International Journal on Recent Researches in Science, Engineering & Technology (IJRRSET)
A Journal Established in early 2000 as National journal and upgraded to International journal in 2013 and is in existence for the last 10 years. It is run by Retired Professors from NIT, Trichy. Journal Indexed in JIR, DIIF and SJIF.
Research Paper
Available online at: www.jrrset.com
UGC Approved Journal No: 45483
Received on: 17.01.2018 Published on: 29.01.2018 Pages : 39-42

ISSN (Print) : 2347-6729
ISSN (Online) : 2348-3105
Volume 6, Issue 1
January 2018.
JIR IF : 2.54
DIIF IF : 1.46
SJIF IF : 1.329

CONCURRENT TRAIL USING DATA WAREHOUSE

Dhanunath.R[1], Neethu Krishna[2],Freeshma Karunan[3]
1Department of Computer Science and Engineering,
Sri Vellappally Natesan College of Engineering, KTU, India
[2,3]Assistant Professor, Department of Computer Science and Engineering, Sri Vellappally Natesan College of Engineering, KTU, India

Abstract

The aim of the project is to update scheduling in streaming data warehouses, which combines the features of traditional data ware houses and data stream systems. Warehouse tables are horizontally partitioned, where the separation of recent and historical data are done. Updating of jobs are handling through append mode. Scheduling decisions depends on the effect of update jobs on data streams. The main innovation is the Max Benefit Look Ahead algorithm for handling the large and heterogeneous job sets. Dynamic views not handled and efficient handling of multiple jobs at the same time is the problems addressed in the current data mining field. The project concerns this criterion in an effective manner. Tables in a streaming warehouse are partitioned by time, thus for many classes of views, updates only need to access one or a few of the most recent partitions. It considers the sequence that never leaves the system idle. It considers future jobs scheduled to arrive before the current job with highest marginal benefit is expected to complete. An important property of the data streams in this motivating application is that they are append-only in nature.

Keywords: Streaming data warehouse, Data staleness, Data freshness.

1. Introduction

The 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 a materialized view defined as an SQL query over one or more tables. Each base or derived table T_j has a user-defined priority p_j and a time-dependent staleness function S_j (τ) that will be defined shortly. 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 modelled as an atomic, non preemptible task. The purpose of an update scheduler is to decide which of the released update jobs to execute next, the need for resource control prevents from always executing update jobs as soon as they are released.

The warehouse completes all the update jobs thus data cannot be dropped. Furthermore, if multiple updates to a given table are pending, they must be completed in chronological order. Thus it cannot load a batch of new data into a table if there is an older batch of data for the table that has not yet loaded. In practice, warehouse tables are horizontally partitioned by time so that only a small number of recent partitions are affected by updates. Historical partitions are usually stored on disk or in

memory. Updates of base tables simply append new records to recent partitions. Affected partitions of derived tables may be recomputed from scratch or updated incrementally. In addition to being updated regularly as new data arrive, derived tables often store large amounts of historical data. It can reduce the number of partitions per derived table by using small partitions for recent and large partitions for historical data.

In some cases, it may want to update several tables together. For instance, if a base table is a direct source of a set T of many derived tables, it may be more efficient to perform a single scan of this base table to update all the tables in T. To do so in this model, here define a single update job for all the tables in T. In the existing framework it is difficult to pre-empt, which for a heterogeneous workload is to allow preemptions and also transparent updating and deletion of row are not maintained. In the proposed system it includes the notion of average staleness as a scheduling metric and presented scheduling algorithms designed to handle the complex environment of a streaming data warehouse.

2. Related Works

Streaming warehouses maintains 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. This has many applications which include online stock trading, Credit card detection or telephone card detection. Streaming warehouses in this context focused on Extract-Transform-Load process [1]. Each view reflects a consistent state of its base data [2], even if different base tables are scheduled for updates at different times [3]. The real-time community has developed the notion of Pfair scheduling for real-time scheduling on multiprocessors [4]. This framework helps to enable on continuously inserting a streaming data feed at bulk-load speed [5]. The great deal of data warehousing has focused on efficient maintenance of various classes of materialized views [6].

3. Proposed Description

In the proposed method, the notion of average staleness as scheduling metric and presented scheduling algorithms designed to handle the complex environment of a streaming data warehouse. Max Benefit with Lookahead algorithm is used for choosing the next job to be executed. In this framework it considers as follows:J_i the released job with the highest marginal benefit. For each J_k whose expected release time r_k is within E_i of the current time and whose marginal benefit is higher than that of J_i. For each J_k whose expected release time r_k is within E_i of the current time and whose marginal benefit is higher than that of J_i. For each set S of released jobs J_m such that $r_k \leq \sum_m (E_m) < E_j$.

$$B[s] = \frac{\sum(p_m \Delta F_m) + p_k \Delta F_k}{\sum(E_m) + E_k)}.$$

If max_sB[s] > marginal benefit of J_i then it schedule the job with highest marginal benefit from set $S^* = argmax_S$B[S] else it would schedule the job J_i. Here J_1 has the highest marginal benefit, but a job with a higher marginal benefit J_3 will be released before J_1 is completed. The Lookahead algorithm needs to find alternate sequences of jobs whose running time is between 1 and 2, and compute their B[S] values intuitively; B[S] represents the marginal benefit of running all the tasks in S followed by J_3. The Lookahead algorithm employs a number of heuristics to prune the number of alternate job sets. It considers the sequences that never leave the system idle $r_k <= \sum(E_m)$). It considers future jobs scheduled to arrive before the current job with highest marginal benefit is expected to complete.

Database Monitoring

For analyzing the current status of the tables and to view the update as quickly as possible, database monitoring would be implemented. At first the job is created based on the user requirement and needs. The job is created with the parameters such as priority and staleness, which would be different for different applications. After the job is created, it would be sent to the server for its approval, if the job

is approved it would be placed in job pool. Scheduling of jobs depend on Max Benefit with Lookahead algorithm. The job would be scheduled according to the increasing Max Benefit with Lookahead value. Here it helps the user to view the Jobs details and Data scheduling value to update in every minute. It helps the client and server interaction to the database which is used to dynamically create the table based on the server entering value. On the server side, approval of jobs for scheduling takes place. The parameters of the job as priority and staleness are taken in to account for its approval. The main feature of this framework is it enables to see all the jobs skill based details in single search. The user to search the events based on the requirements but the admin to view all users and admin created Jobs details. The approved job is placed in job pool. System meter is provided for analyzing the CPU and disk usage. If a job set is heterogeneous with respect to the periods and execution times, scheduler performance is likely to benefit if some fraction of the processing resources are guaranteed to short jobs.

The execution time of an update job is a function of the amount of new data to be loaded. Let n be the time interval of the data to be loaded. Then define the execution time of an update job J_i as :

$$E_i(n)=\alpha_i + \beta_i * n$$

Where α_i corresponds to the time to initialize the ETL process, acquire locks, and β_i represents the data arrival rate. Clearly α_i and β_i may vary across tables. Then can estimate the values for α_i and β_i may vary across tables. Then can estimate the values for α_i and β_i from recently observed execution times; the value of n for a particular update job may be approximated by its freshness delta. A new update job is released whenever a batch of new data arrives, the multiple jobs may be pending for the same table if the warehouse was busy with updating other tables. For now, it assume that all such instances of pending update jobs are merged into a single update job that loads all the available data into that table. This strategy is more efficient than executing each such update job separately because here pay the fixed cost α_i only once.

This framework uses the extension of Max Benefit algorithm [7]. The benefit of executing a job J_i may be defined as $p_i \Delta F_i$ that is its priority weighted freshness delta. Since the marginal benefit does not depend on the period, it can use Max Benefit for periodic and aperiodic update jobs. During transient overload, low-priority jobs are deferred in favour of high priority jobs. When the period of transient overload is over, the low-priority jobs will be scheduled for execution. Since they have been delayed for a long time, they will have accumulated a large freshness delta and therefore a large execution time and therefore might block the execution of high-priority jobs. A solution to this problem is to chop up the execution of the jobs that have accumulated a long freshness delta to a maximum of their periods. This technique introduces a degree of preemptibility into long jobs, reducing the chances of priority inversion [8]. Materialized view hierarchies can make the proper prioritization of jobs difficult. If a high priority view is sourced from a low priority view, then it cannot be updated until the source view is which might take a long time since the source views need to inherit the priority of their dependent views. The interaction of queries and updates in a firm real-time database, i.e., how to install updates to keep the data fresh [9] but also ensure that read transactions meet their deadlines [10]. In the context of web databases, this aims to balance the quality of service of read transactions against data freshness [11]. Focus on streaming data warehousing includes the system design [12], real-time ETL processing [13], and continuously inserting a streaming data feed at bulk-load speed [14].

Performance evaluation:

Time management is acquired positively during this framework. Also consistency of data is achieved through the database monitoring. The simulator framework generates periodic data arrivals, monitors data arrivals, monitors track usage, and generates a call to the scheduler on every data arrival or job completion. An advantage of using a simulation rather than a prototype of a streaming data warehouse is the ability to perform a very large number of tests in reasonable time and under precisely controlled conditions.

4. Analysis

This framework has many applications in different sectors. With the database monitoring concept used in this context enables the view of both the current and historical data which is stored in the database. Efficient handling of multiple jobs at a time increases the performance of the system. Consistency of the data is achieved through this scenario. Staleness parameter of the data helps to avoids disambiguates of handling the multiple jobs at the same time by the server.

5. Conclusion

This paper handles the effectiveness of time management of the data. Through this scenario consistency of data is also achieved. It formalized and solved the problem of nonpreemptively scheduling updates in a real-time streaming warehouse. The main feature of this 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

6. References

1. N. Polyzotis, S.Skiadopoulos, P. Vassiliadis, A. Simitsis, and N.-E. Frantzell, "Supporting Streaming Updates in an Active Data Warehouse," Proc. IEEE 23rd Int'l Conf. Data Eng. (ICDE), pp. 476-485, 2007.
2. L. Colby, A. Kawaguchi, D. Lieuwen, I. Mumick, and K. Ross, "Supporting Multiple View Maintenance Policies," Proc. ACM SIGMOD Int'l Conf. Management of Data, pp. 405-416, 1997.
3. Y. Zhuge, J. Wiener, and H. Garcia-Molina, "Multiple View Consistency for
4. DataWarehousing," Proc. IEEE 13th Int'l Conf.Data Eng. (ICDE), pp. 289-300, 1997.
5. S. Baruah, N. Cohen, C. Plaxton, and D. Varvel, "Proportionate Progress: A Notion of
6. Fairness in Resource Allocation," Algorithmica, vol. 15, pp. 600-625, 1996.
7. S. Baruah, "The Non-preemptive Scheduling of Periodic Tasks upon Multiprocessors," Real Time Systems, vol. 32, nos. 1/2, pp. 9-20, 2006.
8. A.Gupta and I. Mumick, "Maintenance of Materialized Views:Problems, Techniques, and Applications," IEEE Data Eng. Bull., vol. 18, no. 2, pp. 3-18, 1995.
9. L.Golab, T.Johnson, "Scalable Scheduling of Updates in Streaming Data Warehouses", vol. 24, NO.6, 2012.
10. A.Burns, "Scheduling Hard Real-Time Systems: A Review," Software Eng. J., vol. 6, no. 3, pp. 116-128, 1991.
11. 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.
12. K. D. Kang, S. Son, and J. Stankovic, "Managing Deadline Miss Ratio and Sensor Data Freshness in Real-Time Databases," IEEE Trans. Knowledge and Data Eng., vol. 16, no. 10, pp. 1200-1216, Oct.2004.
13. P. Tucker, "Punctuated Data Streams," PhD thesis, Oregon Health & Science Univ., 2005.
14. L. Golab, T. Johnson, J.S. Seidel, and V. Shkapenyuk, "Stream Warehousing with Datadepot," Proc. 35th ACM SIGMOD Int'l Conf. Management of Data, pp. 847-854, 2009.
15. N. Polyzotis, S. Skiadopoulos, P. Vassiliadis, A. Simitsis, and N.-E. Frantzell, "Supporting Streaming Updates in an Active Data Warehouse," Proc. IEEE 23rd Int'l Conf. Data Eng. (ICDE), pp. 476- 485, 2007.
16. C. Thomsen, T.B. Pedersen, and W. Lehner, "RiTE: Providing On-Demand Data for Right-Time Data Warehousing," Proc. IEEE 24th Int'l Conf. Data Eng. (ICDE), pp. 456-465, 2008.