Minimize Staleness and Stretch in Streaming Data Warehouses

S. M. Subhani[1], **M. Nagendramma**[2]

[1, 2]Department of CSE, BVSREC, Chimakurthy, A.P, India

Abstract: ***We study scheduling algorithms for loading data feeds into real time data warehouses, which are used in applications such as IP network monitoring, online financial trading, and credit card fraud detection. In these applications, the warehouse collects a large number of streaming data feeds that are generated by external sources and arrive asynchronously. We discuss update scheduling in streaming data warehouses, which combine the features of traditional data warehouses and data stream systems. In our setting, external sources push append-only data streams into the warehouse with a wide range of inter-arrival times. While traditional data warehouses are typically refreshed during downtimes, streaming warehouses are updated as new data arrive. 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.***

Keywords: Data warehouse maintenance, online scheduling

1. Introduction

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.

Data warehouses integrate information from multiple operational databases to enable complex business analyses. In traditional applications, warehouses are updated periodically and data analysis is done off-line [3]. In contrast, real time warehouses [1], also known as active warehouses [4], continually load incoming data feeds to support time-critical analyses. For instance, an Internet Service Provider (ISP) may collect streams of network configuration and performance data generated by remote sources in nearly real time. New data must be loaded in a timely manner and correlated against historical data to quickly identify network anomalies, denial-of-service attacks, and inconsistencies among protocol layers. Similarly, online stock trading applications may discover profit opportunities by comparing recent transactions in nearly real time against historical trends. Banks may be interested in analyzing incoming streams of credit card transactions to protect customers against identity theft. Since the effectiveness of a real time warehouse depends on its ability to ingest new data, we study problems related to data staleness. In our setting, each table in the warehouse collects data from an external source. The arrival of a set of new data releases an update that seeks to append the data to the corresponding table. Since existing data are not modified, the processing time of an update is at most proportional to the amount of new data.

Our first objective is to nonpreemptively1 schedule the updates on one or more processors in a way that minimizes the total staleness of all tables. Our first contribution answers a question implicit in [2] regarding the difficulty of this problem. We prove that even in the purely online model, any on-line non preemptive algorithm achieves staleness at most a constant factor times optimal, provided that no processor is ever voluntarily idle and provided that the processors are sufficiently fast.

2. System Model

2.1Warehousing Architecture

Figure 1 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. [5]

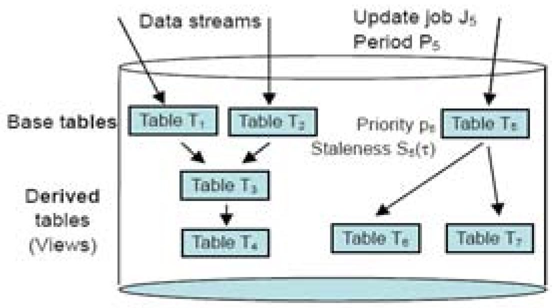

Figure 1: Stream data warehouse

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 a materialized view defined 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 $s_j(\tau)$ that will be defined shortly. Relationships among source and derived tables form a (directed and acyclic) 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.In practice, warehouse tables are horizontally partitioned by time so that only a small number of recent partitions are affected by updates [6][7].

2.2 Earliest Deadline First (EDF)

EDF has been proven to be an optimal uniprocessor scheduling algorithms .This means that if a set of tasks is unschedulable under *EDF*, then no other scheduling algorithm can feasible schedule this task set. The *EDF* algorithm chooses for execution at each instant in the time currently active job(s) that have the nearest deadlines. The *EDF* implementation upon uniform parallel machines is according to the following rules, No Processor is idled while there are active jobs waiting for execution, when fewer then ***m*** jobs are active, they are required to execute on the fastest processor while the slowest are idled, and higher priority jobs are executed on faster processors. In Earliest Deadline First scheduling, at every scheduling point the task having the shortest deadline is taken up for scheduling. A task is schedule under *EDF*, if and only if it satisfies the condition that total processor utilization (*Ui*) due to the task set is less than 1.

The Aim of this work is to provide a sensitivity analysis for task deadline context of multiprocessor system by using a new approach of EFDF (Earliest Feasible Deadline First) algorithm. In order to decrease the number of migrations we prevent a job from moving one processor to another processor if it is among them higher priority jobs. Therefore, a job will continue its execution on the same processor if possible (*processor affinity*). The result of these comparisons outlines some situations where one scheme is preferable over the other. Partitioning schemes are better suited for hard real-time systems, while a global scheme is preferable for soft real-time systems.

The final EDF – partitioned scheduling algorithm is following

```
1. Sort the released jobs by the local algorithm
2. For each job $j_i$ in sorted order
   a) If $j_{i's}$ home track is available, schedule $j_i$ on its home track
   b) Else, if there is an available free track, schedule $j_i$ on the free track
   c) Else, scan through the tracks r such that $j_i$ can be promoted to track r
      i) If track r is free and there is no released job remaining in the sorted list for home track r,
      ii) Schedule $j_i$ on track r
   d) Else, delay the execution of $j_i$
```

3. Minimizing Staleness

We call an algorithm eager, or work-conserving, if it leaves no processor idle while at least one pending update exists. We first state the rather-inscrutable Theorem 3.1, followed by an easy-to-read corollary, which implies that for any C < (v3 - 1)/2, there is a constant (dependent on C) such that the staleness of any eager algorithm is at most that constant factor times optimal, provided that each ai is at most Cp/t.

Theorem 3.1. Fix p, t. For any and d such that 0 <, d < 1, define C, d = vd (1 -)/v3 > 0. Given p processors and t tables, pick any a such that a/ [1 - a/ (p/t)] = C, d p/t. Then the penalty incurred by an eager algorithm is at most (1 + a) 2(1/ 4) (1/ (1 - d)) times LOW, provided that each ai = a.

Since LOW is a lower bound on the staleness achieved by any algorithm, even the optimal, prescient one, and penalty is an upper bound on the staleness achieved by any eager algorithm, the corollary implies the claimed competitiveness Proof: B be the set of batches in this run. For some batch Bi B, let ci be the length of the first update, di be the wait time, and bi be the total length of the batch, i.e., the sum of the lengths of its updates. Clearly,

ci = bi = ci + di, (1)

since ci = bi is obvious and since ci + di is the duration in time from the start (not release) time of the first job in the batch till the update for the batch starts, and this duration is clearly at least the length bi of the batch. For the penalty of this batch, denoted by i, we take the square of the own time, i.e., the length ci of the first update plus the wait time di plus the processing time of the entire batch:

i= [(ci+di) +abi] 2, by the definition of penalty (2)
= (1+a) 2(ci+di) 2, by (1). (3)

Figure 2 illustrates the quantities bi, ci and di using the same example as that in Figure 1; in particular, we consider the batch consisting of updates arriving at times ri, 1 and ri, 2.

Let A be the set of all updates. From the definition of LOW, each update i A has a budget of a2 units, where ai is the length of update i. Our proof requires the use of a charging scheme. A charging scheme specifies what fraction of its budget each update pays to a certain batch. Let us call a batch Bi tardy if ci < (ci + di) (where comes from Theorem 3.1); otherwise it is punctual. Let us denote the corresponding sets by Bt and Bp respectively. More formally, a charging scheme is a matrix (vij) of nonnegative values, where vij shows the extent of dependence of batch i on the budget available to batch j, with the following two properties.

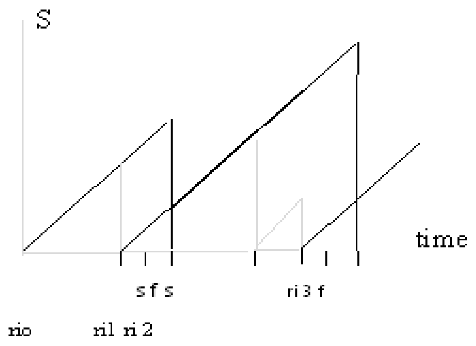

Figure 2: A plot of the staleness of table i over time

4. Conclusion

In this paper, we studied the complexity of scheduling data-loading jobs to minimize the staleness of a real time stream warehouse. We proved that any on-line non-preemptive algorithm that is never voluntarily idle achieves a constant competitive ratio with respect to the total staleness of all tables in the warehouse, provided that the processors are sufficiently fast.

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.

The **main feature** of our framework is the ability to reserve resources for short jobs that dften correspond to important frequently refreshed tables while avoiding the inefficiencies associated with partitioned scheduling techniques. **Feature work** is needed for choosing the right scheduling granularity when it is more efficient to update multiple tables together.

References

[1] L. Golab, T. Johnson, J. S. Seidel and V. Shkapenyuk, Stream Warehousing with Data Depot, SIGMOD 2009, 847-854.
[2] L. Golab, T. Johnson, and V. Shkapenyuk, Scheduling Updates in a Real Time Stream Warehouse, ICDE 2009, 1207-1210.
[3] W. Labio, R. Yerneni, and H. Garcia-Molina, Shrinking the Warehouse Update Window, SIGMOD1999, 383-394.
[4] N. Polyzotis, S. Skiadopoulos, P. Vassiliadis, A. Simits is, and N.-E. Frantzell, Supporting Streaming Updates in an Active Data Warehouse, ICDE 2007, 476-485.
[5] Scalable Scheduling of Updates in Streaming Data Warehouses Lukasz Golab, Theodore Johnson and Vladislav Shkapenyuk AT&T Labs – Research, Florham Park, NJ, 0793.
[6] N. Folkert, A. Gupta, A. Witkowski, S. Subramanian, S. Bel lamkonda, S. Shankar, T. Bozkaya, and L. Sheng, optimizing refresh of a set of materialized views, VLD B 2005, 1043- 1054.
[7] L. Golab, T. Johnson, J. S. Seidel, and V. Shkapenyuk Stream warehousing with Data Depot, SIGMOD 2009, 847-854.