International Journal of Advanced Computer and Mathematical Sciences
ISSN 2230-9624. Vol 3, Issue 3, 2012, pp 362-367
http://bipublication.com

SCHEDULING OF UPDATES IN DATA WAREHOUSES

P. URMILA, K.SIVA RAMA KRISHNA, P. RAJA PRAKASH RAO
DEPT OF CSE, TRR ENGINEERING COLLEGE, HYDERABAD, A.P, INDIA

[Received-09/09/2012, Accepted-27/09/2012]

ABSTRACT

A stream warehouse enables queries that seamlessly range from realtime alerting and diagnostics to long-term data mining. Continuously loading data from many different and uncontrolled sources into a real-time stream warehouse introduces a new consistency problem: users want results in as timely a fashion as possible, but "stable" results often require lengthy synchronization delays. In this paper we develop a theory of temporal consistency for stream warehouses that allows for multiple consistency levels. We model the streaming warehouse update problem as a scheduling problem, where jobs correspond to processes that load new data into tables, and whose objective is to minimize data staleness over time.

INTRODUCTION

Tremendous and potentially infinite volumes of *data streams* are often generated by real-time surveillance systems, communication networks, Internet traffic, on-line transactions in the financial market or retail industry, electric power grids, industry production processes, scientific and engineering experiments, remote sensors, and other dynamic environments. Unlike traditional data sets, stream data flow in and out of a computer system *continuously* andwith varying update rates. They are *temporally ordered, fast changing, massive, and potentially infinite.* It may be impossible to store an entire data stream or to scan through it multiple times due to its tremendous volume. Moreover, stream data tend to be of a rather low level of abstraction, whereas most analysts are interested in relatively high-level dynamic changes, such as trends and deviations. To discover knowledge or patterns from data streams, it is necessary to develop single-scan, on-line, multilevel, multidimensional stream processing and analysis methods. Traditional data warehouses are updated during down times [25] and store layers of complex materialized views over terabytes of historical data. On the other hand, Data Stream Management Systems (DSMS) support simple analyses on recently arrived data in real time. Streaming warehouses such as Data Depot [15] combine the features of these two systems by

maintaining a unified view of current and historical data.

The goal of a streaming warehouse is to propagate new data across all the relevant tables and views as quickly as possible. Once new data are loaded, the applications and triggers defined on the warehouse can take immediate action. This allows businesses to make decisions in nearly real time, which may lead to increased profits, improved customer satisfaction, and prevention of serious problems that could develop if no action was taken. Recent work on the streaming warehouse has focused on speeding up the Extract –Transform-Load (ETL) process [28] [32]. There has also been work on supporting various warehouse maintenance policies, such as immediate, deferred (update views only when queried) and periodic [10]. There has been little work on choosing, of all the tables that are now out-of-date due to the arrival of new data, which one should update next. This is exactly the problem we study in this paper. [36] Instant view maintenance appear to be a reasonable solution for a stream. Whenever new data arrive, we instantly update the corresponding "base" table T. after T has been updated; we activate the updates of all the materialized views sourced from T, followed by all the views defined over those views, and so on. The problem with this approach is the new data may arrive on multiple streams, but there is no mechanism for restraining the number of tables that can be updated simultaneously. Running too many parallel updates can degrade performance due to memory and CPU-cache thrashing, disk-arm thrashing, context switching etc.

Data consistency.

Similarly to previous work on data warehousing, we want to ensure that each view reflects a "consistent" state of its base data [10] [35]. In addition to understanding data semantics and query results, another use for consistency is to minimize the number of base table and view updates in a warehouse. The *update consistency* of a table is the minimal consistency required by queries on it or its dependent tables, and determines when to refresh its partition(s). A partition of a table is computed only when it can be inferred to have a query consistency matching the desired update consistency. [37]

Hierarchies and priorities.

A data warehouse stores multiple layers of materialized views, e.g., a fact table of fine-grained performance statistics, the performance statistics rolled up to a coarser granularity, the rolled-up table joined with a summary of error reports, and so on. Some views are more important than others and are assigned higher priorities. For example, in the context of network data, responding to error alerts is critical for maintaining a reliable network, while loading performance statistics is not. We also need to prioritize tables that server as sources to a large number of materialized views. If such a table is updated, not only does it reduce its own staleness, but it also leads to updates of other tables. [36]

Heterogeneity and non-preemptibility.

Different streams may have widely different inter-arrival times and data volumes. For example, a streaming feed may produce data every minute, while a dump from an OLTP database may arrive once per day. This kind of heterogeneity makes real-time scheduling difficult. Suppose that we have three recurring update jobs, the first two being short jobs that arrive every ten time units and take two time units each to complete, and the third being a long job that arrives every 100 time units and takes 20 time units to complete. The expected system utilization of these jobs is only 60 percent – in an interval of length 100. We spend 20 time units on the long job, plus 2*10=20 time units on ten instances of each of the short jobs. However, serially executing these jobs starves the short ones whenever the long one is being executed.

This is because the execution time of the long job is longer than the inter-arrival time of the short ones. This means that tables or views whose update jobs are short may have high staleness. However, short jobs correspond to tables that are updated often, which are generally important. One way to deal with a heterogeneous workload is to allow preemptions. However, data warehouse updates are difficult to preempt for several reasons. For one, they use significant non-CPU resources such as memory, disk I/Os, file locks and so on. Also, updates may involve complex ETL processes, parts of which may be implemented outside the database. [36]

Another solution is to schedule a bounded number of update jobs in parallel. There are two variants of parallel scheduling. In *partitioned scheduling*, we cluster similar jobs together and assign dedicated resources (e.g, CPUs and /or disks) to each cluster [27]. In *global scheduling*, multiple jobs can run at the same time, but they use the same set of resources. Clustering jobs according to their lengths can protect short jobs from being blocked by long ones, but it is generally less efficient than global scheduling since one partition may have a queue of pending jobs while another partition is idle [4] [12]. Furthermore adding parallelism to scheduling problems generally makes the problems more difficult; tractable scheduling problems become intractable, real-time guarantees loosen, and so on [11]. The real-time community has developed the notion of fair scheduling for real-time scheduling on multi-processors [5].

Transient overload.

Streaming warehouses are inherently subject to overload in the same way the DSMSs are. For example, in a network data warehouse, a network problem will generally lead to a significantly increased volume of data flowing into the warehouse. At the same time, the volume of queries will increase as network managers attempt to understand and deal with the event. A common way to deal with transient overload in a real-time system is to temporarily discard jobs. Discarding some data is acceptable in a DSMS that evaluates simple queries in a single pass and does not store any data persistently [31].

SYSTEM MODEL

Warehousing Architecture

Figure illustrates a streaming data warehouse. Each data stream is generated by an external source, with a batch of new data, consisting of one or more records, being pushed to the warehouse with period pi. If the period of a stream is unknown or unpredictable, we let the user choose a period with which the warehouse should check for new data. Examples of streams collected by an Internet Service Provider include router performance statistics such as CPU usage; system logs; routing table updates; link layer alerts, etc. An important property of the data streams in our motivating applications is that they are append-only, i.e., existing records are never modified or deleted. For example, a stream of average router CPU utilization measurement may consist of records with fields (timestamp, router_name, CPU_utilization), and a new data file with updated CPU measurement for each router may arrive at the warehouse every five minutes. [36]

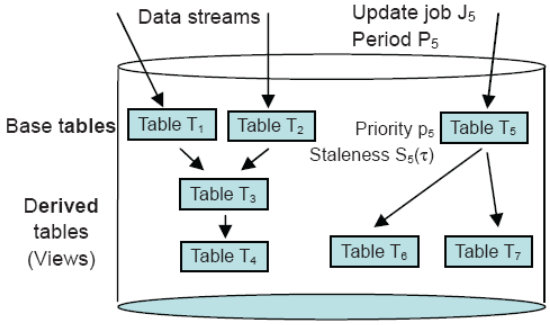

Figure: Stream data warehouse [36]

A streaming data warehouse maintains two types of tables: base and derived. Each table may be

stored partially or wholly on disk. A base table is loaded directly from a data stream. A derived table is materialized view defined over one or more (base or derived) tables. Each base or derived table T_j has a user-defined priority p_j and a time-dependent staleness function $S_j(\tau)$. Relationships among source and derived table form a dependency graph. For each table T_j, we define a set of its ancestor tables as those which directly or indirectly serve as its sources, and a set of its dependent tables as those which are directly or indirectly sourced from T_j. For example, T_1, T_2 and T_3 are ancestors of T_4, and T_3 and T_4 are dependents of T_1. When new data arrive on stream *i*, an update job J_i is released, whose purpose is to execute the ETL tasks, load the new data into the corresponding base table T_i, and update any indices. When this update job is completed, update jobs are released for all tables directly sourced from T_i in order to propagate the new data that have been loaded into T_i. When those jobs are completed, update jobs for the remaining dependent tables are released in the breadth-first order specified by the dependency graph. Each update job is modeled as an atomic, non-preemptible task. The purpose of an update scheduler is to decide which of the released update jobs to execute next; as mentioned earlier, the need for resource control prevents us from always executing update jobs as soon as they are released. The external sources push append-only data streams into the warehouse with a wide range of inter-arrival times. [36]

This proposed system allows business to make decisions in real time. This data stream management system supports simple analyses on recently arrived data in real time.

On-Line stock trading, where recent transactions generated by multiple stock exchange are compared against historical trend in nearly real time to identify profit opportunities in the proposed system.

The traditional data warehouses are typically refreshed during downtimes, streaming warehouses are updated as new data arrive. Where traditional data warehouse store layers of complex materialized views over terabytes of historical data. This existing system does not support to make decisions in real time and immediately. This existing system not suitable for data warehouse maintenance. This Existing system will not suitable for online scheduling problem. Because it does not allow to take immediate decisions.

It does not support various warehouse maintenance policies such as immediate, deferred and periodic.

It enables real time decision support for business critical applications.

It is used to compare multiple stock exchanges against historical trends in nearly real time to identify profit, performance, defect and loss.

Data stream management system support simple analyses on recently arrived data in real time.

CONCLUSION

We solved the problem of scheduling updates in a real-time streaming warehouse. We projected the notion of averages staleness as a scheduling metric and presented scheduling algorithms designed to handle complex environment of a streaming data warehouse.

We then proposed a scheduling framework that assigns jobs to processing tracks and also uses the basic algorithms to schedule jobs within a same.

REFERENCE

1. Tamassia, R., Triandopoulos, N. Efficient Content Authentication over Distributed Hash Tables. *ACNS*, 2007.
2. Tao, Y., Yi, K., Sheng, C., Kalnis, P. Quality and Efficiency in High Dimensional Nearest Neighbor Search. *SIGMOD*, 2009
3. Korn,F.,Sidiropoulos,N.,Faloutsos,C.,Siegel,E.,Protopapas,Z.Fast Nearest Neighbor Search in Medical Image Databases. *VLDB*, 1996.

4. Beyer, K., Goldstein, J., Ramakrishnan, R., Shaft, U. When is Nearest Neighbor Meaningful? *ICDT*,1999.
5. Keogh, E.,J., Ratanamahatana,C., A.Exact Indexing of Dynamic Time Warping. *Knowl. Inf. Syst.*, 7(3), 2005.
6. Scalable Scheduling of Updates in Streaming Data Warehouses Lukasz Golab, Theodore Johnson and Vladislav Shkapenyuk *AT&T Labs – Research, Florham Park, NJ, 07932, USA* {lgolab,johnsont,vshkap}@research.att.com
7. A Burns. Scheduling hard real-time systems: a review, software Engineering Journal, 6(3):116-128 (1991).
8. D.Carney, U.Cetintemel, A Rasin, S. Zdonik, M. Chermiack, and M Stonebraker, Operator scheduling in a data stream manager, VLDB 2003, 838-849.
9. J.Cho and H. Garcia-Molina, synchronizing a database to improve freshness, SIGMOD 2000, 117-128.
10. L.Colby, A.Kawaguchi, D. Lieuwen, I .Mumick, and K.Ross, Supporting multiple view maintenance policies, SIGMOD 1997, 405-416.
11. M.Dertouzos and A.Mok, Multiprocessor on-line scheduling of hard-real-time tasks, IEEE Trans. On Software. Eng., 15(12):1497-1506 (1989).
12. U.Devi and J.Anderson Tardiness bounds under global EDF scheduling Real-Time Systems, 38(2):133-189 (2008).
13. N.Folkert, A.Gupta, A.Witkowski, S.Subramanian, S.Bellamkonda, S.Shankar, T.Bozkaya, and L.Sheng, Optimizing refresh of a set of materialized views, VLDB 2005, 1043-1054.
14. M.Garey and D.Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, New York:W.H. Freeman, 1979.
15. L.Golab, T.Johnson, J.S.Seidel, and V.Shkapenyuk, Stream warehousing with DataDepot, SIGMOD 2009, 847-854.
16. L.Golab, T.Johnson, and V.Shkapenyuk, Scheduling updates in a real-time stream warehouse, ICDE 2009, 1207-1210.
17. H.Guo, P.A. Larson, R.Ramakrishnan, and J.Goldstein, Relaxed currency and consistency: How to say “good enough” in SQL, SIGMOD 2004, 815-826.
18. A Gupta and I. Mumick, Maintenance of materialized views: Problems, techniques, and applications, In IEEE Data Eng. Bulletin, Special Issue on Materialized Views and Data Warehousing, 18(2):3-18 (1995).
19. M.Hammand, M.Franklin, W.Aref, and A.Elmagarmid, Scheduling for shared window joins over data streams, VLDB 2003, 297-308.
20. K.D. kang, S.Son, and J.Stankovic, Managing deadline miss ratio and sensor data freshness in real-time databases, IEEE Trans. On Knowledge and Data Eng., 16(10):1200-1216(2004).
21. B.Kao and H.Garcia-Molina, An overview of real-time database systems, In Advances in Real-Time Systems (ed. S.H. Son), 463-486, Prentice Hall, 1995.
22. G.Koren, D.Shasha Dover, an optimal on-line scheduling algorithm for an overloaded real-time system, RTSS 1992, 292-299.
23. G.Koren, D.Shasha. An approach to handling overloaded systems that allow skips, RTSS 1995, 110-119.
24. Y.K. Kwok and I. Ahmad, Static scheduling algorithms for allocating directed task graphs to multiprocessors, ACM Computing Surveys, 31(4):406-471(1999).
25. W.Labio, R. Yemeni, and H Garcia-Molina, Shrinking the warehouse update window, SIGMOD 1999, 383-394.
26. A. Labrinidis and N. Roussopoulos, Update propagation strategies for improving the quality of data on the Web, VLDB 2001, 391-400
27. Y. Oh and S.H. Son Tight performance bounds of heuristics for a real-time scheduling problem Tech report CS-93-24, U. Virginia (1993).
28. N.Polyzotis, S.Skiadopoulos, P.vassiliadis, A.Simitsis, and N-E Frantzell, Supporting Streaming Updates in an Active Data Warehouse, ICDE 2007, 476-485.
29. M.Sharaf, P.Chrysanthis, A. Labrinidis, and K Pruhs, Algorithms and metrics for processing multiple heterogeneous continuous queries, ACM Trans. On Database Sys, 33(1) (2008).
30. R. Srinivasan, C. Liang, and K. Ramamritham, Maintaining temporal coherency of virtual data warehouses, RTSS 1998, 60-70.
31. N.Tatbul, U.Cetintemel, S.Zdonik, M.Chemiack, and M. Stonebraker, Load shedding in a data stream manager, VLDB 2003, 309-320.

32. C. Thomsen, T.B.Pedersen, and W.Lehner, RiTe Providing on-demand data for right-time data warehousing, ICDE 2008, 456-465.
33. P.Tucker, Punctuated Data Streams, Ph.D Thesis, Oregon Health & Science University, 2005.
34. H. Qu and A. Labrinidis, Preference-aware query and update scheduling in Web-databases, ICDE 2007, 356-365.
35. Y.Zhuge, J. Wiener, and H.Garcia-Molina, Multiple view consistency for data warehousing, ICDE 1997, 289-300.
36. Scalable Scheduling of Updates in Streaming Data Warehouses Lukasz Golab, Theodore Johnson and Vladislav Shkapenyuk *AT&T Labs – Research, Florham Park, NJ, 07932, USA*{lgolab,johnsont,vshkap}@research.att.com
37. Lukasz Golab and Theodore Johnson, Consistency in a Stream Warehouse, 5th *Biennial Conference on Innovative Data Systems Research (CIDR '11)* January 9-12, 2011, Asilomar, California, USA.