Load Shedding for Aggregation Queries over Data Streams

Brian Babcock Mayur Datar Rajeev Motwani

Department of Computer Science
Stanford University, Stanford, CA 94305
{babcock, datar, rajeev}@cs.stanford.edu

Abstract

Systems for processing continuous monitoring queries over data streams must be adaptive because data streams are often bursty and data characteristics may vary over time. In this paper, we focus on one particular type of adaptivity: the ability to gracefully degrade performance via "load shedding" (dropping unprocessed tuples to reduce system load) when the demands placed on the system cannot be met in full given available resources. Focusing on aggregation queries, we present algorithms that determine at what points in a query plan should load shedding be performed and what amount of load should be shed at each point in order to minimize the degree of inaccuracy introduced into query answers. We report the results of experiments that validate our analytical conclusions.

1 Introduction

As information processing systems grow in complexity, they become increasingly difficult to administer and control. Consequently, an emphasis of much recent work (see Section 6 for examples) has been to create systems that are to some degree *self-regulating*, reducing the time, effort, and expertise required of system administrators. Adaptive, self-regulating systems are particularly appropriate for environments where data and query rates are dynamic or unpredictable, as is frequently the case for data stream processing systems. Many sources of streaming data are quite bursty [10, 19, 20], meaning data rates and system load are liable to be highly variable over the lifetime of a long-running continuous query. In this paper, we focus on one type of adaptivity, the ability to gracefully degrade performance when the demands placed on the system cannot be met given available resources, in the context of continuous monitoring queries over data streams.

Data streams arise naturally in a number of monitoring applications in domains such as networking (traffic engineering, intrusion detection, sensor networks) and financial services (arbitrage, financial monitoring). These data stream applications share two distinguishing characteristics that limit the applicability of standard relational database technology: (1) the volume of data is extremely high, and (2) on the basis of the data, decisions are arrived at and acted upon in close to real time. Traditional data processing approaches, where effectively data is loaded into static databases for offline querying, are impractical due to the combination of these two factors.

Many data stream sources (for example, web site access patterns, transactions in financial markets, and communication network traffic) are prone to dramatic spikes in volume (e.g., spikes in traffic at a corporate web following the announcement of a new product or the spikes in traffic experienced by news web sites and telephone networks on September 11, 2001). Because peak load during a spike can be orders of magnitude higher than typical loads, fully provisioning a data stream monitoring system to handle the peak load is generally impractical. However, in many monitoring scenarios, it is precisely during bursts of high load that the function performed by the monitoring application is most critical. Therefore, it is particularly important for systems processing continuous monitoring queries over data streams to be able to automatically adapt to unanticipated spikes in input data rates that exceed the capacity of the system. An overloaded system will be unable to process all of its input data and keep up with the rate of data arrival, so *load shedding*, i.e., discarding some fraction of the unprocessed data, becomes necessary in order for the system to continue to provide up-to-date query responses. The question we study is which tuples to drop, and where in the query plan to drop them, so that the degree of inaccuracy in the query answers introduced as a result of load shedding is minimized.

The use of load shedding as a technique to achieve graceful degradation in the face of unmanageable system load has been suggested in earlier work on data stream systems ([5, 21, 22]). While some heuristics for load shedding have been proposed earlier, a systematic approach to load shedding with the objective of maximizing query accuracy has

Proceedings of the 20th International Conference on Data Engineering (ICDE'04)
1063-6382/04 $ 20.00 © 2004 IEEE

been lacking. The main contributions of this paper are:

1. We formalize the problem setting for load shedding as an optimization problem where the objective function is minimizing inaccuracy in query answers, subject to the constraint that system throughput must match or exceed the data input rate.

2. We describe a load shedding technique based on the introduction of random sampling operations into query plans, and we give an algorithm that finds the optimum placement of sampling operations for an important class of monitoring queries, viz., sliding window aggregate queries over data streams. Our algorithm takes into account the effects of operator sharing among queries having common sub-expressions.

We also present extensions to our techniques to handle set-valued queries and queries with differing quality-of-service requirements.

Overview of Approach We propose a technique involving the introduction of load shedding operators, or *load shedders*, at various points in the query plan. Each load shedder is parameterized by a sampling rate p. The load shedder flips a coin for each tuple that passes through it. With probability p, the tuple is passed on to the next operator, and with probability $1 - p$, the tuple is discarded. To compensate for the lost tuples caused by the introduction of load shedders, the aggregate values calculated by the system are scaled appropriately to produce unbiased approximate query answers.

The decisions about where to introduce load shedders and how to set the sampling rate for each load shedder are based on statistics about the data streams, including observed stream arrival rates and operator selectivities. We use statistical techniques similar to those used in approximate query processing systems to make these decisions in such a way as to achieve the best attainable accuracy given data input rates.

Road Map The rest of this paper is organized as follows: We begin in Section 2 by formally stating our model and assumptions. Section 3 presents our algorithms for optimal sampling for different scenarios, and some extensions to the basic algorithm are described in Section 4. Section 5 gives the results of an experimental evaluation of our techniques. We describe related work in Section 6, before ending with our conclusions in Section 7.

2 Problem Formulation

The class of continuous monitoring queries that we consider are *sliding window aggregate queries*, possibly including filters and foreign-key joins with stored relations, over continuous data streams. Monitoring queries involving joins between multiple streams or non-foreign-key joins between streams and stored relations are comparatively rare in practice and therefore not considered in this paper.[1] A *continuous data stream* S is a potentially unbounded sequence of tuples $\{s_1, s_2, s_3, \ldots\}$ that arrive over time and must be processed online as they arrive. A *sliding window aggregate* is an aggregation function applied over a sliding window of the most recently-arrived data stream tuples (for example, a moving average). The aggregation functions that we consider are SUM and COUNT, though our techniques can be generalized to other functions such as AVG and MEDIAN. Sliding windows may be either *time-based*, meaning that the window consists of all tuples that have arrived within some time interval w of the present, or *tuple-based*, meaning that the window consists of the N most recently arrived tuples. A *filter* is a local selection condition on tuples from a data stream.

We believe this class of queries is important and useful for many data stream monitoring applications, including network traffic engineering, which we will use as an example application domain throughout this paper. Network analysts often monitor sliding window aggregates covering multiple timescales over packet traces from routers, typically filtering based on the internet protocol used, source and destination port numbers, autonomous subnetwork of packet origin (identified via user-defined functions based on longest prefix matching of IP addresses), and similar considerations [17]. Foreign-key joins or semijoins with stored relations may be used in monitoring queries to perform filtering based on some auxiliary information that is not stored in the data stream itself (e.g., the industry grouping for a security in a financial monitoring application). For our purposes, such joins have the same structure and effects as an expensive selection predicate or a user-defined function.

Most data stream monitoring scenarios involve multiple continuous queries that must be evaluated as data streams arrive. Sharing of common sub-expressions among queries is desirable to improve the scalability of the monitoring system. For this reason, it is important that a load shedding policy take into account the structure of operator sharing among query plans rather than attempting to treat each query as an isolated unit.

The input to the load shedding problem consists of a set of queries $q_1, \ldots, q_n$ over data streams $S_1, \ldots, S_m$, a set of query operators $O_1, \ldots, O_k$, and some associated statistics that are described below. The operators are arranged into a *data flow diagram* (similar to [5]) consisting of a directed acyclic graph with m source nodes representing the

[1] Furthermore, hardness results [7] for sampling in the presence of joins apply equally to the load shedding problem, making load shedding for non-foreign-key joins a difficult problem.

Proceedings of the 20th International Conference on Data Engineering (ICDE'04)
1063-6382/04 $ 20.00 © 2004 IEEE

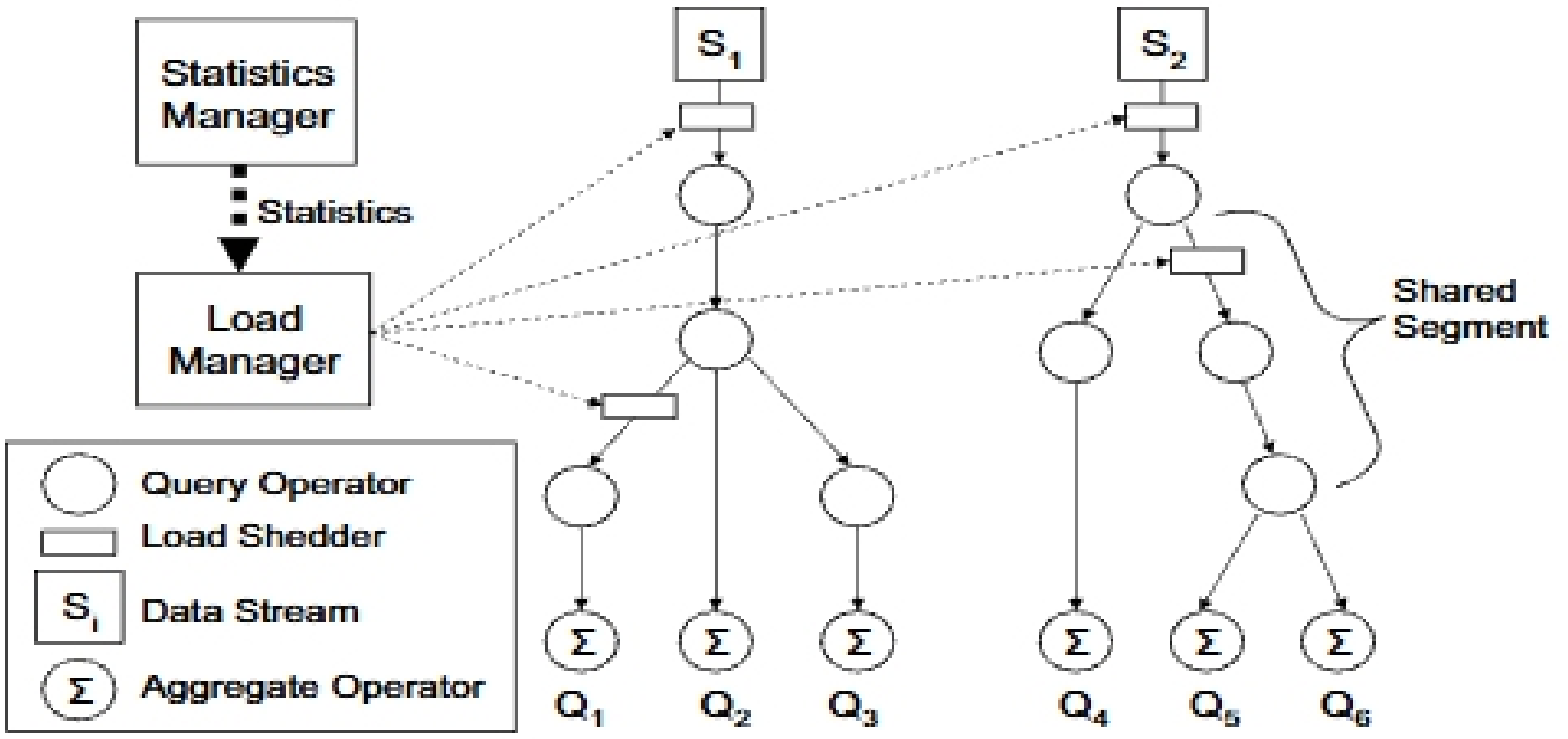

Figure 1. Data Flow Diagram

data streams, n sink nodes representing the queries, and k internal nodes representing the query operators. (Please refer to Figure 1.) The edges in the graph represent data flow between query operators. For each query q_i, there is a corresponding path in the data flow diagram from some data stream S_j though a set of query operators $O_{i_1}, O_{i_2}, \ldots, O_{i_p}$ to node q_i. This path represents the processing necessary to compute the answer to query q_i, and it is called the *query path* for query q_i. Because we do not consider joins between data streams, the data flow diagram can be thought of as being composed of a set of trees. The root node of each tree is a data stream S_j, and the leaf nodes of the tree are the queries that monitor stream S_j. Let $T(S_j)$ denote the tree of operators rooted at stream source S_j.

Every operator O_i in the data flow diagram is associated with two parameters: its selectivity s_i and its processing time per tuple t_i. The selectivity of an operator is defined as the ratio between the number of output tuples produced by the operator and the number of input tuples consumed by the operator. The processing time per tuple for an operator is defined as the average amount of time required by the operator to process each input tuple. The last operator along any query path is a windowed aggregate operator. The output of this operator is the final answer to the query and therefore not consumed by any other operator, so the selectivity of such an operator can be considered to be zero. Each SUM aggregate operator O_i is associated with two additional parameters, the mean μ_i and standard deviation σ_i of the values in the input tuples that are being aggregated. The final parameters to the load shedding problem are the rate parameters r_j, one for each data stream S_j. Rate parameter r_j represents the average rate of tuple arrival on stream S_j, measured in tuples per unit time.

Although we have described these input parameters (selectivity, stream rate, etc.) as known, fixed quantities, in reality their exact values will vary over time and cannot be known precisely in advance. In STREAM [21], our data stream management system, we have a Statistics Manager module that estimates the values of these parameters. Our query operators are instrumented to report statistics on the number of tuples processed and output by the operator and the total processor time devoted to the operator. Based on these statistics, we can estimate the selectivity and processing times of operators as well as the data stream rates. During times when statistics gathering is enabled, our SUM aggregation operator additionally maintains statistics on the sum and sum-of-squares of the aggregate values of tuples that it processes, allowing us to estimate the mean and standard deviation. As stream arrival rate and data characteristics change, the appropriate amount of load to shed and the right places to shed it may change as well. Therefore, in our system, estimates for the load shedding input parameters are periodically refreshed by the Statistics Manager, and load shedding decisions are periodically revisited.

2.1 Accuracy Metric

Let $A_1, A_2, \ldots, A_n$ be the answers to queries $q_1, q_2, \ldots, q_n$ at some point in time, and let $\tilde{A}_1, \tilde{A}_2, \ldots, \tilde{A}_n$ be the answers produced by the data stream monitoring system. If the input rates are high enough that load shedding becomes necessary, the data stream monitoring system

Proceedings of the 20th International Conference on Data Engineering (ICDE'04)
1063-6382/04 $ 20.00 © 2004 IEEE