How to Screen a Data Stream

Quality-Driven Load Shedding in Sensor Data Streams

(Research Paper)

Anja Klein, Gregor Hackenbroich
SAP Research Center Dresden
SAP AG, Germany
<anja.klein, gregor.hackenbroich>@sap.com

Wolfgang Lehner
Database Technology Group
University of Technology Dresden, Germany
wolfgang.lehner@tu-dresden.de

Abstract: As most data stream sources exhibit bursty data rates, data stream management systems must recurrently cope with load spikes that exceed the average workload to a considerable degree. To guarantee low-latency processing results, load has to be shed from the stream, when data rates overstress system resources. There exist numerous load shedding strategies to delete excess data. However, the consequent data loss leads to incomplete and/or inaccurate results during the ongoing stream processing.
In this paper, we present a novel quality-driven load shedding approach that screens the data stream to find and discard data items of minor quality. The data quality of stream processing results is maximized under the adverse condition of data overload. After an introduction to data quality management in data streams, we define three data quality-driven load shedding algorithms, which minimize the approximation error of aggregations and maximize the completeness of join processing results, respectively. Finally, we demonstrate their superiority over existing load shedding techniques at real-life weather data.

Introduction

Typical data stream sources provide potentially high arrival rates (transactions in financial markets, and production monitoring events, etc.), but sufficient resources may not be available for the required workload of numerous queries [8]. For example, the critical resources during stream aggregations are computational power and stream bandwidth, while joins suffer from limited memory capacity.
Furthermore, data streams tend to have dramatic spikes in data volume (evening web traffic, high event rates during critical states in production processes, etc.), so that peak load can be orders of magnitude higher than typical loads. Frequently, it is impractical or impossible to provide resources to fully handle the spike load. However, accurate data stream processing is most critical in such

situations of high and bursty data load. There are two basic approaches to resolve this situation: Provide approximate processing results instead of accurate ones to ensure high performance by discarding a controllable fraction of input stream data [2], [11] or provide accurate results by buffering overload with the risk of failing to keep up with the input rate [21].
In this paper, we present data quality-driven load shedding, which follows the first approach, but outperforms existing load shedding strategies with respect to the achieved quality of stream processing results. Prior load shedding techniques range from simple random sampling to biased sampling optimized for specific stream query types. They discard information from the stream (data loss) and introduce an error to the processing result. Since the true outcome of data stream *aggregations* can only be approximated upon incorporation of load shedding, the correctness of processing results is decreased. During load shedding in combination with data stream *joins*, potential join partners may be deleted from the stream, so that only an incomplete subset of the original join result set is computed.
Moreover, most data stream sources suffer from limited data quality from the beginning, whereas data quality expresses the "'suitability"' of a given data for the respective application task and is defined as a set of data quality dimensions. In [9] we present an elaborated data quality definition for data streams together with methods for data quality recording. The dimension *accuracy* for example is decreased by restricted sensor precisions, by typos in text elements, or by RFID readers failing to scan an item properly. The *completeness* of a data stream is reduced whenever a true world event is missed due to sensor or system malfunction. In many scenarios, such missing items are estimated (e.g., interpolated) to prevent from cascading null-values in data processing.
The novel data quality-driven load shedding improves the overall data quality of processing results significantly by discarding data stream items of low data quality with higher probability than "good" ones. In this way, the approximation error during aggregations is reduced and in some scenarios even compensated completely. Further, the completeness of join result sets is increased with regard to missing values due to malfunctions. For the field of data quality management and data quality-driven load shedding our contributions are as follows.

- We present the novel concept of data quality-driven load shedding that improves the quality of query processing results.
- We propose three algorithms for data quality-driven load shedding. *MaxDQ* samples stream items of minor data quality with high probability and improves the overall quality for arbitrary queries. *MaxDQcompensate* first balances the approximation error before improving further quality dimensions to optimize aggregations. *MaxCompleteness* aims at the most complete result set for data stream joins.
- We investigate methods to calculate the introduced approximation error and to measure the level of data quality improvement due to DQ-driven load shedding.
- We evaluate the data quality-driven load shedding with the help of real-life weather data and show its superiority in comparison with existing load shedding techniques.

This paper is organized as follows: After a discussion of related work in Section 2, we present the concept of data quality-driven load shedding in Section 3. Section 4 defines the specific algorithms MaxDQ, MaxDQcompensate and MaxCompleteness. We conclude this paper with an evaluation in Section 5 and a summary of our contributions in Section 6.

Related Work

Multiple publications [3], [7] underline benefits of the data quality management in data warehouses and databases. To define the term data quality, different sets of data quality dimensions are discussed i.a. in [12] and [22]. Further, there are different approaches to structure data quality metadata in databases [18], [19].

We presented the first approach for data quality management in data streaming environments in [9]. The proposed quality propagation model (QPM) allows for the efficient measurement and transfer of data quality information using so called jumping data quality windows. The stream is partitioned into consecutive, non-overlapping windows $w(k)$ $(1 \leq k \leq \kappa)$, each of which is identified by its starting point t_b, its end point t_e and the window size ω. Beyond the data stream items $x(j)(t_b \leq j \leq t_e)$, the window contains $|Q|$ data quality information q_w, each obtained by averaging the tuple-wise DQ information over the window.or each specified data quality dimension, the window-wise data quality q_w describing window $w = [t_b, t_e]$ is taken as the average of incoming tuple-wise data quality information $q_w = (1/w) \sum_{j=t_b}^{t_e} q(j)$. Further, the QPM provides methods for the data quality processing to track influences of applied data processing operators (joins, aggregations, etc.) on the transferred DQ information to compute the data quality of processing results.

Load shedding aims at the reduction of the data stream volume. In situations of high workload, more data items than the system can cope with may arrive at the data stream processing nodes. Significant delays and memory overflows would result, if all data items shall be processed. The quality of service (QoS) would decrease. Therefore, the load has to be decreased by skipping a certain amount of stream tuples until a manageable data volume or an appropriate processing delay is reached, respectively.

There exists a wide range of load shedding algorithms. They all aim to answer the following questions. *How much* data tuples have to be discarded from the stream? *Where* in the processing path, must the load shedding take place? *Which* tuples have to be deleted from the stream? The first two questions concerning the load distribution between multiple queries and load shedder placement have been studied in detail in prior art. The load shedding rate is computed based on stream and operator statistics (varying stream rates, selectivities, etc.) and the available memory or transfer capacities. To answer the second question, load shedding techniques locate tuple "drop boxes" at root nodes (split points) of processing paths shared in multiple data stream queries [1], [2], [20].

We significantly extend this work by incorporating data quality information in the load shedding process. In the following, we compare different strategies that try to answer the third question to illustrate the benefits of our data quality-driven load shedding approach. The generic load shedding [20] creates a simple random sample of incoming data stream tuples. The information loss due to deleted tuples leads to faulty results in later on data processing. The *correctness of aggregation results* is decreased. Furthermore, the *completeness of join result sets* is reduced, because relevant join partners were deleted from the data stream. Sophisticated strategies have been developed to reduce either the correctness or completeness error by performing semantic load shedding with the help of biased sampling.

Kang et al. present sophisticated memory allocation strategies to maximize the output set of moving window joins over unbounded data streams based on a frequency model of stream arrival [8]. A contrary approach is illustrated in [17], where stream tuples are sampled using an age-based stream model statistic. Longbo et al. follow a correlated load shedding strategy for different incoming

data streams [11]. The result set is maximized by partitioning the domain of the join attribute into sub-domains, and filter out certain input tuples based on their join values.
Although these algorithms aim to maximize the join result set, certain join partners are unrecoverably lost. Further, missing values, which have been interpolated prior to load shedding, are ignored as they can not be distinguished from original measured data. The data quality-driven load shedding *MaxCompleteness* considers both sources of incompleteness: missing values and lost join partners during load shedding. This way, *MaxCompleteness* improves the overall stream completeness compared to existing strategies declared above.
[20] focuses on aggregation queries with sliding windows. Complete data stream windows are dropped to compute *correct* but *incomplete* aggregations with limited CPU power. In the contrary, Babcock et al. present a load shedding algorithm to *approximate* aggregation results to produce a *complete* result set [2]. The introduced correctness error due to shed data tuples is minimized based on stream statistics and operator selectivity. In [6] a semantic load shedding strategy is described, that samples stream overload depending on the standing queries and thus combines the above described algorithms.
While all these algorithms use stream statistics to determine tuples to delete, [14] focuses on *important* join partners to improve the correctness of subsequent aggregations. Moreover, the tuple *utility* is used together with frequency statistics to bias the deletion of data tuples in [1].
We summarize, that different load shedding algorithms have been developed, which are optimized for join and/or aggregation execution. All existing load shedding techniques introduce an error by reducing the amount of processed data. Either tuples are missing in the result set (decreased completeness) and/or aggregation values are approximated (reduced correctness). The quality-driven load shedding cracks this hard problem. By concentrating the load shedding on tuples of minor quality, the overall stream data quality is improved, the introduced error is minimized and may even be compensated.

Quality-Driven Load Shedding

In the following section, we first present the quality-driven ordering of data stream tuples. Then, we propose the novel concept of data quality-driven load shedding (DQLS) and introduce three metrics to measure the quality of processing results to compare our approaches to existing load shedding strategies.

Quality-Driven Tuple Ordering

To enable data quality-driven load shedding, data stream tuples must be ordered according to their total data quality. The starting point for such an ordering are data quality values assigned to each data stream tuple. We assume there are $|Q|$ data quality dimensions, where $q \in Q$ denotes any of these dimensions. We restrict ourselves to scalar-valued DQ information [15], i.e. each q can be expressed as a numerical value.
As an example, [9] describes the data quality dimensions accuracy, confidence and completeness. The accuracy a defines the systematic error of a data stream value x (e.g., sensor measurements, counter of website hits) due to imprecision of the measuring method itself. The confidence ϵ states statistical errors due to random disturbances. The overall numerical correctness α of final processing results consists in the sum of accuracy and confidence: $\alpha = a + e$. Small values of α imply high data quality. The completeness c monitors sensor failures leading to missing data items, which are

interpolated to prevent the processing of null-values. Note that high completeness signifies high data quality. Below, it is convenient to have a uniform notion of data quality in the sense that small values of q imply a high data quality; for all data quality dimensions that initially to not comply with this notion, we redefine q according to $q \rightarrow -q$.
The total data quality is obtained by aggregation of the quality dimensions into a scalar value $\theta = \theta(q_1, q_2, \ldots, q_{|Q|})$, where θ is a weighted average of normalized data quality dimensions; a suitable choice of normalization and weights is described in Definition 3. We use θ to unify the human interpretation of "very good" to "very bad" with respect to the monotonic ordering of numerical data quality values.

Definition 1 The stream tuple x_1 exceeds stream tuple x_2 (expressed by the binary relation $\succ$), iff the total quality of x_1 exceeds the total quality of x_2, i.e. iff the total data quality θ_1 is smaller than θ_2.

$$x_1 \succ x_2 \equiv \theta_1 < \theta_2$$

Now, data stream tuples can be ordered[1] to allow for a quality-driven load shedding.

Overall Approach

The data quality-driven load shedding builds upon the load shedding scheme of Babcock et al. that aims at the lowest processing time per tuple focussing on the question *where* to shed load [2]. The load shedding structure resulting form this approach remains stable as long as the query workload remains unchanged. Moreover, the effective load shedding rate is determined by existing stream statistics. Thus, the problem of effective load shedding reduces to the question of *which* tuples to delete.
To address this problem, it is tempting to directly apply Definition 1 and use a quality-guided ordering to determine "good" and "bad" data stream tuples. However, the naive sorting of stream tuples according to quality would block the data processing stream and is, therefore, out of scope. Thus, we resort to the quality distribution of the stream yet processed. For a given load shedding rate r_{LS}, defined as the ratio of tuples remaining in the stream and the total number of tuples, we determine the data quality bound b (Figure 1). It separates high-quality tuples remaining in the stream from low-quality tuples that will be deleted.

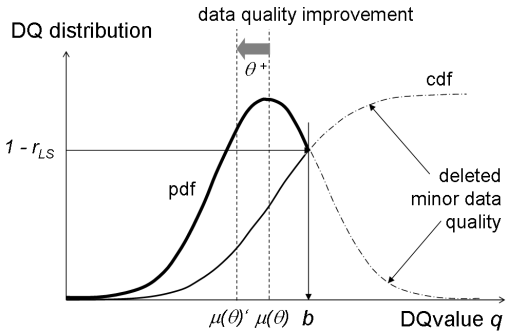

Figure 1: DQ bound and DQ improvement

[1]Formally, θ only defines a quasi order as θ provides no ordering for distinct tuples of the same total data quality. This does not impose a restriction for our approach, as tuples of the same data quality may be ordered arbitrarily, i.e. according to their timestamp.

We perform the load shedding statistically based on the distance between window-quality and DQ bound b. Form this distance we determine the probability for tuples within the window to remain in the data stream, such that -on average- more tuples are removed from bad data quality windows than from windows with good data quality.

Definition 2 Data quality-driven load shedding performs a Bernoulli sampling with a probability $p_{Bernoulli}$ for tuples to remain in the data stream. Here, $p_{Bernoulli} = d_{\succ}(\theta, b)$ is computed from the distance function $d_{\succ}$ between window-quality θ and quality bound b.

Specific choices of $d_{\succ}$ for the developed load shedding algorithms are given in Section 4. Assuming independence of data quality and stream values, quality-driven load shedding performs a non-biased, random sampling of overloaded stream partitions. Bernoulli sampling is applied individually for each data tuple in a DQ window.
We note that data quality-driven load shedding is performed based on the distribution of past quality values. In cases where the quality decreases over time, tuples that were deleted as bad quality tuples at the beginning of the stream processing might be considered of good quality at a later time. This is a consequence of the quasi-static analysis of data stream snapshots which is crucial for the continuous clearing of data overload. A globally optimal load shedding would require a complete analysis of the data stream which would fundamentally contradict the spirit of the streaming approach.

Data Quality Metrics

Although load shedding constitutes a mighty tool to ensure the quality of service in data stream systems, it significantly affects the data quality of processed results. For example, consider the data series 1,1,1,1,1,1,1,1,1,100 of 10 values and true average 10.9 which is sampled with a sampling rate of 50%. If the sample contains the value 100, the sample average is 20.8. Otherwise, the sample average is computed to 1. The deviation between sample and full average is the aggregation approximation error. In [4], Haas provides an estimation of this approximation error based on tuple variance σ^2, sample size n and load shedding rate r_{LS}.

$$\epsilon^+ = \frac{\rho \cdot \sigma}{\sqrt{n}} \cdot \sqrt{1 - r_{LS}} \tag{1}$$

Here, the parameter ρ denotes the $(1 - p/2)$ quantile of the probability p that the interval $[x - \epsilon^+, x + \epsilon^+]$ around the sample average x contains the true average over the full data stream. For the example data series given above, the introduced approximation error is computed to $\epsilon^+ = 19.4$ for the probability $p = 95\%$. As the approximation error represents a statistical sampling error, it is captured in the DQ dimension confidence $\epsilon'_w = \sqrt{\epsilon_w^2 + \epsilon^{+2}}$ to calculate the overall quality of processing results.
The correctness α of approximate aggregation results is described by the DQ dimensions accuracy a and confidence ϵ, respectively. As a data stream aggregation summarizes a certain group g of incoming stream tuples, the accuracy and confidence values of this group have to be summarized. The correctness is defined as sum of the average of all incoming window accuracies a_w and the root mean square of the incoming statistical confidence errors ϵ_w (including approximation errors).

$$\alpha = \frac{1}{|g|} \sum_{|g|} a_w + \sqrt{\frac{1}{|g|} \sum_{|g|} \epsilon_w^2} \tag{2}$$

In order to measure the performance of MaxCompleteness, we compute the recall rec as result set size during join execution with load shedding compared to the result set size in a perfect-world-scenario, where no load shedding has to be performed. The result set integrity I incorporates missing items due to prior sensor failures defined in the DQ dimension completeness $c_w = (1 - \#failures)/\omega$ as well as missing join partners due to load shedding given as recall rec.

$$I = rec \cdot (1 - \bar{c}_w) \quad (3)$$

$$= \frac{|resultWithLS|}{|resultWithoutLS|} \cdot (1 - \frac{1}{n} \sum \omega \cdot c_w) \quad (4)$$

Finally, the overall improvement of the data quality θ^+ due to the load shedding activity is reflected in the decrease of mean values $\theta^+ \equiv \mu(\theta) - \mu'(\theta)$ before and after load shedding (see Figure 1).

Algorithms

Based on the load shedding concept described above, we developed the load shedding algorithm *MaxDQ* for arbitrary queries, *MaxDQcompensate* to improve the correctness of approximate data stream aggregations and *MaxCompleteness* to maximize the completeness of data stream joins under the constraint of limited memory capacity.

MaxDQ - Total DQ Improvement

The MaxDQ algorithms implements the concepts introduced in Sections 2 and 3: The total data quality θ is determined at the end of each data quality window. Then the quality bound b is computed upon analyzing the data quality distribution in the data stream history. Finally, a Bernoulli sampling is performed in the quality window based on the distance between total quality and quality bound.
As described before, we combine all data quality dimensions by performing the transformation to the standard normal distribution and aggregating the quality dimensions into the total data quality θ. This process involves the risk of degradation of important DQ dimensions, upon deleting tuples with high quality in specific dimensions of interest due to low quality in other dimensions. To avoid this and enable to focus on quality dimensions of interest, we introduce dimension weights $weight_i$ with $\sum weight_i = 1$ in the definition of the total quality.

Definition 3 The total data quality θ of a stream tuple is defined as the weighted average of the transformed data quality information in each DQ dimension q_i with mean $\mu(q_i)$ and standard deviation $\sigma(q_i)$.

$$\theta = \frac{1}{|Q|} \sum_{i=1}^{|Q|} weight_i \cdot \frac{q_i - \mu(q_i)}{\sigma(q_i)} \quad (5)$$

The parameters of the normal distribution $\mu(q_i)$ and $\sigma(q_i)$ are calculated with the help of streaming DQ history of each dimension q_i. In the evaluation of the MaxDQ algorithm we take all weights to be equal $weight_i = 1/|Q|$, hence θ reduces to the arithmetic average over quality dimensions. The quality bound b is determined based on the load shedding rate r_{LS}. It is taken as the level of total data quality which exceeds the fraction of r_{LS} data stream tuples: $r_{LS} = 1/n \cdot |x_{\theta<b}|$. We

derive this bound from the history of previously processed tuples' total data quality, expressed with the help of incrementally updated $\mu(\theta)$ and $\sigma(\theta)$. To overcome the ambiguity of the probability density function *pdf* of the normal distribution shown in Figure 1, we apply the cumulative density function *cdf*.

Definition 4 The data quality bound b is defined as the inverse cumulative normal distribution at the load shedding rate r_{LS}, where the underlying normal distribution has the mean value $\mu(\theta)$ and standard deviation $\sigma(\theta)$.

$$b = \Phi^{-1}(r_{LS}) \tag{6}$$

To approximate the inverse *cdf*, we propose to use the algorithm by Peter Acklam [16], which is the most accurate (relative error less than $1.15 \cdot 10^{-9}$) and highly efficient implementation currently freely available. Finally, we use a specific quality distance $d_{\succ}$ to compute the Bernoulli sampling probability

Definition 5 The Bernoulli probability $p_{Bernoulli} = d_{\succ}(\theta, b)$ is computed from the sigmoid quality distance.

$$d_{\succ}(\theta, b) = -0.5 \cdot tanh(b - \theta) + 0.5 \tag{7}$$

This choice of distance provides a sampling probability $p \to 1$ when $\theta \ll b$ and a small sampling probability $p \to 0$ if $b \ll \theta$. The case $b = \theta$ is modelled with $p_{Bernoulli} = 0.5$. By applying the Bernoulli sampling, a fraction of the current data quality window is deleted from the stream.

MaxDQcompensate - Error Minimization

MaxDQcompensate balances the approximation error by focussing the load shedding on tuples of low accuracy and/or confidence. While tuples of high numeric correctness remain in the stream, the overall stream correctness is increased and compensates the introduced load shedding error. Therefore, MaxDQ is extended by a prior analysis of the attainable correctness improvement. The optimal load shedding decision for error compensation is evaluated against the requested load shedding rate r_{LS}.

The first step in the algorithm nominates deletion or preservation of the current stream tuple, as there are two aspects of correctness improvement. The deletion of specific data may decrease the approximation error, if the data standard deviation $\sigma(x)$ of measurement values x is decreased. Otherwise, the preservation in the stream may increase the overall correctness due to good accuracy a_w and/or confidence ϵ_w. The dominant aspect determines the temporary decision and suggests a new load shedding rate. The second step evaluates this decision with respect to the requested load shedding rate r_{LS} by applying a process control strategy.

For the first algorithmic step $getLSrate()$ shown in Algorithm 1, $w(k)$ depicts the current data quality window, while r_j and $size_j$ describe the currently achieved load shedding rate and resulting data stream size, respectively.

First, the deletion of the respective tuple is assumed and the suggested load shedding rate r_{j+1} is updated to calculate the impact on the approximation error $\Delta\epsilon$ (row 2-3). The alteration of the overall correctness in case of preservation $\Delta\alpha$ is calculated in row 4-5. A negative impact $\Delta\epsilon$ indicates an increase of the approximation error, while a negative correctness alteration $\Delta\alpha$ shows a degradation of the result correctness and vice versa. If both impacts (deletion and preservation) are negative, the minor degradation is suggested as temporary load shedding result. If both effects are positive, the activity with stronger improvement should be performed. The temporary decision is presented in the parameter *shed_temp* and suggested load shedding rate r_{j+1}.

Algorithm 1: getLSrate()

Input: x current tuple, a_w, ϵ_w current window accuracy & confidence,
$size_j$ current sample size, r_j current load shedding rate
Output: r_{j+1} new load shedding rate, *shed_temp* temporary load shedding suggestion

1 $r_{j+1} = \frac{r_j \cdot (size_j+1)}{size_j+r_j}$;
2 $\epsilon_{j+1} = \text{updateIntroducedError}(x, \epsilon_j)$;
3 $\Delta\epsilon = \epsilon_k - \epsilon_{j+1}$;
4 $\alpha_{j+1} = \text{updateCorrectness}(a_w, \epsilon_w, \alpha_j)$;
5 $\Delta\alpha = \alpha_j - \alpha_{j+1}$;
6 **if** $\Delta\epsilon > \Delta\alpha$ **then**
7 $\quad shed\_temp = \text{TRUE}$;
8 **else**
9 $\quad shed\_temp = \text{FALSE}$;
10 $\quad r_{j+1} = \frac{r_j \cdot size_j}{size_j+r_j}$;

Algorithm 2: checkLSrate()

Input: r_{j+1} suggested load shedding rate, *shed_temp* temporary load shedding activity,
Q data quality information
Output: *shed* load shedding decision

1 $EWSA_{j+1} = \beta \cdot r_{j+1} + (1-\beta) \cdot EWSA_j$;
2 $\text{updateControlIntervalBounds}(EWSA_{j+1})$;
3 **if** $r_{j+1} > upperBound$ **then**
4 $\quad shed = \text{FALSE}$;
5 **else if** $r_{j+1} < lowerBound$ **then**
6 $\quad shed = shed\_temp \vee \text{MaxDQ}(Q, r_{LS})$;
7 **else**
8 $\quad shed = shed\_temp$;
9 **if** *shed* **then**
10 $\quad size_{j+1} = size_j + 1$;

The second step of MaxDQcompensate *checkLSrate*() (Algorithm 2) checks the temporary decision against the requested load shedding rate r_{LS}. To guarantee the required load discharge, we now apply standard statistical process control techniques [13]. This allows both to monitor whether the load shedding rate remains within acceptable bounds and to apply corrective measures otherwise. We use the exponentially weighted smoothed average (EWSA) of the monitored load shedding rate to compute a statistically valid trend. When the load shedding rate r_{j+1} is suggested, the average is updated as shown in row 1 of Algorithm 2, where the parameter β defines the sensitivity of the trend to changing rates. Typical values used in process control are $\beta \leq 0.05$. It can be shown [13] that EWSA has a normal distribution with mean value 0.5 and standard deviation $\sigma(EWSA) = 2 \cdot \sqrt{1/12 \cdot \beta/(2-\beta)}$.
Afterwards, the control interval $[lowerBound, upperBound]$ around the requested rate r_{LS} is updated (row 2), where ρ describes the $(1-p/2)$-quantile of the chosen probability p .

$$upperBound = r_{LS}, \tag{8}$$

$$lowerBound = r_{LS} - \rho \cdot \sigma(EWSA) \tag{9}$$

Algorithm 3: MaxCompleteness

Input: w current data quality window,
T set of stored tuples

```
1 forall x ∈ w do
2     if ∃y ∈ T|c(x) < c(y) then
3         exchange(x, y);
4 end
```

If the temporary load shedding rate r_{j+1} lies within these bounds, the optimal load shedding activity suggested in $getLSrate()$ is performed (row 9-11). The overall correctness of the approximate aggregation result is improved by either decreasing the introduced approximation error or improving the underlying stream correctness. If the $upperBound$ is exceeded, the requested load reduction could not be achieved by the suggested activity (row 3-5). Although the function $getLSrate()$ might have recommended a preservation of the analyzed window, it has to be deleted from the stream. The desired correctness improvement could not be executed.
In contrast, capacity remains unused if the currently suggested load shedding rate is below the $lowerBound$ (row 6-8). Although the deletion might be suggested with respect to the data correctness ($shed\_temp$ = FALSE), the data stream may benefit from the tuple as it constitutes good quality in further DQ dimensions. The algorithm MaxDQ, which improves the overall data quality, is applied. By allowing MaxDQ to change the decision of $getLSrate()$, we reduce the priority of correctness in order to not waste capacity in the streaming system. This allows better inclusion of further data quality aspects, such as completeness or data basis.

MaxCompleteness - Maximizing Join Sets

The load shedding of potential join partners is not the only reason for incomplete data stream processing result sets. Instead, in real-world applications, there are multiple sources of incompleteness ranging from sensor failures to lost data packages during data transfer. MaxCompleteness is the first load shedding approach which incorporates information about such prior reductions to maximize the integrity of join result sets. As restricted memory capacity is the critical resource during data stream join processing, potential join partners may be deleted before contributing to the result set. We apply the tuple ordering given in Definition 1 to keep those tuples in memory, which provide best results in the data quality dimension completeness.
MaxCompleteness is summarized in Algorithm 3. Whenever a data quality window w arrives at the join processing node, its completeness is compared to the temporarily stored tuple set T. Tuples of minor completeness are replaced by the incoming window tuples until either all tuples of w have been incorporated in T or the reservoir of minor completeness has been consumed. The quality bound b defined in the overall approach in Section 3.2 is given as the lowest completeness value in the storage space T. The load shedding probability $p_{Bernoulli}$ equals 0, if the bound is exceeded, and 1 otherwise.

$$p_{Bernoulli}(x) = \begin{cases} 0 & c(x) < c(b), \\ 1 & else. \end{cases} \tag{10}$$

MaxCompleteness benefits from data quality information to reduce completeness deficiencies of query results caused by sensor failures or other sources of tuple deficits. However, it will most probably entail a high tuple loss during join execution, when stream tuples are eliminated from the temporary storage space before relevant join partners could be processed. We developed two

hybrid load shedding algorithms to combine the benefits of MaxCompleteness and prior load shedding strategies: age-based MaxCompleteness and frequency-based MaxCompleteness according to Srivastava [17] and Kang [8], respectively. The load shedding probability p_{hybrid} incorporates the sampling decision according to stream statistics (age or frequency, respectively) and completeness information.

$$p_{hybrid}(x) \quad = \quad p_{Statistics}(x) \cdot p_{Bernoulli}(x) \tag{11}$$

By integrating data stream statistics in the tuple load shedding, the recall of produced join result sets is improved. Hence, the hybrid algorithms overcome the disadvantage of strict MaxCompleteness.

Evaluation

In this evaluation, we examine the load shedding algorithms at real-world data streams to empirically answer the following questions.

1. Do our load shedding algorithms provide considerable benefit over existing strategies?

2. Can the DQ-driven load shedding algorithms compete with prior state-of-the-art in terms of processing time per tuple?

We have implemented the load shedding algorithms described in this paper using PIPES [10]. This flexible and extensible infrastructure provides fundamental building blocks to implement a data stream management system for continuous data-driven query processing over autonomous data sources.

Experimental Settings

We ran our experiments on a 1.6GHz Intel Pentium IV processor with 2GB of main memory, running Microsoft Windows XP Professional 2002. All Java-based systems were executed using JRE Version 6. We use the real-world weather dataset available at [5] which consists of cloud and weather measurements over several years recorded at land stations and ships all over the globe. Each tuple is identified by time (year, month, day, hour) and place (latitude, longitude) of recording and contains information about present weather, cloud cover, solar attitude, etc. To evaluate our approaches for approximate aggregations we chose the weather measurements taken at land stations in the month of June 1990. We applied a set of five aggregation queries with varying filter and grouping criteria, which were executed in parallel with shared sub-expressions. For example, the query

```
SELECT AVG(total_cloud_cover)
FROM june90
WHERE latitude > 37470 AND latitude < 37480 AND longitude > 12220 AND longitude < 12230
GROUP BY day
```

determines the average daily cloud cover in June 1990 for San Francisco. As we don't intend to compare the efficiency of different load shedding schemes, we were not interested in the specific query overlapping and load shedder placement. Rather, we compare the achieved data quality of query results. For the evaluation of load shedding for window-wise data stream joins, we appended

the readings of June 1991. With the help of the join attributes longitude and latitude we combined weather measurements taken at all land stations for June of two consecutive years. For example, this join strategy could be used to identify weather or temperature shifting over several years.
We assumed a systematic error of 1%, while the statistical measurement error was derived from the measurements' variance. To initialize the data completeness, we identified missing weather measurements by comparing the record history of the weather stations to the planned sensor rate of one measurement every three hours $r = 1/3h$. As the weather dataset provides equidistant timestamps, we simulated varying data stream rates by integrating time delays between tuple readings from the dataset. By this means, we model load factors from 1 to 10 leading to effective load shedding rates of $1 \geq r_{LS} \geq 0.1$[2].
We explore the overall data quality in general and correctness in particular, achieved by simple load shedding using random sampling (*Simple*), the approach proposed by Babcock et al. which minimizes inaccuracy (*Babcock*), and our algorithms *MaxDQ* and *MaxDQcompensate*. Further, we compare the overall completeness of join result sets obtained by the basic age-based (*Srivastava*) and frequency-based (*Kang*) load shedding to our algorithm *MaxCompleteness* and the hybrid approaches *HybridSrivastava* and *HybridKang*. Finally, we measure the processing time of the given approaches to evaluate their processing overhead.

Results

In our first experiment, we analyze the metric of aggregation correctness α averaged over the query result as given in Equation 2. Figure 2 shows the percental decline of the correctness in proportion to aggregation without load shedding for increasing load shedding factors (the lower sub graph shows the zoom into the upper subgraph for the better illustration of the relation between MaxDQ and the Babcock approach). The higher the load factor, the higher is the introduced approximation error that impairs the overall correctness. The simple random sampling ranks lowest, as no means for any DQ improvement are incorporated. The Babcock approach minimizes the introduced approximation error, which improves the correctness significantly. It outperforms MaxDQ for low load factors, where the load shedding rate is too small to delete a sufficient partition of "bad" data stream tuples. However, during high load given in bursty data streams, the DQ-driven load shedding leads to superior results. MaxDQcompensate even has a positive impact on the stream correctness. Due to the sophisticated selection of data stream tuples, the introduced approximation error is compensated.
In Figure 3 we extend the analysis to the total data quality improvement θ^+ for aggregation results. The diagram presents similarities with Figure 2. While the simple random sampling introduces the highest error (highest negative improvement), MaxDQcompensate provides the best results as the correctness error is compensated. The dominance of MaxDQ shifts to lower load shedding factors. As Babcock focuses correctness only, it is sooner exceeded by MaxDQ that aims at maximizing the total data quality improvement.
Next, we evaluate the DQ-driven load shedding for data stream joins, which suffer from restricted memory capacity. The result set integrity I combining missing values due to sensor failures as well as missing join partners due to load shedding is illustrated in Figure 4. To prove the practicability of MaxCompleteness to form reasonable join result sets, we further took a closer look on the recall *rec* of the load shedding algorithms in Figure 5. The allocated memory size is shown as percentage of the amount required to retain the entire data stream window currently under consideration. In

[2]For example, the load factor 2 signifies that the arriving data stream exhibits twice the workload processable by the given resources requesting a load shedding rate of $r_{LS} = 0.5$.

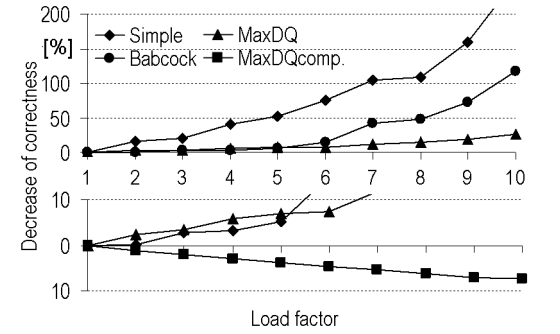

Figure 2: Correctness vs. load factor

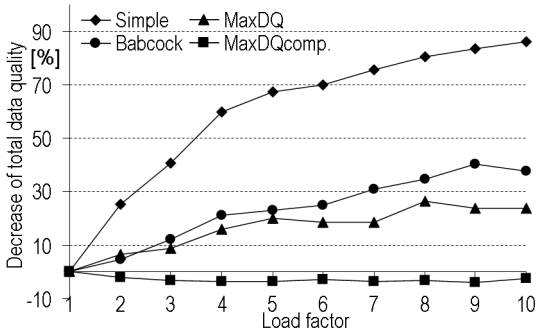

Figure 3: DQ improvement vs. load factor

general, the wider the provided memory, the less stream tuples have to be deleted. The recall approaches 1, and the result set integrity converges to the initial data stream completeness. MaxCompleteness performs a simple random sampling concerning the probability of future join partners as DQ dimension completeness and data stream measurement values are not correlated. Thus, it attains the smallest recall, which leads to medium performance concerning the result set integrity. The basic algorithms by Kang and Srivastava provide the best recall, but sample tuples of low completeness with the same probability as complete ones, which reduces the result set integrity. The hybrid approaches combine the reduced completeness error of MaxCompleteness and the superior recall of Kang and Srivastava, respectively. Thus, they present the best result set integrity for all memory configurations. On the one hand, their recalls are only slightly inferior to the basic age- and frequency-based algorithms. On the other hand, they increase the data stream completeness by eliminating tuples of minor initial completeness due to sensor or reading failures. In Figure 6 we compare the processing time per incoming data stream tuple for each load shedding approach. The minor processing time of join load shedding is caused by the considerably higher complexity of this stream operator. While MaxDQ lies in the order of the simple random sampling, MaxDQcompensate is only slightly slower than the Babcock approach, both minimizing the introduced approximation error. Similarly, the hybrid load shedding approaches present only small slowdowns compared to the basic algorithms. Not including tuple-value information, MaxCompleteness analyzes incoming stream tuples at the level of data quality windows, which results in the smallest processing time for join load shedding.

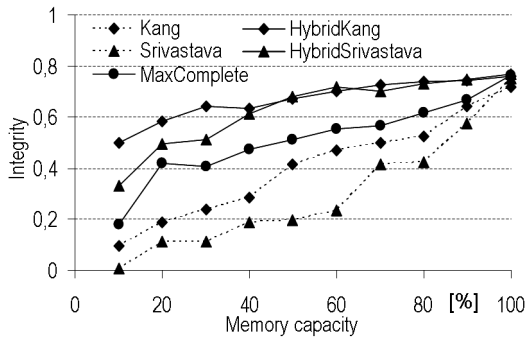

Figure 4: Completeness vs. memory capacity

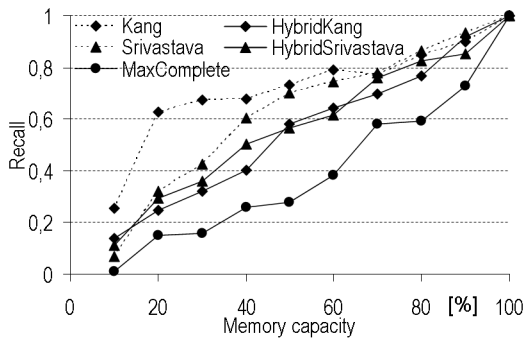

Figure 5: Recall vs. memory capacity

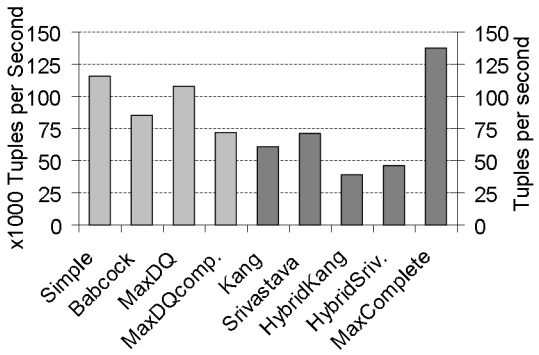

Figure 6: Processing time per tuple

In conclusion, the novel approaches of data quality-driven load shedding significantly outperform the compared existing strategies in the context of data quality. In particular, the correctness of approximated aggregations as well as the completeness of join result sets in case of limited memory is improved. The deficiencies in processing time result from the incorporated analysis of data quality information. In our opinion, they are well justified by the considerable data quality improvement for aggregation as well join query result sets.

Conclusion

In this paper we addressed data stream load shedding to meet limited resource such as computation capability and memory size. The wide range of existing load shedding strategies shares one big problem: the data loss due to discarded overload stream tuples. Data stream aggregation results incorporate an approximation error, while stream joins produce incomplete result sets as important join partners were deleted.
To solve these problems, we presented the concept of data quality-driven load shedding. By integrating data quality information into the *MaxDQ* load shedding process, we were able to delete "bad" data tuples from the stream to improve the overall data quality of stream query results. *MaxDQcompensate* is especially focused on the data quality dimensions accuracy and confidence to minimize the approximation error of data stream aggregations. As not alone load shedding leads to completeness deficiencies of data stream joins, *MaxCompleteness* and its hybrid extensions also incorporate DQ information concerning missing items to maximize the completeness of join result sets. The evaluation in this paper showed, that the data quality-driven load shedding enables the considerable improvement of stream aggregation correctness and join result set completeness.

References

[1] D. Abadi, D. Carney, U. Çetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S. Zdonik. Aurora: A new model and architecture for data stream management. *Special Issue on Best Papers of VLDB 2002*, 12(2):120–139, 2003.

[2] B. Babcock, M. Datar, and R. Motwani. Load shedding for aggregation queries over data streams. In *ICDE'04*, pages 350–361, 2004.

[3] C. Batini and M. Scannapieco. *Data Quality: Concepts, Methodologies and Techniques (Data-Centric Systems and Applications)*. 2006.

[4] P. J. Haas. Large-sample and deterministic confidence intervals for online aggregation. In *SSDBM'97*, pages 51–63, 1997.

[5] C. J. Hahn, S. G. Warren, and J. London. Edited synoptic cloud reports from ships and land stations over the globe, 1982-1991, 2008.

[6] T. Johnson, S. Muthukrishnan, V. Shkapenyuk, and O. Spatscheck. Query-aware sampling for data streams. In *ICDE'07 Workshops*, pages 664–673, 2007.

[7] J. M. Juran. *Juran's quality control handbook*. 4. ed edition, 1988.

[8] J. Kang, J. F. Naughton, and S. D. Viglas. Evaluating window joins over unbounded streams. In *VLDB'02*, pages 341–352, 2002.

[9] A. Klein. Incorporating quality aspects in sensor data streams. In *PIKM'07*, pages 77–84, 2007.

[10] J. Kraemer and B. Seeger. Pipes - a public infrastructure for processing and exploring streams. In *SIGMOD'04*, pages 925–926, 2004.

[11] Z. Longbo, L. Zhanhuai, W. Zhenyou, and Y. Min. Semantic load shedding for sliding window join-aggregation queries over data streams. In *ICCIT'07*, pages 2152–2155, 2007.

[12] F. Naumann and C. Rolker. Assessment methods for information quality criteria. In *ICIQ'00*, pages 148–162, 2000.

[13] N.N. Engineering statistics handbook, 2008.

[14] A. Ojewole, Q. Zhu, and W.-C. Hou. Window join approximation over data streams with importance semantics. In *CIKM'06*, pages 112–121, 2006.

[15] L. Peng and K. S. Candan. Data-quality guided load shedding for expensive in-network data processing. In *ICDE'07*, pages 1325–1328, 2007.

[16] W. Shaw. Refinement of the Normal Quantile, Simple improvements to the Beasley-Springer-Moro method of simulating the Normal Distribution, and a comparison with Acklam's method and Wichura's AS241, Kings College working paper, 2007.

[17] U. Srivastava and J. Widom. Memory-limited execution of windowed stream joins. In *VLDB'04*, pages 324–335, 2004.

[18] V. C. Storey and R. Y. Wang. An analysis of quality requirements in database design. In *ICIQ'03*, pages 64–87, 1998.

[19] D. M. Strong, Y. W. Lee, and R. Y. Wang. Data quality in context. *ACM Communications*, 40(5):103–110, 1997.

[20] N. Tatbul and S. Zdonik. Window-aware load shedding for aggregation queries over data streams. In *VLDB'06*, pages 799–810, 2006.

[21] T. Urhan and M. J. Franklin. Xjoin: A reactively-scheduled pipelined join operator. *IEEE Data Engineering Bulletin*, 23:27–33, 2000.

[22] R. Y. Wang and D. M. Strong. Beyond accuracy: What data quality means to data consumers. *Journal of Management Information Systems*, 12(4):5–33, 1996.