Propagation of Densities of Streaming Data within Query Graphs

Michael Daum, Frank Lauterwald, Philipp Baumgärtel, Klaus Meyer-Wegener
{md, frank, snphbaum, kmw}@i6.cs.fau.de

Dept. of Computer Science, University of Erlangen-Nuremberg, Germany

Abstract. *Data Stream System*s (DSSs) use cost models to determine if a DSS can cope with a given workload and to optimize query graphs. However, certain relevant input parameters of these models are often unknown or highly imprecise. Especially selectivities are stream-dependent and application-specific parameters. In this paper, we describe a method that supports selectivity estimation considering input streams' attribute value distribution. The novelty of our approach is the propagation of the probability distributions through the query graph in order to give estimates for the inner nodes of the graph. For most common stream operators, we establish formulas that describe their output distribution as a function of their input distributions. For unknown operators like *User-Defined Operator*s (UDOs), we introduce a method to measure the influence of these operators on arbitrary probability distributions. This method is able to do most of the computational work before the query is deployed and introduces minimal overhead at runtime. Our evaluation framework facilitates the appropriate combination of both methods and allows to model almost arbitrary query graphs.

1 Introduction

Over the past few years, the field of data stream research has matured, and first commercial products have appeared. There is still a lot of need of and potential for research in distributing, estimating, and optimizing of queries.

Estimation of costs of data processing in data streams is crucial, especially if data stream processing is distributed over battery-powered sensor nodes. This estimation needs to know the data rates and the distribution of values. In our project *Data Stream Application Manage*r (DSAM) [1], we distribute queries over a network of heterogeneous *Stream Processing System*s (SPSs) and *Wireless Sensor Network* (WSN) nodes. This paper presents some of the theoretical results regarding the cost estimator of DSAM.

Like in a *Database System* (DBS), a query in a *Data Stream System* (DSS) consists of a graph of operator instances (nodes) that are connected by inner streams. Each operator in a query may manipulate both the data rate and the distribution of values in a data stream. The outgoing data rate of an operator is often influenced by the distribution of values. In order to achieve good estimates for the whole query, characteristics of intermediate data streams between operators are also relevant. This requires the propagation of estimates through the entire graph. The manipulation of these characteristics (change of variables' formula) can be either measured, computed by analytic methods or it can

The original publication is available at
www.springer.com/computer/database+management+%26+information+retrieval/book/978-3-642-13817-1

be approximated by numerical methods. Basically, an analytic method is preferable as it is more precise and less computationally intensive. Some operators are unfortunately not manageable with analytic methods. For the support of complex query graphs, however, it will be necessary to combine analytic and numerical propagation methods.

In simple cases, each data element of a data stream consists of one value. There are some approaches that support density estimation of one-dimensional data stream items. Heinz and Seeger present approaches that use kernels and wavelets to estimate densities of one-dimensional data streams [2,3]. These approaches use measurements in order to determine the density estimators.

To our knowledge, this is the first work addressing the propagation of densities within query graphs of DSSs by using analytic or numerical methods. Further, it supports multi-dimensional data stream elements. Our contribution to the problem of estimating multi-dimensional densities operates on two levels: First, we propose both analytic and numerical propagation of densities for relevant streaming operators and argue which method is appropriate for each operator. Second, we propose a method for combining these approaches as each operator has a best fitting method for propagating densities, but a query graph may contain both kinds of operators.

We address related work in more detail in the next section. The core of this paper starts with basic explanations and definitions for the estimation of *Probability Density Function*s (PDFs) in Sect. 3. In the following two sections, we present the analytic propagation method and the numerical propagation method. We make a synthesis of these two methods using *Kernel Density Estimator*s (KDEs) that includes some experimental results before we conclude.

2 Related Work

There are several “historical” approaches that use probabilities for the estimation of databases’ statistics. In [4], multi-dimensional histograms model attribute value distributions. There are several formulas for the impact of operators on distributions. We could not adopt these formulas directly for data streams as on the one hand they only consider densities represented as histograms and on the other hand they postulate discrete probability distributions. As they consider databases, e.g. the projection operator uses duplicate elimination.

The *Detailed Database Statistics Model* (DDSM) [5] considers one-dimensional discrete distributions, but with additional matrices that represent dependencies between subdomains of attributes. This paper also describes statistics for estimating join sizes. The influence of operators is evaluated for selection and join.

A survey [6] gives an overview of estimation of statistical profiles in database systems. This paper discusses the influence of statistical methods on cost estimation in different database systems. Further, it discusses different methods of density estimation and the influence of database operators on probability densities without giving concrete formulas.

As mentioned in the introduction, [2] and [3] investigated density estimators over data streams. These papers only consider measuring densities of existing streams. The densities of intermediate streams are still unknown, if there is no actual system executing the respective query.

Simulating a query on a second system allows to estimate the costs of a query graph [7]. This circumvents the need to estimate densities of intermediate streams, but requires a second system only for simulation purposes. We summarize the benefits and drawbacks of our approach (modeling) and the simulation approach in Table 1.

The usefulness of density estimates for the selectivity estimation of range queries is described in [8]. This usefulness is further detailed in [9].

In [10], the cost model uses join and filter selectivities to estimate the rate of intermediate streams in a DSS. This work assumes already known selectivities. Our contribution allows to estimate the unknown selectivities.

Meyerhöfer [11] introduces a method to estimate the performance of software assemblies that is based on [12]. This is done by partitioning the range of a method's input variables by certain characteristics into input subdomains that are mapped to output subdomains. We use query graphs analogously to the software assemblies.

	Modeling	Simulation
Method	estimating	measuring
Complexity	complex cost model	no cost model required
Effort	lean implementation	full system for simulation
Impediments	imprecise a high load	impossible at high load
Calculation Time	fast results	time for simulation
Optimizability	high potential for optimizations	simulation hardly optimizable

Table 1. Comparison between modeling and simulation

3 Basics

In this section, we introduce our data stream model and describe the basics for estimating *Probability Density Function*s (PDFs) using kernels. We further present the assumptions we made.

3.1 Data Stream Model

Like [13], we assume unbounded data streams of independent and identically distributed tuples $\boldsymbol{X}_1, \boldsymbol{X}_2, \ldots \in \mathbb{R}^d$ with d real valued attributes. Timestamps are not part of this model. There can be dependencies between attributes of one tuple. Hence the attribute value distribution of a single stream is multidimensional.

3.2 Estimation of PDFs

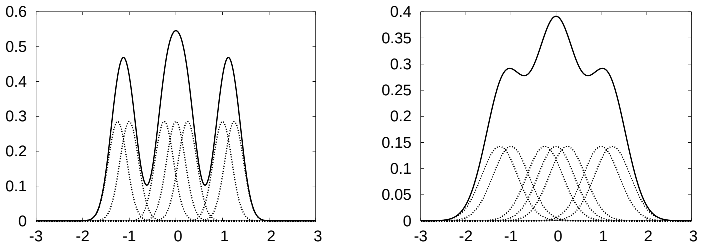
There are different options to estimate the PDF of a data stream. If the underlying parametric model is known, the estimation of the unknown parameters based on a sample is sufficient to determine the density of the stream. Nonparametric estimation is an alternative approach with no need for information about the underlying model. One well-known example for nonparametric estimation is KDE [14, 15].

A *Kernel Density Estimator* (KDE) is constructed from a sample $\boldsymbol{X}_1, \boldsymbol{X}_2, \ldots, \boldsymbol{X}_k$ with $\boldsymbol{X}_i = (X_{i,1}, \ldots, X_{i,d})$, $X_{i,j} \in \mathbb{R}$ as follows:

$$\hat{f}^{(k)}(\boldsymbol{x}) = \frac{1}{k} \sum_{i=1}^{k} \prod_{j=1}^{d} \frac{1}{h_j^{(k)}} K\left(\frac{x_j - X_{i,j}}{h_j^{(k)}}\right), \boldsymbol{x} \in \mathbb{R}^d. \tag{1}$$

The kernel function K is an arbitrary one-dimensional symmetric PDF with mean 0. This means that every tuple in the sample is represented by a "bump" (kernel) and the KDE is the normalized sum of these kernels.

$h_j^{(k)}$ is the bandwidth in dimension j for a sample of k tuples. The bandwidth determines the width of the kernels. Different values for the bandwidth lead to different resulting density estimates (Fig. 1).



(a) $h^{(k)} = 0.2$ (b) $h^{(k)} = 0.4$

Fig. 1. The influence of the bandwidth on KDEs

One goal of density estimation is to minimize the difference between the estimated and the real density. Hence the choice of the right bandwidth is crucial. The following formula presented in [14] is one possible way to estimate the optimal bandwidth in dimension j for a sample of k d–dimensional tuples:

$$h_{j,\text{opt}}^{(k)} \approx \sigma_j \left(\frac{4}{d+2}\right)^{1/(d+4)} \cdot k^{-1/(d+4)}. \tag{2}$$

σ_j is the standard deviation of the sample tuples in dimension j.

3.3 Assumptions

For every input stream, we assume constant rates and fixed densities. The number of tuples in a window is therefore known and fixed at runtime. Varying data rates could lead to varying densities, if e.g. an operator has a time-based window and if the data rate of the input stream and therefore the number of tuples in the window is varying.

If rates or densities change in a relevant way during runtime, a recalculation might be necessary.

Our underlying model uses probability distributions over continuous values. With limited precision, it works well for discrete data streams, if there is a sufficiently high number of uniformly distributed distinct elements in the stream. We cannot support nominal attributes like strings. Specific values are approximated by defining an interval; e.g. equi-joins are simulated by overlapping ranges. As grouping relies on finding tuples with identical values in the grouping attributes, our approach does not support grouping. A definition of interval-based grouping is out of the scope of this paper.

4 Analytic Propagation

To propagate the densities of streams in a query graph, we need to estimate the output densities of all operators used in the graph. This can be achieved by modeling the operators' influence on their input densities. In this section, we describe the formulas we found for the operators described in [16], as it contains both adapted relational and stream operators. Table 2 describes all variables used in the formulas that are not explained in the text.

Identifier	**Description**
f	density of the output stream
n	number of tuples in a window
f_n	density of an aggregate with windowsize n
$g, \tilde{g}$	density of an input-stream
$p \in [0, 1]$	filter predicate
k	number of attributes to be projected out
m	number of attributes not to be projected out
ℓ	number of input-streams
g_j	density of the j-th input-stream
λ_j	rate of the j-th input-stream
o	number of attributes of the input-stream
o_j	number of attributes of the j-th input-stream
$\boldsymbol{x}$	$= (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_\ell) = (x_{1,1}, \ldots, x_{1,o_1}, \ldots, x_{\ell,1}, \ldots, x_{\ell,o_\ell})$
$\boldsymbol{t}$	$= (\boldsymbol{t}_1, \ldots, \boldsymbol{t}_\ell) = (t_{1,1}, \ldots, t_{1,o_1}, \ldots, t_{\ell,1}, \ldots, t_{\ell,o_\ell})$
$\boldsymbol{y}$	$= (y_1, \ldots, y_o)$
$\boldsymbol{s}$	$= (s_1, \ldots, s_o)$
$\boldsymbol{G}$	mapping function
$\boldsymbol{G}^*$	inverse mapping function
$J_{\boldsymbol{G}}$	Jacobian determinant of $\boldsymbol{G}$

Table 2. List of identifiers

4.1 Union

The *Union* operator has ℓ input streams and one output stream. All input streams have the same schema, but they usually have different densities and input rates. Every tuple that arrives on one input stream is immediately forwarded to the output stream. Hence the output density is a weighted sum of the input streams' densities.

The weight for input stream i is $\frac{\lambda_i}{\sum_{j=1}^{\ell} \lambda_j}$. The more tuples arrive on one input stream the higher its weight. The weights are normalized to guarantee that the result is a density. The output density is calculated as follows:

$$f(\boldsymbol{y}) = \frac{1}{\sum_{j=1}^{\ell} \lambda_j} \sum_{j=1}^{\ell} \lambda_j g_j(\boldsymbol{y}). \tag{3}$$

4.2 Filter

A *Filter* is an operator with a predicate P, one input stream, and one output stream. A tuple is only forwarded to the output stream if it satisfies the *Filter*'s predicate. The support of the output stream's density is therefore the support of the input stream's density minus the subset of $\mathbb{R}^n$ for which the predicate P is not satisfied. The selectivity of the *Filter* $\int g(\boldsymbol{s})p(\boldsymbol{s})\,\mathrm{d}\boldsymbol{s}$ is needed to normalize the resulting density.

A function p is defined as:

$$p(\boldsymbol{y}) = \begin{cases} 1 \text{ if } \boldsymbol{y} \text{ satisfies the predicate } P \\ 0 \text{ else} \end{cases}.$$

The resulting density can therefore be described as follows:

$$f(\boldsymbol{y}) = \frac{1}{\int g(\boldsymbol{s})p(\boldsymbol{s})\,\mathrm{d}\boldsymbol{s}} g(\boldsymbol{y})p(\boldsymbol{y}). \tag{4}$$

4.3 Join

The *Join* operator has ℓ input streams and one output stream. Each input stream may have a different schema and a different rate. A query defines a window for each input stream. *Join* tests each possible combination of tuples in these windows if it matches the join predicate P and forwards it to the output stream, if it satisfies the predicate. The attribute value distribution of a tuple which is part of one of those combinations is independent of the window sizes and the input streams' rates. Hence the resulting density is basically a product density with a filter applied to it. The function p is likewise defined as it was for the filter.

$$f(\boldsymbol{x}) = \frac{1}{\int p(\boldsymbol{t}) \prod_{j=1}^{\ell} g_j(\boldsymbol{t}_j)\,\mathrm{d}\boldsymbol{t}} p(\boldsymbol{x}) \prod_{j=1}^{\ell} g_j(\boldsymbol{x}_j) \tag{5}$$

If the predicate P is always satisfied, the *Join* would basically be a Cartesian product between the windows over the input streams. Then the resulting density could simply be described as:

$$f(\boldsymbol{x}) = \prod_{j=1}^{\ell} g_j(\boldsymbol{x}_j). \tag{6}$$

Operator	**Output Density**
Filter	$f(\boldsymbol{y}) = \frac{1}{\int g(\boldsymbol{s})p(\boldsymbol{s})\,\mathrm{d}\boldsymbol{s}} g(\boldsymbol{y})p(\boldsymbol{y})$
Join	$f(\boldsymbol{x}) = \frac{1}{\int p(\boldsymbol{t}) \prod_{j=1}^{\ell} g_j(\boldsymbol{t}_j)\,\mathrm{d}\boldsymbol{t}} p(\boldsymbol{x}) \prod_{j=1}^{\ell} g_j(\boldsymbol{x}_j)$
Projection	$f(x_1,\dots,x_m) = \int g(x_1,\dots,x_m,\tau_1,\dots,\tau_k)\,\mathrm{d}\tau_1 \dots \mathrm{d}\tau_k$
Aggregate Sum	$f_n(x) = \underbrace{(g * \dots * g)}_{n \text{ times}}(x)$
Aggregate Avg	$f_n(x) = n \cdot \underbrace{(g * \dots * g)}_{n \text{ times}}(nx)$
Aggregate Max	$f_n(x) = ng(x) \left(\int_{-\infty}^{x} g(\tau)\,\mathrm{d}\tau \right)^{n-1}$
Aggregate Min	$f_n(x) = ng(x) \left(\int_{x}^{\infty} g(\tau)\,\mathrm{d}\tau \right)^{n-1}$
Aggregate Count	$f_n(x) = \delta(x - n)$
Union	$f(\boldsymbol{y}) = \frac{1}{\sum_{j=1}^{\ell} \lambda_j} \sum_{j=1}^{\ell} \lambda_j g_j(\boldsymbol{y})$
Map	$f(\boldsymbol{x}) = g(\boldsymbol{G}^*(\boldsymbol{x})) \frac{1}{\lvert J_{\boldsymbol{G}}(\boldsymbol{G}^*(\boldsymbol{x}))\rvert}$
Map (affine)	$f(\boldsymbol{x}) = g(\boldsymbol{A}^{-1}(\boldsymbol{x} - \boldsymbol{c})) \frac{1}{\lvert\det(\boldsymbol{A})\rvert}, \qquad \boldsymbol{G}(\boldsymbol{x}) = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{c}$
BSort	$f(\boldsymbol{y}) = g(\boldsymbol{y})$
Resample	$f(\boldsymbol{y}_1, \boldsymbol{y}_2) = g_1(\boldsymbol{y}_1) \cdot g_2(\boldsymbol{y}_2)$

Table 3. Operators' output densities

4.4 Projection

A *Projection* operator removes certain attributes from each tuple that arrives on its single input stream. Therefore the resulting stream has a different schema. In the case of data stream processing, only duplicate preserving projection is considered. *Projection* could also rename the attributes in the schema, but this does not affect the output stream's density.

For the sake of simplicity we assume that the $m + k$ attributes are ordered in a way that the last k attributes will be deleted by the *Projection*. If the input stream's attribute value distribution is $g : \mathbb{R}^{m+k} \to \mathbb{R}$, the output stream's distribution would be:

$$f(x_1, \ldots, x_m) = \int g(x_1, \ldots, x_m, \tau_1, \ldots, \tau_k) \, \mathrm{d}\tau_1 \ldots \mathrm{d}\tau_k. \tag{7}$$

4.5 Aggregate

The *Aggregate* operator applies an aggregate function to all tuples in a window over its single input stream. We assume that these tuples only have one attribute. All other attributes are projected out as follows:

$$g(x_j) = \int \cdots \int \tilde{g}(x_1, \ldots, x_o) \, \mathrm{d}x_1 \ldots \, \mathrm{d}x_{j-1} \, \mathrm{d}x_{j+1} \ldots \, \mathrm{d}x_o.$$

Because we assume the rate to be constant, the number of tuples in a window is known, even for time based windows.

Sum The output density of the sum of n values is the convolution of the input densities.

$$f_n(x) = \underbrace{(g * \ldots * g)}_{n \text{ times}}(x) \tag{8}$$

Average By means of the change of variables formula, the output density for the average of n values can be derived from the formula for the sum of n values.

$$f_n(x) = n \cdot \underbrace{(g * \ldots * g)}_{n \text{ times}}(nx) \tag{9}$$

Maximum The distribution function of the maximum of n values is:

$$F_n(x) = \left(\int_{-\infty}^{x} g(\tau) \, \mathrm{d}\tau \right)^n.$$

Hence the PDF of the output stream is:

$$f_n(x) = F_n'(x) = n g(x) \left(\int_{-\infty}^{x} g(\tau) \, \mathrm{d}\tau \right)^{n-1}. \tag{10}$$

Minimum The density of the minimum of n values is as follows:

$$f_n(x) = n g(x) \left(\int_{x}^{\infty} g(\tau) \, \mathrm{d}\tau \right)^{n-1}. \tag{11}$$

Count Let δ be the Dirac delta function, with the property:

$$\int_{-\infty}^{\infty} \delta(x)\,\mathrm{d}x = 1.$$

The density of the count of n values can be described as:

$$f_n(x) = \delta(x-n) = \begin{cases} +\infty, & x = n \\ 0, & x \neq n \end{cases}. \tag{12}$$

In reality, the rate is usually not constant but determined by the distribution of the inter-arrival times. Therefore the output density depends on the PDF of the inter-arrival times.

4.6 Map

The *Map* operator applies a function $\boldsymbol{G} : \mathbb{R}^m \to \mathbb{R}^m$ to every single input tuple $\boldsymbol{x} = (x_1, \ldots, x_m)$ producing output tuples $\boldsymbol{y} = (y_1, \ldots, y_m)$. If $\boldsymbol{G}$ fulfills the requirements of the change of variables formula, the output density is:

$$f(\boldsymbol{x}) = g(\boldsymbol{G}^*(\boldsymbol{x}))\frac{1}{|J_{\boldsymbol{G}}(\boldsymbol{G}^*(\boldsymbol{x}))|}. \tag{13}$$

If $\boldsymbol{G}$ is an affine transformation $\boldsymbol{G}(\boldsymbol{x}) = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{c}$ then the output stream's density can be described as:

$$f(\boldsymbol{x}) = g(\boldsymbol{A}^{-1}(\boldsymbol{x} - \boldsymbol{c}))\frac{1}{|\det(\boldsymbol{A})|}. \tag{14}$$

4.7 BSort

The *BSort* operator approximately sorts its input stream by applying a certain number of bubble-sort passes. Because the output tuples are the same as the input tuples but possibly in a different order, the operator's output density is the same as its input density.

$$f(\boldsymbol{y}) = g(\boldsymbol{y}) \tag{15}$$

4.8 Resample

The *Resample* operator has two input streams and aligns tuples coming from the second input stream with tuples from the first input stream. To achieve this, the *Resample* operator interpolates tuples of the second input stream to match the time stamp of a tuple from the first input stream.

Under certain assumptions, a stream consisting of perfectly interpolated tuples has the same attribute value distribution as the original stream. These assumptions are manifold but out of the scope of this paper.

Let g_1 be the density of the first input stream and g_2 the density of the second input stream. Assuming a perfect interpolation, the resulting density is:

$$f(\boldsymbol{y}_1, \boldsymbol{y}_2) = g_1(\boldsymbol{y}_1) \cdot g_2(\boldsymbol{y}_2). \tag{16}$$

4.9 Summary

In this section, we have explained our formulas for the analytic estimation of probability distributions for several operators. These include the most important stream operators (*Filter*, *Join*, *Projection*, *Union*, and *Map*) as well as several aggregates, the *BSort*, and the *Resample* operator. For reference, the formulas are summarized in Table 3.

5 Numerical Propagation

In the previous section, we described an analytic method for estimating densities. This approach relies on finding formulas that describe each operator's output density. Thus, it is not applicable for operators for which no formulas are known (yet). This is a problem as there is currently no consensus about the basic set of data stream operators. Additionally, some *User-Defined Operator*s (UDOs) are hard or even impossible to model.

In order to overcome this problem, we developed a method for numerically estimating the output densities of operators.

This method consists of two steps: In the first step (configuration phase), we build a numerical model that describes an operator instance's behavior. As the model depends on the operator's configuration (e.g. filter predicate), it is computed when the operator is instantiated and its configuration is known. This model is independent of the actual input distributions which need not be known at this time.

The second step (application phase) is performed when the query graph is instantiated. It combines knowledge of the actual input distributions with the model of the operator in order to describe the output distributions.

The advantage of this two-step approach is that the computationally expensive model building needs to be performed only once for each operator instance. The second step has to be done every time an operator graph is instantiated. During optimization, several query graphs may be instantiated for a single query. It is the optimizer's duty to choose the best one. Fortunately, the second step can be computed relatively quickly.

After outlining some requirements and limitations of this approach, we will describe both steps in detail.

5.1 Requirements

In order to apply the numerical estimation, a number of requirements have to be fulfilled. These requirements are detailed below.

Determinism Obviously, operators have to act deterministically. An indeterministic operator may behave differently between the test and the real execution, rendering the test results worthless. Additionally, an operator's output may only depend on its input but not on its state because it is not feasible to model all possible states. We model windows as additional input dimensions.

Continuity As an operator cannot be tested with all possible input values, we must assume that small variations in the input values only lead to small variations in the output values. More formally, an operator has to be continuous on subsets of its domain.

Known Domain In order to test an operator with an uniform input distribution, the domain of the input streams has to be known in advance as it is not possible to test with an uniform distribution on an infinite domain. This knowledge may be gained by testing and stored in the metadata catalog.

For operators that have intermediate streams as input, the domain of the intermediate streams has to be known as well. We solve this problem by topologically ordering the query graph and deriving the domain of the intermediate streams from the knowledge about input streams and intermediate operators. This requires the operator graph to be acyclic.

Independence Of Rates As we do not model the influence of rates on an operator's behavior, we require that an operator is independent of the rates of its input streams. The most common rate-dependent operators are time-based windows. We sketch an idea for coping with time-based windows in Sect. 5.4.

5.2 Test with Uniform Distribution (Configuration Phase)

In order to describe the influence of an operator on its output streams, it is tested with synthetic inputs. This yields a model of an operator's behavior. The model is built when the operator is deployed, as it depends on the configuration of the operator as well as knowledge of its input domains.

The operator is tested with k input values that are randomly and uniformly chosen from its input domains. The value of k has to be carefully chosen to balance accuracy versus memory requirements. Each input value is stored together with its corresponding output value. If a given input value does not produce any output, it is stored nevertheless, together with a flag that denotes that no output was produced. This will be required for selectivity estimation. Together the input and output values form the operator model.

5.3 Estimating Densities (Application Phase)

When the operator graph is instantiated, we measure the actual densities of input streams and estimate the densities of intermediate streams. The input densities are then combined with the operator model in order to yield the actual output densities.

For each combination of input and output values in the model, we compute a weight which is based on the input densities. The estimated output density is a weighted sum of kernels based on the output values in the previously calculated operator model.

The weight of each kernel is given by the following equation:

$$g_i = u(\boldsymbol{G}^*(\boldsymbol{X}_i)).$$

Here, u is the actual input density, $\boldsymbol{G}^*$ is the operator's inverse mapping function of the operator model. $\boldsymbol{X}_i$ is an output value in the operator model.

As the integral over a density has to be 1, the weights have to be normalized. This condition holds if the sum of all weights in (17) equals 1.

$$g_i^* = \frac{g_i}{\sum_{j=1}^{k} g_j}$$

Finally, we just have to apply a weighted version of (1) as suggested in [17]. The output of this equation is the estimated output density.

$$\hat{f}^{(k)}(\boldsymbol{x}) = \sum_{i=1}^{k} g_i^* \prod_{j=1}^{d} \frac{1}{h_j^{(k)}} K\left(\frac{x_j - X_{i,j}}{h_j^{(k)}}\right), \boldsymbol{x} \in \mathbb{R}^d \tag{17}$$

The optimal bandwidths $h_j^{(k)}$ can be estimated with (2) in Sect. 3.2.

Selectivity Estimation The selectivity of an operator can be easily derived from the available information. Remember that input values which produce no output are not excluded from the operator model. Thus, after the weights of the kernels are known for the actual input, the selectivity is just the ratio of the sum of weights for kernels without output and the sum of weights for all kernels.

5.4 Further Work

The approach described in this section may still be improved in numerous ways. We sketch the two extensions which in our opinion promise the greatest benefits.

Compression The models for operators with high dimensions or large domains require a large number of kernels for accurate results. This leads to high memory usage as well as increased computational costs at runtime. One solution to this problem might be cluster kernels [2].

The basic idea is to combine several kernels to a single one by means of clustering. For best results, the right choice of clusters is crucial. This choice largely depends on the function that computes the distance between two kernels. It turns out that neither the distance between input values d_i nor the distance between output values d_o is a good choice. The combination of these distances $d = \sqrt{d_i^2 + d_o^2}$ however gives adequate results. Some additional thought is required to determine which information has to be stored for each kernel. [2] discusses several possibilities.

Time-based Windows As in Sect. 4, windows are modeled as additional dimensions. This is not directly feasible for time-based windows, as their size is unknown which leads to an unknown number of dimensions. It may however be possible to transform an operator in a way that the window size is set to a fixed maximum value. Then the actual window size depending on the rate is modeled as an additional parameter. For this approach, it is necessary to determine an upper bound for the window size.

5.5 Summary

The approach described in this section allows to estimate the value distributions for operators for which no analytic formula is known. If an analytic description of an operator is available, it is preferred to the numerical approach. The numerical approach is however more general and allows to model e.g. user defined operators. As both approaches have their merits, they should be combined.

6 Synthesis

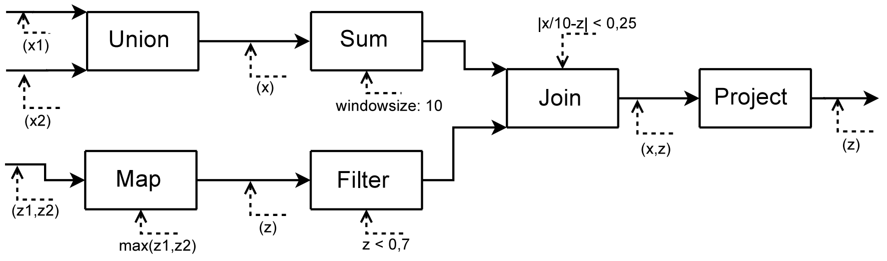


Fig. 2. The operator graph used in our experiments

The propagation of densities works by using the estimated output densities of an operator model as input densities of the successive operator models in the operator graph. If the analytic method is applicable for modeling an operator, it should be used, because it is more precise and less computationally intensive. The numerical approach can model unknown operators such as UDOs. This section gives an overview of the integration of both methods in query graphs and has a complex example query for evaluation purposes. The evaluation uses our implementations of KDEs for the common data-stream operators and a generic implementation that realizes the numerical approach. Finally, we evaluate the results of our estimates and describe the impact of the number of kernels on the quality of our estimations.

6.1 Integration of Analytic and Numerical Propagation

As the numerical method uses KDEs, we decided to implement the analytic methods with KDEs, too. This ensures compatibility of both methods that supports direct combination of both kinds of operator models. As the estimation of the PDFs are implemented as KDEs, all operator models use KDEs and are derived from an abstract class `KernelEstimator`. It defines an interface that supports the determination of the density for any multidimensional point, the dimensionality, and the support of the density. For most of the operators having analytic formulas, we provided an implementation. We did not implement the formulas for *Resample*, *BSort*, and *Aggregate Count* because they are trivial. The *Map* operator would require an implementation of the change of variables formula, which is out of scope of this paper. However, our numerical approach is well suited to estimate the output densities of the *Map* operator.

The decision to use KDEs leads to some problems. A direct implementation of the analytic model for the *Join* operator may create too many kernels. We decided to reduce some kernels by dropping them randomly; a more precise method would be using cluster kernels [2]. The integrals with variable boundaries in the formulas for maximum (10) and minimum (11) prohibit a direct representation as KDEs. Hence our

```
stream S1(x1 float)
stream S2(x2 float)
stream S3(z1 float,z2 float)

UnionStream(x):
  S1 UNION S2

AggStream(x):
  SELECT SUM(x) FROM UnionStream[rows 10]

MapStream(z):
  SELECT MAX(z1,z2) FROM S3

FilterStream(z):
  SELECT z FROM MapStream WHERE z < 0.7

JoinStream(x,z):
  SELECT x,z FROM FilterStream[rows 10],
    AggStream[rows 10]
  WHERE x/10-z < 0.25
    AND x/10-z > -0.25

ProjectStream(z):
  SELECT z FROM JoinStream
```

Listing 1. The CQL query belonging to the operator graph (Fig. 2)

implementation constructs a new KDE, which represents the output density calculated using our exact method.

For the numerical method, we distinguish “configuration phase” and “application phase”, as the generic implementation of the numerical method does not have intrinsic formulas for the manipulation of PDFs. In the configuration phase, it is crucial to create uniformly distributed input values. Input values that don’t generate output values also have to be stored in order to determine the selectivity in the application phase later. In the application phase, i.e. using this instance in a query graph, the output density of the model operator can be determined by weighting the output values according to the input density.

6.2 Evaluation

We tested the methods presented in Sects. 4 and 5 with several synthetic data streams. The densities of these streams consisted of sums of normal distributions $\mathcal{N}(\mu, \sigma^2)$ or multidimensional normal distributions $\mathcal{N}^k(\mu, \sigma^2)$. For this evaluation, we compared output densities estimated with our approach to the actual output densities. We either calculated the real output densities, if it was feasible, or, otherwise, we simulated the operators and measured the resulting output densities with KDEs.

To evaluate the estimation of densities, we utilized the *Mean Squared Error* (MSE). The actual density f was compared to estimated densities $\hat{f}_j^{(n)}$, constructed from n kernels, at 1000 random points $x_{i,j}$ equally distributed over the support of f. Because

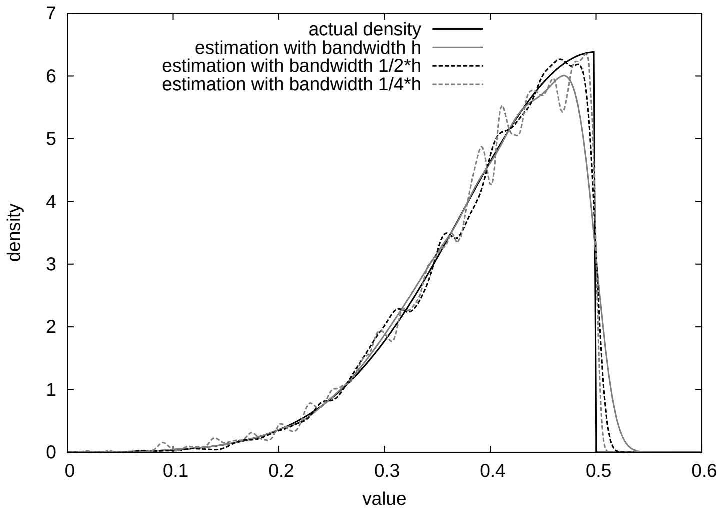


Fig. 3. Output density of a *Filter* operator with the predicate $x < 0.5$

of this randomness, we took the mean of 5 comparisons to get significant results.

$$\text{MSE} = \frac{1}{5} \sum_{j=1}^{5} \frac{1}{1000} \sum_{i=1}^{1000} \left(f(x_{i,j}) - \hat{f}_j^{(n)}(x_{i,j}) \right)^2 \tag{18}$$

We also evaluated the selectivity estimation using the *Mean Relative Error* (MRE). Estimated selectivities $\hat{s}_i^{(n)}$, calculated with n kernels, were compared to the actual selectivity s.

$$\text{MRE} = \frac{1}{20} \sum_{i=1}^{20} \frac{\left| s - \hat{s}_i^{(n)} \right|}{s} \tag{19}$$

We calculated the bandwidth for the KDEs using (2) from [14]. Our experiments showed that this estimate for the bandwidth might not be the optimal choice. Hence we carried out the tests for different bandwidths. In the following, h always represents the bandwidth calculated with (2). Fig. 3 shows the output density of a *Filter* operator, estimated with our implementation of the analytic method using different bandwidths. The input density was $\mathcal{N}(1/2, (1/8)^2)$ and the *Filter*'s predicate was $x < 0.5$. $\frac{1}{2}h$ seemed to be the best choice for the bandwidth in our experiments (Fig. 3). Therefore more precise methods to estimate the optimal bandwidth should be implemented. Some possible methods are described in [15].

We also tested our methods with an operator graph, which is depicted in Fig. 2. The corresponding *Continuous Query Language* (CQL) query [18] is shown in Listing 1. The densities of the input streams of the *Union* operator were $g_{S_1} = \mathcal{N}(1/4, (1/16)^2) + \mathcal{N}(3/4, (1/16)^2)$ and $g_{S_2} = \mathcal{N}(1/2, (1/8)^2)$ with equal rates. The density of the input stream of the *Map* operator was $g_{S_3} = \mathcal{N}^2(1/4, (1/16)^2) + \mathcal{N}^2(3/4, (1/16)^2)$. Both the

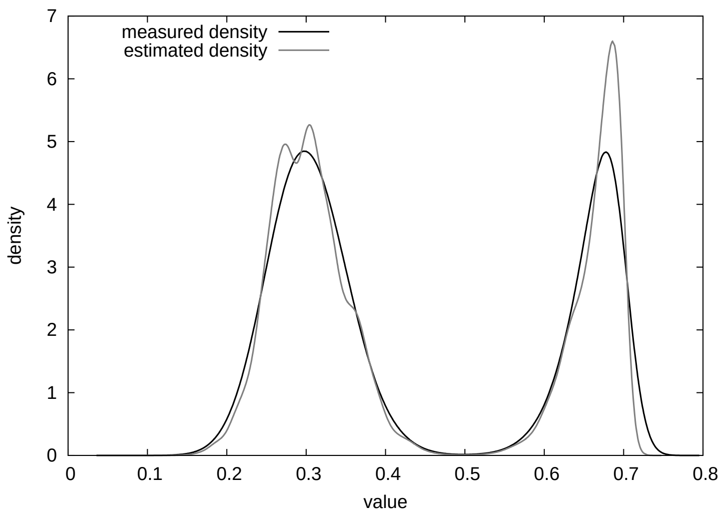


Fig. 4. Output density of the operator graph

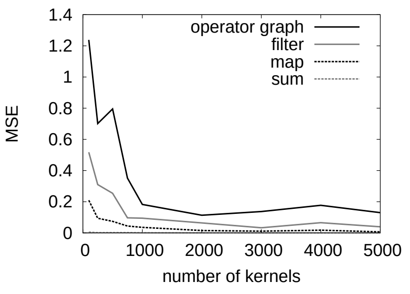
numerical and the analytic method were utilized to estimate the output density of this operator graph. The numerical method was used for the *Map* operator, and the output densities of the other operators were estimated by means of the analytic method. Fig. 4 shows the estimated output density compared to the output density measured using a KDE. Because of the *Filter* operator, the actual density should be zero for every value greater than 0.7. This is neither true for the measured nor for the estimated density, because they were both constructed from KDEs known for blurring discontinuities. In Fig. 5, the MSE is depicted for some operators in the graph and different numbers of kernels and it also shows the MRE for the selectivity estimates for the *Filter* and *Join* operator in the graph.

Our evaluation shows that the prototypic implementation of our methods is able to produce suitable density and selectivity estimates for a complex query graph.

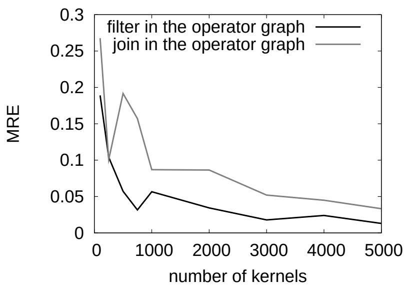
7 Conclusions and Future Work

In this paper, we showed a well working method for estimating densities within query graphs of DSSs. This work is especially important, if data rates of internal streams are relevant for cost models as it is in distributed data stream processing. With our approach, we can estimate densities of both inner and output streams within the query graph without executing the query. This estimation facilitates choosing a good physical plan.

We proposed formulas for calculating output densities for a core set of relevant stream operators and called this “analytic propagation”. For other operators, we proposed a numerical method that works without specific formulas. This might be necessary for application-specific operators like UDOs.



(a) MSE



(b) MRE

Fig. 5. MSE and MRE of the operator graph.

By means of an experimental evaluation, we used KDEs in order to integrate numerical and analytic propagation. The estimation used a concrete query graph. We compared the calculated results of our estimations with the correct results of a simulation. This comparison shows the accuracy of our approach.

In our future work, we plan to use this density estimation as a basis for selectivity estimation in the cost estimator of DSAM. We hope to get better operator placement decisions by having a more precise basis of decision-making.

References

1. Daum, M., Fischer, M., Kiefer, M., Meyer-Wegener, K.: Integration of Heterogeneous Sensor Nodes by Data Stream Management. In: Proceedings of the 10th International Conference on Mobile Data Management: Systems, Services and Middleware (MDM), Los Alamitos, CA, USA, IEEE Computer Society (2009) 525–530
2. Heinz, C., Seeger, B.: Towards Kernel Density Estimation over Streaming Data. In: Proceedings of the 13th International Conference on Management of Data (COMAD), Delhi, India (2006)
3. Heinz, C., Seeger, B.: Adaptive Wavelet Density Estimators over Data Streams. In: Proceedings of the 19th International Conference on Scientific and Statistical Database Management (SSDBM), Washington, DC, USA, IEEE Computer Society (2007) 35
4. Merrett, T.H., Otoo, E.J.: Distribution Models of Relations. In: Proceedings of the 5th International Conference on Very Large Data Bases (VLDB), VLDB Endowment (1979) 418–425
5. Muthuswamy, B., Kerschberg, L.: A Detailed Statistical Model for Relational Query Optimization. In: Proceedings of the 13th ACM Annual Conference, The range of computing : mid-80's perspective, New York, NY, USA, ACM (1985) 439–448
6. Mannino, M.V., Chu, P., Sager, T.: Statistical profile estimation in database systems. ACM Computing Surveys (CSUR) **20**(3) (1988) 191–221
7. Heinz, C., Kramer, J., Riemenschneider, T., Seeger, B.: Toward Simulation-Based Optimization in Data Stream Management Systems. In: Proceedings of the IEEE International Conference on Data Engineering (ICDE). (2008)
8. Blohsfeld, B., Heinz, C., Seeger, B.: Maintaining nonparametric estimators over data streams. In: Proceedings of the GI-Fachtagung für Datenbanksysteme in Business, Technologie und Web (BTW). (2005)

9. Gunopulos, D., Kollios, G., Tsotras, J., Domeniconi, C.: Selectivity estimators for multidimensional range queries over real attributes. The International Journal on Very Large Data Bases (VLDBJ) **14**(2) (2005) 137–154
10. Viglas, S.D., Naughton, J.F.: Rate-Based Query Optimization for Streaming Information Sources. In: Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data (SIGMOD), ACM Press New York, NY, USA (2002) 37–48
11. Meyerhöfer, M.: Messung und Verwaltung von Softwarekomponenten für die Performancevorhersage. PhD thesis, University of Erlangen-Nuremberg (2007)
12. Hamlet, D., Mason, D., Woit, D.: Properties of Software Systems Synthesized from Components. In: Component-Based Software Development: Case Studies. World Scientific Publishing Company (2004) 129–159
13. Heinz, C.: Density Estimation over Data Streams. PhD thesis, University of Marburg (2007)
14. Silverman, B.: Density Estimation for Statistics and Data Analysis. Monographs on Statistics and Applied Probability, London: Chapman and Hall (1986)
15. Scott, D.W.: Multivariate Density Estimation. Wiley-Interscience (1992)
16. Abadi, D.J., Carney, D., Cetintemel, U., Cherniack, M., Convey, C., Lee, S., Stonebraker, M., Tatbul, N., Zdonik, S.: Aurora: a new model and architecture for data stream management. The International Journal on Very Large Data Bases (VLDBJ) **12**(2) (2003) 120–139
17. Zhou, A., Cai, Z., Wei, L., Qian, W.: M-Kernel Merging: Towards Density Estimation over Data Streams. In: Proceedings of the 8th International Conference on Database Systems for Advanced Applications (DASFAA), Washington, DC, USA, IEEE Computer Society (2003) 285–292
18. Arasu, A., Babu, S., Widom, J.: The CQL continuous query language: semantic foundations and query execution. The International Journal on Very Large Data Bases (VLDBJ) **15**(2) (2006) 121–142