Buffer-preposed QoS Adaptation Framework and Load Shedding Techniques over Streams

Rui Zhou, Guoren Wang, Donghong Han,
Pizhen Gong, Chuan Xiao, and Hongru Li

Institute of Computer System, Northeastern University, Shenyang, China
wanggr@mail.neu.edu.cn

Abstract. Maintaining the quality of queries over streaming data is often thought to be of tremendous challenge since data arrival rate and average per-tuple CPU processing cost are highly unpredictable. In this paper, we address a novel buffer-preposed QoS adaptation framework on the basis of control theory and present several load shedding techniques and scheduling strategies in order to guarantee the QoS of processing streaming data. As the most significant part of our framework, buffer manager consisting of scheduler, adaptor and cleaner, is deliberately introduced and analyzed. The experiments on both synthetic data and real life data show that our system, which is built by adding several concrete strategies on the framework, outperforms existing works on both resource utilization and QoS assurance.

1 Introduction

Data stream applications such as network monitoring, on-line transaction flow analysis, intrusion detection and sensor networks pose tremendous challenges to traditional Database Management Systems (DBMSs). To Meet requirements of such scenarios, Data stream Management Systems (DSMSs) such as Aurora [7]/Borealis [8], STREAM [4], NiagaraCQ [10] and TelegraphCQ [11] are built for data management and query processing upon multiple, unbounded, continuous, time-varying input streams, which desire specific processing techniques different from fixed-size stored data sets.

One of the most significant characteristics of DSMSs is to provide assured QoS (such as tuple processing delay [1]) by load shedding [9] in order to cope with excessive incoming tuples and keep up with high-speed streams. In most scenarios, data arrival rates are fluctuating and unpredictable, and average per-tuple processing cost also varies due to inherent uncertainty of processing cost for different tuples, changes of queries (submitting new queries or cancelling old ones) or other urgent incoming tasks taking up part of CPU cycles. Consequently adaptivity is strongly demanded on building a DSMS to deal with these uncertainties and provide stable service for clients.

To achieve adaptivity, Aurora dynamically adjusts its shedding ratio (the fraction of tuples to be dropped) by plus or minus a step value. This solution may lead to overshoot or undershoot and also reacts slowly to bursty traffic. Tu [1]

addressed a control-based adaptation framework and solved the above problem. However, drawbacks still exist when the stream arrival rate fluctuates frequently around CPU's top processing ability, for the reason that if those dropped tuples can be stored temporarily rather than discarded during high stream speed cycles and taken out to be processed when speed falls down, shedding ratio will be reduced and more tuples can be evaluated. Thus we propose a novel buffer-preposed framework to offer a better DSMS adaptation and elaborate the strategies required in constructing a buffer manager. Section 2 investigates related works. The framework and buffer manager are introduced in Section 3 and 4, respectively. Experiments and conclusions are given in Section 5 and 6.

2 Related Work

There has been considerable work on data stream processing. The survey in [12] gives an overview of stream work, and has summarized the issues of building a Data stream Management System. Specialized systems have been mentioned in Section 1. The existing load shedding works can be classified into two categories.

The first one focuses on specific operators (or queries). In [2,3,5,13], approximate joins were extensively studied to give MAX-subset [5] outputs of unbounded streams by dropping the tuples, which produce less join results. The work [6] mainly discussed about minimize the degree of inaccuracy while shedding load on aggregation queries. Our work considers about the other category [1,8,9] that integrates the queries as a whole forming a query network. Data stream Manage Systems monitor the QoS of the outputs, and make corresponding decisions on when, where and how much load to shed. Aurora/Borealis takes advantage of its LSRM to determine the target and amount for adaptation and inserts drop operators into its query network to reduce resource usage. In TelegraphCQ, synopses are built to capture the properties of the tuples, which are dropped from triage queues [15] in case of excessive stream load. Tu et al. [1] proposed a novel framework that regards DSMS as a plant and adjusts incoming flow rate according to current system status by leveraging control theory. Loadstar [14] introduces load shedding techniques to facilitate classifying multiple data streams of large volume and high speed. There are also other topics related to stream applications, such as search for moving object trajectories [16], comparison and contrast between Lp-Norm and edit distance [17], and location-aware topology matching in P2P systems [18].

3 Buffer-preposed QoS Adaptation Framework

In this section, we propose a novel data stream processing system framework, which consists of two parts: upstream part and downstream part. The architecture is depicted in Figure 1. As for downstream part, query processor and CPU scheduler act as the fundamental evaluation components of DSMS, the same as operator network in Aurora/Borealis and STREAM. Monitor sends the QoS of output results to PI controller so that the controller could determine the

number of tuples ($C(k)$) to be injected into the query processor during the next monitoring cycle.

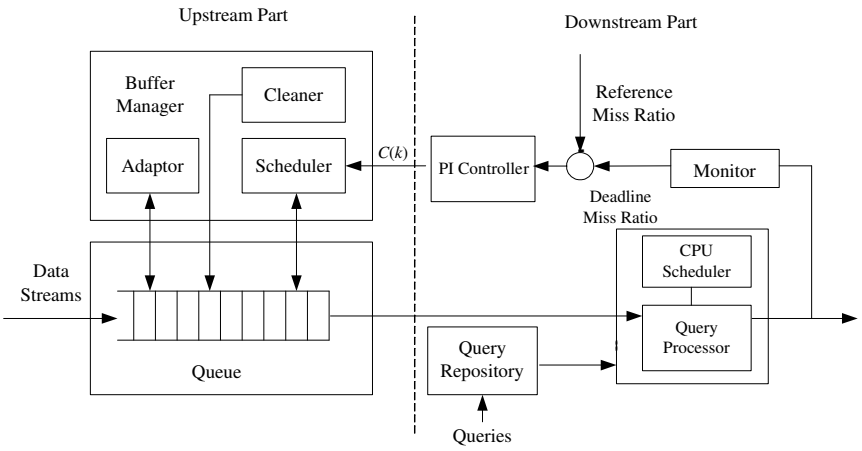


Fig. 1. Buffer-preposed QoS Adaptation Framework

Applying control theory results in a better adaptation, and details can be found in [1]. However, drawbacks still exist when the stream arrival rate fluctuates frequently around CPU's top processing ability as is mentioned in Section 1. Hence, we prepose a buffer regulated by a buffer manager at the upstream part in front of former classic DSMS (the downstream part) to eliminate the problem. Three modules are built in buffer manager: scheduler allocates memory resources and dispatches the tuples in and out of their corresponding queues; adaptor will shed load if the queues are about to overflow; and cleaner is utilized to purge of those QoS-violated tuples. Detailed illustrations are given in Section 4. Note that all the strategies applied in the three modules are orthogonal to our framework, they may be replaced by better ones in the future.

In this paper, we take deadline miss ratio [1] as an example of QoS, note that other QoS metrics also work in our framework. Firstly, *tuple delay* is defined as time elapsed between the generation of the tuple at source and the end of its processing in query processor. A violation of tuple delay requirement is called *deadline miss*, and *deadline miss ratio* is the fraction of missing tuples of the entire data stream. Concerning that, tuple delay is differently defined in [26], whereas both of them could make sense and one can be simply transformed to the other.

4 Buffer Manager

4.1 Scheduler

Scheduler acts not only as a resource manager, but also a communicator between buffer and query processor. Its utility and functions are described as follows:

1. Dynamically allocate memory resource for the incoming streams according to their arrival rates and average per-tuple processing cost. It is believed that higher arrival rate, lower processing cost and higher tuple priority will result in a longer queue in memory. We preset a proper queue length limitation for each stream, and reallocate resources only when remarkable changes of system state take place, such as joining of a new stream, leaving of an old one, or tremendous speed varying of an existing stream, while on the common circumstance, we shed some load to deal with temporary stream speed fluctuation, which will be mentioned in Section 4.2.
2. Receive incoming tuples, dispatch them to their corresponding queue according to some strategies, such as FIFO(First In First Out), EDF [19](Earliest Deadline First), MUF [20](Maximum Urgent First), and pass the number of tuples required by DSMS from the queues to downstream part in terms of a suitable ratio at each sampling cycle. Our work mainly focuses on the situation where tuple priority is not explicit or difficult to determine, which accords with most application scenarios. Therefore FIFO, EDF are studied and compared as our candidate scheduling strategies, noting that other strategies also work in our framework.

4.2 Buffer Adaptor

It may be lack of resources for system to store and process every tuple of the incoming streams, especially when the streams have high arrival rates. We provide a queue with a proper maximum length to each stream. If the queue is about to overflow, we drop some tuples to reduce load in order to make sure DSMS is under a normal state. In this section, we mainly focus on single stream adaptation, and it is easy to be applied to multi-streams. There are many adaptation strategies to maintain the queue. In this section, we will discuss some of them.

Tail drop, Drop front, Random drop If queue length reaches its predefined maximum limitation, the queue will not permit entering of future tuples, and load shedding will be set up. One of the following strategies will be adopted. Tail drop [21] (referred to as TD) means dropping those new incoming tuples, i.e. dropping from the end of the queue, while drop front [23] means dropping from the front of the queue. Random drop [22] is also easy to understand (dropping a tuple in the queue randomly without regarding its position).

Note that the above three strategies are performed only when the queue is full, we consider this condition to be obviously deficient as there is no preserved room for future important tuples. Moreover if stream rate arises and lasts for a period of time, late arriving tuples will most likely be dropped, which will lead to unfairness to some extent. RED′ and PID can avoid this problem.

RED′ RED(Random Early Detection) [24] is dramatically studied for Active Queue Management(AQM)in the field of computer network. Since strategies in each module are orthogonal with the framework, adaptor can be implemented

with any strategy. Now we will simply introduce RED′ adaptation scheme, a variation of RED. The average queue length is calculated by:

$$\tilde{q} = (1 - w_q) \cdot \tilde{q} + w_q \cdot q \tag{1}$$

where q is the current queue length and $\tilde{q}$ is the historical average length. w_q is weight factor, $w_q \in [0,1]$. If $\tilde{q} < min_{th}$ (min_{th}: predefined minimum length of queue), all arriving tuples will enter into the queue; if $\tilde{q} > max_{th}$ (max_{th}: predefined maximum length of queue), all these incoming ones will be dropped ; and if $min_{th} \leq \tilde{q} \leq max_{th}$, the shedding ratio p will be given in Equation 2, where max_p is a maximum shedding ratio.

$$p = max_p \cdot (\tilde{q} - min_{th})/(max_{th} - min_{th}) \tag{2}$$

PID PID [25] (Proportional, Integral and Differential) control is widely used in automatic control and system engineering. First, we define *occupying ratio* as *the number of tuples in queue divided by maximum queue length*. In order to reserve some space for future tuples, we assign an expected *occupying ratio* (denoted as r) as the reference and try to control the queue length to be $r \cdot L$. Symbols of this section are listed in Table 1.

Table 1. Symbols used in PID Adaptation

L	maximum queue length
Y	output signal: occupying ratio of queue
X_{in}	incoming rate of tuples entering the queue
X_{out}	output rate of tuples (i.e. CPU processing ability)
X	queue length increment (i.e. X_{in}-X_{out})
S	stream rate
U	load shedding ratio

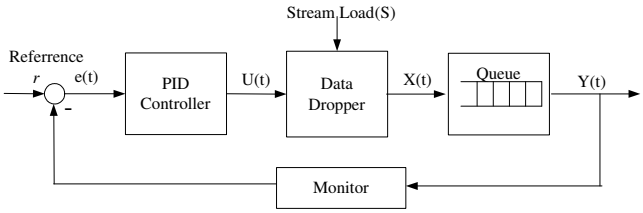


Fig. 2. The Feedback Control Loop

The blocks of closed-loop feedback control are given in Figure 2. As to queue, the plant, it is easy to obtain the following difference equation, which is utilized to model the variation of occupying ratio of the queue,

$$Y(k) = Y(k-1) + \frac{X(k)}{L} \tag{3}$$

and the transfer function in Z-domain is:

$$G(z) = \frac{Y(z)}{X(z)} = \frac{1}{L(1 - z^{-1})} \tag{4}$$

With respect to PID controller, the input signal is calculated as the following:

$$X(k) = K_P \cdot e(k) + K_I \sum_{j=0}^{k} e(j) + K_D \left[e(k) - e(k-1) \right] \tag{5}$$

and we can get controller transfer function:

$$C(z) = \frac{X(z)}{E(z)} = K_P + K_I \frac{1}{1 - z^{-1}} + K_D (1 - z^{-1}) \tag{6}$$

After analyzing the system's closed-loop transfer function (see Equation 7) by means of Root Locus or Frequency Response (available in MATLAB), we conclude that the system is marginally stable, and thus controllable. Parameters K_P, K_I, K_D can be determined through real experiments.

$$T(z) = \frac{C(z)G(z)}{1 + C(z)G(z)} \tag{7}$$

Considering the data dropper and the queue as a whole, we have:

$$X(k) = X_{in}(k) - X_{out}(k) = (1 - U(k)) \cdot S(k) - X_{out}(k) \tag{8}$$

and thus the controller passes the shedding ratio $U(k)$ to data dropper. $U(k)$ can be deduced from Equation 8: $U(k) = 1 - \frac{X(k)+C(k)}{S(k)}$, where $C(k)$ equals to $X_{out}(k)$, and $S(k)$, $C(k)$ can be predicted by $S(k-1)$ and $C(k-1)$ obtained in the last sampling period.

4.3 Cleaner

Cleaner detects the tuples in queues that will not satisfy the QoS of the queries submitted in DSMS if passed to CPU to be processed. For instance, if possible out-of-date tuples which may miss its deadline can be removed from queues by cleaner, CPU cycles will be saved to produce useful results. As is mentioned in previous sections, we adopt deadline miss ratio as the expected QoS of DSMS, and then the cleaning strategy should be invalidating (the same as removing) the tuples whose permitted tuple delay will be probably violated. Now we will introduce the rule of removing a possible outdated tuple. Table 2 gives out the variables used in the following subsections, here t stands for timestamp, and T stands for time period.

Cleaning Strategy At the kth monitoring cycle, the following condition should be satisfied if a tuple is supposed to be purged.

$$T_t^i + T_w^i(k) + T_p^i(k) > T_d \tag{9}$$

Here, T_t^i can be determined by $T_t^i = t_a^i - t_g^i$, note that t_a and t_g can be obtained when the corresponding tuple arrives. As average per-tuple processing cost may be variable in different monitoring cycles, we use $T_w^i(k)$ to associate T_w^i with cycle information k (also the same as to $T_p^i(k)$ and T_p^i) to make a more accurate estimation of waiting time. Details of estimating of $T_w^i(k)$ and $T_p^i(k)$ are presented in the next subsections.

Table 2. Variables used in Cleaning Strategies

t_g	generation timestamp of a tuple (at source)
t_a	arrival timestamp of a tuple (at destination)
T_t	transmission time of a tuple
T_w	waiting time of a tuple before it can be processed
T_p	processing time of a tuple in query networks
T_d	tuple delay predefined by query properties or users
$T_p^i(k)$	processing time of the ith tuple in queue at the kth monitoring cycle
$ET_w^i(k)$	estimated waiting time of the ith tuple in queue at the kth monitoring cycle

Estimation of $T_p^i(k)$ Several reasons that may result in fluctuations of average per-tuple processing cost have been given in Section 1. Therefore, we are supposed to formulate an estimation of $T_p^i(k)$, the predicted cost of a tuple in the kth monitoring cycle, to help judge if the tuple can be invalidated.

We utilize Single Exponential Smoothing to estimate the processing cost, and the formula is given below:

$$ET_p^i(k) = \alpha T_p^i(k-1) + (1-\alpha) ET_p^i(k-1) \tag{10}$$

where $0 < \alpha \leq 1$ and $k \geq 2$. $T_p^i(k-1)$ is the real average per-tuple processing cost in $(k-1)$th monitoring cycle, while $ET_p^i(k-1)$ is the estimated one. $ET_p^i(k)$ is the weighted sum of $T_p^i(k-1)$ and $ET_p^i(k-1)$. Weight α is called smoothing constant. When α is close to 1, latest processing ability dominates the estimation of $ET_p^i(k)$, while when α is close to 0, historical processing cost plays a more important role. We can set α according to specific application scenarios and a proper value can be found through a series of extensive experiments.

Estimation of $T_w^i(k)$ Note that, as for a certain tuple, there exists a long period of time during which the tuple is waiting in the queue after arrival in advance of the time being processed, it is crucial to perform an estimation of the waiting time. First we have:

$$T_w^i(k) = T_w^{i-1}(k) + T_p^{i-1}(k) \tag{11}$$

The waiting time of the ith tuple equals to the sum of waiting time and processing time of the $(i-1)$th tuple (suppose the ith and $(i-1)$th tuple are in the same processing cycle). Substitute $ET_w^i(k)$ and $ET_w^{i-1}(k)$ for $T_w^i(k)$ and $T_w^{i-1}(k)$ in Equation 11, we get:

$$ET_w^i(k) = ET_w^{i-1}(k) + T_p^{i-1}(k) \tag{12}$$

After recurrence substitution, we have:

$$ET_w^i(k) = \sum_{j=1}^{i-1} T_p^j(k) \tag{13}$$

Assume that estimated per-tuple processing cost in the kth monitoring cycle are the same, denoted as $ET_p(k)$, Equation 13 can be reduced to Equation 14.

$$ET_w^i(k) = (i-1) ET_p(k) \tag{14}$$

Summary of Estimation Replacing $T_w^i(k)$ and $T_p^i(k)$ with $ET_w^i(k)$ and $ET_p^i(k)$ in Equation 9, and also combining with Equation 10 and Equation 14, we can get the final conclusion, i.e. the formula of cleaning strategy:

$$T_t^i + i \cdot [\alpha T_p(k-1) + (1-\alpha) ET_p(k-1)] > T_d \tag{15}$$

Cleaning is performed from queue front to queue end, as a result, if a tuple is removed, its processing cost will not add in the estimation of waiting time of its successive tuple. Hence the estimation is probably accurate, and cleaner can reduce excessive stream load gracefully and effectively.

5 Experimental Results

To assess the practical performance of our model, we perform several sets of experiments on both synthetic and real life datasets. First we test our framework with various buffer scheduling strategies, adapting schemes, and cost estimation precisions. After certifying the validity of our framework, we pick a set of concrete strategies and build an adaptation system, referred to as CWB (Control With Buffer), and compare its performance with that of the system mentioned in [1], referred to as COB (Control withOut Buffer). The results show that CWB outperforms COB, achieving higher CPU utilization and lower deadline miss ratio, especially when stream arrival rate fluctuates wildly.

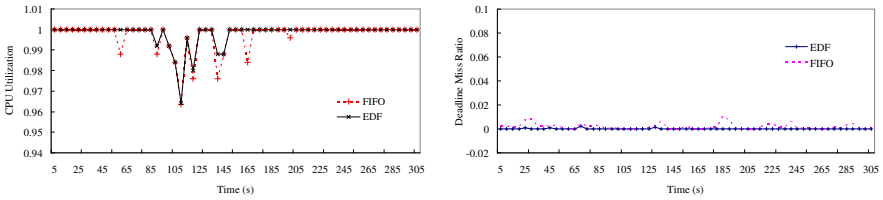
For synthetic data, we generate the tuples with arrival time following exponential distribution, i.e. in each time interval, the number of arriving tuples follows Poisson distribution. For real life data, we use the LBL-PKT-4 dataset from Internet Traffic Archive [27]. Monitoring cycle of downstream part (CPU) and upstream part (buffer) are 5s, 1s, respectively. Tuple delay is randomly chosen from 250ms, 500ms, 1s, 2s. Maximum queue length of incoming stream is set to be 150 tuples. We construct the downstream part according to [1] as stated in Section 3. Our discussion is mainly about the strategies in buffer manager (the upstream part) and comparison between CWB and COB.

5.1 Experiments on Buffer Manager

Buffer manager includes scheduler, adaptor and cleaner. For cleaner, we determine the smoothing constant α as 0.1 through experiments, which obtains a nearly optimal prediction of average per-tuple processing cost.

Scheduling Strategies As for scheduler, FIFO and EDF as tuple transmission strategies are thoroughly studied and compared. Here, we use synthetic dataset, permitted deadline miss ratio 0.01, and the simplest TD strategy for adaptor.

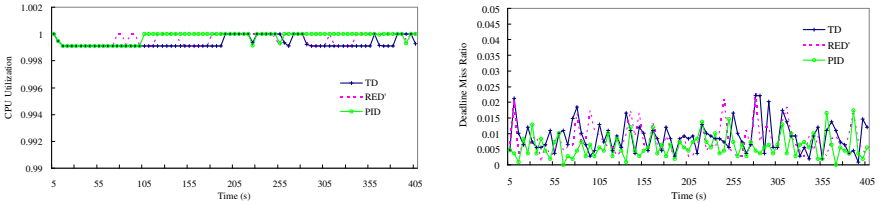
Figure 3 shows that EDF leads to higher CPU utilization and lower deadline miss ratio. That is because some miss-prone tuples can be laid at front and processed earlier by EDF, and thus deadline misses are reduced. Meanwhile this optimization could result in cleaning less tuples, which can reduce the load shedding ratio, i.e. increase the CPU utilization. Therefor EDF rather than FIFO is used in the following experiments.



(a) CPU Utilization (b) Deadline Miss Ratio

Fig. 3. Performance on FIFO and EDF

Adaptation Strategies As for adaptor, we perform the comparison among TD, RED′ and PID. We set the parameters in Equation 1 and 2 as: $w_p = 0.15$, $min_{th} = 30$, $max_{th} = 120$, $max_p = 0.1$ and the parameters in Equation 5 as: $K_P = 30$, $K_I = 18$, $K_D = 7.5$, the reference occupying ratio 0.5. System load is l.2 times of CPU processing ability.



(a) CPU Utilization (b) Deadline Miss Ratio

Fig. 4. Performance on TD, RED′ and PID

As is shown in Figure 4, PID performs the best while TD the worst, and RED′ is also acceptable but exhibits only a second choice. Figure 5 gives the frequencies of queue occupying ratio under three adapting strategies during 1000s execution. Here sampling cycle is 0.1s. The experimental result shows PID effectively reduces the occupying ratio of queue and provides reserved space for future tuples. Like EDF, PID is used exclusively in the following experiments.

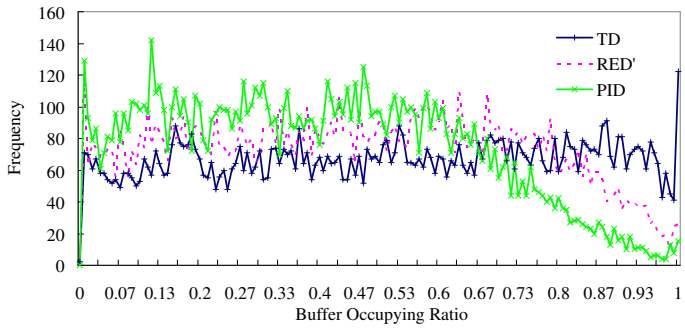
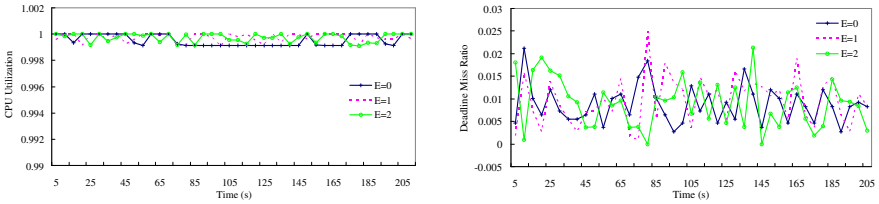


Fig. 5. Frequency of Buffer Occupying Ratio

Precisions of Cost Estimation We also investigate the performance of our framework with respect to different precisions of cost estimation. Let E stand for the precision level. E=0, $T_p = ET_p$; E=1, $T_p \in [0.3ET_p, 1.7ET_p]$; E=2, $T_p \in [0.1ET_p, 4.1ET_p]$. T_p, ET_p are real tuple processing cost and average tuple processing cost, respectively.



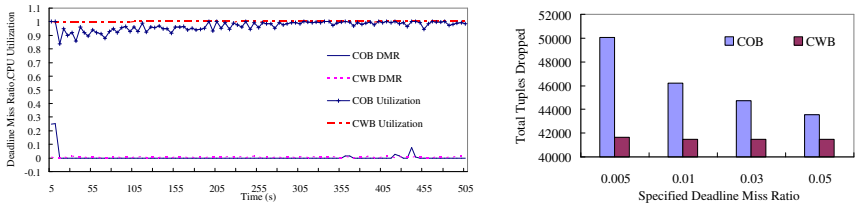
(a) CPU Utilization (b) Deadline Miss Ratio

Fig. 6. Different Precisions of Cost Estimation

From figure 6, we can draw the conclusion that, though the fluctuation of CPU utilization and deadline miss ratio arises as E increases, the affects are not prominent and system is capable to learn the variation and works stably and robustly. E is set to 0 unless particularly specified in other experiments.

5.2 Performance of CWB vs. COB

After discussing about buffer manager, we compare CWB and COB with parameters set according to Section 5.1. From figure 7(a), we know that CWB reacts faster, while COB converges to a steady state after 15 seconds. Although CWB undulates frequently, its fluctuating amplitude is trivial and can be ignored, whereas COB exhibits the opposite. As to CPU utilization, CWB can make full use of CPU resources for the reason that buffered data can be passed to the downstream part if CPU becomes idle. Figure 7(b) illustrates that dropped tuples of CWB is much fewer than COB, which means CWB has processed more tuples. This is accorded with higher CPU utilization of CWB. As the expected deadline miss ratio arises, fewer tuples are dropped. The decrease of CWB is not obvious due to effectiveness of cleaning strategy, for only those tuples prone to miss their deadlines are doomed to be removed.



(a) CPU Utilization and Deadline Miss Ratio (b) Total Tuples Dropped

Fig. 7. Performance on CWB and COB

5.3 Experiments on Real Life Datasets

As is shown in Figure 8, the performance of CWB is much better than that of COB contrasting to the result on synthetic dataset. That is because the arrival rate of LBL-PKT-4 TCP Traffic is more fluctuating than Poisson distribution data. Our framework provides a more adaptive solution in real life applications.

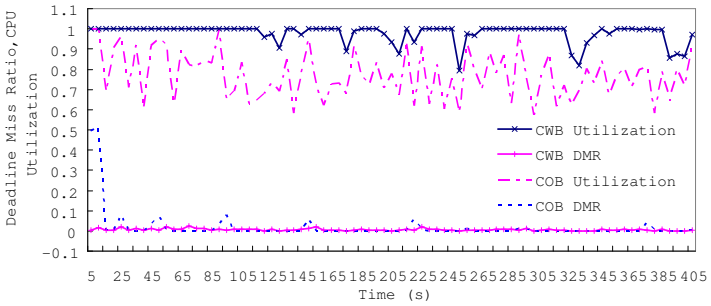


Fig. 8. Experimental Result on Real Life Dataset

6 Conclusions and Future Work

In this paper, we propose a novel QoS adaptation framework including upstream part and downstream part. In the upstream part, buffer manager including scheduler, adaptor and cleaner is deliberately introduced and analyzed. The experiments on both synthetic data and real life data show that our system, which is built by adding several concrete strategies on the framework, outperforms existing works on both resource utilization and shedding ratio. A promising direction for future work is to consider the priority of the tuples, i.e. incoming data are of different importance. On the circumstance, we are supposed to find out a set of specific strategies for scheduler, adaptor and cleaner.

Acknowledgement. This work is partially supported by National Natural Science Foundation of China under grant No. 60573089 and 60473074 and supported by Natural Science Foundation of Liaoning Province under grant no. 20052031.

References

1. Yi-Cheng Tu, Mohamed Hefeeda, Yuni Xia, and Sunil Prabhakar. Control-based Quality Adaptation in Data Stream Management Systems. In *Proceedings of DEXA, pages 746-755*, August 2005.
2. J. Kang, J. F. Naughton, and S. D. Viglas. Evaluating window joins over unbounded streams. In *Proc. of ICDE*, Bangalore, India, March 2003.
3. J.Xie, J. Yang, and Y. Chen. On joining and caching stochastic streams. In *Proc. 2005 ACM SIGMOD Conf.*, Baltimore, Maryland, USA, June 2005.
4. The STREAM Group. STREAM: The Stanford Stream Data Manager. *IEEE Data Engineering Bulletin* ,26(1):19-26,March 2003.

5. A. Das, J. Gehrke, and M. Riedewald. Approximate Join Processing Over Data Streams. In *Proc. 2003 ACM SIGMOD Conf.*, June 2003.
6. B. Babcock, M. Datar, and R. Motwani. Load Shedding for Aggregation Queries over Data Streams. In *Proc. 2004 Int. Conf. on Data Engineering*, Feb. 2004.
7. D. Abadi, D. Carney, et al. Aurora: a new model and architecture for data stream management. *VLDB Journal*, Vol.12(2),pp.120-139,2003.
8. D. Abadi, Y. Ahmad, M. Balazinska, U. Cetintemel, M. Cherniack, J. Hwang, W. Lindner, A. Maskey, A. Rasin, E. Ryvkina, N. Tatbul, and S. Zdonik. The Design of the Borealis Stream Processing Engine. In *Procs. of CIDR*, Jan. 2005.
9. N. Tatbul, U. Cetintemel, S. Zdonik. M. Cherniack, M. Stonebraker. Load Shedding in a Data Stream Manager. In *Proc. 29th int. Conf. on VLDB*, Sep. 2003.
10. J. Chen, D. J. DeWitt, F. Tian, and Y. Wang. NiagaraCQ: A scalable continous query system for internet databasses. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 379-390, 2000.
11. S. Chandrasekaran, A. Deshpande, M. Franklin, et al. TelegraphCQ: Continuous Dataflow Processing for an Uncertain World. In *Proc. of CIDR*, Jan. 2003.
12. B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *Proc. Principles of Database Systems (PODS)*, June 2002.
13. D. Han, R. Zhou, C. Xiao. G. Wang. *et al.* Load shedding for Window Joins over Data Streams. In *Proc. Int. Conf. on WAIM*, Hongkong, June 2006,
14. Yun Chi, Haixun Wang, and Philip S. Yu. LoadStar: Load Shedding in Data Stream Mining. In *Proc. Of the 31st VLDB Conf.*, pages 1302-1305, August 2005.
15. Frederick Reiss and Joseph M. Hellerstein. Data Triage: An Adaptive Architecture for Load Shedding in TelegraphCQ. In *Proc. of ICDE*, pages 155-156, April 2005.
16. L. Chen, M. T. Ösu, and V. Oria, Robust and Fast Similarity Search for Moving Object Trajectories. In *Proceedings of 24th ACM International Conference on Management of Data (SIGMOD'05)*, Baltimore, June 2005, pages 491-502.
17. L. Chen, R. Ng. On the Marriage of Lp-Norm and Edit Distance. In *Proc. of VLDB*, Toronto, Canada, August 2004, pages 792-803.
18. Y. Liu , X.Liu , L. Xiao , Li. Ni, and X. Zhang. Location-Aware Topology Matching in P2P Systems. *IEEE INFOCOM*, 2004
19. Abdelzaher, T., Sharma, V., Lu, C. A Utilization Bound for Aperiodic Tasks and Priority Driven Scheduling. *IEEE Trans. on Computers*, 53, 2004, 334-350.
20. D. B. Stewart and P. K. Khosla, Real-Time Scheduling of Sensor-Based Control Systems, Real-Time Programming, ed. by W. Halang and K. Ramamritham, (Tarrytown, New York: Pergamon Press Inc.), 1992.
21. S. Floyd, V. Jacobson. Traffic phase effects in packet-switched gateways. ACM Comp. Commun. Rev., Vol.21, No.2, pp.26-42, Apr. 1991.
22. Hashem E S. Analysis of random drop for gateway congestion control. MIT Lab for Computer Science : Technical Report, MIT 1989.
23. T Lakshman, A Neidhardt, and T Ott. The drop from front strategy in TCP and in TCP over ATM. In *IEEE INFOCOM*, San Francisco, CA, 1996. pp.1242-1250
24. S. Floyd and V. Jacobson. Random Early Detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, vol. 1, no. 4, August 1997.
25. G. F. Franklin, J. D. Powell, and A. Emami-Naeini. Feedback Control of Dynamic Systems. Prentice Hall, Massachusetts, 2002.
26. Yi-Cheng Tu, Liu Song, Sunil Prabhakar. Load Shedding in Stream Databases: A Control-Based Approach. March 2006. Technical report, Purdue University.
27. V. Paxson and S. Floyd. Wide-Area Traffic: The Failure of Poisson Modeling. *IEEE/ACM Transactions on Networking*, 3(3), pp. 226-244, June 1995. http://ita.ee.lbl.gov/html/contrib/LBL-PKT.html