[14] F. Cellier, A. Nebot, F. Mugica, and A. de Albornoz, “Combined qualitative/quantitative simulation models of continuous-time processes using fuzzy inductive reasoning techniques,” *Int. J. Gen. Syst.*, vol. 24, no. 1–2, pp. 95–116, 1996.
[15] P. Fishwick and P. Luker, *Qualitative Simulation Modeling and Analysis*. New York: Springer-Verlag, 1991.
[16] G. Klir, “Aspects of uncertainty in qualitative systems modeling,” in *Qualitative Simulation Modeling and Analysis*, P. Fishwick and P. Luker, Eds. New York: Springer-Verlag, 1991, pp. 24–50.
[17] S. Case, Q. Shen, R. Banares-Alcantara, and J. Ponton, “Detecting inverse responses in chemical processes with qualitative simulation,” in *Proc. 11th Int. Workshop Qualitative Reasoning*, 1997, pp. 249–255.
[18] P. Struss, “Mathematical aspects of qualitative reasoning,” *Artif. Intell. Eng.*, vol. 3, pp. 156–169, 1988.
[19] I. Miguel and Q. Shen, “Solution techniques for constraint satisfaction problems (part 1: Foundations and part 2: Advanced techniques),” *Artif. Intell. Rev.*, vol. 15, no. 4, pp. 231–245, 269–293, 2001.
[20] P. Fouche and B. Kuipers, “Reasoning about energy in qualitative reasoning.,” *IEEE Trans. Syst., Man, Cybern.*, vol. 22, no. 1, pp. 47–63, Jan. 1992.
[21] P. Struss, “Global filters for qualitative behaviors,” in *Proc. 7th Nat. Conf. Artif. Intell.*, 1988, pp. 275–279.
[22] J. Keppens and Q. Shen, “On compositional modeling,” *Knowledge Eng. Rev.*, vol. 16, no. 2, pp. 157–200, 2001.
[23] B. Williams, “The use of continuity in a qualitative physics,” in *Proc. 3rd Nat. Conf. Artif. Intell.*, 1984, pp. 350–354.
[24] K. Forbus, “Interpreting measurements of physical systems,” in *Proc. 5th Nat. Conf. Artif. Intell.*, 1986, pp. 113–117.
[25] D. Dvorak and B. Kuipers, “Process monitoring and diagnosis,” *IEEE Expert*, vol. 6, no. 3, pp. 67–74, 1991.
[26] E. Manders, G. Biswas, P. Mosterman, L. Barford, and J. Barnett, “Signal interpretation for monitoring and diagnosis: a cooling system testbed,” *IEEE Trans. Instrum. Meas.*, vol. 49, no. 3, pp. 503–509, Mar. 2000.
[27] Q. Shen and R. Leitch, “Diagnosing continuous systems with qualitative dynamic models,” *Artif. Intell. Eng.*, vol. 9, no. 2, pp. 107–125, 1995.
[28] R. Rosenberg and D. Karnopp, *Introduction to Physical System Dynamics*. New York: McGraw-Hill, 1983.
[29] U. Keller, “Qualitative model reference adaptive control (QMRAC),” Ph.D. dissertation, Dept. Comput. Elect. Eng., Heriot Watt Univ., Edinburgh, UK, 1999.

Multiagent Immediate Incremental View Maintenance for Data Warehouses

Gary C. H. Yeung and William A. Gruver

***Abstract*—Data warehouse systems typically designate downtime for view maintenance, ranging from tens of minutes to hours depending on the system size. In this paper, we develop a multiagent system that achieves immediate incremental view maintenance (IIVM) for continuous updating of data warehouse views. We describe an IIVM system that processes updates as transactions are executed at the underlying data sources to eliminate view maintenance downtime for the data warehouse—a crucial requirement for internet applications. The use of a multiagent framework provides considerable process speed improvement when compared with other IIVM systems. Since agents are used to delegate the data sources and warehouse views, it is easy to reorganize the components of the system. Through the use of cooperative agents, the data consistency of IIVM can be easily maintained. The test results from this research show that the proposed system increases the availability of the data warehouse while preserving a stringent requirement on data consistency.**

***Index Terms*—Data warehouse, multiagent, view maintenance.**

I. Introduction

View maintenance requires updating the materialized views in a database system as changes are made to the underlying data. View maintenance is well understood in conventional transaction database systems. Most prior research in view maintenance has been concerned with deferred incremental view maintenance (DIVM), in which updates to the underlying data are logged and applied to modify the materialized views collectively during system downtime. DIVM, the primary view maintenance method adopted by most commercial data warehousing products, assumes that the data warehouse has a convenient system downtime. For data warehouses that provide global access, however, downtime may not be acceptable. The time required for view maintenance requirement is a major limitation on the size of a data warehouse.

Data warehouse views may also be updated by the use of immediate incremental view maintenance (IIVM). In this technique, changes to the underlying data are applied immediately and individually to the materialized views. Potential data inconsistency due to asynchronous messaging, however, constrains its usage in commercial systems. In this study, we propose a multiagent framework for performing IIVM in data warehousing with parallel processing.

Hanson [1] conducted one of the earliest studies of DIVM in which differential tables were maintained on base tables that contain the suspended updates that have not been applied to the database state. Colby [2] applied base logs and differential tables for periodic update of views, a concept that is similar to taking snapshots from every state of change in the base tables. Mumick [3] improved Colby’s method by storing changes of base tables in a Summary-Delta table in which the information is updated to the summary tables during off-hours. Gupta [4] proposed a counting algorithm to keep track on the order of updates for each tuple in a view so that updates can be applied at the correct data warehouse states. Hull [5] decomposed an integrated view

Manuscript received December 3, 2001; revised December 2, 2003 and March 17, 2004. This paper was recommended by Associate Editor Y. Pan.

The authors are with the Intelligent Robotics and Manufacturing Systems Laboratory, School of Engineering Science, Simon Fraser University, Burnaby, V5A 1S6 BC, Canada (e-mail: gruver@cs.sfu.ca).

Digital Object Identifier 10.1109/TSMCA.2005.843385

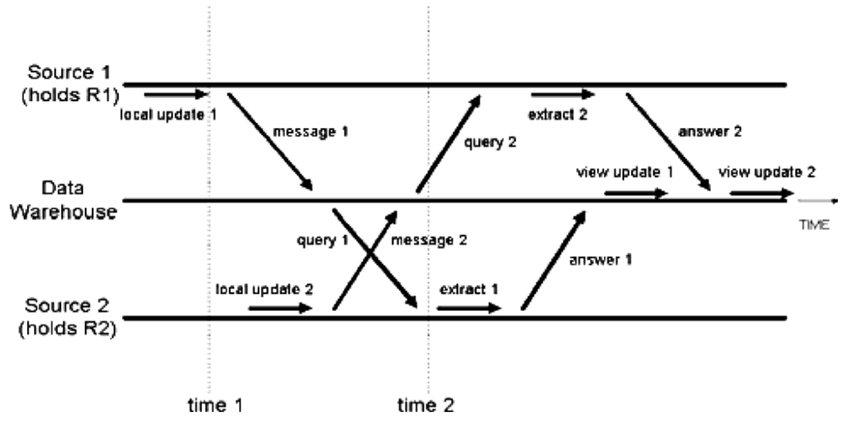

Fig. 1. Concurrent update causing interference.

into materialized, partially materialized, and virtual relations. Mediators were used in his system to optimize view usage by making less frequently accessed relations to be virtual. In research by Kawaguchi [6], a concurrency theory was developed based upon conflicts and serialization graphs in the presence of deferred materialized views.

Zhuge [7] initiated research into IIVM by proposing a series of algorithms (called strobe algorithms) to perform IIVM for select-project-join (SPJ) warehouse views. Strobe algorithms employ a monitoring and merging process to propagate updates to the data warehouse from distributed data sources. Compensation queries are required if interfering updates are detected at the sources. The SWEEP algorithm proposed by Agrawal [9] eliminated the need for compensation queries by a local compensation process, thereby providing a significant reduction in processing time. Ross [10] improved the view maintenance process by creating additional views, which is similar to subtasking view maintenance processes into simpler subprocesses. Ling and Liu [11] investigated message transmission disorders due to network delay. They issued time stamps at the sources to track the order of update notifications. In research by Chen [12], a global counter enabled solution of the same anomaly detection problem.

The main challenge in using IIVM is the need to provide compensation for the concurrent source update that interferes with the view update in process. This is due to the assumption that the underlying data sources are autonomous and independent of the data warehouse. The data warehouse must intelligently perform compensation when a concurrent update occurs. Parallel processing is often avoided due to a high probability of anomalies. The main contribution of our study is a framework for parallel processing within IIVM.

Research has also been conducted to apply agents to data warehousing and data mining. Bose [13] applied software agents for data-mining applications. His approach used agents to perform data selection, data transformation, data mining, and result interpretation. Belo [14] proposed using agents and the contract net protocol (CNP) to perform data trading between data-warehousing systems. Ram [15] proposed using a blackboard-based cooperative system for integrating data from heterogeneous databases with different schemas. The blackboard facilitates communication among human and computational agents.

II. Multiagent System

A. Problem Statement

We consider a system designed to perform IIVM for a data warehouse. The materialized views in the data warehouse are assumed to be SPJ views. The data sources are autonomous and there is no coordination between data sources. The data sources have no knowledge about the existence of a data warehouse. We assume that for a given source, updates and queries are executed atomically. For example, we do not allow an update to be performed on a data item when a query on the same item is in process. We assume that the data warehouse receives the update notifications in the same order as they were sent from the data sources. A violation of this assumption was investigated in [11] and [12].

For a typical warehouse view V defined by

$$V = \prod\nolimits_{\mathrm{Attr}} \sigma_{\mathrm{Cond}}(R_1 \rhd\lhd \ldots \rhd\lhd R_i \rhd\lhd \ldots \rhd\lhd R_n) \tag{1}$$

as an update $\Delta \mathrm{R_i}$ (an update to the base relation $\mathrm{R_i}$) is received at the data warehouse, the incremental change to the view is computed by querying the data sources to compute the join, and then applying the select and project operations. The query to be evaluated for the join is expressed as

$$Q = R_1 \rhd\lhd \ldots \rhd\lhd \Delta R_i \rhd\lhd \ldots \rhd\lhd R_n. \tag{2}$$

Thus, the view must be reassembled by making queries for each relation in the SPJ expression. This entails making queries to each of the sources that hold the relations $R_1, R_2, \ldots, R_\mathrm{n}$.

Using relational algebra to illustrate, assume two relations R_1 and R_2 and a view defined as a natural join over the relations $V = R_1 \rhd\lhd R_2$. As a result of an update ΔR_1, the new view should be

$$V_{\mathrm{new}} = (R_1 + \Delta R_1) \rhd\lhd R_2 = R_1 \rhd\lhd R_2 + \Delta R_1 \rhd\lhd R_2. \tag{3}$$

Since the data warehouse already has $R_1 \rhd\lhd R_2$, it only needs to compute a query $Q_1 = \Delta R_1 \rhd\lhd R_2$ at the data source containing $\mathrm{R_2}$ and incorporate the results into the materialized view. Difficulties arise, however, when two concurrent updates overlap. This scenario is shown in Fig. 1.

A local update for R_2 occurs at Source 2, causing query 1 to return the modified result. The reason for the difficulty is that the source transactions are allowed to occur at Source 2 between times 1 and 2. However, to correctly evaluate $\Delta R_1 \rhd\lhd R_2$, the state of Source 2 may not change between times 1 and 2.

Compensation must be performed to offset the effect of the interfering update. The data warehouse should perform a check after receiving each query result to see whether a concurrent update has altered the source state before the query. The properly compensated view-tuple should be

$$\Delta V = \Delta R_1 \rhd\lhd (R_2 - \Delta R_2). \tag{4}$$

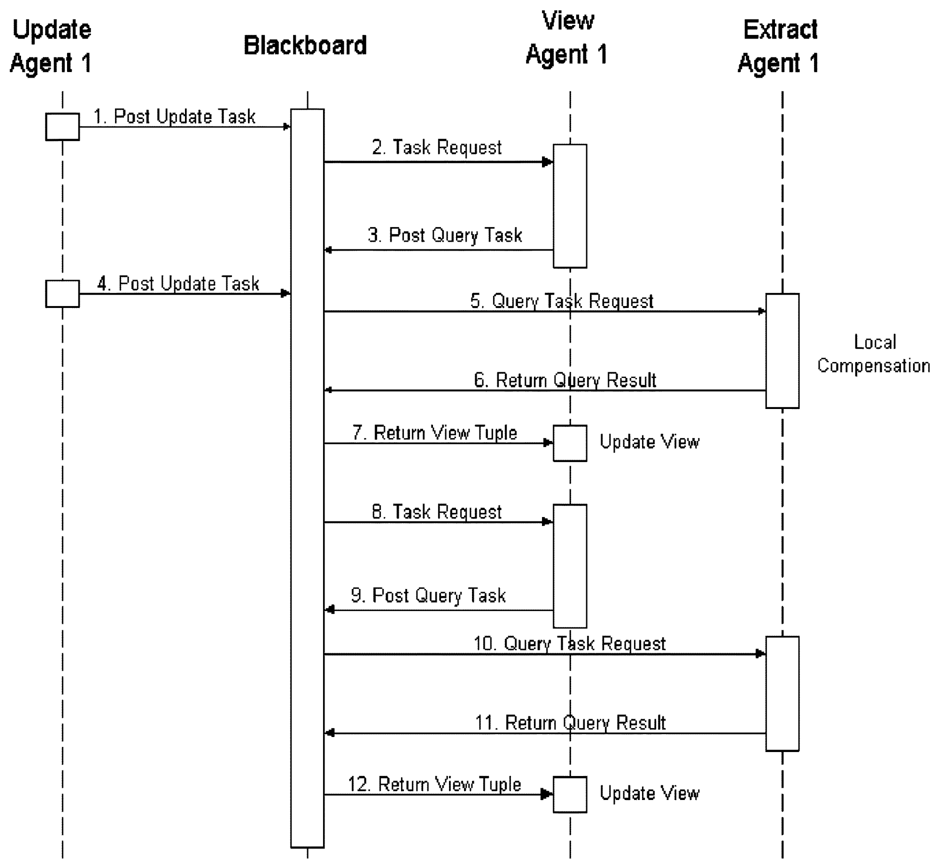

Fig. 2. Sequence diagram for a multiagent IIVM process.

B. Warehouse Agents

In the proposed multiagent system (MAS), we shall design agents to accomplish the tasks involved in IIVM. The MAS resembles an assembly line so that tasks are passed in a strict sequence through the system while suitable agents operate concurrently on the tasks. The finished product would be a set of properly compensated view-tuples for the warehouse views.

The *Update Agent* is responsible for monitoring the source. Each update agent delegates a single data source. The update agent stores information regarding the source (source meta-data), including the locator (URL) for the data table (network address, database name, connection protocol) and the relation identities. The functions of the update agent are: 1) polling a data source for updates and 2) converting the update information into a task request.

The *View Agent* is responsible for managing a data warehouse view. A view agent holds the metadata regarding a view, such as the SPJ view definition and the locators of the view relations. The view agent holds the mapping of the source relations to the view relations. The functions of the view agent are: 1) checking if an update is associated with the view it represents; 2) monitoring for concurrent updates; and 3) modifying warehouse views. If a view agent detects that the incoming updates interferes with another update in process, it passes the incoming update to the cooperating extract agent to perform compensation. Each view agent delegates a single warehouse materialized view.

The *Extract Agent* is responsible for querying the sources upon request. The extract agent interprets the query request and queries the specified sources. It does not possess metadata for a particular source or view. The view and the update agents supply the query information through the task request. If requested by the view agent, the extract agent can perform compensation to the queried result.

Blackboard architecture is adopted as the coordination method for the agents. A *Blackboard* is a common knowledge area for the agents to post and receive task requests. Fig. 2 is the sequence diagram illustrating the interaction between the agents and the blackboard. This example demonstrates how concurrent updates are handled by the MAS through local compensation.

C. Comparison With Other IIVM Systems

We shall give an overview on how MAS provides advantages for different stages of IIVM over other available systems in terms of processing time. When an update notification is received, the data warehouse must search to determine which views should be updated. The C-Strobe [7] and SWEEP [9] systems execute a sequential scan that includes all the views in the system. For the proposed MAS, the update notification is distributed to the view agents to concurrently perform this activity. The view agents can proceed to the following update in the queue without waiting for the current update to be finished.

To assemble the view-tuple ΔV for an SPJ view, the data warehouse issues a series of subqueries to the distributed sources to recompute the join relations. The existing systems perform the subqueries in a sequence. In the proposed MAS, multiple extract agents can perform subqueries of the same or different updates concurrently.

The data warehouse must intelligently compensate for the changes in the source made by a concurrent update. The local compensation method eliminates the need for compensation queries by checking for concurrent updates after each subquery. This method is adopted by the MAS. Since updates are initially assigned to the view agents, concurrent update checking is performed on a view agent's update queue, which is much shorter than update queue of the entire system, as performed by Zhuge [7].

In the proposed MAS, a warehouse view modification is performed by the view agents, according to the arrival order of the update notifications from the sources. If an update affects multiple views, the view agents cooperate to perform update modification simultaneously. This action improves data consistency in the data warehouse. Other systems disregard the fact that an update could affect multiple views and queue the updates sequentially.

D. Failure Handling

In transactional database systems, failure recovery deals with handling transactions when system crashes occur. The central issue is that the atomicity of every transaction must be maintained in the event of a system crash. Either the entire transaction is completed or nothing is completed. The rollback operation is often required to undo partially completed transactions. In transactional database systems, failing to recover an interrupted transaction could cause permanent database errors that cannot be rectified later.

The failure recovery in data warehouse view maintenance is different from that in conventional transaction processing. A data warehouse view maintenance failure could interrupt a view-tuple assembly process and leave it only partially complete. The point of interruption could occur before the update notification is sent, during subqueries, or at view modification. Instead of trying to recover the updates at the time of failure, it is often more convenient in data warehousing to recompute the views from the source data. Therefore, the objective of failure recovery in a data warehousing system should be on failure detection and actions to maintain view consistency for the users at any time, instead of recovering the interrupted operation.

There are many possible reasons for the failure of a source, including networks, hardware (disk storage), or software (database management system). The type of source failure is usually unknown to the data warehouse. Thus, it is uncertain whether transactions are active or updates are sent from the failed sources. Our approach is to maintain the warehouse views in a consistent prior state and when the failure at the source or warehouse is resolved, the warehouse views are recomputed. The failure detection methodology of MAS is described here.

In general, there are two possible types of failure in a data warehousing system: 1) failure of data sources and 2) failure of the data warehouse.

1) Failure of data sources. Such failures can be detected when an update agent fails to query the source for update or an extract agent fails to make queries to the source, causing SQL exceptions. In this case, the IIVM system should stop processing updates and maintain a consistent data warehouse state for the users. Note that if the current update is dropped, the warehouse will simply stay at the prior consistent state.
2) Failure of the data warehouse. Such failures are detected when a view agent fails to connect to the warehouse or update the warehouse view, causing an SQL exception. In that case, the IIVM system should stop processing updates and maintain a consistent prior data warehouse state for the users.

III. Experimental Results

In our experimental studies, we compared the performance of our proposed MAS-IIVM with the traditional IIVM algorithms in terms of processing time required to perform updates. The traditional algorithms were emulated in the same experimental setup by using a single thread for sequential processing.

The results demonstrate that the MAS-IIVM system, in which processes operated in parallel, had better performance than the traditional IIVM system. We verified that MAS-IIVM produces better performance without incurring data inconsistency.

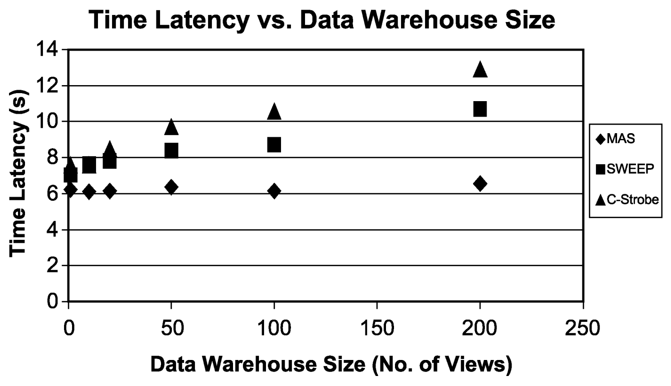

Fig. 3. Time latency versus data warehouse size.

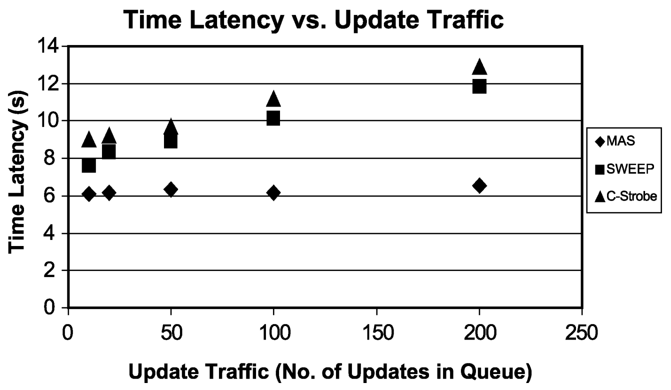

Fig. 4. Time latency versus update traffic.

The experimental MAS-IIVM was programmed in Java. Both the data sources and data warehouse were managed using Microsoft SQL Server 2000 implemented on the same multiple-CPU data server. All data tables were relational data tables and utilized SPJ schema to establish views in the data warehouse. The blackboard, update agents, extract agents, and view agents ran on separate threads.

The effect of the data warehouse size on time latency is presented in Fig. 3. For MAS, view agents check the update in parallel processes. Most of these processes are concurrent with the querying processes by the extract agents. Immediately after the earliest view agent finished checking and posts the query task, an extract agent can start to perform the queries. The C-Strobe and SWEEP systems perform all checking processes and querying processes in a sequence. Therefore, this methodology is directly influenced by the warehouse size.

Fig. 4 shows the time recorded as the update traffic is increased. The update traffic is the number of pending updates in the system queue. Other systems require more processing time, since checking is performed on the system queue for interfering updates after each subquery. For MAS, the extract agent only needs to check the updates assigned to one view agent. The time saving is tremendous if the update traffic is high.

Fig. 5 shows the time latency recorded as the number of views affected by an update increases. The concurrent processing of MAS can provide a significant advantage, since the subqueries and view modifications can be performed in parallel. Other IIVM systems perform the processes sequentially.

The effect of the number of available extract agents on the total query time is presented in Fig. 6. The query processes of the C-Strobe and SWEEP systems are equivalent to the MAS using one extract agent. If a single extract agent is used in MAS, the total query time is the same as other systems. As the number of extract agents increases in

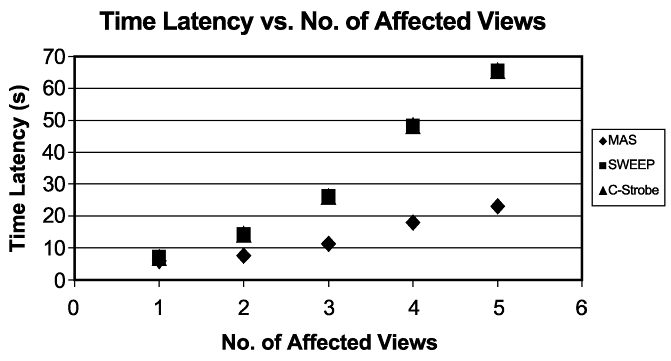

Fig. 5. Time latency versus the number of affected views.

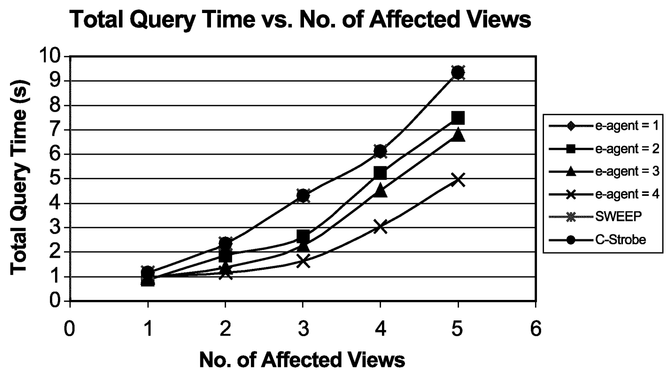

Fig. 6. Comparison of query time versus the number of affected views.

MAS, the total query time decreases since more tasks can be performed concurrently.

In summary, the advantage provided by MAS is that the update subprocesses for different updates can be performed concurrently. The performance gain is more significant as the system size increases. Even though we expect that a parallel system would provide better performance than a sequential system, our test results showed that a parallel process is feasible and is simple to implement with the help of a multiagent framework.

IV. Data Consistency

The data consistency of a data warehouse is intact if data "make sense" when being queried by client programs. DIVM can provide strict consistency since changes of base data are stored and applied in batches during downtime during which no transaction can interfere with the view maintenance process. After the changes are applied, the views in the data warehouse match perfectly with current state of the sources. For a data warehouse that adopts IIVM, the consistency condition must be enforced by the data warehouse so that the views are consistent at any time of operation and no erroneous updates occur. According to prior researches, there are different levels of consistency and a higher consistency level usually requires a more complex algorithm to implement.

Consistency levels can be used to specify the consistency requirement for view maintenance. In general, the following four levels can be considered, in ascending order of consistency level [7]: convergence, weak consistency, strong consistency, and completeness.

Convergence—requires only that the final data warehouse view be consistent with the source data after all update activities have been completed.

Weak Consistency—requires that each data warehouse view reflect a valid state at each source but the data warehouse view could reflect a different time state at each source.

Strong Consistency—requires not only that each data warehouse view reflect a valid source state at each source, but also that the source states reflected by a data warehouse view must be of the same global time state.

Completeness—requires not only that each data warehouse view is strongly consistent with the sources, but also that all activities at the sources must be reflected by the data warehouse views.

Our MAS was designed to enforce the completeness consistency level. The C-Strobe and the SWEEP algorithms also enforce the completeness consistency level, which is the main reason we used these latter systems in our comparison.

For IIVM, if the effect of the concurrent update is not compensated, even the convergence requirement cannot be satisfied. This is because the effect of a concurrent update could be duplicated in the data warehouse view, causing permanent errors in the warehouse view.

Through the use of local compensation, MAS ensures that the effect of the concurrent update is properly compensated under all concurrent update scenarios. The data warehouse is able to capture the original source state in every case since compensation corrects the queried answer when a concurrent update occurred at the source. The concurrent update is then queued according to the order of arrival. Using this method, every source state is reflected in the data warehouse in the same order as the sources. Therefore, the completeness consistency level is maintained.

V. Conclusion

We advocated the need for IIVM to perform continuous view maintenance in real-world data warehouses. MAS described by this study can perform IIVM and achieve a stringent level of data consistency of completeness. The multiagent framework applied in our system improved the sequential update processes found in other IIVM systems. The operation of IIVM is time critical and is usually too complex for applying parallel processing since anomalies can easily occur. We demonstrated that the multiagent IIVM system provides higher data warehouse availability as the update traffic or the warehouse size increases. By testing different IIVM systems, we demonstrated that MAS reduces the update processing time as the system size increases. MAS allows multiple agents to perform query tasks concurrently. We demonstrated that having more agents in the system increases performance, although the number of agents is limited by processing ability. Failures can be detected more quickly and easily by agents since they constantly interact with the sources and the data warehouse. In our proposed MAS, since the metadata for the warehouse views and the sources are completely distributed, the system can be increased or decreased in size by adding or removing agents.

References

[1] E. Hanson, "A performance analysis of view materialization strategies," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, San Francisco, CA, May 1987, pp. 440–453.
[2] L. S. Colby, T. Griffin, and L. Libkin *et al.*, "Algorithms for deferred view maintenance," in *Proc. ACM SIGMOD Conf.*, Montreal, Canada, June 1996, pp. 469–480.
[3] I. Mumick, D. Quass, and B. Mumick, "Maintenance of data cubes and summary tables in a warehouse," in *Proc. ACM SIGMOD Conf.*, Tuscon, AZ, May 1997, pp. 100–111.
[4] A. Gupta, I. S. Mumick, and V. S. Subrahmanian, "Maintaining views incrementally," in *Proc. ACM SIGMOD Conf.*, Washington, DC, May 1993, pp. 157–166.

[5] R. Hull and G. Zhou, "A framework for supporting data integration using the materialized and virtual approaches," in *Proc. ACM SIGMOD Conf.*, Montreal, Canada, Jun. 1996, pp. 481–492.

[6] A. Kawaguchi, D. Lieuwen, I. Mumick, D. Quass, and K. Ross, "Concurrency control theory for deferred materialized views," in *Proc. Int. Conf. Database Theory*, Athens, Greece, Jan. 1997, pp. 306–320.

[7] Y. Zhuge, H. Garcia-Molina, and J. L. Wiener, "Consistency algorithms for multi-source warehouse view maintenance," *J. Distrib. Parallel Databases*, vol. 6, no. 1, pp. 7–40, 1998.

[8] Y. Zhuge, J. L. Wiener, and H. Garcia-Molina, "Multiple view consistency for data warehousing," in *Proc. Int. Conf. Data Eng.*, Birmingham, UK, Apr. 1997, pp. 289–300.

[9] D. Agrawal and A. El Abbadi *et al.*, "Efficient view maintenance at data warehouses," in *Proc. ACM SIGMOD Conf.*, Tucson, AZ, May 1997, pp. 417–427.

[10] K. Ross, D. Srivastava, and S. Sudarshan, "Materialized view maintenance and integrity constraint checking: trading space for time," in *Proc. ACM SIGMOD Conf.*, Montreal, Canada, Jun. 1996, pp. 447–458.

[11] T. W. Ling and Y. Liu, "An efficient view maintenance algorithm for data warehousing," in *Proc. Adv. Database Technol.: ER'98 Workshops Data Warehousing Data Mining, Mobile Data Access, Collaborative Work Support Spatio-Temporal Data Manage.*, Singapore, Nov. 1998, pp. 169–180.

[12] R. Chen and W. Meng, "Precise detection and proper handling of view maintenance anomalies in a multidatabase environment," in *Proc. 2nd IFCIS Int. Conf. Coop. Inform. Syst.*, Jun. 1997, pp. 110–119.

[13] R. Bose and V. Sugumaran, "IDM: an intelligent software agent based data mining environment," in *Proc. IEEE Int. Conf. Syst., Man, Cybern.*, Atlanta, GA, Dec. 1998, pp. 2888–2893.

[14] O. Belo and A. Cuncha, "Integrating agent based information outsourcing techniques on data warehousing systems," in *Proc. IEEE Int. Conf. Syst., Man, Cybern.*, Tokyo, Japan, Oct. 1999, pp. 1025–1030.

[15] S. Ram, "A blackboard-based cooperative system for schema integration," *IEEE Expert*, vol. 10, no. 3, pp. 56–62, June 1995.