2007-07-01

A Dynamic Attribute-Based Load Shedding Scheme for Data Stream Management Systems

Amit Ahuja

Yiu-Kai D. Ng
ng@cs.byu.edu

Original Publication Citation
Amit Ahuja and Yiu-Kai Ng, "A Dynamic Attribute-Based Load Shedding Scheme for Data Stream Management Systems." In Proceedings of the First International Workshop on Data Stream Processing (STREAM 27), pp. 2-25, July 1-6, 27, Silicon Valley, USA.

A Dynamic Attribute-Based Load Shedding Scheme for Data Stream Management Systems

Amit Ahuja
San Jose, CA 95111, U.S.A.
Email: amit_ahuja83@yahoo.com

Yiu-Kai Ng
Department of Computer Science
Brigham Young University
Provo, Utah 84602, U.S.A.
Email: ng@cs.byu.edu

***Abstract*—A data stream being transmitted over a network channel with capacity less than the data transmission rate of the data stream causes sequential network problems. In this paper, we present a new approach for shedding less-informative attribute data from a data stream to maintain a data transmission rate less than the network channel capacity[1]. A scheme for shedding attributes and their data, instead of tuples, becomes imperative in data stream load shedding, since shedding a complete tuple would lead to shedding informative attribute data along with less-informative attribute data in the tuple. Our load shedding approach handles intra-stream, as well as inter-stream, load shedding such that the former sheds less-informative attribute data in a single data stream, whereas the latter sheds less-informative attribute data from multiple streams. Our load shedding approach, (i) handles wide range of data streams in different application domains, and (ii) is dynamic in nature.**

I. Introduction

In recent years we have seen tremendous change in the way data are transferred over the network on which data streams are defined. As the high-rate stresses are introduced in the communication and computing infrastructure, so it may become impossible to (i) transmit the entire input to an application program, (ii) compute sophisticated functions on large pieces of inputs at the rate it is presented, and (iii) store temporarily or archive a data stream. To handle these problems, load shedding has been proposed as a solution.

We note that all data streams have a static schema, which means that the involved attributes and their corresponding data types do not change, but have a high dynamic data transmission rate. Moreover, some of the data values of an attribute vary more often than the values of other attributes, whereas others may remain nearly constant. Many applications process stream data in which all tuples are important, and some attribute values are more "informative" than the others. These scenarios demand a new approach towards shedding stream data by dropping less-informative attribute (values) in tuples, instead of the entire tuples, which is the *attribute-based load-shedding approach* presented in this paper.

Our load shedding approach differs from existing ones, since none of the existing ones consider a load shedding system for data streams with focus on dropping attributes in a tuple, rather than the tuple itself, to minimize information loss during the load shedding process. Our load shedding approach preprocesses all data values in a data stream S using moving averages, which serves the purpose of diminishing any rare irregularities in the data values of S. The data after preprocessing are used to compute the load shedding scheme of S, which comprises of the designated attributes and their data to be dropped. Furthermore, our load shedding approach is (i) *dynamic*, since the load shedding scheme that controls the data of S to be shed is computed and updated in real-time on each sliding window of S and (ii) *adaptive*, since it minimizes unnecessary load shedding scheme re-evaluations by re-evaluating in a pattern according to the data pattern of S which may cause changes of attribute ranking (in determining which attribute data to be shed) in S.

In Section II, we discuss related works in load shedding in data streams. In Sections III, IV, and V, we present our intra-stream and inter-stream load shedding approaches. In Section VI, we include the experimental results, and in Section VII, we give a concluding remark.

II. Related Work

Many efforts have been made in the past to handle load shedding mechanisms in data stream management systems ($DSMS$). Borealis [2] accomplishes load shedding by temporarily adding "drop" operators to the Borealis processing network as a way to shed tuples, either according to the values of the tuples or in a randomized fashion. Borealis uses a *static* QoS based approach for shedding tuples, whereas we use a dynamic approach to determine the attribute data to shed.

Data Triage [8], which treats load shedding as a problem to deal with bursty data arrival, sheds load to maintain low latency but keeps enough data to produce a relatively accurate picture of what happened during the burst, whereas Loadstar [5] considers a QoD-based load shedding, which is more adaptive than QoS. Loadstar, however, lacks the ability to control the communication rates of the data streams.

[6] reduce the amount of data in the sliding windows by prematurely evicting tuples from their windows, which is beneficial, since premature eviction saves any processing time which would have been spent on them before they would have been evicted later. [6], however, does not address the problem on dynamically scaling the optimization applied to reduce the window data to be shed.

[1]This work was partially funded by Cisco Systems, Inc.

Second International Conference on Digital Telecommunications (ICDT'07)
0-7695-2910-0/07 $20.00 © 2007 IEEE

In [4], the authors handle load shedding by introducing load shedder at various points in a query plan, in which every incoming tuple is passed onto the next load shedder with a probability, called the sampling rate. Aurora [1], [10], another $DSMS$, relies on QoS information to guide the load-shedding process by using a QoS monitor that watches over system performance and activates the load shedder whenever an overload is detected. Since Aurora uses the QoS information, which is predefined, it lacks the dynamic nature.

STREAM [7] uses query plans for handling data streams, and its CQL [3] is capable of handling relation-to-relation, stream-to-relation, and relation-to-stream operators. One of the relation-to-stream operators in CQL, the stream-sample operator, drops a specified fraction of stream tuples from its input queue based on a uniform random sample. The designers of STREAM use approximation techniques to reduce the synopsis and queue sizes, and further suggest that if the queues grow too large, then simply dropping packets could also help, which may not be optimal.

III. Our Load Shedding Approaches

In this paper, we propose two different load shedding strategies: the *intra-stream* and *inter-stream* load shedding. The intra-stream load shedding approach sheds less-informative attributes within a data stream to lower the data transmission rate at the source site to meet the limited data transmission channel capacity. Our intra-stream load shedding approach is unique, since it minimizes information loss by shedding less-informative attributes instead of tuples as complete tuples may contain less-informative as well as (more-) informative attributes.

The inter-stream load shedding, on the other hand, deals with shedding on various data streams when multiple streams have to be transmitted over a single local channel with the channel capacity less than the cumulative data rate of the data streams. The inter-stream load shedding may have been preceded by intra-stream load shedding individually on each of the involved data streams. Our inter-stream load shedding approach sheds zero or more less-informative attributes and their data from each involved data stream to bring the cumulative data transmission rate below the local channel capacity. The uniqueness of our inter-stream load shedding is (i) its ability to reduce the load on the transmission channel when multiple streams are being transmitted over the channel, and (ii) each involved data stream co-ordinates with the central load shedder and obtains information to perform (further) load shedding on its own data stream at the inter-stream site.

IV. The Intra-Stream Load Shedding Approach

Our intra-stream load shedding approach first identifies the less-informative attributes, i.e., attributes whose data vary less compared to the data of other attributes, in an incoming data stream. The major function of our intra-stream load shedding strategy, as well as our inter-stream load shedding approach, is to create the *load shedding scheme* of a data stream S, which sheds attribute data from S by the load shedder.

The load shedding scheme generation step is preceded by a preprocessing step that smoothens out any "irregularities" in the source data. (See Section IV-B for details.) After the preprocessing step, the *load shedding scheme* can be generated according to (i) the amount of attribute data to be shed based on the channel capacity at the data stream source site, and (ii) the chosen attributes (and their data) to be shed. To determine the attributes of a data stream S to be shed, we use *standard deviation* to compute the ranking amongst the attributes of S. Hereafter, the data from the source site is shed according to the load shedding scheme, which is re-evaluated in *real-time* and is *dynamic* in nature. Since a complete data stream can not be stored at the source site due to the continously flowing nature of streaming data, the load shedding scheme generation algorithm uses an excerpt of the data stream S, called *sliding window*, to generate the load shedding scheme of S. We use each current sliding window of S to capture such an excerpt of S, and subsequent excerpts can be used to update the load shedding scheme at various time intervals.

A. Sliding Window Size

Different segments of a data stream S, which convey up-to-the-moment information, are separated by the *cycle identifier* (CID, for short), which is defined as either a single attribute or a combination of attributes, of S. The values of a CID follow a fixed-length repetitive cycle in S, which consists of tuples in S such that the order of appearances of various CID values in the tuples fall in the same cycle, and the number of tuples in each fixed-length repetitive cycle in S is called the *cycle length* of S. The cycle length of S is treated as the *size* of each sliding window of S for load shedding. The data streams we deal with have a static schema and regular (repeated) patterns of tuples, such as, the attributes "Company Name", and "City Name" in data streams on Stock exchange information at a stock exchange, and weather information of capitals of different states in a country, demonstrate that cycles of data values in "real-world" data streams exists. In this section, we present a method in determining the CID of S and thus the cycle length of S.

The CID of S is detected during the training phase of S, before our load shedding system actually starts shedding data from S. All the attributes in S whose values individually follow a repetitive pattern *form* the set of replicated attributes ($RepA$s, for short) of S, and whenever a replicated attribute is detected, its *cycle length* is also recorded.

The replicated attributes in $RepA$s are partitioned into sets S_1,..., S_n ($n \geq 1$) according to the cycle length of each attribute such that each S_i ($1 \leq i \leq$ n) contains all the attributes with the same cycle length. Furthermore, all the attributes in each S_i have a one-to-one relationship with each other, i.e., the value of each of these attributes in a tuple t in S_i of S can uniquely identify the values for all the other attributes in t of S_i. Since the replicated attributes in each S_i have a one-to-one relationship with all the other attributes in S_i, only one (i.e., anyone) attribute from each S_i is required to form the chosen attributes, which yield the CID of S. Moreover,

A_1	A_2	A_3	A_4
a1	aaaa	a	29.91
a2	bbbb	b	27.96
a3	cccc	c	30.09
a1	aaaa	d	30.09
a2	bbbb	a	29.91
a3	cccc	b	29.80
a1	aaaa	c	29.77
a2	bbbb	d	30.09

A_1	A_2	A_3	A_4
a3	cccc	a	30.18
a1	aaaa	b	29.94
a2	bbbb	c	29.91
a3	cccc	d	30.00
a1	aaaa	a	30.00
a2	bbbb	b	30.09
a3	cccc	c	29.78
: :	: :	: :	: :

TABLE I
A SAMPLE DATA STREAM TRAINING DATA SET

the *multiplication* of the cycle length of each chosen attribute yields the fixed cycle length of S. The amount of computation required to identify the CID of S is one time.

Example 1: Consider the training data in Table I, in which attributes A_1, A_2, and A_3 are replicated with Cyclen(A_1) = 3, Cyclen(A_2) = 3, and Cyclen(A_3) = 4. Partitioning the attributes in RepA_s into sets, with each set having attributes of the same cycle length, yields sets $S_1 = \{A_1, A_2\}$ and $S_2 = \{A_3\}$. If A_1 is selected to form the CID, then the CID of S is $\{A_1, A_3\}$, and the cycle length of S is Cyclen(A_1)$\times$ Cyclen(A_3) = 3 $\times$ 4 = 12. □

B. Exponential Moving Average

Occasionally, data in a sliding window are found to have sudden and short-lived changes, deviating from the data stream variation properties, i.e., the variation of data values of attributes. For example, the values of "precipitation" would remain more constant than other attributes, such as "temperature," of a desert weather data stream over a long period of time, making "precipitation" less-informative, and the candidate attribute to be shed. However, due to sudden rains, there may be a significant change in precipitation. The precipitation over the next several hours, however, may remain relatively constant. Though this abrupt change does not really represent the weather conditions in the desert on a regular, consistent basis, it may cause other informative attributes, such as "temperature," being treated as less-informative (false positives) and less-informative attribute, i.e., "precipitation," being treated as informative (false negatives). In order to (i) smoothen the data, (ii) suppress any short and sudden change in data, and (iii) reduce the number of false positives and false negatives, *Moving Averages* (MAs) is employed as a preprocessing step for determining less- and more-informative attributes in the current sliding window of a data stream to smoothen the fluctuations so that distortions are reduced to a minimum in volatile data.

The two most popular types of MAs are the *simple moving average* (SMA) and *exponential moving average* (EMA)[2]. We have considered both SMA and EMA as the MA for the preprocessing step in our load shedding approach. Unlike SMA, EMA have the ability to stay closer to the actual data than SMA. Unarguably, the smoothened data is desired to be close to, and as accurate as possible to, the original data. Thus, EMA is an obvious choice as the MA for our preprocessing step to smoothen the data in a current sliding window.

[2]http://www.stockcharts.com/education/IndicatorAnalysis/indic_movingAvg.html

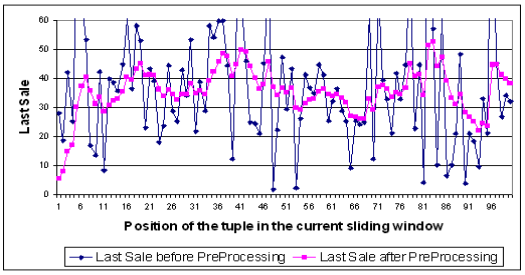


Fig. 1. Values of the *Last Sale* attribute of a stock exchange data stream, quotes.nasdaq.com/quote.dll?page=nasdaq100, collected on September 13, 2005, before and after preprocessing

Example 2: Comparing the data values of the "Last Sale" attribute in a stock exchange data stream in Figure 1, before and after the preprocessing step using EMA, we can clearly see that the curve for the training data values after preprocessing is smoother than the values before preprocessing. □

C. Load Shedding Scheme Generation and Re-Evaluation

Using the first sliding window of a data stream S, the first load shedding scheme of S is created. For the subsequent updates to the load shedding scheme of S, only the current sliding window of S is used. There are two design issues in intra-stream load shedding scheme generation: (i) how much data should be shed, and (ii) which attributes should be shed.

1) Amount of data to shed: According to various studies in computer networks [9], it is known that every channel has a capacity depending on the noise and bandwidth of the channel. If data are transmitted at a rate higher than the capacity of the channel, then data errors and collisions occur exponentially. To overcome these problems, the rate should be lower than the capacity of the channel, and if needed, attributes and their corresponding data are shed at the source site to bring down the data transmission rate.

Our intra-stream load shedding approach is designed to maintain a transfer rate $R^{'}$ not larger than its corresponding channel capacity C. Whenever $R > C$, attributes and their data are shed from its data stream being transmitted over the network, starting with shedding the less-informative attributes such that the transfer rate R falls to $R^{'}$ ($\leq C$). Depending on the sizes of different attributes, one or more less-informative attributes are shed so that R falls to $R^{'}$ ($\leq C$). The amount of data to be shed is r = R - C. Number of attributes to shed to attain the required data shedding r depends on the sizes of the individual attributes, since different attributes may have different sizes.

2) Attribute Shedding Using Standard Deviation: Having determined the amount of attribute data r ($r \geq 0$) to be shed from a sliding window W of a data stream S, we must decide which attributes should be shed from W. We start out by determining a ranking amongst the attributes in W using standard deviation SD, which calculates how spread out the values in a list of data is. The values that are more closely bound, i.e., having less variation (*smaller SD*) in its data values, are "less-informative," whereas values which are less closely bound, i.e., having more variation (*larger SD*) in its data values, are "more-informative." We apply SD to the data

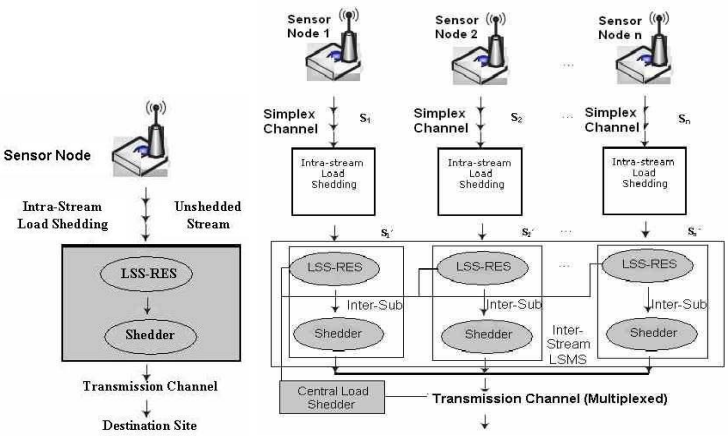
values of each attribute A in W to calculate how closely the data values of A are. After the SD for each attribute in W is computed, all the attributes are ranked, with attributes having $lower\ SD$ ranked $higher$ and attributes having $higher\ SD$ ranked $lower$.

Example 3: Consider the three sample attributes A, B, and C with data items <0, 0, 14, 14>, <0, 6, 8, 14>, and <6, 6, 8, 8>. The SDs are 7, 5, and 1, respectively. The attributes are ranked as A, B, and C, from high to low. □

3) Re-Evaluation of a Load Shedding Scheme: The load shedding scheme of a data stream S requires regular re-evaluation as the standard deviations of different attributes in subsequent sliding windows of S may change, causing the ranking amongst the attributes to change. We may adopt a simple non-adaptive load shedding scheme re-evaluation algorithm to re-evaluate the load shedding scheme at regular intervals. One major problem with using a non-adaptive re-evaluation algorithm is that if the time interval is $short$, the source site would re-evaluate the load shedding scheme too $often$, which imposes a lot of burden on the source site in terms of computational time required for re-evaluations. However, if the time interval is too $large$, the source site would not be re-evaluating the load shedding scheme often $enough$, which creates the risk of an obsolete load shedding scheme being used for a long time.

Our $adaptive$ re-evaluation algorithm starts out with a very small re-evaluation time interval, referred as the *original time interval*. For the first time, the proposed algorithm re-evaluates an existing load shedding scheme after waiting for the original time interval, and then checks if the re-evaluated (i.e., the newly generated) load shedding scheme of the current sliding window of the same data stream with smoothened data (due to EMA preprocessing) is differed (in terms of attributes to be shed) from the previous load shedding scheme. If the attributes to be shed are the same, the time interval is $doubled$ so that the re-evaluation is invoked after a longer interval. However, if the attributes to be shed are different, then (i) the time interval is reset to the original time interval, since a change in the load shedding scheme has just been detected, and we anticipate changes in the load shedding scheme in near future, and (ii) the load shedding scheme is updated. The time interval keeps growing in its usual manner every time the anticipated change in the attributes to be shed is proved incorrect.

Our adaptive load shedding scheme re-evaluation algorithm will notice the change in the load shedding scheme that its non-adaptive counterpart may not notice. Consider a data stream such that attribute A is the least-informative attribute during the first thirty minutes of every hour and attribute B is the least-informative attribute during the last thirty minutes of every hour. Assume that one attribute needs to be shed and a non-adaptive load shedding scheme re-evaluation algorithm is invoked every hour, starting five minutes past the first hour. Since A is the least-informative attribute during the first thirty minutes of every hour, every time the load shedding scheme re-evaluation algorithm is invoked, A is found to be the least-informative attribute and is shed, and the load shedding scheme never changes. In such a scenario, this non-adaptive re-evaluation algorithm would fail to notice the change in the load shedding scheme.



(a) Intra-stream (b) Inter-stream

Fig. 2. Architecture of our load shedding system

Consider our adaptive load shedding scheme re-evaluation algorithm, in the same scenario. Every time our adaptive load shedding re-evaluation algorithm is invoked at the first five minutes, i.e., Δt of the current clock hour, it finds A as the *least-informative* attribute, and it would double the current value of Δt. Eventually, Δt would reach a value such that the current clock time + Δt would fall within the last thirty minutes of the current clock hour, resulting in B being detected as the *least-informative* attribute. At this point, the load shedding scheme is updated, and Δt is reset to the original time interval, i.e., five minutes. Thus, our adaptive load shedding re-evaluation algorithm would notice the change in the load shedding scheme, which cannot be accomplished by the non-adative counterpart.

4) The Architecture of our Intra-Stream Load Shedding Sub-System: Figure 2(a) shows the intra-stream load shedding architecture of our DSMS. The current sliding window of a data stream S is fed into the (intra-stream) load shedding scheme generation and re-evaluation sub-system, denoted *LSS-RES*, which (i) preprocesses the data in the current sliding window, (ii) computes the rankings of the attributes of S, (iii) determines the amount of attribute data to be shed, and (iv) generates and re-evaluates the (intra-stream) load shedding scheme. Hereafter, the unshed current sliding window from the data stream is shed by the *shedder*, if needed. A data stream with or without undergone shedding by the shedder is referred as an '*intra stream*'.

V. Inter-Stream Load Shedding

Our inter-stream load shedding management system, denoted *Interstream-LSMS*, which consists of (i) multiple Inter-Subs, one for each intra stream for (further) load shedding, and (ii) a *central load shedder*, which manages the transmission of multiple intra streams over a single multiplexed channel where the cumulative data rate of all the intra streams is sometimes more than the multiplexed channels capacity. The central load

shedder notifies each Inter-Sub with the amount of data to be (further) shed from its intra stream.

The basic architecture of our Interstream-LSMS is shown in Figure 2(b). Each Inter-Sub consists of the same basic components as in the intra-stream load shedding sub-system. Unshed source data streams, labeled S_1,. . ., S_n in Figure 2(b), are originated from different sources and transmitted through different channels, called *simplex channels* (after data are shedded, if necesssary), to reach the Interstream-LSMS, as intra streams, i.e., S'_1,. . ., S'_n. Each *LSS-RES* of an Inter-Sub individually determines the less-informative attribute values in its intra stream to be shed, and together with the *LSS-RESs* in other Inter-Subs reduce the combined data transmission rate to be less than the multiplexed channel capacity.

Amount of Data to be shed

In our Interstream-LSMS, since each *LSS-RESs* of an Inter-Sub cannot directly sense the capacity of the multiplexed channel, the *central load shedder* must inform each *LSS-RES* individually about the *amount of data to shed* from its intra stream. The central load shedder, which uses two pieces of information, i.e., (i) the data transmission rate of each intra stream and (ii) the channel capacity of the multiplexed channel, computes the amount of data to be (further) shed from each intra stream, and forward this information to its corresponding *LSS-RES*. Consider R as the *cumulative data transmission rate* of all the data streams, which is computed by the central load shedder, and let C be the *channel capacity* of the multiplexed channel, the data transmission rate at which data has to be shed from the accumulation of all intra streams is $R - C$. R must fall to $R^{'}(\leq C)$, if $R > C$, and the cumulative percentage of data to be shed from all the data streams is $(R - C)/R$. If an intra stream n ($n \geq 1$) has data tranmission rate R_n, then the amount of attribute data to be shed from the intra stream n, denoted R_n', is $R_{n'}$ = $((R - C)/R) \times R_n$, where $(R - C)/R$ is the *drop rate*. Again, less-informative attributes of n are shed, until the required amount of attribute data R_n' is shed. Depending on the sizes of the different attributes, one or more less-informative attributes of n will be shed. After the inter-stream load shedding scheme is generated (or updated) at each Inter-Sub, the current (shedded) sliding windows from all the intra streams are then transmitted over the multiplexed channel to the destination asynchronously, in parallel.

VI. Experimental results on load shedding scheme generation

In this section, we evaluate our load shedding scheme generation and re-evaluation approach. Since the verifications of CID detection, less-informative attribute detection, and load shedding scheme generation and re-evaluation apply to both intra-stream and inter-stream load shedding approaches, the verifications apply to both.

A. Verifying the accuracy of detecting CIDs

If the attribute(s) of a training data stream S chosen as the CID of S has (have) the same replicated values (in the same sequence) within each sliding window of S during actual load shedding, then the accuracy of our CID detection approach is confirmed. The *sliding window size* of each data stream S used in the experiments was also verified along with the detection of the CID of S, since the cycle length of each detected CID yields the corresponding window size. The experimental results of detecting CIDs using training and test data are presented in Table II, which shows a 100% accuracy rate.

Data Stream Source (Sets 1, 2, 3)	Detected CID on Training Data	Detected CID on Test Data
Weather Information		
W_1	Location	Location
W_2	. . .	. . .
W_3	. . .	. . .
Stock Exchange Information		
S_1	Company Name	Company Name
S_2	. . .	. . .
S_3	. . .	. . .
Internet Traffic Information		
I_1	Router Name	Router Name
I_2	. . .	. . .
I_3	. . .	. . .

* For each Set i (i = 1, 2, 3), the training data and test data are disjoint

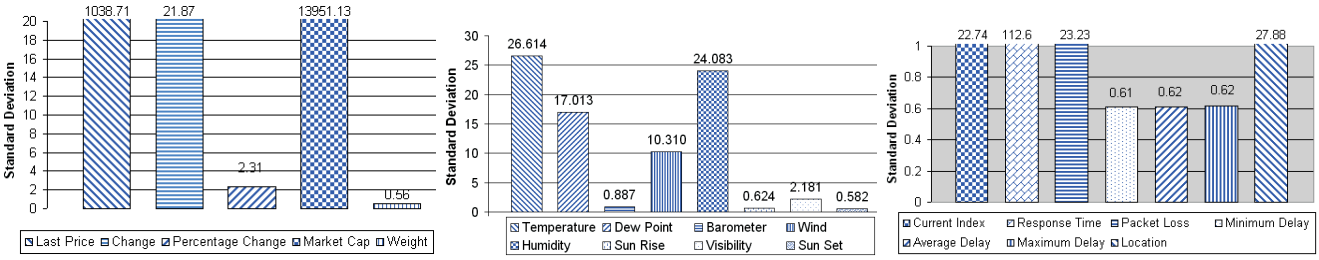
TABLE II
RESULTS ON USING TRAINING AND TEST DATA FOR CID DETECTION WITH 100% ACCURACY USING TRAINING DATA SETS (OF 30MB EACH), AND TEST DATA FROM TABLE III

Data Stream Source (Sets 1, 2, 3)	On Each Data Set: Size (GB)	Number of Tuples	Sliding Window Size	Number of Sliding Windows
Weather Information	**15.6**			
W_1 - weather.yahoo.com/	2	28036790	75	373823
W_2 - www.wunderground.com/	1.2	37063068	500	74126
W_3 - www.weather.com	2	32819280	250	131277
Stock Exchange Information	**21**			
S_1 - quotes.nasdaq.com/quote.dll?page=nasdaq100	3	52516326	100	525163
S_2 - finance.indiamart.com/markets/bse/bse100.html	3	64860370	488	132910
S_3 - www.channelnewsasia.com/cna/finance/sg/stockmonitor.htm	1	33288126	1235	26953
Internet Traffic Information	**12**			
I_1 - www.Internettrafficreport.com europe.htm	2	16332958	96	170134
I_2 - average.miq.net/index.html	1	11689810	50	233796
I_3 - watt.nlanr.net/ampmap_active.php	1	10866072	86	126349

TABLE III
DATA SOURCES OF TEST/TRAINING STREAM DATA COLLECTED ON MARCH 1, 2006 (EXCEPT FOR HTTP://WWW.CHANNELNEWSASIA.COM/CNA/FINANCE/SG/STOCKMONITOR, WHICH WAS COLLECTED ON MARCH 29, 2006)

B. Verifying the accuracy of detecting less-informative attributes

To verify the accuracy of our approach in determining the informativeness of an attribute, i.e., the ranking of attributes, of a data stream S, which determines the attributes of S to be shed, we performed experiments on the test data of various data streams. We captured the standard deviations of different attributes in a data stream S and verified that the attributes detected as less-informative are indeed less varying as compared to the attributes detected as more-informative in S. Figures 3(a), 3(b), and 3(c), show the standard deviation (rankings) for different attributes of a stock exchange, weather, and Internet traffic data stream, respectively. The results show



(a) Stock exchange data stream (b) Weather data stream (c) Internet traffic data stream

Fig. 3. Detecting less-informative attributes on stream data downloaded on March 1, 2006 from Indiamart.com, Yahoo.com, and InternetTrafficReport.com

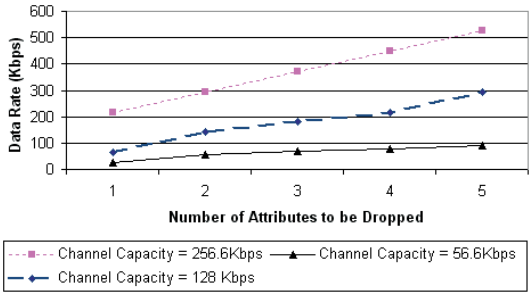


Fig. 4. Variations in the number of attributes to be shed with various channel capacities and data transmission rates on the data from http://www.wunderground.com with seven attributes

that our less-informative attribute detection approach using standard deviation on attribute values correctly identifies all the less-informative attributes, which are "Weight," "Sun Set," and "Minimum Delay" for the stock exchange, weather, and Internet traffic data stream, respectively.

C. Verification of our load shedding scheme generation and re-evaluation approach

As discussed in Section IV-C, our load shedding approach utilizes an adaptive load shedding scheme, which is re-evaluated at various time intervals. The verification of the load shedding scheme involves verifying the correctness of the re-evaluation of the load shedding scheme in between the pre-defined time intervals. The verification results are shown in Table IV. Figure 4 shows how the number of attributes to be shed varies with (i) the capacity of the channel C and (ii) the data transmission rate R of a data stream. As shown in Figure 4, the number of attributes to be shed increases linearly with increase in R when C is kept constant.

VII. Conclusions

In this paper, we propose a dynamic (intra-stream and inter-stream) load shedding approach that shed data while reducing information loss. Our load shedding approach is *dynamic*, since it is re-evaluated in real-time, and is *adoptive*, since it (i) selects less-informative attributes of a data stream to be shed based on the standard deviations of the attributes and is applicable to any kind of data stream, and (ii) minimizes the number of unnecessary load shedding scheme re-evaluations.

We have conducted experiments to verify (i) the correctness of our less-informative attribute load shedding approach, with 100% accuracy in choosing the less-informative attributes of a data stream, and (ii) the accuracy of our load shedding scheme generation and re-evaluation, with 94% accuracy in generating and re-evaluating a load shedding scheme.

Data Stream Source (Sets S_1, S_2, S_3)	# Hits			# Misses		
	S_1	S_2	S_3	S_1	S_2	S_3
Weather Information						
weather.yahoo.com/	94	94	93	6	6	7
www.wunderground.com/	95	94	95	5	6	5
www.weather.com	96	96	96	4	4	4
Stock Exchange Information						
quotes.nasdaq.com/quote.dll?page=nasdaq100	94	95	95	6	5	5
finance.indiamart.com/markets/bse/	95	95	95	5	5	5
www.channelnewsasia.com/cna/finance/sg/stockmonitor.htm	93	94	93	7	6	7
Internet Traffic Information						
www.Internettrafficreport.com	94	95	95	6	5	5
average.miq.net/index.html	92	93	93	8	7	7
watt.nlanr.net/active/maps/ampmap_active.php	95	94	94	5	6	6
Average	**94.3**			**5.7**		

Hit: a match of manually and automatically generated less-informative attribute(s). *Miss*: a mismatch. *Number of Randomly Chosen Sliding Windows* from Table III for each test data set: 100

TABLE IV
EXPERIMENTAL RESULTS OF TEST DATA USED FOR VERIFYING THE CORRECTNESS OF OUR LOAD SHEDDING SCHEME GENERATION AND RE-EVALUATION APPROACH WITH AN AVERAGE NUMBER OF *hits* OF *94.3%* AND AN AVERAGE NUMBER OF *misses* OF *5.7%*

References

[1] D. Abadi, D. Carney, U. Etintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S. Zdonik. Aurora: A New Model and Architecture for Data Stream Management. *Journal of VLDB*, 12(2):120–139, 2003.

[2] Y. Ahmad, B. Berg, U. Cetintemel, M. Humphrey, J. Hwang, A. Jhingran, A. Maskey, O. Papaemmanouil, A. Rasin, N. Tatbul, W. Xing, Y. Xing, and S. Zdonik. Distributed Operation in the Borealis Stream Processing Engine. In *Proc. of SIGMOD*, pages 882–884, 2005.

[3] A. Arasu, S. Babu, and J. Widom. The CQL Continuous Query Language: Semantic Foundations and Query Execution. Technical Report 2003-67, Computer Science Dept., Stanford University, 2003.

[4] B. Babcock, M. Datar, and R. Motwani. Load Shedding for Aggregation Queries over Data Streams. In *Proc. of ICDE*, pages 350–361, 2004.

[5] Y. Chi, H. Wang, and P. Yu. Loadstar: Load Shedding in Data Stream Mining. In *Proc. of VLDB*, pages 1302–1305, 2005.

[6] L. Golab. Querying Sliding Windows over On-Line Data Streams. In *Proc. of ICDE/EDBT Ph.D. Workshop*, pages 1–10, March 2004.

[7] R. Motwani, J. Widom, A. Arasu, B. Babcock, S. Babu, M. Datarand, G. Manku, C. Olston, J. Rosenstein, and R. Varma. Query Processing, Resource Management, and Approximation in a Data Stream Management System. In *Proc. of the Conf. on Innovative Data Systems Research*, pages 1–16, 2003.

[8] F. Reiss and J. Hellerstein. Data Triage: An Adaptive Architecture for Load Shedding in TelegraphCQ. In *Proc. of ICDE*, pages 155–156, 2005.

[9] C. Shannon. *The Mathematical Theory of Information*. Urbana, University of Illinois Press, 1949. (Reprinted 1998).

[10] S. Zdonik, U. Cetintemel, M. Cherniack, C. Convey, S. Lee, G. Seidman, M. Stonebraker, N. Tatbul, and D. Carney. Monitoring Streams - A New Class of Data Management Applications. In *Proc. of VLDB*, pages 215–226, 2002.