clearly only applicable for extraction from source systems that natively support time stamps and have little change activity.

3.1.2 Differential Snapshots

Labio and Garcia-Molina list various algorithms [18] to obtain snapshot deltas and compare the performance of the algorithms analytically. As mentioned earlier, the differential snapshot method requires the use of a database dump and load technique on the source systems and a further computation to compare the before and after snapshots to obtain the delta. As with the time stamp based method, the differential snapshot method suffers from being able to only capture final changes to the data just before a snapshot is obtained. Given these limitations and in comparison to the other three methods discussed, the differential snapshot method may be applicable only when the source systems allow no other way of capturing deltas.

3.1.3 Trigger Based Extraction

Triggers are widely advertised as an easy way to extract deltas from a database. Some commercial delta extraction products are based on triggers. As with the time stamp based method, the trigger method needs to use one of the database dump techniques for exporting the delta (table) out of the operational source systems or execute a stored procedure within the database (putting additional load on the database).

Trigger based extraction is very simple to implement and requires no change to applications at the source system. Since the method is event based, it is attractive if capturing every state change in source systems is a requirement for the data warehouse. Triggers execute in the same transaction context as the triggering event and so there is no inconsistency between the triggering event and the triggered action. Source systems' autonomy is a dominant characteristic that dictates whether triggers are allowed to be implemented. Besides this, we find other significant reasons that may invalidate the use of triggers as an extraction method.

As discussed in Section 2, if source systems are distributed and the business process does not execute as one atomic transaction global non-serializable executions are possible. Reconciling deltas from each source system to obtain the correct values that can be applied to a data warehouse is a significant challenge. A (impractical) mechanism is to capture a global transaction identifier along with the deltas that guarantees uniqueness across the systems. If there is replication in the source systems, reconciliation of the

captured (replicated) delta is difficult. Additionally, there are performance impacts to consider.

The advertised trigger overhead is between 5-13%. In our experiment, we found that this only applies to transactions that only update/delete a single record. In reality, typical OLTP transactions may access more than one record. We varied the transaction size in our experiments and present the results in Figure 2.

Figure 2 shows the performance effects of inserts, deletes and updates on the source table without and with row-level triggers defined on them. The parameter of the experiments is the number of records being inserted/deleted/updated per transaction and the response time of each transaction is measured. When performing update/delete transactions, the size of the source table remains constant at 100,000 rows. The deltas are captured as follows: for insertions into the source tables, the new values being inserted are captured; for updates, the old and new values are captured; and, for deletions, the old values are captured.

For each inserted record, the trigger will perform one insertion, so the overhead of the trigger is a constant (80-100%) with respect to all size of transactions. The overhead includes an additional triggered insertion and overhead of triggering. Each update transaction performs a table scan to update desired records and triggers two insertions (before and after image). The overhead of update triggers is increasing because the per-row update cost is diminishing when transaction size increases, due to internal DBMS optimization. Each delete transaction performs a table scan to delete the selected records and triggers one insertion.

All the triggers in our experiments wrote to tables within the same database. We also ran tests where we wrote the results of a triggering action into a remote database located in the same 10Mb/sec. switched LAN. In our experience, capturing the changes directly to an external system - be it a staging area or the warehouse itself - is in the order of ten to hundred times more expensive (in terms of response time) depending on networking and workload characteristics. In fact, the cost is one order magnitude higher even if the staging area is located in a different database at the same machine. The higher cost is attributable to the penalty for establishing database connections, extra inter-process communications, and, I/O and memory contentions between the databases.

3.1.4 Log Based Extraction

The use of archive logs to extract deltas is attractive from a performance perspective having little performance impact on the source systems (despite the overheads associated with turning on archiving). The shipping of the deltas (logs) to a remote site is off the critical path of the source system's applications. This contributes further in lessening the performance impact of obtaining deltas. Additionally, existing applications are not required to be modified. Many commercial delta extraction and replication products use this method.

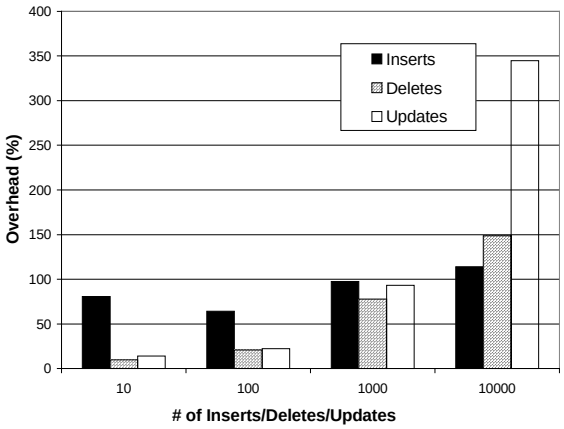

Figure 2: Insert/Delete/Update trigger overhead

The challenges associated with using log-based methods are as follows. Log formats are often proprietary and DBMS vendors do not always support APIs to their logs. Log formats almost always change with major product releases and sometimes even with minor releases. If the formats are not published, vendors do not have a need for logs to be compatible across versions of their products and certainly never across DBMS products. These have one of the following effects on the extraction process: (1) the same DBMS product as the source system must be the first recipient of the delta, or (2) a third party integration product capable of reading various logs versions and supporting integration across them is required. Both options have significant monetary cost for owning and maintaining additional software licenses. Due to the additional step involved, these options extend the duration of the end-to-end incremental maintenance process. A further complication is that log based techniques depend on the schema of the source and the destination to match exactly. This is the reason why the most common usage of this technique are found in database recovery, replication, and database hot-standby. Since many database products implement physiological logging [12], having API access to the database logs is often not sufficient for the extraction process. Additional complexity arises if the warehouse requires that incremental state changes be recorded, the log recipient must be able to version the data. Furthermore, logs are low level and sometimes complicate the extraction process. If heterogeneity, replication, and distribution are involved log based extraction approaches become even more complex.

4 Discussion

The time stamp based and trigger based methods allow for transformation of deltas during the extraction process. The trigger and log based methods can capture state changes. The log based method poses minimal performance impact on the source systems. As discussed in Section 3, each of the methods also have significant

shortcomings in their approach to be used for delta extraction. It may be concluded that none of the current methods as an aggregate is attractive given the requirements posed in Section 2. In this section, we present a technique that we have been developing. We compare it against the delta extraction techniques discussed in Section 3. We present preliminary results from our prototype implementation and compare it against the other techniques. We also present comparative analysis of how deltas extracted using our technique and the techniques of Section 3 can be integrated into a data warehouse.

4.1 Op-Delta Based Extraction

Based on the requirements discussed in Section 2, we have been investigating a technique called Op-Delta [7, 8]. Op-Delta is a log based method that captures deltas as operations that caused the changes. In contrast, the methods discussed in Section 3 capture the values of delta (for clarity we refer to these as value delta). The statement: *"UPDATE status=`revised' from PARTS where last_modified_date > 11/15/99"* may generate a value delta in the size of several thousands records, half of those are the before image of the affected records and another half are the after image of the affected records. However, the SQL statement itself is already an Op-Delta in the size of about 70 bytes. The Op-Delta method is similar to applying the redo log to a database. However, the data warehouse schema is typically an aggregation of the source database schema unlike a recovering database, so appropriate transformations need to be applied to the redo logs.

In [8], we presented algorithms to maintain SPJ views at data warehouses based on Op-delta. In addition, we have identified sufficient conditions that Op-Delta alone is enough to refresh the data warehouse (i.e., self-maintainability with respect to Op-Delta), and for some cases, a hybrid between a partial value delta (the before image portion only) and the Op-Delta is necessary to refresh the data warehouse in a self-maintainable manner. Briefly here are some of the advantages of Op-Delta over value delta (and the other extraction methods discussed).

For deletions and updates at sources, Op-Delta can reduce the "delta" volume and hence the message traffic from source to the data warehouse significantly. This is because the size of an Op-Delta for deletion and update is independent to the size of the transaction, which is measured by the number of affected records. In contrast, the size of value delta is proportional to the size of a transaction. For insertion at sources, the Op-Delta has the same space efficiency as the value delta because the size of a single inserted record is more or less equal to the size of that insert SQL statement.

Op-Delta maintains the original source transaction boundaries. Since, each Op-Delta can be applied as a self-contained transaction to the data warehouse concurrently with the data warehouse queries [8], as demonstrated by our prototype, a data warehouse outage is not required for incremental maintenance. In contrast, value delta methods lose the transaction context at the sources and are need to be applied as an indivisible batch [34]. Maintaining original source transaction boundary is significant to improve the efficiency of update maintenance at a data warehouse. Since the transaction context of value delta is lost, each original transaction will be captured by one or more value delta records and each of which will be translated into a single SQL statements. For instance, each original insert transaction will be captured as one value delta record which will be translated into one insert SQL statement in the data warehouse. However, one original delete transaction that deleted *x* records will be captured as *x* value delta records (the before image of the deletion), and will be finally translated into *x* SQL delete statement at the data warehouse. Likewise, each original update transaction that updated x records will be finally translated into *x* SQL delete statements (from before image) and *x* SQL insert statements (from after image).

We performed experiments to quantify the efficiency of Op-Delta against value delta during data warehouse maintenance. The experiments varied the size of transaction (number of affected records) and measured the average response time for maintaining insertion, deletion, and update at the data warehouse. The results show that the response time of maintaining insertion by Op-Delta and value delta is the same as they both map one original insert transaction into one insert transaction in the data warehouse. For deletions, the data warehouse maintenance window using Op-Delta is on average 31.8% shorter than that of using value delta with transaction size ranging from 10 records per transaction to 10,000 records per transaction. For updates, the maintenance window using Op-Delta is on average 69.7% shorter than that using value delta with transaction size ranging from 10 records per transaction to 10,000 records per transaction.

Our experiments show that Op-Delta can shrink the data warehouse update maintenance window significantly. As mentioned in [19], another possibility to maintain data warehouses is to go "on-line" without shutting down the data warehouse. Op-Delta captures the original transaction context and hence can interleave with OLAP queries without impacting the integrity of the query result. In addition, the efficiency of Op-Delta update maintenance at the data warehouse significantly reduces the resource competition against the OLAP queries.

Finally, in COTS based integrated systems, it may not be feasible to implement low level value delta methods. For example, if the system replicates data, a trigger based or database log based approach will extract several instances of the same data. To obtain one authoritative copy of the data, the different instances now have to be reconciled before being transported to the data warehouse. Alternatively, Op-Delta offers an opportunity

to capture changes at the business transaction level where there is only one authoritative representation of the fact rather than at the lower database level where there is replicated data. Additionally, if the databases are encapsulated Op-Delta may be the only possible mean of extracting delta.

The extraction mechanisms discussed in the reference architecture of Section 1.4 can also be implemented to extract both Op-Delta and value delta. For instance, trigger is a widely suggested method to extract value deltas and is implemented by the DBMS vendors. However, extracting Op-Delta has less performance impact at the source systems. A value delta is composed of the before image and the after image of a state change. At all levels, in order to capture value deltas, the extraction mechanism has to (1) extract the before image, (2) execute the state change operation, (3) extract the after image, and all three steps have to be bracketed in one transaction. Note that Step (1) and (3) are overheads imposed on the operation in order to capture its value delta. As discussed in [8], in some cases, the description of the operation is the only information needed to be captured in an Op-Delta, and in the worst case, the operation description has to be augmented with the before image of the state change. Hence, capturing an Op-Delta has less impact on the original operation than capturing value deltas since the after image, and in some cases the before image too, of a state change are not captured. In [8] we also developed a set of transformation rules to directly apply the Op-Delta to various schema in data warehouses. In comparison, log-based value delta extraction methods always assume the destination schema is same as the source schema.

4.2 Experimental Results

We performed experiments to study the cost to extract Op-Delta from the source system using the same experimental environment described in Section 3. We decided to capture the Op-Delta right before it is submitted to the DBMS to simulate the capture mechanism that will implemented by COTS software or by the wrapper approach. The experiments varied the size of transaction (ranging from 10 to 10,000) which is the number of affected records, each of which is 100 bytes in size. The first set of experiments capture the Op-Delta and stored it transactionally into a database table. The idea is to perform head-to-head comparison with the trigger based extraction method which is widely implemented. Figure 3 shows that the average overhead of capturing an insertion Op-Delta is 66.47%, whereas for deletion and update the average overheads are 2.48% and 3.68% respectively. The overhead of capturing insertion Op-Delta is comparable with using trigger. Although they both capture the same volume of information, the DBMS trigger mechanism incurs additional cost comparing to an external SQL insert statement (to insert the Op-Delta into the database table). The overhead of capturing deletion and update Op-Delta is much smaller than that of using trigger method (overhead ranges from 9-344% in Figure 2) because the size of Op-Delta is much smaller than the equivalent value delta as the transaction size increases.

The second set of experiments write the Op-Delta log into a file and hence reduces the overhead of transaction management. Table 4 shows the response time of the original transactions with database Op-Delta log and file Op-Delta log are enabled with transaction size ranging from 10 to 10,000. Using a file log significantly improves the original transaction response time as excessive database overheads on query processing and transaction management are reduced. If writing the Op-Delta log does not need to be transactional, using a file log could be attractive.

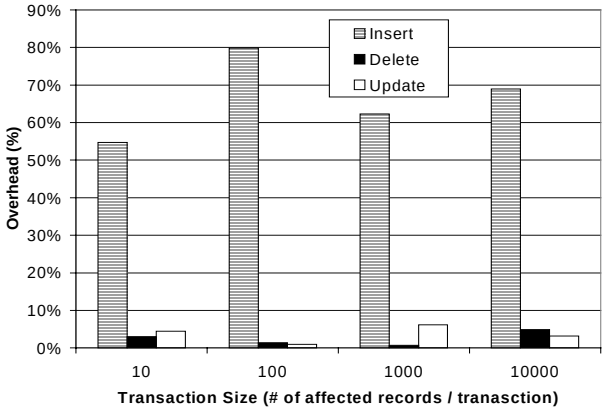

Figure 3: Op-Delta extraction overhead on insert/delete/update

5 Summary

Research and commercial products assume that value deltas normally in the form as a differential file are available for data warehouse maintenance purposes. From our participation in several large-scale data warehouse deployments in Boeing, we observe that this assumption is not valid. Additionally even when value deltas are accessible, the means of obtaining them pose significant performance and other overheads on the source systems. The contributions of this paper are the summarization of currently available methods to extract deltas and an analysis of the each method's strengths and weakness. We presented a new method called Op-Delta that better supports delta extraction from COTS software based integrated systems. Op-Delta supports a more efficient capture of deltas than do value delta based techniques, minimizes the volume of data that needs to be transported from the source systems to a data warehouse, and supports a more efficient integration of the deltas into a data warehouse. Additionally, Op-Delta does not require a data warehouse to be unavailable during the maintenance process as do the value delta based processes.

Each of the currently available extraction methods have their advantages and some notable disadvantages

Table 4 : Response time (ms) - DB log vs file log

Trxn Size	Insert (DBLog)	Insert (FileLog)	Delete (DBLog)	Delete (FileLog)	Update (DBLog)	Update (FileLog)
10	117	75	80	74	69	68
100	862	519	428	427	272	271
1,000	8,081	5,379	4,046	4,004	2,672	2,638
10,000	81,840	55,364	43,962	41,416	27,233	26,571

that make them impractical in many contexts. Through qualitative discussions and experimental results we have highlighted the strengths and weaknesses of each of the methods. Time stamp based methods are most suitable when state changes at the source systems do not need to be captured in the data warehouse and when the amount of delta at the source systems is very small. The method is more flexible than some of the other methods discussed since it allows restricting, sub-setting, and when appropriate aggregating deltas during the extraction process. However, when the volume of deltas is large the performance overhead on the source systems associated with capturing the deltas can be very significant as we have shown. Differential snapshot based methods are applicable when source systems snapshots are only available in lieu of deltas. The performance impact on the source system is the most when compared with each of the other delta extraction methods considered. Value log based methods by far have the least performance impact on the source systems: they do not have a direct impact on the source systems' transactions because redo logs are being captured anyway by the DBMS. Additionally, they can capture state changes in the source systems and do not require any modification to existing applications. The weakness of value log based methods are that it requires the same DBMS product with the same configuration in the source systems and in the data warehouse (or in a staging area), that it cannot deal with the architectural challenges we discussed in Section 2, and that it is very inflexible in that it can only fully re-create a database much like a recovery manager does. Trigger based methods are attractive when capturing state changes is necessary. Just as the value log based methods, they are simple to implement, do not require any modifications to the user applications and the COTS software, and are flexible in allowing portions of deltas to be captured. However, as we have shown in our experiments the performance impact of triggers to capture deltas can be prohibitive. Since triggers execute in the same transaction scope as user transactions their performance impact is directly felt by the user applications. Furthermore, if a trigger fails it also aborts the user transaction.

A common thread among the current value delta based extraction methods cause the data warehouse to be unavailable for extended duration during the incremental maintenance process. Since each of the value delta based methods we have discussed cannot capture transaction semantics, the incremental maintenance process cannot exploit that information. As a result, the maintenance processes require the data warehouse be unavailable during the integration process or cannot guarantee the coherency of the information stored within the data warehouse during the incremental maintenance process. Addressing the architectural hurdles discussed earlier to obtain an authoritative data value for a change activity is another disadvantage these low level delta extraction methods have difficulty addressing.

Our work on Op-Delta at Boeing seeks to overcome these disadvantages. Op-Delta can be applied concurrently with existing user queries to the data warehouses thereby not requiring the data warehouse be shutdown during the incremental maintenance process. Additionally, Op-Delta is a log based approach and hence minimizes the impact on the performance of user transactions at the source systems. It captures deltas in a much more compact manner than do value delta based methods, thereby, reducing the network transmission costs of the data transfer from the source systems to the data warehouse. It is capable of masking out all the architectural hurdles of replication, distribution, and heterogeneity making it easier to obtain authoritative values for deltas. On the other hand, Op-Delta may require that COTS software need to be modified to enable Op-Delta capture. However, if a COTS software maintains the same APIs and the semantics of the operations behind the APIs, user applications should not see any functional effect of the COTS software modifications and will not require re-coding. As discussed earlier, we have found that COTS software vendors do not resist this approach because it opens up their product to better support data warehouses. In addition, Op-delta can be extracted by implementing a wrapper around the information source.

6 References

[1] D. Agrawal, A.E. Abbadi, A. Singh, and T. Yurek. *Efficient View Maintenance at Data Warehouses.* In SIGMOD'97, pp. 417-427, Tucson, Arizona, May 1997.

[2] Arbor Essbase 5 white papers and product demonstration. Arbor Software Corp. 1998.

[3] M. Adiba and B. Lindsay. *Database Snapshots*. Proceedings of VLDB '80.

[4] P. Bernstein and E. Newcomer. Principles of Transaction Processing, Morgan-Kaufmann, 1997.

[5] K. Beyer and R. Ramakrishnan. *Bottom-Up Computation of Sparse and Iceberg CUBESs*, Proceedings of ACM SIGMOD Conference, Philadelphia, 1999.

[6] University of Darmstadt Database Research Group. All about Data Warehousing, http://www.informatik.tu-

darmstadt.de/DVS1/staff/wu/dw.html

[7] L. Do, P. Drew, W. Jin, V. Jumani, D. VanRossum. *Issues in Developing Very Large Data Warehouses*. Proceedings of VLDB, New York, 1998.

[8] L. Do, P. Ram, A. Kawaguchi, and C. Pu. *Op-Delta: An Efficient Approach to Incremental Data Warehouses Maintenance*. Boeing Phantom Works Technical Report 99-002.

[9] L. Do, P. Ram, and P. Drew, *The Need for Asynchronous Distributed Transactions*, in Proceedings of ACM SIGMOD Conference, Philadelphia, 1999.

[10] http://www.eti.com/products/toolkit.html

[11] A. Gupta, H.V. Jagadish, and I.S. Mumick. *Data Integration using Self-maintainable Views*. EDBT'96.

[12] J. Gray and A. Reuter, Transaction Processing: Concepts and Techniques, Morgan-Kaufmann, 1993.

[13] N. Huyn. *Multiple-view Self-maintenance in Data Warehousing Environments*, Proceedings of 23rd VLDB, 1997.

[14] R. Hull and G. Zhou. *A Framework for Supporting Data Integration Using the Materialized and Virtual Systems*. Ed. N.W. Patton, M.H. Willams, Springer - Berlin, pp. 23-39, 1993.

[15] Informatica. http://www.informatica.com/powercapture.html and product demo.

[16] Guide to Informix Enterprise Replication. http://www.informix.com/answers/oldsite/answers/pubs/pdf/7919.pdf 1997.

[17] H.V. Jagadish, P.P.S. Narayan, S. Seshadri, and S. Sudarshan. *Incremental Organization for Data Recording and Warehousing*, in Proc. of 23rd VLDB Conference, 1997.

[18] W.J. Labio and H. Garcia-Molina. *Efficient Snapshot Differential Algorithms for Data Warehousing*. Proceedings of VLDB '96.

[19] W.J. Labio, R. Yerneni and H. Garcia-Molina. *Shrinking the Warehouse Update Window*, Proceedings of ACM SIGMOD Conference, Philadelphia, 1999.

[20] B. Lindsay, L. Haas, C. Mohan, H. Pirahesh, and P. Wilms. *A Snapshot Differential Refresh Algorithm*. Proceedings of SIGMOD '86.

[21] L. Liu, C. Pu, R. Barga, and T. Zhou. *Differential Evaluation of Continual Queries*. In IEEE Proc. of the 16th Intl. Conf. on Distributed Computing Systems, Hong Kong, 1996.

[22] A. Mendelzon. Data Warehousing and OLAP: A Research-Oriented Bibliography. University of Toronto. http://www.cs.toronto.edu/~mendel/dwbib.html.

[23] Oracle Express Database Administration Guide, Rel 2.0.4. Oracle Corp. 1997.

[24] Oracle8 Advanced Replication Product Documentation.

[25] Praxis OmniReplicator. http://www.praxisint.com/OmniEnterprise/omnirepl.asp.

[26] D. Quass, A. Gupta, I.S. Mumick, and J. Widom. *Making Views Self-Maintainable for Data Warehousing*. Proceedings of PDIS'96, Miami Beach, Florida, 1996.

[27] D. Quass and J. Widom. *On-Line Warehouse View Maintenance*, in Proceedings of SIGMOD 1997.

[28] Red Brick Warehouse Table Management Utility Reference Guide. Ver. 5.0. Red Brick Systems, Inc. 1996.

[29] P. Ram and L. Do. *Your Warehouse is Empty: Initial Load and Incremental Update*. Boeing AR&T Tech-Report SSGTECH-98-027.

[30] P. Ram, L. Do and P. Drew, *Distributed Transactions in Practice*, in ACM SIGMOD Record, 28(3) September 1999.

[31] P. Ram, L. Do, P. Drew and T. Zhou, *Object Transaction Service: Experiences and Issues*, Proceedings of the Symposium on Distributed Objects and Applications (DOA '99) held in conjunction with VLDB, Scotland, September, 1999.

[32] SAS Data Warehouse Administration white papers and product demo. SAS Institute, 1998.

[33] A. Sheth and J. Larson. *Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases*, ACM Computing Surveys, 22 (3), September 1990.

[34] M.C. Wu and A. P. Buchmann. *Research Issues in Data Warehousing*, In Datenbanksysteme in Buro, Technik und Wissenschaft, (Proc. BTW'97 Conference), Springer, March 1997.

[35] J.L. Wiener, H. Gupta, W.J. Labio, Y. Zhuge, H. Garcia-Molina, and J. Widom. *A System Prototype for Warehouse View Maintenance*, in Proceedings of Workshop on Materialized Views, June, 1996.

[36] G. Zhou, R. Hull, and R. King. *Generating Data Integration Mediators that use Materialization*. Journal of Intelligent Information Systems, 6(2-3):199-221, June 1996.

[37] Y. Zhuge, H. Garcia-Molina, J. Hammer, and J. Widom. *View Maintenance in a Warehousing Environment*, in Proceedings of SIGMOD 1995.

[38] Y. Zhuge, H. Garcia-Molina, and J.L. Wiener. *The Strobe Algorithms for Multi-Source Warehouse Consistency*, in Proceedings of PDIS, 1996.