Control-Based Load Shedding in Data Stream Management Systems

Yi-Cheng Tu and Sunil Prabhakar
Department of Computer Sciences, Purdue University

1 Introduction

Data stream management has attracted much enthusiasm from the database community in recent years. A very important characteristic of data stream management systems (DSMSs) is that both data arrival and query processing are continuous. As a result, the system needs to handle a large number of streams and queries concurrently. This brings the problem of maintaining Quality-of-Service (QoS) (i.e., parameters describing timeliness, reliability, and precision) in query processing and data refreshing. Important QoS parameters include: processing delay, data loss, and sampling rate, etc. Many applications of DSMS place strong real-time constraints on query processing. Hard (e.g., in stock price analysis) or soft (e.g., in network monitoring) deadlines are normally set and the value of query results drops dramatically if these deadlines are missed. However, delays in data processing in DSMSs are difficult to control due to resource sharing of multiple streams/queries and unpredictable pattern of resource consumption of these entities. System overloading, which inevitably degrades QoS, is very common in such environments.

Load shedding techniques have been exploited to overcome system overloading [6, 1]. The idea is to decrease the input load by discarding data tuples from the query engine. Essentially, it targets at maintaining critical QoS (e.g., processing delay) with the price of less critical ones (e.g., data loss). The specific problem we are interested in is: how can we control processing delays to be under an appropriate level with as little data loss as possible? Current DSMSs employ simple heuristic methods in attempt to solve this problem. For example, Fig. 1 shows the load shedding algorithm utilized in Aurora [6] and STREAM [1] DSMSs.

```
1 **for** every $T$ time units
2     **if** incoming load $L$ is greater than
           the system processing capacity $L_0$
3       **do** shedding load with amount $L - L_0$
4     **else** allow $L_0 - L$ more load to come
```

Figure 1. Generic load shedding algorithm

The intuition behind the above algorithm is: to control processing delays, we need and only need to make sure that the input load to DSMS is smaller than its processing capacity. The above algorithm works well when input load changes infrequently. However, this is generally not the case in practice. Streaming data are intrinsically dynamic with respect to the arrival patterns [5, 7]: input load may fluctuate within a wide range. Among others, we show two examples where the current method (Fig.1) will fail to maintain processing delay properly.

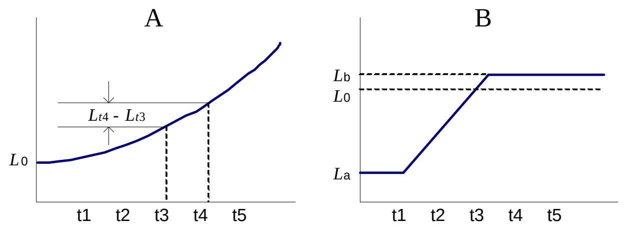

Figure 2. Two different cases of load arrivals.

Example 1. As a very typical situation in streams, the incoming data rate may keep increasing in a lasting period of time (Fig. 2A). At any time t_i, we sense the load L_{t_i} is higher than L_0 and decide to shed load with amount $L_{t_i} - L_0$ in the next period. The problem is: the incoming load in the next period $L_{t_{i+1}}$ is greater than L_{t_i} thus we are not discarding enough load and the number of outstanding data tuples q increases. If this keeps happening, processing delays of tuples can increase unboundedly.

Example 2. Note that system would normally tolerate tuple delays to a certain level (denoted as y_r). When the incoming data rate changes from a stable small value L_a to a higher value L_b that is slightly greater than L_0, the algorithm will discard data with amount of $L_b - L_0$ (Fig. 2B). However, more data should be allowed to enter the DSMS because the queue is empty (i.e., $q = 0$) before the change. In this case, although the delay time is smaller than the target value y_r, the extra data loss is unnecessary.

The above examples are just two of many cases that any simple and intuitive load shedding algorithm cannot handle. Things are more complicated when we consider the fact that the per-tuple processing cost also changes over time. Another thing missing from Fig. 1 is how to set the monitoring period T. Therefore, we need a systematic solution to adjust quality levels of streams in response to fluctuations of system status and input rates.

Related Work. Very little attention has been paid to the development of a unified framework to support quality of service in DSMSs. In [1], load shedding strategies that minimize the loss of accuracy of aggregation queries in DSMS are discussed. Closely related to our work is the QoS-driven load shedding introduced in the context of the Aurora project [6]. In Aurora, three basic questions are raised for load shedding in DSMSs: *when*, *how much*, and *where* to discard load. The *Aurora* work focuses more on the last question. In order to decide where in the query network load should be discarded, a load shedding roadmap (LSRM) is constructed. The LSRM generates a list of possible shedding plans and sorts them by their expected returns on CPU cost to utility loss ratio. Simple heuristics are used to determine the time and amount of load shedding (recall Fig. 1). To some extent, our work aims to provide a better answer to the first two questions (i.e., *when* and *how much*) in a systematic way. We have presented the basic approaches and some simulation results in a short paper [7].

2 Overview of Our Approach

We follow a push-based data stream query processing model, similar to those of STREAM [10] and Aurora [4]. As in [6], we assume CPU power is the bottlenecking resource. We allow the system administrator(s) to specify a (either fixed or time-varying) target delay y_r. Given a monitoring period T[1], the goal is to make the average delay y of all tuples arriving within each period remain under y_r, with as little data loss as possible. In case of overloading, we perform load shedding by discarding tuples from the waiting queues in the network of operators. The specific problems we need to solve are the time to shed load and the amount of load to be discarded. We assume i) data arrival rate (denoted as f_i) fluctuates all the time and it is impossible to predict it; and ii) average processing (CPU) cost (denoted as c) of tuples also changes over time.

2.1 A Solution Based on Feedback Control

We view quality adaptation under the DSMS setup as a *control* problem. Generally, the term *control* refers to the operations to manipulate some outputs of a system by adjusting the inputs to it. Here the output is the average tuple delay y, and input is the incoming load f_i to the DSMS. As we've mentioned (Fig. 2), the failure of the simple solution is due to its being unaware of the system status (i.e., the current value of y and q). We remedy this by using feedback control techniques: we use the current output signal (feedback) in making decisions for the next period of time. Feedback control is used in many areas of engineering to deal with systems subject to unpredictable disturbances.

Why feedback control? Denote the nominal model of the controlled system as a (i.e., a transfer function in the diagram in Fig. 3). The goal of adding a controller to the picture is to make the system output y track the target value y_r. Thus, the best thing one can do is to design a controller with transfer function $1/a$ because we would then have $y = y_r \frac{1}{a} a = y_r$[2], i.e., the output is exactly the target value. However, real-world dynamic systems are always affected by input and output uncertainties (i.e., d_i and d_o, respectively) named *disturbances*, and system modeling errors d_m. Considering these, the actual output becomes

$$y = y_r + y_r \frac{1}{a} d_m + (a + d_m) d_i + d_o. \tag{1}$$

Clearly, the control suffers from poor accuracy due to the existence of d_i, d_o, and d_m, and there is no easy way to reduce their effects. On the other hand, if we use y as a feedback and send the difference between y_d and y (i.e., *control error*) as input to the controller (Fig. 3B), the value of y in the next round becomes

$$y = \frac{K(a + d_m) y_r}{1 + K(a + d_m)} + \frac{(a + d_m) d_i}{1 + K(a + d_m)} + \frac{d_o}{1 + K(a + d_m)}$$

where K is the feedback controller designed based on the system model. Now we see that, if the controller is set to be large enough, i.e., $K(a + d_m) >> 1$, y is approximately $y \approx y_r + \frac{1}{K} d_i + \frac{1}{K} d_o$ and the effects of d_i, d_o, and d_m can be reduced by a factor of K. Due to the existence of a complete loop in the control diagram (Fig. 3B), feedback control is also called *closed-loop* control while the control in Fig. 3A is called *open-loop* control. Note that the algorithm shown in Fig. 1 is open-loop in nature.

Closed-loop in our problem. Taking the advantages of feedback control in dealing with environmental/internal dynamics, we propose to build a load shedding framework based on a feedback control loop. Specifically, we design a concrete control loop model (Fig. 4) from the preceding conceptual model (Fig. 3B). The controlled system is the query engine of the DSMS. Periodically, we do the following: first, the output signal y - average delay of tuples in each period - is looped back. The control error ($e = y_r - y$) is sent to the controller. Then the controller generates the control signal u (based on e) and send it to the load shedder. The latter will decide where in the query network to shed/add load such that the total input load to the query engine equals u. Note that the controller is the only new component to be built (on top of current DSMSs). The fluctuations in the arrival of input data and the variable processing

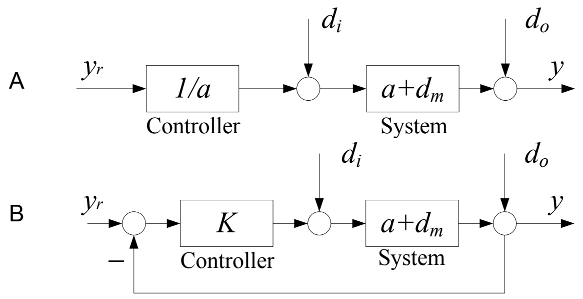

Figure 3. Open-loop (A) and feedback (B) control systems.

[1]We will talk about the choice of T later.

[2]All control-related analysis is performed in the z-domain [9] where convolution in the diagram becomes a multiplication.

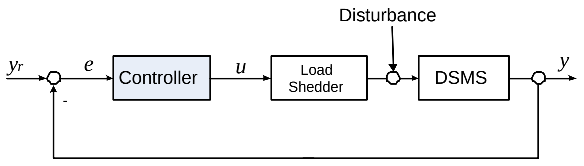

Figure 4. The closed-loop load shedding framework.

cost c are treated as disturbances. The quantity u is essentially our answer to the question of *how much* load to shed.

It is easy to see that the critical part of the control loop is the controller. Feedback control theory provides a series of mathematical tools to analyze and tune the controller so that guaranteed performance can be achieved. We use the open-source Borealis [3] DSMS (whose query engine is derived from Aurora) as our experimental platform.

2.2 Issues

The following technical issues/challenges have to be addressed in order to make the idea work.

Modeling Borealis. Specifically, we need a dynamic model that describes the response of the system to input signals. Since models based completely on rigorous analysis are very difficult to derive in studying complex systems such as a DSMS, we use *system identification* [9] techniques to generate the model. The basic idea is to initially model the plant as a difference equation with unknown parameters. Then we can determine the order and parameters of the difference equation experimentally. Generally, we can start from some knowledge of the model and then go back and forth between experiments and hypotheses thus a more refined model is built each time.

Controller design. As we've mentioned, the control goal is to let y closely track the target value y_r. Although user satisfaction is not affected when we have $y < y_r$, it implies that more data is lost than what is necessary (under a overloading situation). Thus, we should set our design goal to fast convergence, meaning that the controlled system is capable of bringing y back to the desired value y_r very quickly when y deviates from y_d in either direction. For this purpose, a controller design based on pole placement is desirable to achieve guaranteed performance[3].

Determining control period. This essentially answers the question of 'when' to shed load. Most likely, we need to find a tradeoff between the following two rules in selecting T: i) *Sampling theorem.* In order to manage the disturbances, we need to effectively capture the moving trends of those disturbances. According to information theory, a higher sampling frequency (i.e., shorter period) is preferred; ii) *Stochastic feature of system signals.* We take both the signals y and c as expectations of those for a series of tuples. Therefore, control period cannot be too short. Otherwise, the y and c signals we measure will subject to errors due to the lack of enough samples.

[3]System poles are the roots of the denominator polynomial of the closed-loop transfer function. The location of system poles can tell how fast and smoothly the system responds to inputs therefore it is directly related to our design goal of fast convergence.

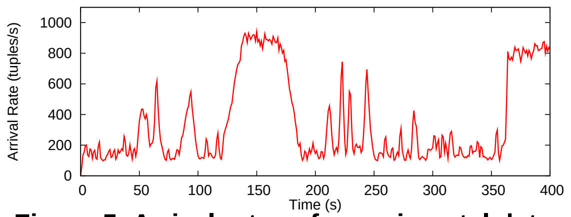

Figure 5. Arrival rates of experimental data.

2.3 Contributions

1. We propose the idea of using feedback control techniques to provide quantitative guidance to load shedding in data stream systems. This approach, as compared to current (open-loop) methods, is more effective in maintaining QoS under dynamic inputs and internal factors;

2. We develop a system model that describes system responses to incoming data inputs. Based on this model, we design a feedback controller via well-established techniques in control theory. The controller dynamically determines what amount of load should be discarded/admitted; and

3. We implement and evaluate our control-based load shedding framework in a real DSMS - the Borealis data stream manager. Unlike previous works that only experimented on simulators [8], our work on Borealis provides better understanding of how control theoretical techniques can be applied in a real software system.

3 Preliminary Results

By extensive analysis and experiments, we develop the following model for the Borealis system (in the z-domain):

$$G(z) = c \cdot T/(z-1). \tag{2}$$

Setting the convergence rate to three control periods, we develop the following feedback controller.

$$u(k) = \frac{0.4e(k) - 0.31e(k-1)}{cT} + 0.8u(k-1) \tag{3}$$

where $u(k), e(k)$ are control signal and control error of the current control period and $u(k-1), e(k-1)$ are their counterparts in the previous period. Note that we treat c as a constant at this moment and leave it as future work to handle time-variant c. Due to space limitations, we skip the details about the derivation of the above equations.

Experiments. We implemented our control-based load shedding framework in Borealis. We test both the original and controller-enhanced Borealis systems with synthetic and real streaming data. The synthetic data are generated such that the arrival rate follows a Pareto distribution to simulate various levels of burstiness. A sample data stream is

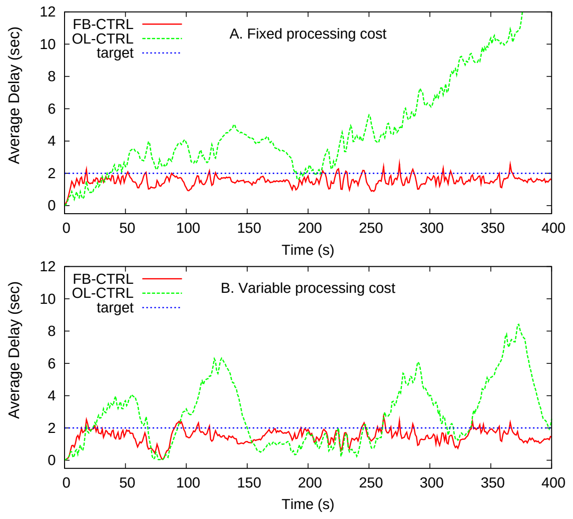

Figure 6. Performance of different load shedding algorithms.

shown in Fig. 5. First we test both systems with a query network with a constant average processing cost. As we can see from Fig. 6A, our approach (FB-CTRL) achieves much better control of average delay: most of the recorded y values are under the target value of two seconds. For those rare occasions when $y > y_r$, the errors are small. The simple method (OL-CTRL), on the other hand, generates much more delay violations and renders the system unstable after the 200th second (i.e., y increases unboundedly). One thing to point out is: data loss for both methods are approximately the same (1 to 0.985). Other situations unchanged, we also make the average processing cost c to change in a sinusoidal pattern. Again, the delays recorded (Fig. 6B) in the FB-CTRL method are under the target value with very few exceptions. For the OL-CTRL method, the average delays seem to fluctuate sinusoidally, following the pattern of the changes of c. Similar to the previous experiment, data losses are almost the same in both methods.

4 Future Work

In addition to the tasks listed in Section 2.2, this study can be extended in the following directions.

Adaptive control. So far we have worked on a system whose internal mechanisms are fixed. However, system model could evolve over time. For example, we ended up using the average processing cost c as a parameter in our system model. Although experimental results show that the system works well even under variable c (Fig.6B), we cannot prove the stability of the controller. A more convincing way to handle this would be to use a second (outer) loop to capture the dynamics of the system model itself and send it as a feedback to the current (inner) loop. By doing this, we can i) handle more dramatic variations of c, ii) get better control results in terms of fewer undershoots (i.e., the case of $y < y_d$) which could lead to less data loss.

Adaptation other than load shedding. Adaptation strategies other than load shedding are also very popular in dealing with overloading. Generally speaking, the goal of all adaptation methods is to adjust the load that the system needs to process. For example, in the case of sampling rate reduction, we generally accomplish this by resetting the width of some adaptive filters on the stream sources [2]. The relationship between the width of the adaptive filters and the resulting data sending rate is generally not linear and hard to quantify. We need to investigate the behavior of our system and decide if re-modeling is needed.

More Sophisticated Quality Model. In this paper, we set a target only for the processing delay and data loss is basically regarded as the cost of achieving the main control goal of maintaining delays. The control on data loss is done in a best-effort manner. A more palpable model would involve setting targets for multiple QoS dimensions thus a single-in-multi-out system model has to be utilized. The complexity of such models could make a solution following this path very difficult but it is worth further investigations.

5 Conclusions

In this paper, we study the problem of quality adaptation in data stream systems from a new angle. Our approach takes advantage of proven techniques from the field of feedback control theory. Compared to current solutions, our approach is designed to maintain tuple delays more effectively under dynamic incoming load and variable data processing costs. We have implemented our design in the Borealis data stream manager and experimental results support our expectations on the performance of our solution.

References

[1] B. Babcock et al. Load Shedding for Aggregation Queries over Data Streams. In *Procs. of ICDE 2004*.

[2] C. Olston et al. Adaptive Filters for Continuous Queries over Distributed Data Streams. In *Procs. of ACM SIGMOD 2003*.

[3] D. Abadi et al. The Design of the Borealis Stream Processing Engine. In *Procs. of CIDR 2005*.

[4] D. Carney et al. Monitoring Streams - A New Class of Data Management Applications. In *Procs. of VLDB 2002*.

[5] M. Zhang et al. Data Mining Meets Performance Evaluation: Fast Algorithms for Modeling Bursty Traffi c. In *Procs. of ICDE 2002*.

[6] N. Tatbul et al. Load Shedding in a Data Stream Manager. In *Procs. of VLDB 2003*.

[7] Y.-C. Tu et al. Control-based Quality Adaptation in Data Stream Management Systems. In *Procs. of DEXA 2005*.

[8] K.-D. Kang et al. Managing Deadline Miss Ratio and Sensor Data Freshness in Real-Time Databases. *IEEE TKDE*, 16(10):1200–1216, October 2004.

[9] G. F. Franklin, J. D. Powell, and A. Emami-Naeini. *Feedback Control of Dynamic Systems*. Prentice Hall, 2002.

[10] T. S. Group. STREAM: The Stanford Stream Data Manager. *IEEE Data Engineering Bulletin*, 26(1):19–26, March 2003.