Cluster Kernels: Resource-Aware Kernel Density Estimators over Streaming Data

Christoph Heinz and Bernhard Seeger

Abstract—A variety of real-world applications heavily rely on an adequate analysis of transient data streams. Due to the rigid processing requirements of data streams, common analysis techniques as known from data mining are not directly applicable. A fundamental building block of many data mining and analysis approaches is density estimation. It provides a well-defined estimation of a continuous data distribution, a fact which makes its adaptation to data streams desirable. A convenient method for density estimation utilizes kernels. The computational complexity of kernel density estimation, however, renders its application to data streams impossible. In this paper, we tackle this problem and propose our Cluster Kernel approach, which provides continuously computed kernel density estimators over streaming data. Not only do Cluster Kernels meet the rigid processing requirements of data streams, but they also allocate only a constant amount of memory, even with the opportunity to adapt it dynamically to changing system resources. For this purpose, we develop an intelligent merge scheme for Cluster Kernels and utilize continuously collected local statistics to resample already processed data. We validate the efficacy of Cluster Kernels for a variety of real-world data streams in an extensive experimental study.

Index Terms—Data streams, stream mining, statistical modeling, kernel density estimation.

1 Introduction

Recent technological advances have facilitated the automatic collection of data in a variety of heterogeneous application scenarios. As data often arrives continuously and its nature is transient, we refer to it as a data stream. In order to get meaningful insights into the characteristics of a data stream, data mining and analysis techniques are a natural starting point. They provide, for instance, functionality to reveal interesting patterns, detect outliers, or identify critical regions of a data stream. The sheer volume of data streams as well as their rapid nature, however, make the use of common techniques from the area of data mining impossible; data accumulates faster than it can be analyzed. In fact, an analysis technique has to meet stringent processing requirements to be applicable to data streams [1].

Taking those requirements into account, we address in this paper the adaptation of a fundamental building block of many analysis techniques, namely, density estimation, to the data stream scenario. The main objective of density estimation is to reveal the unknown probability density function of a distribution, solely given a representative sample of values. A density estimator is a comprehensive statistical model of the distribution described by the sample values. With a well-defined density estimator at hand, a variety of data analysis issues can be addressed [2]: "*In general, density estimation provides a classical basis across statistics for virtually any kind of data analysis in principle, including clustering, classification, regression, time series analysis, active learning, and so on....*" With this work, we aim to provide a foundation for the application of these analysis tasks to data streams by adapting density estimation in compliance with the aforementioned processing requirements.

In most real-world applications with streams, we have no a priori knowledge about the stream characteristics. For that reason, the class of *nonparametric* density estimators is very appealing as they make no assumptions on the unknown density function; "*the data speak strictly for themselves*" [3]. Since data streams are often discrete observations, i.e., samples, of continuous real-valued distributions like temperature or heart rate, we confine the subsequent considerations to continuous density estimators. A theoretically well-founded and practically approved approach for the nonparametric estimation of continuous distributions utilizes kernels [3], [4]. Kernel-based density estimators can approximate *any* distribution arbitrarily good (in probabilistic terms), provided the associated sample is sufficiently large [3]. Hence, an adaptation of kernel density estimation to data streams seems to be a highly promising approach. However, the heavy computational cost of kernel density estimators is a severe obstacle; their memory allocation is linear in the size of the sample, accompanied by linear evaluation cost. Since these facts violate the processing requirements of data streams, we cannot directly build kernel density estimators over streams.

In this paper, we circumvent these restrictions and propose Cluster Kernels, our approach to kernel density estimation over streaming data in compliance with those processing requirements. We not only elaborate Cluster Kernels for one-dimensional streams but also consider the case of multidimensional streams. Generally, during runtime we keep a constant number of so-called *Cluster Kernels*, which constitute the essential building blocks of a kernel density estimator over a data stream. Each of those Cluster

• *The authors are with the Department of Mathematics and Computer Science, University of Marburg, Hans-Meerwein-Str., 35032 Marburg, Germany. E-mail: {heinzch, seeger}@informatik.uni-marburg.de.*

Manuscript received 16 Aug. 2006; revised 19 Dec. 2007; accepted 21 Dec. 2007; published online 11 Jan. 2008.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number TKDE-0390-0806.
Digital Object Identifier no. 10.1109/TKDE.2008.21.

1041-4347/08/$25.00 © 2008 IEEE Published by the IEEE Computer Society

Kernels represents a "local cluster" of processed elements. More precisely, each Cluster Kernel maintains local statistics that describe the behavior of the data stream in different data localities. These local statistics are used to resample approximately all processed elements, thereby increasing the evaluation accuracy of the estimator. In order to keep the number of Cluster Kernels constant, we provide an intelligent merge scheme based on a suitable notion of closeness between Cluster Kernels. A valuable benefit of this merge is the ability to adapt the allocated amount of memory dynamically to changing system resources. This aspect is of utmost importance in complex applications that run a large number of analysis tasks over multiple data streams simultaneously. Another important aspect is to keep pace with evolving data streams. We address this aspect by coupling Cluster Kernels with *exponential smoothing*, a weighting scheme from time series analysis.

In order to substantiate the efficacy of Cluster Kernels, we discuss the results of an extensive experimental study with a variety of one-dimensional real-world data streams [5]. In the experiments, we compared Cluster Kernels with a simple sampling-based approach and with M-Kernels [6], a competitive technique for kernel density estimation over streams.

This paper is organized as follows: In Section 2, we present related work. We prepare the ground for Cluster Kernels by introducing our data model, its processing requirements, and kernel density estimation in Section 3. We give a general overview of our approach in Section 4. Section 5 presents Cluster Kernels for univariate data streams and Section 6 presents M-Kernels. A generalization of Cluster Kernels for multivariate streams is elaborated in Section 7. In Section 8, we present the results of our experimental study. Finally, we conclude with a summary and an outlook on future work in Section 9.

2 Related Work

The analysis and mining of transient data streams has come to the fore in recent years. Concerning stream mining, Gaber et al. [7] give a comprehensive overview of arising questions, challenges, and associated techniques. Several approaches develop stream mining systems, e.g., MAIDS [8]. For an adaptation to data streams, mining algorithms have to meet specific processing requirements [1]. Many core mining algorithms were already successfully adapted to streams, e.g., change detection [9]. In [10], the authors specifically address the problem of clustering evolving data streams. They use so-called microclusters to periodically store local and temporal summary statistics of the current clusters. A special type of Cluster Kernels uses the same statistics, but for a different task. Concerning resource awareness, only a few approaches like [11] take this aspect into account.

In this work, we specifically tackle the adaptation of kernel density estimation to data streams. A short paper [12] sketches some aspects of this adaptation and [13] gives more details. A comprehensive and detailed discussion of density estimation over data streams, including Cluster Kernels, is given in [14]. Density estimation as a research topic in mathematical statistics is thoroughly discussed in [3], [4]. Density estimation can also be addressed with support vector machines [15]. In data mining and analysis, it serves as a building block for a plethora of mining and analysis methods, e.g., biased sampling [16]. Database research also reaps the benefits of kernel density estimation, e.g., in estimating the selectivity of range queries [17], [18].

The computational complexity of kernel density estimation renders its application to massive data sets difficult. To reduce this complexity, Gray and Moore [2] provide an approximate solution for offline data sets. The basic idea is to establish a dual-tree structure: One tree partitions a given set of training data and the other one the query set. A simultaneous traversal of both trees allows an efficient evaluation of sets of query points. Another approximate solution for multidimensional data relies on a multipole-based algorithm [19]. This technique provides online computable kernel density estimators by maintaining a multivariate Taylor series expansion for the estimator. Concerning their applicability to data streams, both approaches do not meet the requirement of a constant amount of allocated memory [1] (see also Section 3.1). They also cannot track evolving streams.

There are also initial approaches for kernel density estimation over data streams. Procopiuc and Procopiuc [20] present a kernel-based solution for the selectivity estimation of range queries over multidimensional spatial streams. Its basic idea is to update local variances with a kd-tree-like structure on top of a continuously maintained sample of the stream. However, this approach has difficulty in capturing evolving streams. M-Kernels are kernel density estimators over one-dimensional data streams [6]. Since we use M-Kernels as competitive technique in our experiments, we take a closer look at them in Section 6. To cluster streams with evolving cluster dynamics, Tasoulis et al. [21] use window-based kernel density estimators extended to a spatiotemporal setting. In [22], an online variant of kernel density estimators, also based on windows, is used to detect outliers in sensor networks.

3 Preliminaries

First of all, we present the underlying data stream model as well as its processing requirements. Then, we give a brief introduction to kernel density estimation and show that the severe constraints of data stream processing prevent its direct application.

3.1 Data Stream Model

A data stream is an unbounded sequence $X_1, X_2, \ldots$ of numbers with $X_i \in \mathbb{R}^d$ for $i \in \mathbb{N}$. Except where otherwise stated, we assume that the stream represents, at each time instant, a sample with independent and identically distributed (iid) observations of an unknown continuous random variable X. The premise of independence of two arbitrary stream elements is reasonable for most applications as data sources typically send their elements autonomously, i.e., previous elements do not affect their successors. The premise of an identical distribution is weakened in Section 4.3, which shows how to focus on recent changes in evolving streams.

Except for the iid-assumption, we do not have other assumptions on the data stream, e.g., value range or size. It

is worth mentioning that, even if this assumption is violated, our technique still produces reasonable results. More precisely, our empirical study has revealed that our technique also delivers good estimators for the case of autocorrelated time series, which exhibit correlations between temporally close elements. We touch on this subject from a theoretical point of view in Section 3.3.

3.2 Processing Requirements

In order to keep pace with transient and volatile data streams, an analysis technique has to meet the following stringent processing requirements [1]:

1. Each element is processed only once.
2. The per-element processing time is constant.
3. The amount of allocated memory is constant.
4. A valid model is available anytime during processing, i.e., it does not require the complete stream to be built.
5. The models should keep pace with changes in the data stream like concept drifts.
6. The provided models should be equivalent to their best offline counterparts, which have unlimited resources at hand and arbitrary access to all elements.

As the practical applicability of an analysis technique also depends on whether it can be integrated into complex applications, we add another requirement:

7. A model can adapt its allocated memory to changing system resources anytime.

With respect to these requirements, we specifically aim to provide kernel density estimators over data streams.

3.3 Kernel Density Estimation

One of the core concepts in mathematical statistics is the **probability density function** (pdf). Each continuous random variable X has a unique pdf f. A pdf is a positive real-valued function that integrates to one. The pdf completely describes X and its characteristics, e.g., the probability of all possible outcomes of X can be determined by integrating f. The pdf can also be exploited to determine important characteristics of X like mean, quantiles, Fourier transform.

In real-world scenarios, however, neither X nor its pdf is known. Typically, we only have observations of X in the form of a sample $X_1, \ldots, X_n$ with $X_i \in \mathbb{R}^d$ for $i = 1, \ldots, n$. To overcome this problem, density estimation methods from the area of mathematical statistics use the data to estimate the underlying pdf. Parametric estimation approaches assume that f falls in a specific parametric family, e.g., Gaussian densities, whereas nonparametric approaches do not assume any specific form of f. The latter are very appealing as they are solely based on the sample and thus avoid specifying the wrong parametric family as possible with parametric approaches.

A theoretically well-founded and practically approved nonparametric approach is kernel density estimation [3], [4]. A **kernel density estimator** (KDE) with **kernel function** K and **bandwidth** $h^{(n)}$ is defined as

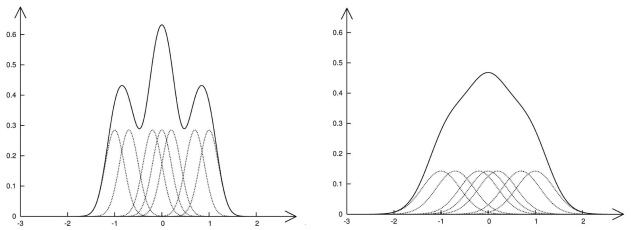

Fig. 1. Kernel density estimates and underlying kernels with (a) bandwidth = 0.2 and (b) bandwidth = 0.4.

$$\hat{f}^{(n)}(x) := \frac{1}{n}\sum_{i=1}^{n} \frac{1}{\left(h^{(n)}\right)^d} K\left(\frac{1}{h^{(n)}}(x - X_i)\right), x \in \mathbb{R}^d, \qquad (1)$$

for independent and identically distributed (iid) observations $X_1, \ldots, X_n$ with $X_i \in \mathbb{R}^d$. Essentially, a KDE is the overall sum of "bumps" centered at each observation X_i. While the bandwidth $h^{(n)}$ determines the width of each bump, the kernel function determines its shape. Henceforth, we refer to those bumps as **kernels**.

Fig. 1 displays two KDEs based on different bandwidths for a sample consisting of seven observations and with the Gaussian kernel as kernel function. The KDEs in this figure indicate that the bandwidth significantly affects the shape of a KDE. Changing the bandwidth from 0.2 to 0.4 transforms a trimodal KDE into a unimodal one. Generally, if the bandwidth is chosen too high, the KDE is oversmoothed and hides important details. If the bandwidth is chosen too low, the KDE is undersmoothed and introduces spurious details. In fact, an adequate setting of the bandwidth is crucial to the quality of a KDE and a large body of work exists on this topic [3], [4], [23]. To guarantee probabilistic convergence, the bandwidth has to decrease for an increasing sample size [3]. Note that it may be appropriate in some cases to use a separate bandwidth for each data dimension [3], i.e., a vector of bandwidths instead of a single bandwidth as in (1). In this work, we concentrate on a single bandwidth, but our approach can be easily adapted to cope with bandwidth vectors.

In contrast to the bandwidth setting, the setting of the kernel function is of minor concern. To ensure that $\hat{f}^{(n)}$ is a density, the kernel function itself must be a pdf. As a KDE is the sum of kernels, it inherits continuity and differentiability from its underlying kernel function. For practical reasons, the kernel function should be bounded as evaluating a KDE based on an unbounded kernel function requires to evaluate all kernels.

Given the definition of a KDE, let us briefly sketch the case of the iid-assumption being violated. More precisely, we consider the data to be a time series, i.e., a discrete stochastic process, which can additionally exhibit autocorrelation, i.e., the process correlates with itself at different points in time. According to [24], the asymptotically optimal bandwidth for independent data also delivers good results for strongly dependent data. However, a deeper discussion of this complex topic is beyond the scope of this paper. For more details, we refer to [24] and, in particular, to [25]. The latter source shows that, under mild conditions, the same convergence rates and the same asymptotic distribution as in the iid case can be achieved.

As already mentioned in Section 2, kernel density estimation has become highly relevant in various application scenarios. "*Apart from the histogram, the kernel estimator is probably the most commonly used estimator and is certainly the most studied mathematically*" [3]. Its broad acceptance results from the combination of simplicity with powerful mathematical properties [4]. First, a KDE only relies on the sample without a priori distribution assumptions. Second, a KDE is asymptotically unbiased. Third, a KDE is consistent in terms of the mean integrated squared error, i.e., the more sample points, the better the estimation quality. Fourth, compared to histograms, the common technique in database systems, KDEs have a higher rate of convergence, can produce smooth estimates, and do not need to know the range of the support.

3.4 Kernel Density Estimation over Streaming Data

The benefits of kernel density estimation highly recommend its adaptation to data streams as the resulting estimate gives a comprehensive statistical model of the distribution described by the stream data. The viability of a density estimate results from the fact that it can be utilized for various exploratory analysis tasks, e.g., visualization, computation of summary measures, and detection of sparse regions and hot spots. A statistically reliable density estimate can also serve as a building block in data mining and analysis approaches.

Since we assume a data stream to be an iid sample of an unknown continuous distribution (see Section 3.1), the adaptation of kernel density estimation seems straightforward. However, the computational cost of KDEs collides with the processing requirements presented in Section 3.2: According to (1), a KDE requests memory linear in the sample size, i.e., in the size of the stream. Even if large amounts of data could be stored, the use of KDEs will become unfeasible due to high evaluation cost. Furthermore, common bandwidth strategies require access to the complete sample. Consequently, each processed element of the stream would have to be accessible at any time. Hence, kernel density estimation in its original form cannot be directly applied to data streams.

A naive approach for an adaptation is to continuously maintain a constant-size sample of the already processed elements. Then, one can build a KDE anytime on top of the current sample elements. However, the estimation quality will not improve any more after the sample has been initialized since the consistency of KDEs presupposes an increasing sample size.

4 Overview of Our Approach

Our goal is to develop a suitable technique for kernel density estimation over data streams that takes the rigid processing restrictions presented in Section 3.1 into account. However, before we go into the details, we first provide a grasp of our basic idea.

Recall the general form of a KDE as described in (1). Basically, we establish specific functions $CK_i^{(n)}$, termed **Cluster Kernels**, so that, for $x \in \mathbb{R}^d$,

$$\hat{f}^{(n)}(x) = \frac{1}{n}\sum_{i=1}^{n} \frac{1}{\left(h^{(n)}\right)^d} K\left(\frac{x - X_i}{h^{(n)}}\right) \approx \frac{1}{n}\sum_{i=1}^{m} CK_i^{(n)}(x) \quad (2)$$

holds. There, $\hat{f}^{(n)}$ refers to the offline KDE with bandwidth $h^{(n)}$ based on the first n elements of the data stream, quasi the optimal, current KDE computed with unlimited resources. While the number n of processed stream elements continuously increases, we keep the total number m of Cluster Kernels constant to meet the restriction of a constant amount of memory. Note that m is given by the system. As we will see, the Cluster Kernels themselves change during runtime. Each Cluster Kernel $CK_i^{(n)}$ represents a local group of elements of the stream, called **partition**. To be more specific, each Cluster Kernel represents the kernels over the elements of the partition. We consider m partitions $S_i^{(n)}$, $i = 1, \ldots, m$ of the stream, where each one is associated with a separate Cluster Kernel $CK_i^{(n)}$. As we cannot store all elements of a partition, we equip each Cluster Kernel with simple local statistics $stat_i^{(n)}$ to summarize them continuously. This includes, among others, $c_i^{(n)}$, the current number of elements in $S_i^{(n)}$.

Generally, a Cluster Kernel $CK_i^{(n)}$ is characterized by a well-defined partition representative $\bar{X}_i^{(n)}$ called **mean** as well as bandwidth, local statistics, and merge costs:

$$< \bar{X}_i^{(n)}, \hat{h}_i^{(n)}, stat_i^{(n)}\left(mergecosts_{i,j}^{(n)}\right)_{j=1,\ldots,i-1,i+1,\ldots,m} > . \quad (3)$$

Section 4.2 gives an explanation of $\bar{X}_i^{(n)}$ and $mergecosts_{i,j}^{(n)}$. To cope with arbitrary bandwidth strategies, each Cluster Kernel has, in our general approach, its own bandwidth $\hat{h}_i^{(n)}$. We later present an approximate online computation of a theoretically well-founded *global* bandwidth setting. Additionally, we present a kernel function that simplifies the subsequently explained merge of Cluster Kernels as well as their evaluation.

4.1 Evaluation of Cluster Kernels

Let us consider the evaluation of a Cluster Kernel. With respect to the partitions $S_1^{(n)}, \ldots, S_m^{(n)}$, Cluster Kernels must guarantee that, for $i = 1, \ldots, m$,

$$CK_i^{(n)}(x) \approx \sum_{X_j \in S_i^{(n)}} \frac{1}{\left(h^{(n)}\right)^d} K\left(\frac{1}{h^{(n)}}(x - X_j)\right) \quad (4)$$

holds. Hence, Cluster Kernels have to deliver results equivalent to those obtained by evaluating the kernels over all elements of a partition. Note that this corresponds to processing requirement 6 of Section 3.1. With the local statistics $stat_i^{(n)}$, we resample elements $\hat{X}_{i,j}^{(n)}, j = 1, \ldots, c_i^{(n)}$ of the partition and use them for the evaluation of the associated Cluster Kernel $CK_i^{(n)}$:

$$CK_i^{(n)}(x) = \sum_{j=1}^{c_i^{(n)}} \frac{1}{\left(\hat{h}_i^{(n)}\right)^d} K\left(\frac{1}{\hat{h}_i^{(n)}}\left(x - \hat{X}_{i,j}^{(n)}\right)\right). \quad (5)$$

A necessary premise of Cluster Kernels is that (5) can be converted into a closed formula; otherwise, the evaluation cost for all Cluster Kernels would be $O(n)$. The kernel function and the resampling strategies presented in Section 5 guarantee that this requirement is met.

TABLE 1
Description of Parameters

Parameter	Description
n	# processed stream elements
X_i	i-th element of the stream
K	kernel function
$h^{(n)}$	optimal bandwidth for the first n elements
$\hat{f}^{(n)}$	optimal KDE for the first n elements
m	maximum number of Cluster Kernels
$CK_i^{(n)}$	i-th Cluster Kernel after n processed elements
$\bar{X}_i^{(n)}$	mean of $CK_i^{(n)}$
$\hat{h}_i^{(n)}$	bandwidth of $CK_i^{(n)}$
$mergecosts_{i,j}^{(n)}$	merge costs between $CK_i^{(n)}$ and $CK_j^{(n)}$
$stat_i^{(n)}$	local statistics of $CK_i^{(n)}$
$S_i^{(n)}$	partition associated with $CK_i^{(n)}$
$c_i^{(n)}$	# of elements in $S_i^{(n)}$
$\hat{X}_{i,j}^{(n)}$	j-th resampled element of $S_i^{(n)}$

Combining (5) with (2) delivers

$$\hat{f}^{(n)}(x) \approx \frac{1}{n}\sum_{i=1}^{m}\sum_{j=1}^{c_i^{(n)}} \frac{1}{\left(\hat{h}_i^{(n)}\right)^d} K\left(\frac{1}{\hat{h}_i^{(n)}}\left(x - \hat{X}_{i,j}^{(n)}\right)\right). \quad (6)$$

This equation reflects the main objectives of Cluster Kernels: They continuously keep summaries of the stream partitions and use them to resample the partition elements; the evaluation of all resampled elements of all partitions, in turn, approximates the optimal KDE over all processed elements. As each Cluster Kernel represents the kernels over a local "cluster" of elements, we decided on the name Cluster Kernel. For the sake of clarity, Table 1 summarizes the essential parameters.

4.2 Merge of Cluster Kernels

Up to now, we have discussed the components of Cluster Kernels and their evaluation after n processed elements. In case a new element X_{n+1} arrives, we determine whether it equals the mean of an existing Cluster Kernel. If so, we update the local statistics $stat_i^{(n)}$ of this Cluster Kernel, which includes incrementing $c_i^{(n)}$. If not, we build a new Cluster Kernel with mean X_{n+1} and properly initialized local statistics.

If the current number of Cluster Kernels plus the new one is $m+1$, we merge the two Cluster Kernels *closest to each other* into one Cluster Kernel. As this merge will typically be lossy, we introduce a **loss function** $loss(CK_i^{(n)}, CK_j^{(n)}, CK)$. This function describes the loss in accuracy that results from substituting $CK_i^{(n)}$ and $CK_j^{(n)}$ by Cluster Kernel CK. By means of this loss function, we determine the **merge costs** of a Cluster Kernel pair as follows:

$$mergecosts_{i,j}^{(n)} := min\{loss(CK_i^{(n)}, CK_j^{(n)}, CK) : CK\ Cluster\ Kernel\}. \quad (7)$$

We define the **merge kernel** of $CK_i^{(n)}$ and $CK_j^{(n)}$ as the Cluster Kernel that minimizes the loss function. The mean of the merge kernel will be located between $\bar{X}_i^{(n)}$ and $\bar{X}_j^{(n)}$, depending on the concrete loss function. Concerning the other components of the merge kernel, we have to set its bandwidth and determine its local statistics by merging $stat_i^{(n)}$ and $stat_j^{(n)}$.

If the number of Cluster Kernels is $m+1$ and has to be reduced, we choose among all pairs of Cluster Kernels the one with *overall* minimum merge costs and substitute them by their merge kernel. The number of potential merge pairs is $(m+1)m$. In case the Cluster Kernels can be totally ordered by their means $\bar{X}_i^{(n)}$, we can reduce this number and, thus, the computational effort. Concretely, the loss function has to ensure that the merge costs are *monotonous* with respect to the means of the corresponding Cluster Kernels, i.e., $mergecosts_{i,j}^{(n)} \leq mergecosts_{i,k}^{(n)}$ for $\bar{X}_i^{(n)} \leq \bar{X}_j^{(n)} \leq \bar{X}_k^{(n)}$. Hence, the merge of Cluster Kernels with closer means must cause lower merge costs. Given all Cluster Kernels totally ordered by mean, it then suffices to compute the merge costs only between adjacent Cluster Kernels, which reduces the number of merge pairs to m. Thus, each Cluster Kernel only has to store the merge costs with its right neighbor, i.e., $CK_i^{(n)} =< \bar{X}_i^{(n)}, \hat{h}_i^{(n)}, stat_i^{(n)}, mergecosts_{i,i+1}^{(n)} >$, with respect to (3). Note that the total ordering by mean is the prerequisite for this pruning of potential merge pairs. Univariate data streams are totally ordered, but multivariate ones with $d > 1$ are only partially ordered.

The merge of Cluster Kernels renders their seamless integration into complex applications possible as it offers anytime-adaptation to changing system resources. In case of an increased maximum number m, we build separate Cluster Kernels for each new element, except duplicates, until the total number of Cluster Kernels equals m. In case of a decreased m, we perform the merge step sufficiently often.

4.3 Capturing Evolving Streams

An inherent difficulty for stream analysis techniques is that the characteristics of a data stream can vary over time. Financial data, for instance, typically has a more volatile nature rather than being stable over time. Hence, an analysis technique also has to properly take changes of the stream into account. To fulfill this need, we provide an extension of Cluster Kernels that allows us to focus on recent trends of an evolving stream.

Let us examine an evolving stream from a formal point of view. Recall that we consider a data stream as a sample of an unknown random variable. Up to now, we assumed the stream to be stable, i.e., all stream elements follow the same distribution. However, the distribution underlying an evolving stream will change over time and, thus, so will the underlying pdf we want to estimate. In order to concentrate on the latest trends, we couple Cluster Kernels with **exponential smoothing**, a convenient weighting scheme from time series analysis.

The basic idea of this coupling is to give older data less weight in the evaluation. We smoothly fade them out, resulting in a kind of "smooth" sliding window. Let us consider the current estimator after the insertion of a new element X_n and before the merge step is performed (the element shall not be a duplicate):

$$\hat{f}^{(n)}(x) = \frac{1}{n}\sum_{i=1}^{m} CK_i^{(n-1)}(x) + \frac{1}{n} CK^{(n)}(x), \tag{8}$$

for $n \geq 2$ and $\hat{f}^{(1)}(x) = CK_1^{(1)}(x)$. Each Cluster Kernel is equally weighted with $1/n$. With exponential smoothing, these equal weights are replaced by exponentially decreasing ones. More precisely, given $\alpha \in (0, 1)$, the Cluster Kernel of a new element receives weight α and triggers a rescaling of older weights by a factor $(1 - \alpha)$:

$$\hat{f}_\alpha^{(n)}(x) = (1-\alpha)\hat{f}_\alpha^{(n-1)}(x) + \alpha CK^{(n)}(x), \tag{9}$$

for $n \geq 2$ and $\hat{f}_\alpha^{(1)}(x) = CK_1^{(1)}(x)$. With the smoothing parameter α, we adjust the impact of old and new data.

For illustration purposes, we assume the maximum number m of Cluster Kernels to be unbounded and determine the resulting weighting sequence. For $n \geq 2$, it follows:

$$\begin{aligned}\hat{f}_\alpha^{(n)}(x) = {} & (1-\alpha)^{n-1} CK_1^{(n)}(x) \\ & + \sum_{i=2}^{n-1} (1-\alpha)^{n-i} \alpha CK_i^{(n)}(x) + \alpha CK_n^{(n)}(x).\end{aligned} \tag{10}$$

Instead of equal weights $1/n$ for each Cluster Kernel, we now have "discounted" weights: $(1-\alpha)^{n-i}\alpha$ and $(1-\alpha)^{n-1}$ for older Cluster Kernels and α for the new Cluster Kernel. The weighting sequence sums to 1 for each $n \in \mathbb{N}$. This is a necessary prerequisite for KDEs as, otherwise, the integration to 1, a fundamental property of each pdf, is violated. M-Kernels, a competitive technique, also support a weighting by means of a fadeout function [6], but the resulting weighting scheme violates this prerequisite.

4.4 Viability of Cluster Kernels

Overall, the Cluster Kernel approach meets the processing requirements of data streams postulated in Section 3.1. Algorithm 1 gives an abstract view of the general procedure (without smoothing). A major advantage of our approach is its modular design. Due to its exchangeable modules for the main algorithm components, it constitutes a framework for KDEs over streaming data. In order to build Cluster Kernels, the following components must be defined: settings for kernel function and bandwidth, definition of local statistics, definition of a loss function, and development of resampling strategies. In the following sections, we elaborate on concrete strategies and settings for those components for univariate as well as for multivariate streams. For the sake of applicability, we also attend to the implementation of Cluster Kernels and propose suitable solutions.

Algorithm 1: Cluster Kernels over data stream

```
for each incoming element X do
    if ∃i ∈ {1,...,m} : X = X̄_i^(n) then
        update local statistics stat_i^(n);
    else
        insert new Cluster Kernel for X;
    end if
    while # Cluster Kernels exceeds m do
        choose i, j with mergecosts_{i,j}^(n) = min;
        substitute CK_i^(n), CK_j^(n) by their merge kernel;
    end while
    if evaluation demanded then
        for i = 1,...,m do
            resample elements of S_i^(n) based on stat_i^(n);
            evaluate CK_i^(n) for resampled elements;
        end for
        return (Σ_{i=1}^m evaluated CK_i^(n))/n;
    end if
end for
```

5 Cluster Kernels over Univariate Data Streams

We now discuss our specialization of Cluster Kernels for univariate streams, using Algorithm 1 as the starting point. With regard to the applications with Cluster Kernels, the space constraints require us to focus on their evaluation and the development of the related formulas. The integration of Cluster Kernels, the computation of summary measures like mean and variance, as well as other applications, are thoroughly discussed in [14].

5.1 Kernel Function and Bandwidth Settings

We utilize a kernel function with bounded support as those with unbounded support require evaluating all kernels to evaluate a given point (see (1)). Specifically, we decided to use the **Epanechnikow kernel** as the underlying kernel function. It is defined as

$$K(x) = 0.75 \cdot (1 - x^2) \cdot 1_{[-1,1]}(x), x \in \mathbb{R}. \tag{11}$$

Not only does this kernel function have a simple form, but its efficiency is also asymptotically optimal among all kernels [2]. We will see that the computation of the merge costs, as well as the evaluation of Cluster Kernels, strongly relies on the simple form of this kernel.

As mentioned in Section 3.3, the bandwidth as the second parameter of a KDE is vital to the estimation quality. Theoretically founded and practically approved bandwidth strategies assign a global bandwidth to all kernels. Complex bandwidth strategies [23] typically come with a heavy computational burden that violates the stream processing requirements, e.g., the one-pass paradigm. For example, the popular least squares cross validation strategy for setting the bandwidth requires minimizing a sum consisting of n parameterized estimators, which are based on the n sample points. Therefore, we concentrate on the **normal scale rule**, a simple yet convenient bandwidth strategy, which is defined in (12). For the sake of an online application of this rule, we have to estimate the standard deviation $\sigma^{(n)}$ of the data stream in an online fashion in amortized constant time. A suitable estimate of $\sigma^{(n)}$ is the sample standard deviation, which itself can be estimated in one pass with a numerically stable algorithm [26]. Given this estimate $\hat{\sigma}^{(n)}$, we can continuously compute a global bandwidth $\hat{h}^{(n)}$ while processing the stream:

$$h^{(n)} := 1.06 \cdot \sigma^{(n)} \cdot n^{-\frac{1}{5}} \approx \hat{h}^{(n)} = 1.06 \cdot \hat{\sigma}^{(n)} \cdot n^{-\frac{1}{5}}. \tag{12}$$

The first bandwidth $\hat{h}^{(1)}$ is set to 1. Due to the global bandwidth, it follows that $\hat{h}_i^{(n)} = \hat{h}^{(n)}$ for $i = 1, \ldots, m$. Therefore, we skip the index i of the bandwidth.

5.2 Local Statistics of Univariate Cluster Kernels

We present two settings for the local statistics $stat_i^{(n)}$ of a univariate Cluster Kernel $CK_i^{(n)}$: one computes the minimum and maximum of the associated partition $S_i^{(n)}$ and the other one the mean and variance of $S_i^{(n)}$. Both incorporate $c_i^{(n)}$, the number of elements in $S_i^{(n)}$.

The local statistics with minimum and maximum are defined as $stat_i^{(n)} := \{c_i^{(n)}, min_i^{(n)}, max_i^{(n)}\}$. A new Cluster Kernel $CK_i^{(n+1)}$ for a new element X_{n+1} has initial local statistics $stat_i^{(n+1)} = \{1, X_{n+1}, X_{n+1}\}$. In case of an update of a Cluster Kernel, we increment $c_i^{(n)}$; minimum and maximum are not affected. In case of a merge, the merge kernel receives local statistics $\{c_i^{(n)} + c_j^{(n)}, min\{min_i^{(n)}, min_j^{(n)}\}, max\{max_i^{(n)}, max_j^{(n)}\}\}$.

The variance-based local statistics are similar to the microclusters in [10]. Concretely, we store the sum $sum_i^{(n)}$ and squared sum $sqrdSum_i^{(n)}$ of the partition elements and utilize them to compute mean and sample standard deviation $\sigma_i^{(n)}$ of $S_i^{(n)}$. Thus, the local statistics of a new Cluster Kernel for X_{n+1} are initially $stat_i^{(n)} := \{1, X_{n+1}, X_{n+1} \cdot X_{n+1}\}$. In case of an update, we get $\{c_i^{(n)} + 1, sum_i^{(n)} + \bar{X}_i^{(n)}, sqrdSum_i^{(n)} + \bar{X}_i^{(n)} \cdot \bar{X}_i^{(n)}\}$. In case of a merge, the local statistics of the merge kernel are $\{c_i^{(n)} + c_j^{(n)}, sum_i^{(n)} + sum_j^{(n)}, sqrdSum_i^{(n)} + sqrdSum_j^{(n)}\}$.

5.3 Loss Function for Univariate Cluster Kernels

Our definition of a loss function for two Cluster Kernels $CK_i^{(n)}$, $CK_j^{(n)}$ is based on the following objective: Consider the sum of kernels weighted with $c_i^{(n)}$ and $c_j^{(n)}$ over means $\bar{X}_i^{(n)}$ and $\bar{X}_j^{(n)}$, respectively. We set their merge kernel as the kernel with weight $c_i^{(n)} + c_j^{(n)}$ whose suitably chosen mean ensures an optimal approximation of the sum. Thus, the means and the weights are the crucial factors during merge.

We examine the mean squared deviation, a common measure for the similarity of real-valued functions, to quantify the accuracy loss induced by the merge of two Cluster Kernels $CK_i^{(n)}$, $CK_j^{(n)}$:

$$\begin{aligned} loss_{L_2}(X) &:= loss_{L_2}\left(CK_i^{(n)}, CK_j^{(n)}, CK\right) \\ &:= \int_{-\infty}^{\infty} \left(\frac{c_i^{(n)}}{\hat{h}^{(n)}} K\left(\frac{x - \bar{X}_i^{(n)}}{\hat{h}^{(n)}}\right) + \frac{c_j^{(n)}}{\hat{h}^{(n)}} K\left(\frac{x - \bar{X}_j^{(n)}}{\hat{h}^{(n)}}\right) \right. \\ &\quad \left. - \frac{c_i^{(n)} + c_j^{(n)}}{\hat{h}^{(n)}} K\left(\frac{x - X}{\hat{h}^{(n)}}\right) \right)^2 dx, \end{aligned} \tag{13}$$

with X the mean of CK. We abbreviate $loss_{L_2}(CK_i^{(n)}, CK_j^{(n)}, CK)$ to $loss_{L_2}(X)$ to emphasize the fact that the mean X of CK is the only variable in (13). Since we set the mean of the merge kernel as the minimum of $loss_{L_2}(X)$, we minimize the accuracy loss of the merge, i.e., the merge is optimal with respect to the mean squared deviation. For illustration purposes, Fig. 2 shows the shape of the loss function for two Cluster Kernels and a global bandwidth $\hat{h}^{(n)} = 1$. The left y-axis describes the weighted kernels and the right one the loss function, while the x-axis describes the support of the kernel functions as well as the possible means of the merge kernel.

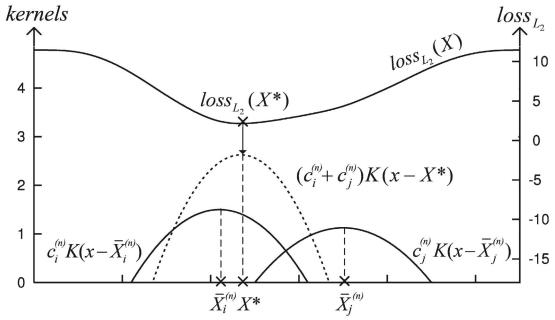

Fig. 2. Merge of two Cluster Kernels with means $\bar{X}_i^{(n)}$, $\bar{X}_j^{(n)}$ and global bandwidth $\hat{h}^{(n)} = 1$.

For the existence and the computation of the minimum, $loss(X)$ holds:

Theorem 1. *For arbitrary Cluster Kernels CK_i, CK_j, the minimum of $loss(X)$ exists and can be computed in constant time.*

For the proof of this theorem, we refer to the Appendix, which can be found in the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TKDE.2008.21.

5.4 Evaluation of Univariate Cluster Kernels

Each Cluster Kernel $CK_i^{(n)}$ is associated with a partition $S_i^{(n)}$ of the data stream with the objective—see also (2)—to approximate the KDE over all elements in $S_i^{(n)}$. We serve this purpose with a resampling of partition elements based on the local statistics $stat_i^{(n)}$. Due to our strict processing requirements, a resampling strategy must ensure constant cost for the evaluation of all resampled elements. Due to this necessity, the following resampling strategies all provide a closed formula for (5). For more detailed discussion and evaluation formulas, we refer to the Appendix, which can be found in the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TKDE.2008.21.

One-value-resampling: The mean $\bar{X}_i^{(n)}$ of a Cluster Kernel is the representative of the partition $S_i^{(n)}$. By utilizing $c_i^{(n)}$, the number of elements in $S_i^{(n)}$, we generate $c_i^{(n)}$ identical instances of the mean: $\hat{X}_{i,j}^{(n)} := \bar{X}_i^{(n)}$, $j = 1, \ldots, c_i^{(n)}$. Even though this strategy is simple and has minimum storage requirements for its local statistics, it also has its drawbacks. Only one value represents the complete partition of a Cluster Kernel. If the partition has a wide range, we run the risk of not adequately representing it. Moreover, large streams may induce an estimator that essentially consists of spikes at the means due to a decreasing bandwidth.

Min-max-resampling with equal distances: The local statistics based on minimum $min_i^{(n)}$ and maximum $max_i^{(n)}$ additionally incorporate $c_i^{(n)}$. To cover the complete partition with resampled elements, **min-max-resampling with equal distances** distributes $c_i^{(n)}$ elements equidistantly over $[min_i^{(n)}, max_i^{(n)}]$. Apparently, this strategy does not run the risk of producing spikes. Intuitively, we expect it to be

particularly suitable for locally smooth distributions. However, it may have problems with distributions exhibiting many outliers. An outlier will "expand" the range of the resampled elements even though the outlier itself is per se locally isolated or in a sparse region.

Min-max-resampling with exponential distances: An alternative strategy for resampling elements of $S_i^{(n)}$ also relies on distributing $c_i^{(n)}$ elements over $[min_i^{(n)}, max_i^{(n)}]$. **Min-max-resampling with exponential distances** distributes the majority of resampled elements around the mean $\bar{X}_i^{(n)}$ as it is the representative of the partition. More precisely, we place them left and right to the mean with exponentially increasing distances. This strategy is less sensitive to outliers compared to the one based on equal distances. It is designed for nonsmooth distributions with local irregularities.

Mean-var-resampling with equal or exponential distances: Two other resampling strategies employ the variance-based local statistics. Analogous to the latter two strategies, we distribute $c_i^{(n)}$ elements with equal or exponential distances in the interval $[\bar{X}_i^{(n)} - 2\sigma_i^{(n)}, \bar{X}_i^{(n)} + 2\sigma_i^{(n)}]$. The closed formulas for **mean-var-resampling** are identical to those for min-max-resampling except that $min_i^{(n)}$ and $max_i^{(n)}$ are replaced by $\bar{X}_i^{(n)} - 2\sigma_i^{(n)}$ and $\bar{X}_i^{(n)} + 2\sigma_i^{(n)}$, respectively.

5.5 Implementation of Univariate Cluster Kernels

For practical purposes, we discuss two suitable implementations of univariate Cluster Kernels with respect to Algorithm 1. While one implementation is based on a sorted list, the other one is based on trees. They differ in that the list-based implementation updates all merge costs continuously and the tree-based one only locally.

5.5.1 List-Based Implementation

Similarly to the implementation of M-Kernels presented in Section 6, we organize all Cluster Kernels $CK_i^{(n)} =< \bar{X}_i^{(n)}, \hat{h}^{(n)}, stat_i^{(n)}, mergecosts_{i,i+1}^{(n)} >, \ i = 1, \ldots, m$, in a list sorted by mean $\bar{X}_i^{(n)}$. With respect to Algorithm 1, this list has to support the insertion of new Cluster Kernels as well as the merge of adjacent Cluster Kernels.

Insertion step. If a new arriving element equals the mean of an existing Cluster Kernel, we update the local statistics according to Section 5.2. If not, we insert a new Cluster Kernel in compliance with the ordering by mean.

A new element triggers the recomputation of the bandwidth due to its dependency on the number of processed elements. As the bandwidth is part of our loss function—see (13)—we have to update the merge costs of all Cluster Kernels if a new element arrives. While performing this update, we can determine the current Cluster Kernel pair with overall minimum merge costs to simplify the merge step.

Merge step. If the number of Cluster Kernels exceeds m, we replace the adjacent Cluster Kernels with overall minimum merge costs by their merge kernel. The merge kernel is located between these Cluster Kernels, i.e., the substitution does not violate the list ordering. After a merge, we update the merge costs between the merge kernel and its left as well as its right neighbor (if existent).

Algorithm analysis. The insertion of a new element has complexity $O(m)$ due to the update of all merge costs. Provided we already determined the Cluster Kernel pair with overall minimum merge costs, the subsequent merge as well as the update of the affected merge costs has complexity $O(1)$. Overall, the complexity of the list-based implementation is $O(m)$.

5.5.2 Tree-Based Implementation

The tree-based implementation has a substantially lower complexity compared to the list-based one. Its implementation is approximate because we do not recompute the merge costs of all Cluster Kernels after an update of the bandwidth. We only recompute the merge costs of those Cluster Kernels that are "locally" affected by an insertion or a merge.

Let us examine the requirements for processing a set of m Cluster Kernels. On the one hand, an ordering by mean is desirable as it facilitates the insertion and search of Cluster Kernels. On the other hand, an ordering by merge costs is desirable as it facilitates the detection of the Cluster Kernel pair with minimum merge costs. We unify these contrary requirements in a data structure consisting of a binary search tree and a priority search tree. We store the Cluster Kernels $CK_i^{(n)} =< \bar{X}_i^{(n)}, \hat{h}^{(n)}, stat_i^{(n)}, mergecosts_{i,i+1}^{(n)} >, \ i = 1, \ldots, m$, in a binary search tree (**mean tree**) with the mean $\bar{X}_i^{(n)}$ as ordering criterion. We maintain a priority search tree (**merge costs tree**) with entries $\langle mergecosts_{i,i+1}^{(n)}, \bar{X}_i^{(n)} \rangle$ and the merge costs as ordering criterion.

Insertion step. With regard to Algorithm 1, a new inserted element implies that either the local statistics of an already inserted Cluster Kernel are updated or a new Cluster Kernel is inserted into the mean tree. In both cases, we recompute the merge costs between the symmetric predecessor (if it exists) and the corresponding Cluster Kernel as well as between the corresponding Cluster Kernel and its symmetric successor (if it exists). To keep both trees consistent, we remove the associated "old" merge costs from the merge costs tree and insert the new merge costs.

Merge step. If the overall number of Cluster Kernels exceeds m after an insertion, we merge the adjacent Cluster Kernels with minimum merge costs. We remove the minimum merge costs from the merge costs tree and determine the associated Cluster Kernel in the mean tree. In compliance with the mean tree ordering, we replace this Cluster Kernel with its merge kernel and remove its symmetric successor. Finally, we update the merge costs between the symmetric predecessor (if it exists) and the merge kernel as well as between the merge kernel and the new symmetric successor (if it exists). While doing so, we also keep the merge costs tree consistent by removing and inserting the associated merge costs.

Algorithm analysis. The insertion of a new element as well as the merge step has complexity $O(\log m)$. Hence, the overall complexity of the tree-based implementation is $O(\log m)$, compared to $O(m)$ for the list-based one.

Comparison with list-based implementation. Only kernels locally affected by an insertion or a merge receive

an update of their merge costs with respect to the current bandwidth, i.e., the tree-based implementation is "approximate." The list-based implementation recomputes all merge costs in case of an updated bandwidth at the expense of higher processing cost. However, merges are most likely to occur in dense data regions where the probability of new elements will be higher than in sparse regions. For the tree-based implementation, it follows that the merge costs in these regions are, with a high probability, up-to-date. The results of our experimental study in Section 8 will show that the loss in accuracy by this approximation has only minor effects on the overall quality of the resulting estimators, whereas the processing time substantially improves compared to the list-based implementation.

5.5.3 Implementation of Exponential Smoothing

According to (9), the Cluster Kernel of a new element receives weight α, while the other Cluster Kernels are rescaled with $(1-\alpha)$. Thus, the application of Cluster Kernels coupled with exponential smoothing causes insertion cost of $O(m)$ for list-based and tree-based implementation. Thus, the performance of the latter one deteriorates.

6 M-Kernels over Univariate Data Streams

M-Kernels [6] initially inspired us to do this work. We will briefly describe the essence of M-Kernels as, on the one hand, we use them as a comparative technique in our experiments and, on the other hand, to clarify the differences with our approach. As M-kernels can be described as a univariate specialization of our general approach, we discuss them with respect to Algorithm 1. In our terminology, an M-Kernel corresponds to a Cluster Kernel.

Kernel Function and Bandwidth Settings. The authors use the Gaussian kernel as an underlying kernel function. Concerning the bandwidth, each M-Kernel has a separate one initially set to 1. In case of a merge of M-Kernels, their merge kernel receives a new bandwidth.

Local Statistics of M-Kernels. In contrast with Cluster Kernels, an M-Kernel only stores the number of elements it represents as local statistics. These local statistics are initialized and updated analogous to Cluster Kernels.

Loss Function for M-Kernels. Similarly to our approach, two M-Kernels are merged by replacing them with their merge kernel. The mean and the bandwidth of the merge kernel are chosen so that the L_1 distance, i.e., the mean absolute deviation, between the two M-Kernels and their merge kernel is minimized. It is important to note that, in contrast to our approach, the bandwidth is also a variable of this loss function. Hence, computing the merge kernel becomes a two-dimensional problem. As there is no closed formula for the solution of this problem, the authors propose using numerical approximations instead.

Evaluation of M-Kernels. Given the element number as local statistics, M-Kernels use one-value-resampling, the most simple resampling strategy presented in Section 5.4, for the evaluation.

Implementation of M-Kernels. For the organization of M-Kernels, the authors provide a list-based implementation similar to the one we presented in Section 5.5.1, i.e., the list is sorted by means of the M-Kernels.

Drawbacks of M-Kernels. The following shortcomings severely limit the applicability of M-Kernels:

- The unbounded support of the Gaussian kernel, which is used as an underlying kernel function, exacerbates an efficient evaluation of M-Kernels.
- The proposed bandwidth setting has no theoretical foundation. It is not ensured that the bandwidth decreases for an increasing sample size, which is the main premise for the consistency of KDEs [3].
- The numerical approximation of mean and bandwidth of the merge kernel causes additional computational effort and leads to less accurate values.
- The evaluation strategy for M-Kernels leads, for large stream sizes, to estimators with singular spikes at the partition representatives.
- M-Kernels capture evolving data streams by means of a fade-out function. However, the resulting estimator violates the prerequisite of an integration to 1.

7 Cluster Kernels over Multivariate Data Streams

Up to this point, we have discussed Cluster Kernels over univariate data streams. The strategies for this case can be generalized to the case of multivariate streams, using Algorithm 1 as the foundation. Due to space constraints, we only sketch the general procedure and pinpoint the main problems of this generalization and how to solve them.

The starting point is the definition of a KDE over multivariate data in (1). Following our general approach, we maintain local statistics for m partitions of the stream and exploit them to resample the partition elements. Then, we evaluate the associated Cluster Kernels with respect to these elements. The entirety of Cluster Kernels constitutes the estimator for the already processed stream elements as described in (2).

Kernel Function and Bandwidth Settings. The Epanechnikow kernel as well as the normal scale rule both have multivariate analogs.

The multivariate Epanechnikow kernel is for $x \in \mathbb{R}^d$, defined as

$$K(x) := \begin{cases} \frac{1}{2c_d}(d+2)(1-x^T x), & x^T x < 1 \\ 0, & \text{otherwise,} \end{cases} \tag{14}$$

with c_d the volume of the unit d-dimensional sphere [3].

Analogously to the univariate case, the problem of the optimal bandwidth can only be overcome with an approximate solution. Due to its simplicity, we use a multivariate analog of the normal scale rule. For the Epanechnikow kernel as the underlying kernel function, it is defined as

$$h^{(n)} = \sigma \left(\frac{8}{c_d}(d+4)(2\sqrt{\pi})^d \right)^{\frac{1}{d+4}} n^{\frac{-1}{d+4}}, \tag{15}$$

with σ^2 as the average marginal variance [3]. Section 5.1 already discussed the problem of computing the sample variance in an online manner. Note that, as in the univariate case, the use of more complex bandwidth strategies is difficult due to their computational complexity. Provided that an online approximation of such a strategy is given, it can be seamlessly integrated.

Local Statistics and Resampling Strategies. The local statistics $stat_i^{(n)}$ of a multivariate Cluster Kernel $CK_i^{(n)}$ can

be defined similarly to the univariate case. A counter $c_i^{(n)}$ documents the number of elements in the associated partition. For the local statistics based on minimum and maximum, $min_i^{(n)}$ and $max_i^{(n)}$ are d-dimensional vectors that store minima and maxima for each dimension. The variance-based local statistics store sum and squared sum for each dimension. For both local statistics, updates and merges are defined componentwise.

The local statistics prepare the ground for the resampling of partition elements. The essential requirement for resampling strategies is that a closed formula for the evaluation of a Cluster Kernel over all resampled elements can be provided. The same requirement must be met for their integration and for the computation of summary measures. For univariate data, we developed, in Section 5.4, different resampling strategies that meet this requirement. While one-value-resampling can be directly adapted to the multivariate case, the adaptation of min-max-resampling and mean-var-resampling is more difficult. Concerning the evaluation of univariate Cluster Kernels, in the proofs of Theorems 2 and 3 in the Appendix which can be found in the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TKDE.2008.21, we determined indexes to confine the kernels to evaluate for a given point x. With respect to these indexes, we derived closed formulas for the evaluation of a Cluster Kernel. In order to proceed analogously in the multivariate case, we have to determine all $\hat{X}_{i,j}^{(n)}$, $j \in \{1, \ldots, c_i^{(n)}\}$ with $(x - \hat{X}_{i,j}^{(n)})^T (x - \hat{X}_{i,j}^{(n)}) < (\hat{h}^{(n)})^2$, provided that the multivariate Epanechnikow kernel is used.

Loss function. In the same way as in (13), we can utilize the mean squared deviation for the definition of a loss function for multivariate Cluster Kernels. Let us emphasize that the corresponding integral is over $\mathbb{R}^d$ and that the variable X is d-dimensional, which renders the computation of the minimum difficult. However, we can always compute an approximate solution with the help of numerical approximation techniques. For example, a suitable and performant numerical technique to solve this problem even for high dimensions is the downhill simplex algorithm [27].

Implementation of multivariate cluster kernels. A simple but inefficient approach for the implementation of multivariate Cluster Kernels is their organization in a list. Generally, an implementation of Cluster Kernels has to provide an efficient insertion, search, and merge of Cluster Kernels. For the multivariate case, the lack of a total ordering for $d > 1$ exacerbates an efficient implementation of the merge step in which the Cluster Kernel pair with minimum merge costs is determined. In the univariate case, the total ordering combined with the monotony of the loss function allowed us to restrict the number of possible merge pairs to the pairs of adjacent Cluster Kernels. Even though the multidimensional loss function is also monotonous, the partial ordering makes the number of possible merge pairs larger in the multidimensional case. Given a Cluster Kernel for which we want to determine its possible merge partner, we have to examine all of its closest neighbors. More precisely, we iteratively determine its nearest neighbors with respect to their means. Due to the monotony of the loss function, we can stop this computation for the first neighbor dominated by one of the previous neighbors. Essentially, this problem is a so-called relative skyline query [28], with the considered Cluster Kernel as the query point.

We propose using two data structures for the implementation: a priority search tree for the merge costs and a multidimensional tree structure for the Cluster Kernels, which is organized by their means. A suitable starting point for the second tree, particularly with regard to the above problem of merge pairs, is the R-tree as it naturally supports k-nearest neighbors queries as well as relative skyline queries.

8 Experimental Evaluation

We scrutinized Cluster Kernels in an extensive experimental study whose essential results we present in this section. With the experiments, we addressed the following questions: How do Cluster Kernels perform for different real-world data streams? How is their runtime behavior in terms of processing time? How do Cluster Kernels react to sudden changes in their available amount of memory?

8.1 Experimental Settings

8.1.1 Techniques

In our experiments, we concentrated on KDEs over univariate data streams. We examined different types of univariate Cluster Kernels as presented in Section 5. Concretely, we examined Cluster Kernels based on the list and on the tree implementation, respectively. For the tree implementation, we applied all resampling strategies discussed in Section 5.4 and, for the list implementation, only one-value-resampling for the sake of clarity. In our experiments, we also examined M-Kernels as a competitive technique. As another competitive technique, we included KDEs based on a continuously maintained iid sample of the stream. The computation of the sample, whose size is constant, relies on reservoir sampling [29]. We implemented all techniques with PIPES, our Java library for advanced stream processing and analysis [30].

8.1.2 Data Sets

In order to assess these techniques, it is crucial to consider different real-world data streams. Because of this, we examined a variety of heterogeneous one-dimensional data streams from the time-series archive of the University of California Riverside [5]. The streams originate from diverse fields like facility monitoring, networking, and medicine and exhibit different characteristics, e.g., noisy/smooth, stationary/nonstationary, and autocorrelation. We additionally included a synthetic data set, called CLAW, whose density is a mixture of Gaussian densities.

8.1.3 Quality Measure

In Table 1, we referred to the optimal KDE and bandwidth for the first n elements of the stream as $\hat{f}^{(n)}$ and $h^{(n)}$, respectively; both were determined with unlimited computational resources. Let $\hat{f}^{(opt)}$ and $h^{(opt)}$ be the optimum KDE and bandwidth for the *complete* data stream, i.e., the best offline KDE. In case of CLAW data, our synthetic data stream, we set $\hat{f}^{(opt)}$ as the *original* density. The comparison with $\hat{f}^{(opt)}$ allows us to form an opinion about the quality of an estimator computed with one of the upper techniques.

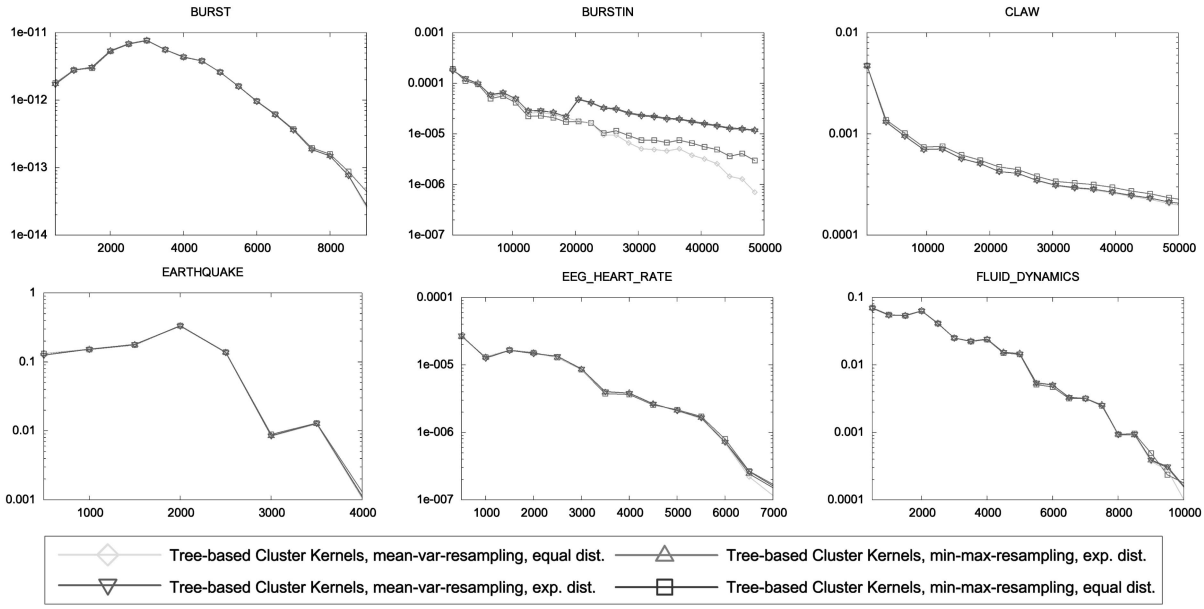

Fig. 3. Performance of tree-based Cluster Kernels with min-max-/mean-var-resampling and equal/exponential distances; the number of processed elements is on the x-axis and the MSE is on the logarithmically scaled y-axis.

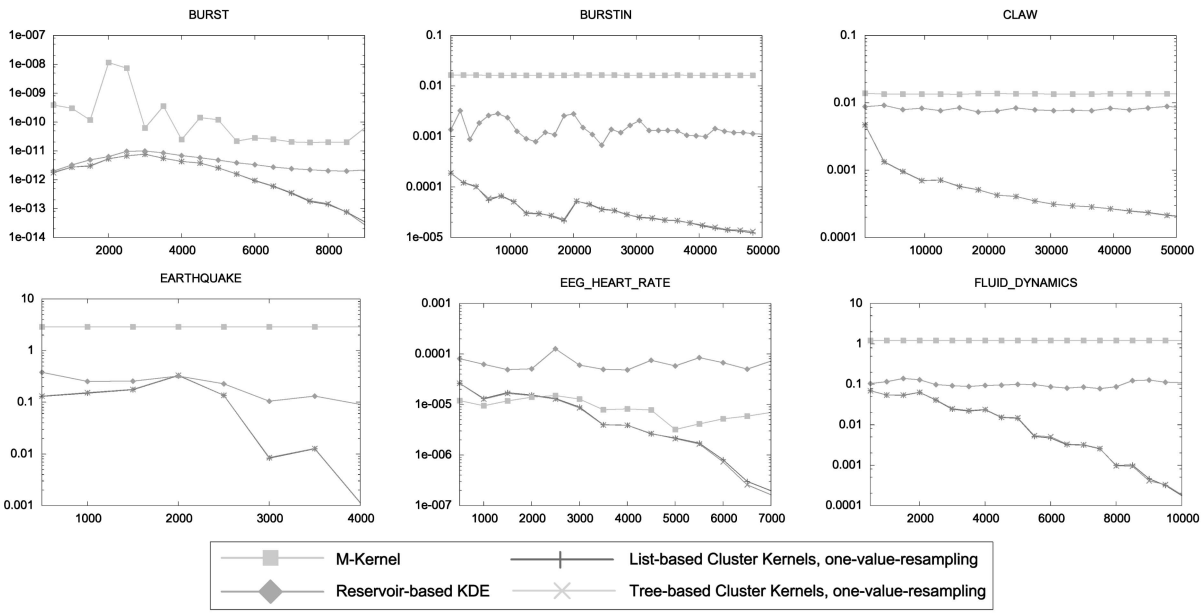

Fig. 4. Performance of M-Kernels, reservoir-based KDEs, list-based and tree-based Cluster Kernels with one-value-resampling; the number of processed elements is on the x-axis and the MSE is on the logarithmically scaled y-axis.

We measured the quality of such an estimator $\hat{g}^{(n)}$ after n processed elements in terms of the mean squared error:

$$MSE(n) := \frac{1}{500}\sum_{i=1}^{500}\left(\hat{f}^{(opt)}(x_i) - \hat{g}^{(n)}(x_i)\right)^2, \qquad (16)$$

with $x_1, \ldots, x_{500}$ an equidistant partition of the support of $\hat{f}^{(opt)}$.

8.2 Estimation Quality

A core question is whether Cluster Kernels converge in terms of a decreasing MSE for an increasing number of processed elements. To answer this question, we continuously compared the optimal KDE $\hat{f}^{(opt)}$ with the current estimator $\hat{g}^{(n)}$. We evaluated the current MSE always after 500 elements have been processed. The maximum number of M-Kernels and Cluster Kernels was set to $m = 100$. The reservoir size of the sample-based technique was also set to 100. Due to space constraints, we display only the results for a subset of the streams examined. The other results are given in the Appendix, which can be found in the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TKDE.2008.21. For the sake of clarity, we arrange the experimental results in two figures. In the charts in Figs. 3 and 4, the x-axis represents the number of processed elements and the logarithmically scaled y-axis the MSE. It is worth mentioning that, for CLAW data,

where we compared the estimators to the original density, the same trends hold. Generally, we observed the following trends:

8.2.1 Impact of Resampling Strategies on Quality

For tree-based Cluster Kernels, we examined, in addition to one-value-resampling, also min-max-resampling and mean-var-resampling with equal and exponential distances, respectively. As Fig. 3 clarifies, the Cluster Kernels produced, in the majority of cases, estimations of nearly the same quality in terms of MSE. Only for BURSTIN data were min-max-resampling and mean-var-resampling with exponential distances inferior. The differences in comparison to one-value-resampling—see Fig. 4—were also marginal in most cases.

8.2.2 List-Based versus Tree-Based Cluster Kernels

The fundamental difference between list and tree-based Cluster Kernels is their update policy for the merge costs with respect to the current bandwidth. List-based Cluster Kernels are always exact at the expense of higher processing costs. Tree-based Cluster Kernels only update locally at the expense of merge costs not always being up-to-date. As the results in Fig. 4 suggest, the differences in quality between tree and list-based Cluster Kernels were negligible; their performance was nearly equal. If we take the higher processing cost of list-based Cluster Kernels into account, we state that tree-based Cluster Kernels are the better choice for practical purposes.

8.2.3 Performance of Reservoir-Based KDEs

The quality of reservoir-based KDEs relies on the current sample, whose size is constant. However, the probabilistic convergence of KDEs requires increasing sample sizes. Hence, we expect reservoir-based KDEs not to improve any further after an initialization phase. The results in Fig. 4 validate this theoretically expected behavior: The MSE of reservoir-based KDEs is constant on average. We could only achieve an improvement by increasing the sample size. However, in a few cases, reservoir-based KDEs produced estimates that were temporarily of comparable quality or superior to Cluster Kernels.

8.2.4 Performance of M-Kernels

According to Fig. 4, M-Kernels mostly failed to capture the unknown density. In the majority of cases, their MSE was constant on average; M-Kernels also did not improve any further. A closer examination revealed that they basically suffered from an inappropriately chosen bandwidth. This mostly induced an oversmoothed estimate that hid important details.

8.2.5 Performance of Cluster Kernels

Overall, Cluster Kernels achieved excellent rates of convergence for all examined data streams. They outperformed M-Kernels and reservoir-based KDEs in most cases by orders of magnitude. In a few cases, the quality of Cluster Kernels temporarily worsened, indicated by an increased MSE. This can be explained by a temporary emphasis on features that receive less weight in the overall KDE. However, Cluster Kernels have proven to be a very robust choice for estimating the densities underlying heterogeneous real-world data streams.

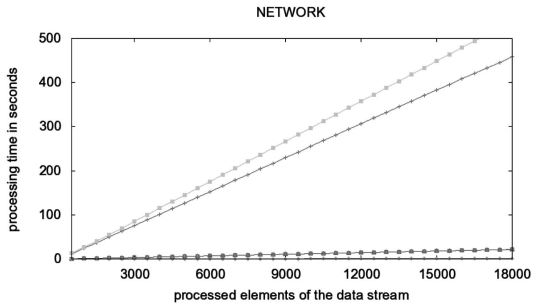

Fig. 5. Processing time in seconds.

8.3 Processing Time

A crucial aspect is the processing time of an online technique. We provide a notion of the computational complexity of the techniques described above by comparing their processing time. We set the parameters as in the last experiment and measured the time in seconds while processing the stream. Fig. 5 displays the results with line types as in Figs. 3 and 4. Overall, reservoir-based KDEs had the shortest processing time due to their low computational complexity. In contrast, M-Kernels had the longest processing time. This can be traced back to the effort of determining the mean of the merge kernel with numerical approximations. List-based Cluster Kernels, in turn, were faster than M-Kernels. However, they were also clearly inferior to tree-based Cluster Kernels due to local merge cost updates instead of a global update. Concerning the resampling strategies, one-weight and min-max-resampling nearly had the same processing time as did mean-var-resampling, which is not displayed.

8.4 Resource-Awareness

We emphasized in this work the necessity of resource-awareness as it is a fundamental premise for using an online analysis technique in a complex application. For that reason, we examined how Cluster Kernels react to sudden changes of their available amount of memory. We studied the effects which arise for tree-based Cluster Kernels with one-value-resampling over a stream of CLAW data. While processing the stream, we randomly varied the number of Cluster Kernels from a minimum of 10 to a maximum of 100, always after 5,000 elements had been processed. By examining the continuously computed MSE, we can study the impact of those memory modifications on the quality of the Cluster Kernels.

Fig. 6 summarizes the results of this experiment. While the x-axis displays the number of processed elements, the left y-axis displays the current MSE and the right x-axis the current number of Cluster Kernels. We observe that Cluster Kernels react very flexibly and adaptively to changes of their maximum number. Even significant decreases of their number only caused a momentary loss in accuracy; afterward, Cluster Kernels "recovered" again, indicated by a henceforth decreasing MSE.

9 Conclusions and Future Work

In this work, we developed Cluster Kernels, a sophisticated technique for resource-aware kernel density estimation over

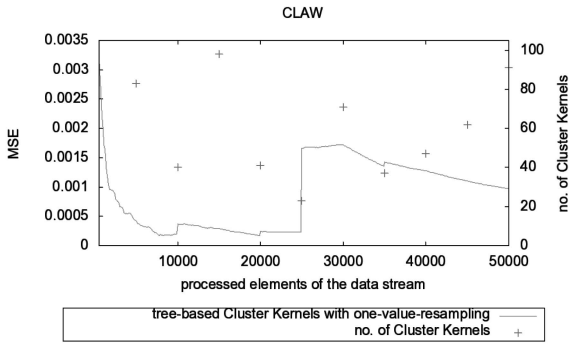

Fig. 6. Quality of Cluster Kernels for varying amounts of memory.

streaming data. We provided a generic algorithm for Cluster Kernels, whose modular design allows an online computation of KDEs by specifying exchangeable strategies for kernel function and bandwidth settings, computation of local statistics, and defining a loss function. A major advantage of Cluster Kernels is that they are assumption free and, given preset strategies, require no expert knowledge for their use. Cluster Kernels can also be seamlessly integrated into complex applications as their resource-awareness allows a well-defined adaptation to changing system resources.

We discussed Cluster Kernels for univariate data and extended the presented strategies to the multivariate case. Besides an adaptation of theoretically well-founded settings for kernel function and bandwidth, we defined a suitable loss function and elaborated resampling strategies based on simple local statistics.

An extensive experimental study substantiated the suitability of Cluster Kernels. They combined low processing cost with high rates of convergence for a variety of heterogeneous real-world data streams. They outperformed sample-based KDEs as well as M-Kernels in terms of quality by orders of magnitude.

In future work, we will address the following issues: To improve the partition representation, we aim to develop additional resampling strategies. We will also investigate whether more complex bandwidth strategies can be adapted to data streams by means of approximations. With regard to the theoretical foundation of Cluster Kernels, we will examine their consistency in dependency on the available memory. For multivariate Cluster Kernels, we aim to elaborate on the approximate computation of the merge kernel and the adequate setting of the underlying data structure.

ACKNOWLEDGMENTS

This work was supported by the German Research Society (DFG) under Grant SE 553/4-3.

REFERENCES

[1] P. Domingos and G. Hulten, "A General Framework for Mining Massive Data Streams," *J. Computational and Graphical Statistics,* 2003.

[2] A. Gray and A. Moore, "Nonparametric Density Estimation: Toward Computational Tractability," *Proc. Third IEEE Int'l Conf. Data Mining,* 2003.

[3] B. Silverman, *Density Estimation for Statistics and Data Analysis.* Chapman and Hall, 1986.

[4] D.W. Scott, *Multivariate Density Estimation: Theory, Practice, and Visualization.* John Wiley & Sons, 1992.

[5] E. Keogh and T. Folias, "The UCR Time Series Data Mining Archive," www.cs.ucr.edu/~eamonn/TSDMA, 2002.

[6] Z. Cai, W. Qian, L. Wei, and A. Zhou, "M-Kernel Merging: Towards Density Estimation over Data Streams," *Proc. Eighth Int'l Conf. Database Systems for Advanced Applications,* 2003.

[7] M. Gaber, A. Zaslavsky, and S. Krishnaswamy, "Mining Data Streams: A Review," *SIGMOD Record,* vol. 34, no. 2, 2005.

[8] L. Auvil, Y.D. Cai, D. Clutter, J. Han, G. Pape, and M. Welge, "MAIDS: Mining Alarming Incidents from Data Streams," *Proc. ACM SIGMOD,* 2004.

[9] S. Ben-David, J. Gehrke, and D. Kifer, "Detecting Change in Data Streams," *Proc. 30th Int'l Conf. Very Large Data Bases,* 2004.

[10] C. Aggarwal, J. Han, J. Wang, and P. Yu, "A Framework for Clustering Evolving Data Streams," *Proc. 29th Int'l Conf. Very Large Data Bases,* 2003.

[11] W.-G. Teng, M.-S. Chen, and P.S. Yu, "Resource-Aware Mining with Variable Granularities in Data Streams," *Proc. Fourth IEEE Int'l Conf. Data Mining,* 2004.

[12] C. Heinz and B. Seeger, "Resource-Aware Kernel Density Estimators over Streaming Data," *Proc. 15th ACM Int'l Conf. Information and Knowledge Management,* 2006.

[13] C. Heinz and B. Seeger, "Towards Kernel Density Estimation over Streaming Data," *Proc. 13th Int'l Conf. Management of Data,* 2006.

[14] C. Heinz, "Density Estimation over Data Streams," PhD dissertation, Univ. of Marburg, 2007.

[15] V. Vapnik and S. Mukherjee, "Support Vector Method for Multivariate Density Estimation," *Advances in Neural Information Processing Systems,* vol. 12, 2000.

[16] G. Kollios, D. Gunopulos, N. Koudas, and S. Berchtold, "An Efficient Approximation Scheme for Data Mining Tasks," *Proc. Int'l Conf. Data Eng.,* 2001.

[17] D. Gunopulos, G. Kollios, V.J. Tsotras, and C. Domeniconi, "Selectivity Estimators for Multidimensional Range Queries over Real Attributes," *The VLDB J.,* vol. 14, no. 2, 2005.

[18] B. Blohsfeld, D. Korus, and B. Seeger, "A Comparison of Selectivity Estimators for Range Queries on Metric Attributes," *Proc. ACM SIGMOD,* 1999.

[19] C. Lambert, S. Harrington, C. Harvey, and A. Glodjo, "Efficient On-Line Nonparametric Kernel Density Estimation," *Algorithmica,* vol. 25, no. 1, 1999.

[20] C.M. Procopiuc and O. Procopiuc, "Density Estimation for Spatial Data Streams," *Proc. Ninth Int'l Symp. Spatial and Temporal Databases,* 2005.

[21] D.K. Tasoulis, N.M. Adams, and D.J. Hand, "Unsupervised Clustering in Streaming Data," *Proc. Sixth IEEE Int'l Conf. Data Mining,* 2006.

[22] S. Subramaniam, T. Palpanas, D. Papadopoulos, V. Kalogeraki, and D. Gunopulos, "Online Outlier Detection in Sensor Data Using Non-Parametric Models," *Proc. 32nd Int'l Conf. Very Large Data Bases,* 2006.

[23] B. Turlach, "Bandwidth Selection in Kernel Density Estimation: A Review," 1993.

[24] P. Hall, S.N. Lahiri, and Y.K. Truong, "On Bandwidth Choice for Density Estimation with Dependant Data," *Annals of Statistics,* vol. 23, 1995.

[25] D. Bosq, *Nonparametric Statistics for Stochastic Processes.* Springer, 1998.

[26] T.F. Chan, G. Golub, and R. LeVeque, "Algorithms for Computing the Sample Variance: Analysis and Recommendations," *The Am. Statistician,* vol. 37, 1983.

[27] J. Nelder and R. Mead, "A Simplex Method for Function Minimization," *Computer J.,* vol. 7, 1965.

[28] E. Dellis and B. Seeger, "Efficient Computation of Reverse Skyline Queries," *Proc. 33rd Int'l Conf. Very Large Data Bases,* 2007.

[29] J.S. Vitter, "Random Sampling with a Reservoir," *ACM Trans. Math. Software,* 1985.

[30] J. Krämer and B. Seeger, "PIPES—A Public Infrastructure for Processing and Exploring Streams," *Proc. ACM SIGMOD,* 2004.

Christoph Heinz received the PhD degree in computer science from the University of Marburg, Germany, in 2007. During his PhD studies, he worked on a project on query processing over data streams funded by the German Research Foundation (DFG). His research interests include statistical modeling and mining of data streams, in particular the nonparametric estimation of the probability density function of streams. He is a senior member of the development team of the open source library XXL.

Bernhard Seeger received the PhD degree in computer science from the University of Bremen, Germany. He is a professor of database systems in the Institute of Computer Science at the University of Marburg, Germany. His research interests include spatial and temporal database systems, indexing and query processing techniques, and data stream management. He is an associate editor of *The VLDB Journal* and has been a committee member for all major database conferences. He is the head of the development team of the open source library XXL.