ISSN:0975-9646

An Efficient Algorithm for Update Scheduling in Streaming Data Warehouses

Bolla Saikiran[1], Kolla Morarjee[2]

[1]M.Tech Scholar, [2]Assistant Professor
Department of Computer Science and Engineering, CMR Institute of Technology
Medchal, Hyderabad, Andhra Pradesh, India

Abstract: Traditional data warehouses are designed to hold the Historical Data. The data warehouse allows update only policy. Data Stream is a continuously growing Data from Various Sources. Update scheduling in streaming data warehouses, which combines the features of traditional data warehouses and data stream systems. External sources push append-only data streams into the warehouse with a wide range of inter arrival times. Traditional data warehouses are typically refreshed during downtimes, streaming ware houses are updated as new data arrive. We explore the model the streaming warehouse update problem as a scheduling problem, where processes that load new data into tables, and whose objective is to minimize data staleness over time. . We choose a scheduling framework that handles the complications encountered by a stream warehouse: view hierarchies and priorities, data consistency, inability to preempt updates, heterogeneity of update jobs caused by different inter arrival times and data volumes among different sources, and transient overload. A feature of our framework is that scheduling decisions do not depend on properties of update jobs, but rather on the effect of update jobs on data staleness. We present EDF-P, Max Benefit update scheduling algorithms and proves that Max Benefit is giving best performance that EDF-P by minimizing data staleness over the time.

Keywords : Data warehouse maintenance, Update scheduling, Data streaming.

I. INTRODUCTION

The Application of Data Stream Mining Demands the Newly Arrived Data, not on the Historical Data. The Problem is to Combine Traditional Data Warehouses and Data Stream Systems[1]. In existing system Data Stream Management Systems (DSMS) it Supports Simple Analysis and Recently Arrived Data. Data Depot also combines the Features of Traditional Data Warehouses and Data Stream Management System (DSMS).

Traditional data warehouses are updated during downtimes 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 these two systems by maintaining a unified view of current and historical data. This enables a real-time decision support for business-critical applications that receive streams of append-only data from external sources. Applications include: Online stock trading, where recent transactions generated by multiple stock exchanges are compared against historical trends in nearly real time to identify profit opportunities. Credit card or telephone fraud detection, where streams of point-of-sale transactions or call details are collected in nearly real time and compared with past customer behavior. Network data warehouses maintained by Internet Service Providers (ISPs), which collect various system logs and traffic summaries to monitor erasure correcting code in the file distribution preparation to provide redundancies and guarantee the data constancy[2]. This construction drastically reduces the communication and storage overhead as compared to the traditional replication. 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 streaming warehouses has focused on speeding up the Extract-Transform-Load (ETL) process which was explained in Figure1. This Figure1 also describes the Data flow in Data Warehouse with the ETL Process[3]. Data coming from various external sources to the Data Warehouse and loading into the warehouse tables by utilizing the ETL Process.

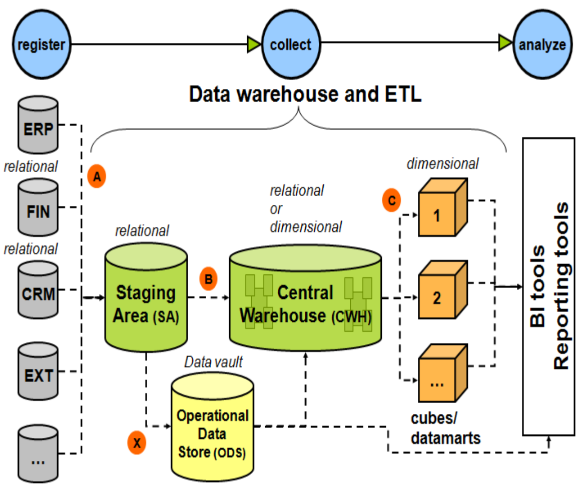

Figure 1: Data warehousing with ETL Process.

There has also been work on supporting various warehouse maintenance policies, such as immediate, deferred, and periodic. However, there has been a little work on choosing, of all the tables that are now out-of-date due to the arrival of new data, which one should be updated next. This is exactly the problem we study in this paper. Immediate view maintenance may appear to be a reasonable solution for a streaming[4]. That is, whenever new data arrive, we immediately update the corresponding "base" table T. After T has been updated, we trigger 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 that new data may arrive on multiple streams, but there is no mechanism for limiting the number of tables that can be updated simultaneously[5]. Running too many parallel updates can degrade performance due to memory and CPU-cache thrashing, disk-arm thrashing, context switching, etc. This motivates the need for a scheduler that limits the number of concurrent updates and determines which job to schedule next.

II. Related Work

This section compares the contributions of this paper with previous work in three related areas: scheduling, data warehousing, and data stream management.

Scheduling

The closest work to ours is, which finds the best way to schedule updates of tables and views in
order to maximize data freshness. Aside from using a different definition of staleness, our Max Benefit basic algorithm is analogous to the max-impact algorithm from Labrinidis and Roussopoulos, as is our "Sum" priority inheritance technique. Our main innovation is the multi track Proportional algorithm for scheduling the large and heterogeneous job sets encountered by a Streaming warehouse additionally, we propose an update chopping to deal with transient overload. Another closely related work is which studies the complexity of minimizing the staleness of a set of base tables in a streaming warehouse[6]. In general, interesting scheduling problems are often NP hard in the offline setting and hard to approximate offline. This motivates the use of heuristics such as our greedy Max Benefit algorithm. While we believe that update scheduling in a streaming warehouse is novel, our solution draws from a number of recent scheduling results. In particular, there has been work on real-time scheduling of heterogeneous tasks on a multiprocessor to address the tension between partitioned scheduling and global scheduling. The Pair algorithm and its variants have been proposed when tasks are perceptible, however we must assume that data loading tasks are nonpreemptible. Our Proportional algorithm attempts to make a fair allocation of resources to nonpreemptible tasks in a multitrack setting, and is the first such algorithm of which we are aware. Overload management has been addressed in, e.g.. However, these algorithms handle overload by dropping jobs, while a data warehouse can defer updates for a while, but cannot drop them. Derived table relationships are similar to precedence constraints among jobs, e.g., a derived table cannot be updated until its sources have been updated. Previous work on scheduling with precedence constraints focused on minimizing the total time needed to compute a set of nonrecurring jobs. Our update jobs are recurring, and we use the notion of freshness delta to determine when a derived table update should be scheduled[7]. There has also been work on adding real-time functionality to databases. However, most of this work focuses on scheduling read-only transactions to achieve some quality of- service guarantees. The works closest to ours are and which discuss the interaction of queries and updates in a firm real-time database, i.e., how to install updates to keep the data fresh, but also ensure that read transactions meet their deadlines. However, their system environments are significantly different from ours: transactions can be preempted, all tables in the database are "snapshot" tables, and updates are very short lived, meaning that they can be deferred until a table is queried. Similar work has also appeared in the context of web databases, which aims to balance the quality of service of read transactions against data freshness. It also assumes a "snapshot" model rather than our append-only model.

Data Warehousing

There has been some recent work on streaming data warehousing, including system design, real-time ETL Processing, and continuously inserting a streaming data feed at bulk-load speed. These efforts are complementary to our work they also aim to minimize data staleness, but they do so by reducing the running time of update jobs once the jobs are scheduled. A great deal of existing data warehousing research has focused on efficient maintenance of various classes of materialized views, and is orthogonal to this paper. In and discuss consistency issues under various view maintenance policies. As discussed earlier, maintaining consistency in a streaming data warehouse is simpler due to the append-only nature of data streams. There has also been work on scheduling when to pull data into a warehouse to satisfy data freshness guarantees. This work does not apply to the push-based stream warehouses studied in this paper, which do not have control over the data arrival patterns. Quantifying the freshness of a data warehouse was addressed in several works[8]. For instance, Adelberg et al. propose two definitions: maximum age, which corresponds to the definition used in this paper, and unapplied update, which defines staleness as the difference between the current time and the arrival time of the oldest pending update. Unapplied update is not appropriate in a real-time stream warehouse that must handle sources with various arrival rates and inter arrival times. For example, suppose that two updates have arrived simultaneously, one containing 2 minutes of recent data from stream 1, and the other carrying one day of data from stream 2. Clearly, the table sourced by stream 2 should be considered more out-of-date, yet both are equal under unapplied update. Cho and Garcia-Molina propose to measure the average freshness over time, but their definition of freshness is far simpler than ours: a database object is assumed to have a freshness of one if it is up-to-date and zero otherwise.

Data Stream Management

One important difference between a DSMS and a data stream warehouse is that the former only has a limited working memory and does not store any part of the stream permanently. Another difference is that a DSMS may drop a fraction of the incoming elements during overload, whereas a streaming data warehouse may defer some update jobs, but must eventually execute them. Scheduling in DSMS has been discussed in, but all of these are concerned with scheduling individual operators inside query plans.

III. SYSTEM MODEL

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[1]. 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, and a new data file with updated CPU measurement for each router may arrive at the warehouse every 5 minutes.

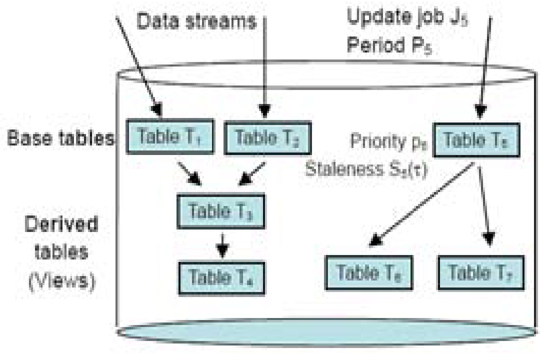

Figure 2: Streaming Data Warehouse

Warehouse Consistency Following the previous work on data warehousing, we want derived tables to reflect the state of their sources as of some point in time[9]. Suppose that D is derived from T1 and T2, which were last updated at times 10:00 and 10:05, respectively as shown in figure 2. If T1 and T2 incur arbitrary insertions, modifications, and deletions, it may not be possible to update D such that it is consistent with T1 and T2 as of some point in time, say, 10.00.However, tables in a streaming warehouse are not "snapshots" of the current state of the data, but rather they collect all the (append-only) data that have arrived over time. Since the data are append-only, each record has exactly one "version." For now, suppose that data arrive in time stamp order. We can extract the state of T2 as of time 10:00 by selecting records with time stamps up to and including 10:00. Using these records, we can update D such that it is consistent with T1 and T2 as of time 10:00. Figure2 shows T1 and T2 are Base Tables, Derived table has been created from based tables namely T3 and again created another derived table T4 created from table T3.we applied EDFP and Maxbenefit algorithms on the tables and conducted multiple Experiments. Hence update job as Ji and priority as pi.

Data Staleness we illustrate the staleness as a function of time for a base table Ti. Suppose that the first batch of new data arrives at time 4. Assume that this batch contains records with time stamps up to time 3.Staleness accrues linearly until the completion of the first update job at time 5. At that time, Ti has all the data up to time 3, and therefore its staleness drops to 2.

Scheduling Model

Let Ji be the update job corresponding to table Ti. For base tables, the period of Ji is equal to the period of its source stream. We estimate the period of a Ji corresponding to a derived table as the maximum period of any of Ti's ancestors[10]. For base and derived tables, we define the freshness delta of Ti, call it _Fi, as the increase in freshness after Ji is completed. For instance, when the second batch of new data arrives in (at time 7), the table contains data up to time 3. Since the update contains data up to time 7, the freshness delta is 4.

Prioritized EDF (EDF-P)

We tries to order the jobs by assigning priorities. In this process ties are breaking by deadlines [7]. We estimate the deadline ri +Pi where ri is the last time Ti's freshness delta changed from zero to nonzero, Pi is period of derived table [7]. EDF-Partitioned Strategy The EDF-partitioned algorithm assigns jobs to tracks in a way that ensures that each track has a feasible non preemptive EDF schedule. A feasible schedule means that if the local scheduler were to use the EDF algorithm to decide which job to schedule next, all jobs would meet their deadlines. EDF-partitioned strategy is compatible with any local algorithm for scheduling individual tracks[2]. The EDF-partitioned algorithm has some weaknesses, a collection of jobs with identical periods might be partitioned among several tracks[11]. The track promotion condition among these jobs and tracks is the same as the condition which limits the initial track packing and therefore no track promotion will be done. the Earliest-Deadline-First (EDF) algorithm orders released jobs by proximity to their deadlines. EDF(Earlier Dead Line First) is known to be an optimal hard real-time scheduling algorithm for single track. Prioritized EDF orders jobs by their priorities, breaking ties by deadlines. Here jobs are performed based on the Deadline.

Max Benefit

Max Benefit Recall that the goal of the scheduler is to minimize the weighted staleness [7]. In this context, the benefit of executing a job Ji may be defined as pi Δ Fi, i.e., its priority weighted freshness delta (decrease in staleness).

Similarly, the marginal benefit of executing Ji is its benefit per unit of execution time: pi Δ Fi=EΔ(Fi). A natural online greedy heuristic is to order the jobs by the marginal benefit of executing them. We will refer to this heuristic as Max Benefit. Since marginal benefit does not depend on the period, we can use Max Benefit for periodic and a periodic update jobs. one may argue that Max Benefit ignores useful information about the release times of future jobs. This algorithm is used to minimize the weighted staleness.

IV. System Evaluation

Framework to Generate Update Jobs Automatically when the Environment is a Simulation Environment and it produces a Bunch of Jobs and Send to Data Warehouse for Every 5mins automatically.

Table Relational Graph

All the Tables of the Warehouse are devided into two types. One is Base Table another is a Derived Table (view). Base Table is the Primary Tables Whenever New Data is Arrived the tables are updated First. The Derived Tables are the Views of an SQL Query. The Derived Tables Depends on the Base (or) Source Tables. This Modules Prepares Dependency Graph. Assignment Priorities to tables/Views Base and Derived Tables has to be Assigned Priority Values Based on Dependency Graph. The Priority Values of Tables are Used at the Time of Processing (or) Prioritizing Jobs.

Job Partitioning

The System will Recieve a Bulk Number Update Jobs Among all the Jobs, Which Job has to be Given High Priority and to avoid Datastaless Job Partioning is Required. The Module are EDF Partioned Strategy, Leveraging Idle Resources Via Track Promotion and Proportional Partitioning Strategy.

Update Chopping

The Traditional of a Single Job Updates all the Jobs into a Single Queue of Jobs. Lineally all the will be Executed. The Update Chopping Process Divides the Jobs into Priority Jobs and Stale Data Jobs. The Stale Data jobs are Given Low priority. The High Priority Jobs Executed First.

Experiments are conducted with EDFP and Maxbenefit algorithms for updated scheduling in streaming data warehouses by creating base tables T1 and T2. Derived table has been created from based tables namely T3 again created another derived table T4 created from table T3.we applied EDFP and Maxbenefit algorithms on these tables and proved that Maxbenefit giving the accurate results.

V. Conclusions

In this paper, we motivated, formalized, and solved the problem of non preemptively scheduling updates in a real-time streaming warehouse. We proposed the notion of average staleness as a scheduling metric and presented scheduling algorithms designed to handle the complex environment of a streaming data warehouse. We then proposed a scheduling framework that assigns jobs to processing tracks and uses basic algorithms to schedule jobs within a track. The main feature of our framework is the ability to reserve resources for short jobs that often correspond to important frequently refreshed tables, while avoiding the inefficiencies associated with partitioned scheduling techniques. We have implemented some of the proposed algorithms in the Data Depot streaming warehouse, which is currently used for several very large warehousing projects within AT&T. We tested with EDF-P and Max Benefit algorithms. Finally we proved that Max Benefit is giving best performance that EDF-P by minimizing data staleness over the time. As future work, we plan to extend our framework with new basic algorithms. We also plan to fine-tune the Proportional algorithm in our experiments, even the aggressive version with "all" allocation still exhibits signs of multiple operating domains, and therefore can likely be improved upon (however, it is the first algorithm of its class that we are aware of). Another interesting problem for future work involves choosing the right scheduling "granularity" when it is more efficient to update multiple tables together, as mentioned. We intend to explore the tradeoffs between update efficiency and minimizing staleness in this context.

References

[1] A.Sreeja, I.V, Sailakshmiharitha, N.Bhaskar, "Load Balancer Scheduling Over Streaming Data in Federated Databases" IJERT, Vol.2.Issue 8, August 2013.

[2]Lukas Golab, Theodore Johnson, and Vladislav Shkapenyuk," Scalable Scheduling of Updates in Streaming Data Warehouses", IEEE Transactions On KDE, Vol/24, No.6, June2012.

[3] M.H. Bateni, L. Golab, M.T. Hajiaghayi, and H. Karloff, "Scheduling to Minimize Staleness and Stretch in Real-time Data Warehouses," Proc. 21st Ann. Symp. Parallelism in Algorithms and Architectures (SPAA), pp. 29-38, 2009.

[4] S. Baruah, "The Non-preemptive Scheduling of Periodic Tasks upon Multiprocessors," Real Time Systems, vol. 32, nos. 1/2, pp. 9- 20, 2006.

[5] S. Babu, U. Srivastava, and J. Widom, "Exploiting K-constraints to Reduce Memory Overhead in Continuous Queries over Data Streams," ACM Trans. Database Systems, vol. 29, no. 3, pp. 545- 580, 2004.

[6] B. Babcock, S. Babu, M. Datar, and R. Motwani, "Chain: Operator Scheduling for Memory Minimization in Data Stream Systems," Proc. ACM SIGMOD Int'l Conf. Management of Data, /pp. 253-264, 2003.

[7] D. Carney, U. Cetintemel, A. Rasin, S. Zdonik, M. Cherniack, and M. Stonebraker, "Operator Scheduling in a Data Stream Manager," Proc. 29th Int'l Conf. Very Large Data Bases (VLDB), pp. 838- 849, 2003.

[8] J. Cho and H. Garcia-Molina, "Synchronizing a Database to Improve Freshness," Proc. ACM SIGMOD Int'l Conf. Management of Data, pp. 117-128, 2000.

[9] S. Baruah, N. Cohen, C. Plaxton, and D. Varvel, "Proportionate Progress: A Notion of Fairness in Resource Allocation," Algorithmica,vol. 15, pp. 600-625, 1996.

[10] B. Adelberg, H. Garcia-Molina, and B. Kao, "Applying Update Streams in a Soft Real-Time Database System," Proc. ACM SIGMOD Int'l Conf. Management of Data, pp. 245-256, 1995.

[11] A. Burns, "Scheduling Hard Real-Time Systems: A Review, "Software Eng. J., vol. 6, no. 3, pp. 116-128, 1991.