International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Impact Factor (2012): 3.358

Algorithms used for Updates in Streaming Data Warehouses by Scalable Scheduling

Shraddha S. Deshpande[1], S. A. Kinariwala[2]

[1]PG Student, Department of Computer Science and Engineering, Dr. BAMU University, Aurangabad, India

[2]Professor, Department of Computer Science and Engineering, Dr. BAMU University, Aurangabad, India

Abstract: ***This project includes jobs correspond to processes that load new data into tables, and whose objective is to minimize data staleness over time. The proposed framework 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 interarrival times and data volumes among different sources, and transient overload. A novel feature of our framework is that scheduling decisions depend on the effect of update jobs on data staleness.***

Keywords: Data Stream Management Systems (DSMS), Update scheduling, warehouse maintenance, Data streaming, Scalable scheduling.

1. Introduction

Data mining is the process by which accurate and previously unknown information is extracted from large volumes of data. This information should be in a form that can be understood, acted upon, and used for improving decision processes that can be used to increase revenue, cuts costs, or both. Data mining software is one of a number of analytical tools for analyzing data. It allows users to analyze data from many different dimensions or angles, categorize it, and summarize the relationships identified. Technically, data mining is the process of finding correlations or patterns among dozens of fields in large relational databases.
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 DataDepot [15] combine the features of these two systems by maintaining a unified view of current and historical data. 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 network performance and detect network attacks.

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.

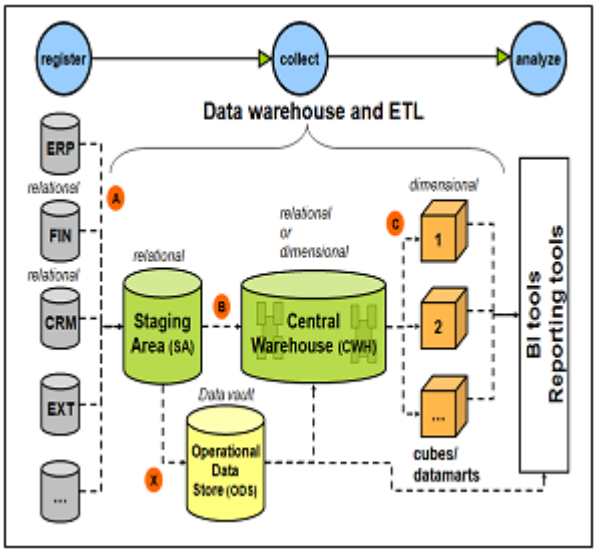

Figure 1: Data warehousing with ETL Process

There has also been work on supporting various warehouse maintenance policies, such as immediate deferred and periodic. 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. Instant view maintenance appears 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.

Licensed Under Creative Commons Attribution CC BY

Scheduling metric: Many metrics have been considered in the real-time scheduling literature. In a typical hard real time system, jobs must be completed before their deadlines a simple metric to understand and to prove results about. In a firm real-time system, jobs can miss their deadlines, and if they do, they are discarded. The performance metric in a firm real-time system is the fraction of jobs that meet their deadlines. However, a streaming warehouse must load all of the data that arrive therefore no updates can be discarded. In a soft real-time system, late jobs are allowed to stay in the system, and the performance metric is lateness which is the difference between the completion times of late jobs and their deadlines. Instead, we will define a scheduling metric in terms of data staleness, roughly defined as the difference between the current time and the time stamp of the most recent record in a table.

1. Data consistency: We want to ensure that each view reflects a "consistent" state of its base data.
2. 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 serves 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.
3. Heterogeneity and nonpreemptibility: Different streams may have widely different interarrival 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. 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. 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 (e.g., with respect to their expected running times) and assign dedicated resources (e.g., CPUs and/or disks) to each cluster. 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. Furthermore, adding parallelism to scheduling problems generally makes the problems more difficult; tractable scheduling problems become intractable, real-time guarantees loosen, and so on. The real-time community has developed the notion of Pfair scheduling for real-time scheduling on multiprocessors. However, Pfair scheduling requires preemptible jobs.
4. Transient overload: Streaming warehouses are inherently subject to overload in the same way that DSMSs are. For example, in a network data warehouse, a network problem will generally lead to a significantly increased volume of data (system logs, alerts, etc.) 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. However, all the data must be loaded into a warehouse, so we cannot drop updates, just defer their execution. During overload, a reasonable scheduler defers the execution of update jobs corresponding to low-priority tables in favor of high-priority jobs. When the overload subsides and low-priority tables can finally be scheduled, they may have accumulated a large amount of work (i.e., multiple "chunks" of new data may have arrived). As a result, these low-priority jobs become long-running and may now starve incoming high-priority updates.

2. Literature Survey

2.1 Soft Real-Time Database System

The Proposed efficiently export a materialized view but to knowledge none have studied how to efficiently import one. To install a stream of updates, a real-time database system must process new updates in a timely fashion to keep the database fresh, but at the same time must process transactions and meet their time Constraints. Various properties of updates and views that affects this trade-off. Examining through simulation, four algorithms for scheduling transactions and installing updates in a soft real time database.[2]

2.2 Multiple View Consistency for Data Warehouse

The proposed data warehouse stores integrated information from multiple distributed data sources. In effect, the warehouse stores materialized views over the source data. The problem of ensuring data consistency at the warehouse can be divided into two components: ensuring that each view reflects a consistent stare of the base data, and ensuring that multiple views are mutually consistent. Guarantying multiple view consistency (MVC) and identify and define formally three layers of consistency for materialized views in a distributed environment.[3]

2.3 Synchronizing a Database to Improve Freshness

The proposed a method to refresh a local copy of an autonomous data source to maintain the copy up-to-date. As the size of the data grows, difficult to maintain the fresh copy making it crucial to synchronize the copy electively. Two fresh Metrics, such as change models of the underlying data and synchronization policies.[4]

2.4 Operator Scheduling For Memory

The proposed many applications involving continuous data streams, data arrival are busty and data rate fluctuates over time. Systems that seek to give rapid or real-time query responses in such an environment must be prepared to deal gracefully with bursts in data arrival without compromising

Licensed Under Creative Commons Attribution CC BY

system performance. Strategies for processing burst streams adaptive, load-aware scheduling of query operators to minimize resource consumption during times of peak load. Chain scheduling, an operator scheduling strategy for data stream systems that is near-optimal in minimizing run-time memory usage for any collection of single stream queries involving selections, projections, and foreign-key joins with stored relations. Chain scheduling also performs well for queries with sliding-window joins over multiple streams, and multiple queries of the above types.[5]

a) Existing System Model

The existing system that is traditional datawarehouse does not support the accuracy in data warehouse maintenance. The system does not support to make decisions in real time . This system is not suitable for online scheduling problem. Because it does not allow to take immediate decisions. The problem with this system 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.

b) Proposed 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. 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.

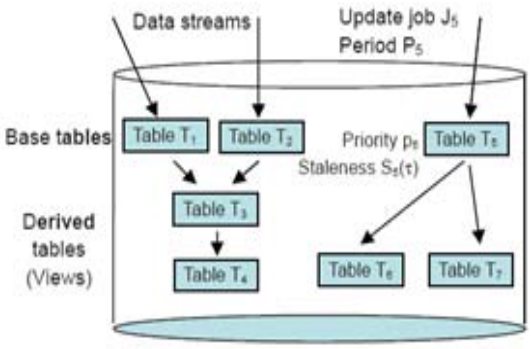

Figure 2: Streaming Data Warehouse

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. 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.

3. Scheduling Model

3.1.1 Prioritized EDF (EDF-P)

In this technique jobs are ordered by their assigning priorities and ties are breaking by deadline. 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.

3.1.2 Max benefit

In this context, the benefit of executing a job *Ji* may be defined as *piΔFi,* 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: *piΔ 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.

3.1.3 EDF-Partitioned

This algorithm assigns jobs to tracks such that each track has a feasible nonpreemptive 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.

3.1.4 Proportional Partitioning Strategy

In this algorithm, clusters of similar jobs are indentified and choosing a small value of k may create many small clusters of jobs whose execution times and periods are similar; as with the EDF-partitioned algorithm, if any tracks are left over, they are free tracks. In this case the scheduling algorithm will need to share the track among the clusters.

Licensed Under Creative Commons Attribution CC BY

The Proportional strategy uses the following scheduling algorithm:

1. Sort the released jobs using the local algorithm.
2. For each job *Ji* in sorted order,
   a) Let *j* be the cluster of Ji.
   b) If a track between *track_lo[j]+1..track_hi[j]* is available, schedule *Ji* on that track.
   c) Else, if *track track_lo[j]* is available, schedule *Ji* on that track.
   d) Else, if a free track is available, schedule *Ji* on that track.
   e) Else, if there is an available track *r* numbered between*0* and *track_ lo[j]- 1* such that there is no released job remaining in the sorted list that would be scheduled on *r,* schedule *Ji* on *r.*
   f) Else, delay the execution of *Ji.*

3.1.5 Round Robin

Each job is assigned a time interval, called its quantum, which is allowed to run. If the job is still running at the end of quantum, the CPU is preempted and given to another job. If the job has blocked or finished before the quantum has elapsed, the CPU switching is done. Of course, round robin is easy to implement.

4. Conclusion

We solved the problem of non preemptively scheduling updates in a real-time streaming warehouse. We projected the notion of averages staleness as a scheduling metric and presented various scheduling algorithms such as max benefit, round robin, EDF-P, EDF-Partitioned, proportional partitioning 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 track. The main feature of 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.

5. Acknowledgement

The authors express gratitude to Principal, Head of Department (CSE) Dr. Radhakrishna Naik. Marathwada Institute of Technology College of Engineering, Aurangabad, and Maharashtra India. They also express their sincere thanks all the faculty members of CSE Department MIT College of Engineering, Aurangabad, and Maharashtra, India for their constant support and enthusiasm.

References

[1] Lukas Golab, Theodore Johnson, and Vladislav Shkapenyuk," Scalable Scheduling of Updates in Streaming Data Warehouses", IEEE Transactions On KDE, Vol/24, No.6, June2012.
[2] 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.
[3] Y. Huge, J. Wiener, and H. Garcia-Molina, "Multiple View Consistency for Data Warehousing," Proc. IEEE 13th Int'l Conf. Data Eng. (ICDE), pp. 289-300, 1986.
[4] J. Cho and H. Garcia-Molina, "Synchronizing a Database to Improve Freshness," Proc. ACMSIGMOD Int'l Conf. Management of Data, pp.117-128, 2000.
[5] 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.
[6] Bolla Saikiran, Kolla Morarjee, "An Efficient Algorithm for Update Scheduling in Streaming Data Warehouses" (IJCSIT) International Journal of Computer Science and Information Technologies, Vol. 5 (2) ,pp. 1082-1085,2014.

Author Profile

Shraddha S. Deshpande received the B.E degree in computer Science from P.E.S. College of Engineering from Aurangabad in 2010, at present appearing for M.E degree in Computer Science and Engineering department at Marathwada Institute of Technology, Aurangabad. Her research interest in Algorithms used for updates in streaming data warehouses by scalable scheduling.

Licensed Under Creative Commons Attribution CC BY