CLARO: Modeling and Processing of High-Volume Uncertain Data Streams

Thanh Tran, Liping Peng, Boduo Li, Yanlei Diao, Anna Liu
University of Massachusetts Amherst

Abstract— **Uncertain data streams, where data is *incomplete*, *imprecise*, and even *misleading*, have been observed in many environments. Feeding such data streams to existing stream systems produces results of unknown quality, which is of paramount concern to monitoring applications. In this paper, we present the Claro system that supports stream processing for uncertain data naturally captured using *continuous random variables*. Claro employs a unique data model that is flexible and allows efficient computation. Built on this model, we develop evaluation techniques for complex relational operators, i.e., aggregates and joins, by exploring advanced statistical theory and approximation. Evaluation results show that our techniques can achieve high performance while satisfying accuracy requirements, and outperform a state-of-the-art sampling-based method significantly. A case study further shows that our techniques can enable a tornado detection system (for the first time) to produce detection results at stream speed and with much improved quality.**

I. Introduction

Uncertain data streams, where data is *incomplete*, *imprecise*, and even *misleading*, have been observed in a variety of environments, such as sensor networks measuring temperature and light [10], [17], radio frequency identification (RFID) networks [19], [31], GPS systems [20], and weather radar networks [23]. As these data streams are collected by monitoring applications, they often undergo sophisticated query processing to derive useful high-level information. However, feeding uncertain data streams directly to existing stream systems can produce results of unknown quality. This issue is of paramount concern to monitoring applications that trigger actions based on the derived information.

Our work is particularly motivated by two emerging applications. The first is object tracking and monitoring using RFID readers [31]. RFID data streams are highly noisy due to the sensitivity of sensing to the orientation of reading and environmental factors such as metal objects and interference. When such streams are used to support tracking and monitoring, for instance, to detect safety violations regarding flammable objects, the quality of the alerts raised is a critical issue to the end application.

The second application is tornado detection [23], where meteorological data streams are collected from a radar network and processed in a real-time stream system. Data uncertainty can arise from environmental noise, device noise, and inaccuracies of various radar components. Such uncertainty can propagate all the way through the stream system, making tornado detection results error-prone. Given the potential social impact, it is absolutely vital that such systems capture the quality of detection results.

In this paper, we address uncertain data stream processing for data that is naturally modeled using *continuous random variables*, such as many types of sensor data and financial data. Given such data, our work supports relational query processing under uncertainty. For each relational operator, we aim to fully characterize the distribution of each tuple produced from uncertain data. Such distributions, called *result tuple distributions*, can be used to return any statistics needed, e.g., mean, variance, and confidence regions. They also allow the stream system to feed these tuples as input to other operators and characterize the results of further processing—it is evident that merely having statistics such as mean and variance for those tuples is not enough to do so.

Challenges. Uncertain data stream processing as described above raises two challenges: First, it is computationally difficult to obtain result distributions when input tuples are modeled using continuous random variables. Such computation often involves multivariate integrals or requires new algorithms if an integral-based solution does not exist. Second, such computation must be performed efficiently for high-volume data streams. While approximation is a common approach to improving efficiency, the technique must be able to achieve a small bounded error while meeting performance requirements.

Despite a large body of recent work on uncertain data management, the above challenges have not been adequately addressed. Most probabilistic databases [1], [2], [5], [26], [28], [32] and stream systems [6], [18], [21] model tuples using *discrete* random variables and evaluate queries using the possible worlds semantics. The continuous nature of our data, however, precludes the use of these techniques as the possible values of a continuous random variable are infinite and cannot be enumerated.

Recent work that considers continuous random variables has taken two approaches to handling aggregates: The integral-based approach [5] performs exact derivation of result distributions. While it is accurate, its computation is too slow for stream processing, as we shall show later in this paper. The sampling-based approach [14], [29] employs approximation by discretizing continuous distributions and sampling from the resulting distributions. However, for real-world data it is difficult, sometimes impossible, to find the right number of samples that guarantees both accuracy and efficiency, as we shall also show in our performance study. Joins of continuous random attributes have only been addressed at the modeling level [29], lacking both evaluation techniques and demonstrated performance for data streams.

Scope and contributions. In this paper, we present a probabilistic data stream system, which we call Claro, that supports relational processing of uncertain data streams modeled by continuous random attributes. The architectural design of Claro, as described in [12], is to extend the box-arrow paradigm for stream processing [3]

such that tuples carry distributions to describe uncertainty and relational operators transform these distributions while processing tuples. This paper, in particular, focuses on the data model and processing algorithms for two complex operators, *aggregates* and *joins*. These operators are crucial to our target applications but have not been sufficiently addressed as described above. Further in the streaming context, our goal is to perform such complex operations at high speed, e.g., thousands of tuples per second. More specifically, our contributions include:

Data model. The foundation of Claro is a unique data model based on Gaussian Mixture distributions. This model is highly flexible in that it subsumes common Gaussian distributions and can model arbitrary real-world distributions. It further allows efficient computation by using powerful statistical methods for continuous variables and Gaussian properties. Our data model stands in contrast to those based on histograms [14] and weighted particles [20] which indicate the use of samples in computation. Moreover, our model has the potential to support a variety of relational operators. While our work focuses on aggregates and joins, an extension to other operators is discussed at the end of the paper.

Aggregates. Our data model empowers us to explore advanced statistical theory, such as characteristic functions, to obtain exact result distributions of aggregation while completely eliminating the use of integrals (in contrast to using multiple integrals in [5]). However, the formulas for result distributions that the exact algorithm produces grow exponentially in the number of aggregated tuples. Hence, we provide two approximation methods to simplify the formulas for result distributions. These techniques, when combined into a hybrid solution, can satisfy arbitrary application-specified accuracy requirements while achieving high speed in stream processing.

Joins. Our data model also enables efficient and accurate techniques for joins. To produce proper results for different application semantics, we propose two types of joins. The first type models an equi-join between a continuous random attribute and a deterministic attribute using an outer join over a probabilistic view. Our techniques include efficient regression to construct the view and a closed-form solution to representing result distributions in the form of Gaussian Mixture models (GMMs). The second type of join is modeled by a cross-product followed by a selection. Claro supports such joins with result distributions also in GMMs and an efficient method to remove the results with low existence probabilities.

Evaluation. We perform a thorough evaluation of our techniques for joins and aggregates, and compare them with sampling-based methods ([14] for aggregates and a home-grown method for joins). Results of this study demonstrate our advantages in both accuracy and speed over the sampling-based methods, due to the use of our data model and techniques for continuous random variables. We further perform a case study in the tornado detection application, in which a real trace collected from a tornadic event is fed to the Claro system. Our results show that fitting data to the Claro model and using its processing techniques allows the tornado detection algorithm (for the first time) to produce detection results at stream speed and with much improved quality.

The remainder of the paper is organized as follows. We detail our motivating applications in §II. We present our data model in §III, and main techniques for aggregates and joins in §IV and §V. Evaluation results are described in §VI. §VII covers related work. Finally, §VIII concludes the paper with remarks on future work.

II. Motivating Applications

In this section, we present two motivating applications.

A. Object Tracking and Monitoring

In the first application, radio frequency identification (RFID) readers are used to monitor a large area such as a warehouse, a retail store, or a library. RFID data is known to be highly noisy [19], [31] due to environment factors such as occluding metal objects and interference. Moreover, mobile RFID readers may read objects from arbitrary angles, hence particularly susceptible to variable read rates. Our prior work [31] provides techniques to transform raw RFID readings into a stream of location tuples. Each location tuple contains ($time$, tag_id, x^p, y^p), where x^p and y^p denote the inferred location of the object and are probabilistic in nature.

Despite the data uncertainty, monitoring applications want to run queries on the location stream to derive high-level information. The first query illustrates an example in fire monitoring: "trigger an alert when a flammable object is exposed to a high temperature." This query takes two inputs: a location stream as described above for flammable objects, and a temperature stream from sensors in the same area with attributes ($time$, $sensor\_id$, x, y, $temp^p$), where the temperature can be uncertain due to sensing noise. The query joins the location stream with the temperature stream based on the location. The query is written as if the location of an object and the temperature at a location were precise.

```
Q1: Select Rstream(R.tag_id, R.x, R.y, T.temp)
    From   FlammableObject [Now] As R,
           Temp [Partition By sensor_id Rows 1] As T
    Where  T.temp > 60 °C and
           R.x = T.x and R.y = T.y
```

The second query detects violations of a shipping policy by the Food and Drug Administration (FDA): "food with and without peanuts cannot be located closely in the supply chain." This query takes two location streams and checks for the proximity of two types of food.

```
Q2: Select Rstream(R.tag_id, R.x, R.y, S.tag_id)
    From   PeanutFreeFood [Range 3 minutes] As R,
           PeanutFood [Range 3 minutes] As S
    Where  |R.x - S.x| < 3 ft and |R.y - S.y| < 3 ft
```

B. Hazardous Weather Monitoring

The CASA Research Center is developing weather radar networks to detect hazardous weather events such as tornados and storms [23]. A four-radar testbed has been deployed in southwestern Oklahoma, a region that

receives an average of four tornado warnings and 53 thunderstorm warnings each year [23].

A CASA radar node rotates to scan. It sends around 2000 pulses per second, alternating between 54 high frequency pulses and 40 low frequency ones. The raw data obtained is partitioned into high and low frequency segments accordingly, and further partitioned based on the distance to the radar. Overall, the raw data is generated at a rate of 175Mb per second. Such data is highly noisy due to electronic device noise, instability of transmit frequency, quality issues of the system clock and the antenna, and finally environmental noise.

Such high-volume noisy raw streams are fed into a stream system for real-time weather event detection. The current CASA system addresses both data volume and noise issues by means of taking average. Fig. 1 shows a simplified diagram of the system. The top box depicts the generation of wind velocity from raw data. This module applies Fast Fourier Transform (FFT) to each stream segment and performs signal processing. It then outputs a single velocity value for each segment and averages the values for adjacent high and low frequency segments. The reflectivity analysis, shown in the lower part of Fig. 1, uses similar average operations. While such average operations can reduce data volume and gain a smoothing effect, the resulting data is still highly noisy, causing *low quality detection results* and *long running time*.

In our case study (detailed in §VI-C), we explore the use of distributions, rather than simple average values, to separate useful data from noise while controlling the data volume. The output of our data analysis contains tuple streams with distributions, i.e., (*time*, *azimuth*, *distance*, *velocity*^{p} or *reflectivity*^{p}). We also study the transformation of these distributions through CASA operations, specially, the frequently used aggregation operations. By doing so, we expect to gain better tornado detection results yet with shorter execution time.

III. The CLARO Data Model

The foundation of the Claro system is a data model based on Gaussian mixture distributions that can capture a variety of uncertainties for continuous attributes and further allow fast relational processing. In this section, we introduce Gaussian mixture distributions and describe the complete Claro data model for relational processing.

A. Gaussian Mixture Models (GMMs)

Gaussian Mixture Models (or distributions), abbreviated as GMMs, are traditionally used for data clustering and density estimation. As an instance of probability mixture models, a GMM describes a probability distribution using a convex combination of Gaussian distributions.

Definition 1 *A Gaussian Mixture Model for a continuous random variable X is a mixture of m Gaussian variables X_1, X_2, $\cdots$, X_m. The probability density function (pdf) of X is:*

$$f_X(x) = \sum_{i=1}^{m} p_i f_{X_i}(x),$$

$$f_{X_i}(x) = \frac{1}{\sigma_i\sqrt{2\pi}} e^{-\frac{(x-u_i)^2}{2\sigma_i^2}} \qquad (X_i \sim N(u_i, \sigma_i^2)),$$

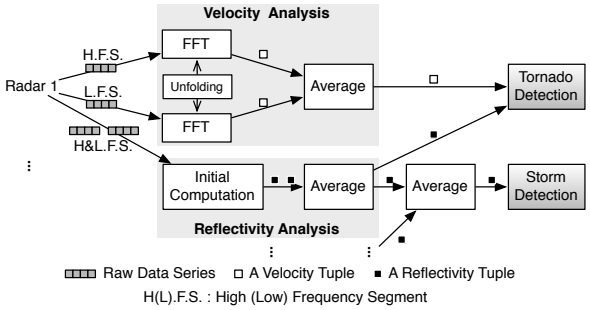


Fig. 1. Simplified stream processing in the CASA radar system

where $0 \le p_i \le 1$, $\sum_{i=1}^{m} p_i = 1$, and each mixture component is a Gaussian distribution with mean u_i and variance σ_i^2.

Definition 2 *A multivariate Gaussian Mixture Model for a random vector* $\mathbf{X}$ *naturally follows from the definition of multivariate Gaussian distributions:*

$$f_{\mathbf{X}}(\mathbf{x}) = \sum_{i=1}^{m} p_i f_{\mathbf{X}_i}(\mathbf{x}),$$

$$f_{\mathbf{X}_i}(\mathbf{x}) = \frac{1}{(2\pi)^{k/2}|\Sigma_i|^{1/2}} e^{-\frac{1}{2}(\mathbf{x}-\mathbf{u}_i)^T\Sigma_i^{-1}(\mathbf{x}-\mathbf{u}_i)} \quad (\mathbf{X}_i \sim N(\mathbf{u}_i, \Sigma_i)),$$

where k is the size of random vector, and each mixture component is a k-variate Gaussian distribution with mean $\mathbf{u}_i$ *and covariance matrix* Σ_i.

The Claro system adopts Gaussian Mixture Models due to several key benefits of these models. First, GMMs are a natural extension to Gaussian distributions which are widely used in scientific sensing and financial applications. Hence, they can be easily accepted by end users such as the CASA scientists we are working with.

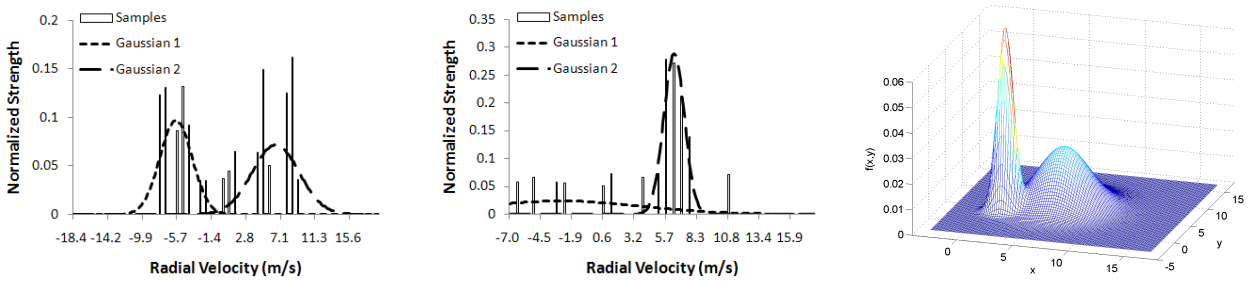
Second, GMMs are highly flexible in modeling arbitrary real-world distributions. For instance, Fig. 2(a) shows a detected bimodal distribution of velocity at the boundary between a positive velocity area and a negative velocity area in a tornadic event. In contrast, Fig. 2(b) shows a velocity distribution in a positive velocity area, where one Gaussian component captures the peak and the other captures the noise spread across the entire spectrum. In the RFID application, Fig. 2(c) shows the inferred location distribution of a recently moved object [31]. Here, the bivariate, bimodal GMM represents the possibilities of the old and new locations using two mixture components; each component is a bivariate Gaussian modeling the joint distribution of x and y locations.

The third benefit of GMMs is efficient computation based on Gaussian properties and advanced statistical theory. First, the mean and variance of GMMs can be computed directly from those of the mixture components:

$$E[X] = \sum_{i=1}^{m} p_i E[X_i] \tag{1}$$

$$Var[X] = \sum_{i=1}^{m} p_i (Var[X_i] + (E[X_i])^2) - (E[X])^2 \tag{2}$$

Furthermore, the cumulative distribution function (cdf) of a GMM with a single variable has an analytic (closed form) expression based on a known error function. Values of the error function are precomputed in any numerical



(a) CASA: Velocity distribution after FFT in Area(430, 281.9°) in a tornadic event

(b) CASA: Velocity distribution after FFT in Area(430, 282.3°) in a tornadic event

(c) RFID: Location distribution of a recently moved object detected using RFID readers

Fig. 2. **Gaussian Mixture Models for real-world data collected from our target applications**

library. Hence, computing $\int_a^b f_X(x)\mathbf{d}x = F_X(a)\text{-}F_X(b)$ using the cdf incurs little cost. Other computation benefits of GMMs, such as the characteristic functions, product distributions, and linear transformation, are described later in relevant sections.

B. Generating GMMs from Real-World Data

Gaussian Mixture Models can be generated from real-world data in a variety of ways.

Samples. Recent studies [20], [31] have developed inference techniques that transform raw data to distributions represented using weighted samples. Given these samples, GMMs can be generated using existing tools for density estimation or function fitting. If an application models data using other distributions, e.g., the Gamma distribution, it is easy to generate samples from this distribution and then fit a GMM as described above.

Correlated time series. Time series data is prevalent in many applications. Values in a time series are temporally correlated so cannot be viewed as samples to fit GMMs. Two techniques can be applied in this case:

Fast Fourier Transform (FFT) translates a correlated data sequence in the time domain to an uncorrelated sequence in the frequency domain. The latter is essentially a discrete distribution that can be used to fit a GMM. Although FFT has the $O(n \log n)$ complexity, where n is the sequence length, doing so for short subsequences of data does not incur high overhead. In fact, the CASA system has already applied FFT to the streams arriving at 175 Mb/sec. We will show the use of this technique in our case study in Section VI-C.

Another method is to use the autoregressive moving average (ARMA) model which restricts data correlations to the past n time steps. Our previous study applied ARMA fitting to the radar data streams with and without a tornado [12]. We discovered that in either case, the ARMA model with n=5 satisfies the statistical condition for fitting. Given such models, we can perform sampling at the frequency of once every n+1 values and feed the samples to fit GMMs.

The Claro system offers all above methods for data providers to transform raw data to tuples modeled using GMMs. Below, we assume that the input to Claro already has uncertain data modeled using GMMs.

C. CLARO Data Model

We now present the complete Claro data model for relational processing. An uncertain data stream is an infinite sequence of tuples that conform to the schema $\mathcal{A}^d \cup \mathcal{A}^p$. The attributes in $\mathcal{A}^d$ are *deterministic attributes*, such as the tuple id and the fixed x-y location of a sensor. The attributes in $\mathcal{A}^p$ are real-valued *probabilistic attributes*, such as the temperature of a location and wind velocity in an area. The probabilistic attributes are further partitioned into independent attributes $A_i^p \in \mathcal{A}^p$ and groups of correlated attributes $\mathbf{A}_j^p \subseteq \mathcal{A}^p$. An independent probabilistic attribute is modeled using a *continuous random variable* following a Gaussian Mixture distribution, denoted by $f(A_i^p)$. A set of correlated attributes $\mathbf{A}_j^p$ is modeled using a multivariate Gaussian Mixture distribution, denoted by $f(\mathbf{A}_j^p)$. Then the *tuple distribution* is defined as:

$$f(\mathcal{A}^p) = \prod_i f(A_i^p) \prod_j f(\mathbf{A}_j^p),$$

which is a multivariate Gaussian Mixture distribution.

In some scenarios, tuples in a stream can be correlated. Inter-tuple correlations can be modeled using joint tuple distributions or lineage [2]. Our current model does not include such correlations for two reasons: First, while raw data is often temporally correlated, our methods for transforming raw data to tuples, such as FFT and ARMA-based sampling, have already taken such correlations into account. Second, given stringent performance requirements stream systems may sometimes have to sacrifice inter-tuple correlations. For instance, the CASA system ignores spatial correlations in any processing before final tornado detection, and probabilistic stream systems [20], [25] ignore inter-tuple correlations, all for performance reasons. A thorough treatment of tuple correlations in stream processing is a focus of our future work.

In the rest of the paper, we present evaluation techniques for aggregates and joins under the Claro model.

IV. Aggregation of Uncertain Tuples

We first address aggregation of uncertain tuples. In this work, we focus on `sum` and `avg` because they are crucial to our target applications but have not been

sufficiently addressed in the literature.[1] Recent work on aggregation in probabilistic databases [1], [8] mostly focuses on discrete attributes and employs possible worlds semantics for query processing. For continuous attributes, however, the computation for aggregation is by nature a multivariate integral. For instance, with n tuples modeled using continuous random variables $X_1, \cdots, X_n$, $\int \cdots \int_{x_1+\cdots+x_n<x} f_{X_1}(x_1) \cdots f_{X_n}(x_n)\ \mathrm{d}x_1 \cdots \mathrm{d}x_n$ computes the cumulative distribution function for the sum of these tuples. Since multivariate integration is a prohibitively expensive operation, there have been two main approaches to addressing this issue:

The *integral-based approach* derives exact result distributions by manipulating continuous random variables. To sum n such variables, the state-of-the-art solution [5] integrates two variables at a time, hence using n-1 integrals for the aggregation. As we shall show in § IV-A, this technique is infeasible for stream processing.

The *sampling-based approach* [14], [29] generates a fixed number of samples from the distribution of each input tuple, computes aggregate values from these samples, and constructs the result distribution using the aggregate values. Despite its generality, its approach has two main drawbacks: First, it is unknown how many samples are needed a priori. Using a small number of samples trades off accuracy for performance; using a large number has the opposite problem. Second, the sampling-based approach does not acquire knowledge of the true result distribution and hence cannot adapt to varying data characteristics and accuracy requirements.

Our work departs from existing approaches by exploring advanced statistical theory to obtain *exact result distributions* while completely eliminating the use of integrals. However, the formulas for result distributions that the exact algorithm produces grow exponentially in the number of aggregated tuples. Hence, we provide two *approximation* techniques to simplify the formulas for result distributions while satisfying accuracy requirements and achieving high speed in stream processing. It is important to note that such accuracy cannot be guaranteed without the knowledge of the true result distribution.

A. A Basic Algorithm using Characteristic Functions

We first introduce *characteristic functions* and describe a basic algorithm to derive the result distribution for `sum` of a set of tuples. The modification to `avg` is straightforward and hence omitted in the following discussion.

In probability theory, characteristic functions (CFs) are used to "characterize" distributions. Specifically, the CF of a random variable U is defined as (chapter 2, [4]):

$$\Phi_U(t) = E[e^{iUt}],$$

where E denotes the expected value and i is the complex number $\sqrt{-1}$. The pdf of U then can be obtained by the inverse transformation of the CF:

$$f_U(x) = \frac{1}{2\pi} \int_{-\infty}^{+\infty} e^{-itx} \Phi_U(t) dt.$$

Now let us consider `sum`(A), with the attribute A in n tuples modeled using random variables X_1, ..., X_n . Let $U = X_1 + X_2 + ... + X_n$. The CF of U is:

$$\Phi_U(t) = Ee^{iUt} = Ee^{i(X_1+X_2+...+X_n)t} \quad (3)$$
$$= \Phi_{X_1(t)}\Phi_{X_2(t)}...\Phi_{X_n(t)} \quad (4)$$

That is, the CF of U can be written as the product of the CFs of the input tuples based on the independence assumption. This suggests a simple algorithm for `sum`: (1) get the CF of each input tuple, (2) take the product of these functions, (3) for a given value x, apply the inverse transformation at x to yield $f_U(x)$, which we call a *parameterized integral*.

In the context of Gaussian Mixture Models (GMMs), the CFs can be expressed in closed form. For example, for a Gaussian mixture of two components:

$$f(x) = p_1 \frac{1}{\sigma_1\sqrt{2\pi}} e^{-\frac{(x-\mu_1)^2}{2\sigma_1^2}} + p_2 \frac{1}{\sigma_2\sqrt{2\pi}} e^{-\frac{(x-\mu_2)^2}{2\sigma_2^2}},$$

its CF can be written directly as:

$$\Phi_X(t) = p_1 e^{i\mu_1 t - \frac{1}{2}\sigma_1^2 t^2} + p_2 e^{i\mu_2 t - \frac{1}{2}\sigma_2^2 t^2}$$

Thus, step 1 above does not involve any integration. The main computation is the parameterized integral for the inverse transformation. This gives a boost in performance compared to the two-variable convolution method, which requires n-1 parameterized integrals [5].

The main limitation of this approach is that the formula of the result distribution involves an unresolved parameterized integral. To get sufficient knowledge of the result distribution (e.g., calculating its mean and variance), one needs to repeat the inverse transformation for a large number of points. To understand the cost of such repeated integration, we used a **numerical solution** called adaptive quadrature [27] to compute integrals. The task is to average over 10 tuples and compute the pdf values for 20 points. We applied optimizations such as restricting the range of integration and limiting the maximum number of iterations. The throughput obtained is less than 200 tuples/second. This indicates that this technique is inefficient for our data stream applications. Moreover, it is unknown if the result distribution is a GMM.

B. Exact Derivation of Result Distributions

The discussion in the previous section motivated us to seek a solution without using numerical integration. For GMMs, it turns out that we can obtain the **closed-form** solution to the inverse transformation. In addition, when input tuples are Gaussian mixtures and independent, the result of `sum` over those tuples is also a Gaussian mixture that can be directly obtained from the input tuples.

Theorem IV.1 *Let each $X_i, (i = 1..n)$ be a mixture of i_m components identified by the parameters $(p_{ij}, \mu_{ij}, \sigma_{ij}), (j = 1..i_m)$. The result distribution for $U = \sum_{i=1}^{n} X_i$ is a Gaussian*

[1] Our system also supports `min` and `max`. Our techniques are similar to those in [5] and hence omitted in this paper.

mixture of $\prod_{i=1}^{n} i_m$ components, each of which corresponds to a unique combination that takes one component from each input Gaussian mixture $\{i_j\}, (i = 1..n, j \in \{1..i_m\})$ and is identified by (p_k, μ_k, σ_k):

$$p_k = \prod_{i=1}^{n} p_{i_j};\ \mu_k = \sum_{i=1}^{n} \mu_{i_j};\ \sigma_k = \sqrt{\sum_{i=1}^{n} \sigma_{i_j}^2}. \quad (5)$$

Proof: See Appendix.

To the best of our knowledge, we are not aware of any state-of-the-art book on mixture models [24], [13] showing this result.

This technique gives an exact solution so the accuracy is guaranteed. The computation involved is to enumerate and compute all components of the result Gaussian mixture. Let N be the number of input tuples and M be the average number of mixture components in each input tuple. The result formula size is then $O(M^N)$. Computing one component of the result formula requires multiplication and addition of N input tuple parameters as shown in Eq. 5; thus, the time complexity is $O(NM^N)$.

As the above analysis shows, the result formula grows exponentially in the number of aggregated tuples, raising a scalability issue with this technique. We next describe two approximation techniques to address this issue.

C. Approximation using the Closed-Form Solution

Our first approximation technique simplifies the result distribution formula while satisfying the accuracy requirement. (This task is similar in nature to approximate lineage [26] by replacing a complex lineage formula with a simpler one.) The idea is to group adjacent Gaussian peaks in the exact result formula and approximate each group using a single Gaussian component. The resulting approximate distribution is more user-friendly and incurs lower cost in subsequent processing. Our algorithm, referred to as **sort-group**, has the following main steps:

Algorithm 1 Approximation using **sort-group**

1: Compute all T components of the result distribution based on Theorem IV.1. Each component is a Gaussian $N(\mu_i, \sigma_i)$ with a coefficient p_i.
2: Sort the components in increasing order of μ_i.
3: Start with $K = 1$. Group all T components into K new Gaussian components: each of them replaces $\lfloor \frac{T}{K} \rfloor$ original components, except for the last replacing the last $(T - (K-1)\lfloor \frac{T}{K} \rfloor)$ original components.
4: Calculate the approximate point-based variation distance (VD) to see if it satisfies the given accuracy constraint. If so, return the mixture of K Gaussian components; otherwise, increase K and go to step 3.

The above algorithm searches for an approximate result GMM with K components that achieves the accuracy requirement. It first generates the exact result GMM and sorts the components by their means. Then it groups the consecutive components first by normalizing them into a new Gaussian mixture, so that mean and variance can be computed, and then by approximating them using a single Gaussian with the computed mean and variance. The search starts with K set to a small number and increases it until the accuracy is satisfied.

In step 4, the point-based VD is a metric for measuring the distance between distributions. For two continuous distributions $D_1(x)$ and $D_2(x)$, it is defined as:

$$VD = \frac{1}{2} \sum_x |D_1(x) - D_2(x)| \Delta x$$

The values of x are evenly spaced in the range where the two distributions have most of their density mass. Δx is the distance between two consecutive x points. The constant $\frac{1}{2}$ ensures that VD is in [0,1]. The metric is similar in idea to that used in [14]. When calculating the approximate VD, we use a small number of points, as opposed to a larger number (i.e., 1000) used in accuracy measurement, to reduce the computation overhead. It is experimentally shown that we can achieve a reasonable approximation of the true VD using 30 points.

We next consider the time complexity of this algorithm. Step 1 has the same complexity as in exact derivation, $O(NM^N)$. In step 2, sorting has a complexity of $O(TlogT)$, with $T = M^N$, which yields $O(NM^N)$. In step 3, grouping involves enumerating M^N components and is repeated K times, hence a cost $O(KM^N)$. In step 4, point evaluations take a time proportional to the number of points used (e.g., 30) and the formula size. The exact result formula size is M^N while the approximate one varies from 1 to K across iterations. So the cost of step 4 is $O(f_P(M^N + K^2))$, where f_P denotes the cost of evaluations for P points. As the analysis shows, our approximation technique results in a simplified formula of size K, but still has a time complexity exponential in the number of tuples (N). In fact, it has a somewhat higher cost compared to exact derivation due to the costs of steps 2-4 above.

D. Approximation using Characteristic Function Fitting

The previous approximation still enumerates all components of the result formula and becomes very inefficient when the number of input tuples is large. We next propose to approximate result distributions by performing function fitting in the Characteristic Function (CF) space. This is based on the property that the CF of `sum` can be compactly represented as a product of n individual CFs (Eq. 4), instead of an exponential number of components (Eq. 5). The goal is to find some Gaussian mixture whose CF best fits this product function.

We devise an approximation algorithm, named **Characteristic Function (CF) fitting**. The sketch of the algorithm is shown below. The algorithm searches for the right number of components by starting with one component Gaussian mixture, running the least squares fitting. If the fitting residual is below a threshold, return the fitted parameters; otherwise increase the number of components and repeat fitting. Note that the objective function for fitting contains both *real* and *imaginary* parts since the CFs are complex functions and both parts contribute to the pdf via inverse transformation.

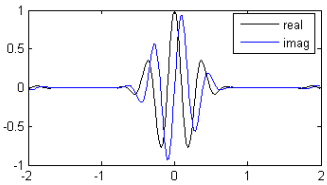


Fig. 3. Example of Characteristic Function for avg of 10 tuples.

Algorithm 2 Approximation using **CF fitting**

1: Obtain the expression of the CF of the sum, $\Phi_{\text{sum}}(t) = \prod_{i=1}^{n} \Phi_{X_i}(t)$. This is a complex function.
2: Take P points $\{t_i\}, (i = 1..P)$ from the domain, and compute $\{\Phi_{\text{sum}}(t_i)\}, (i = 1..P)$.
3: Start with $K = 1$. Consider a Gaussian mixture of K components. The corresponding CF is $\overline{\Phi}(t)$.
4: Run least squares fitting to minimize: $\sum_{i=1}^{P} \left[(Re(\overline{\Phi}(t_i) - \Phi_{\text{sum}}(t_i)))^2 + (Im(\overline{\Phi}(t_i) - \Phi_{\text{sum}}(t_i)))^2\right]$.
5: Get the fitting residual. If this is smaller than a threshold ϵ, return the fitted Gaussian mixture. Otherwise, increase K and go back to step 3.

We further propose several optimizations based on statistical theory to improve performance and accuracy.

Range of CFs: Since the characteristic function $\Phi(t)$ approaches 0 fast as the magnitude of t increases, the range $[C_1, C_2]$ needs to be small and centered around 0. See Figure 3 for an example of CF for avg of 10 tuples. Both real and imaginary parts of the function are shown. Hence, we use $[-1, 1]$ in our algorithm as the range can capture the shape of the CFs in most cases.

Inital guesses: Due to the oscillating behavior of the characteristic functions, the fitting results are quite sensitive to the initial values and can get stuck in local optima. Finding good initial guesses for fitting can be as hard as fitting itself. Theorem IV.1 provides insights into choosing these initial guesses. Specifically, we precompute a fixed number of result components (e.g., 40) and use them to derive the guesses. These components are chosen so that their means are separated to capture different regions of the result distribution. At each iteration, they are grouped into K components that are used as initial guesses.

Choice of fitting residual: The fitting residual ϵ is chosen to guarantee the given VD requirement. We have performed an approximation analysis and derived an upper bound for ϵ. The proof is based on the application of Cauchy-Schwartz inequality to characteristic functions and the property that these functions approach 0 quickly for points far from the origin to restrict the integration bounds. Given our choice of other paramaters, we can show that $\epsilon \leq 6 * 10^{-5} VD^2$ can meet the accuracy requirement VD.

The complexity of the main steps of this algorithm is as follows. In step 2, computing P points, each of which is a product of N complex terms with M components, has a cost linear in P, M, N. For step 4, the complexity of the least squares fitting algorithm we use is $O(p^3)$, where p is the number of parameters [33], and $p = 3k$ in our case. Since fitting is repeated for $k = 1..K$, its total cost is proportional to $I \cdot K^4$, where the factor I accounts for multiple steps required for one fitting on average. While this analysis is only approximate (due to the difficulty of bounding the fitting cost), it shows that this algorithm has eliminated the exponential cost as in sort-group. However, K^4 is a nontrivial cost as when the result distribution has a complex shape, we need many Gaussian components (e.g., K=10) to approximate it. We observe empirically that when the number of tuples N is large, K becomes small as the result distribution becomes smoother. As a result, this technique can be efficient when N is large.

Relation to the Central Limit Theorem. The Central Limit Theorem (CLT) is a special case of our algorithm. It states that the sum of a sufficiently large number of independent random variables is normally distributed [4]. This gives an asymptotic result but our algorithm dynamically determines when this result can apply. Our experiments show that the fitted result distributions are smooth single Gaussians when the number of tuples is large enough (e.g., greater than 20).

E. Hybrid Solution

The discussions above suggest a **hybrid** solution to exploit the advantages of the three algorithms: When the number of tuples is small, we use exact derivation since it is fast and its formula is not complex. When this number is larger but enumerating all the components is still possible, we use sort-group to have an approximate formula of reduced size. After that, we switch to CF fitting. This way, we exploit the advantage of each algorithm in the range where it performs the best. We also observe that the switching points among the three mainly depend on the number of tuples and less so on other data characteristics, as shown in § VI. This implies that once the hybrid solution is configured with those switching points, it can be applied to different workloads.

The hybrid solution also supports the use of windows. It can be directly applied to stream systems using tumbling windows such as CASA [23] and XStream [16]. When sliding windows are used, we employ incremental computation. First, generating the mixture formulas for exact derivation and sort-group (Eq. 5) can be incremental by factoring out the old tuples and adding the new ones. The CLT can be implemented in a similar fashion. For CF fitting and sort-group, while some parts (e.g., function fitting and point evaluation) cannot be incremental, we can use a heuristic of applying the result of the previous batch as the initial guess of the GMM parameters.

V. Joins of Uncertain Tuples

In this section we consider efficient evaluation of joins under the Claro data model. The evaluation strategies of joins vary significantly with the nature of the join attributes. Recent research on probabilistic databases [1], [2], [21], [32] has mostly focused on join attributes modeled by *discrete* random variables. Since it is possible to enumerate the values of a discrete random variable, existing work supports such joins based on the possible worlds semantics (PWS): in every possible world, each

random variable takes a specific value so a join can proceed just as in a traditional database. However, when data uncertainty is captured using *continuous* random variables, join methods based on PWS no longer work because we cannot enumerate the possible values of a continuous random variable.

Below, we propose two types of joins of continuous random attributes to suit different application semantics. One is based on the use of a *probabilistic view*, which has not been considered in previous work. The other is based on the *cross-product* semantics, for which we extend the definition in [29] with join results and efficient evaluation techniques under our model.

A. Joins using Probabilistic Views

Let us first consider query Q1 in § II-A. Given each object location, it retrieves the corresponding temperature. Most notably, the join attributes (x, y) are probabilistic in the object location stream, but are fixed sensor locations in the temperature stream. This type of join is inherently difficult to support for two reasons. First, it is not possible to enumerate the values of the continuous random attributes (x^p, y^p). Second, since each value of (x^p, y^p) has probability 0, any join result that pairs a specific value of (x^p, y^p) and a temperature tuple also has probability 0.

As can be seen, such joins compare continuous random attributes and deterministic attributes based on equality. To attain proper result distributions for them, we introduce the notion of a **probabilistic view.** In the above example, a probabilistic view on the temperature stream is defined to be the conditional distribution $p(temp|x, y)$, i.e., distribution of *temp* given a (x,y) value. Then, the process of iterating all possible values of x^p and y^p and retrieving the corresponding temperature can be compactly represented by $f(x^p, y^p)p(temp|x, y)$, yielding a joint distribution $f(x^p, y^p, temp^p)$.

Formally, let the left input R be $\{A_1^p, \cdots, A_k^p, \bar{R}\}$, where $A_1^p, \cdots, A_k^p$ are the join attributes and $\bar{R}$ denotes the rest of R. Let the right input S be $\{A_1, \cdots, A_k, B_1, \cdots, B_l, \bar{S}\}$, where $A_1, \cdots, A_k$ are the join attributes, also called the *condition attributes*, $B_1, \cdots, B_l$ are the attributes dependent on the condition attributes which the view aims to return, hence called the *view attributes*, and $\bar{S}$ is the rest of S. In the temperature stream for Q1, the condition attributes are x and y and the view attribute is *temp*. Then a probabilistic view can be defined for $B_1, \cdots, B_l$ conditioned on $A_1, \cdots, A_k$, denoted by $\mathbf{V}_{B_1^p,\cdots,B_l^p|A_1,\cdots,A_k}$, and characterized by $p(B_1, \cdots, B_l|A_1, \cdots, A_k)$. Then the join is defined as:

Definition 3 *Given $R = \{A_1^p, \cdots, A_k^p, \bar{R}\}$, $S = \{A_1, \cdots, A_k, B_1, \cdots, B_l, \bar{S}\}$ with $B_1, \cdots, B_l$ dependent on $A_1, \cdots, A_k$, an equi-join of R and S using a probabilistic view ($\bowtie^{\mathbf{V}}$) is a left outer join of R and the probabilistic view $\mathbf{V}_{B_1^p,\cdots,B_l^p|A_1,\cdots,A_k}(S)$:*

$$(R \underset{R.A_1^p=S.A_1,\cdots,R.A_k^p=S.A_k}{\bowtie^{\mathbf{V}}} S) \equiv (R \underset{R.A_1^p,\cdots,R.A_k^p}{\sqsupset\bowtie} \mathbf{V}_{B_1^p,\cdots,B_l^p|A_1,\cdots,A_k}(S))$$

In this definition, the left outer join preserves each tuple in R and extends it with the attributes $B_1^p, \cdots, B_l^p$ from S. Attributes in $\bar{S}$ are not included in the output because they are not captured by the probabilistic view.

Closed-form result distributions. Given the above definition, we next seek a solution to the distribution of each outer join result. Recall that the CLARO data model describes the join attributes in the left input using (multivariate) Gaussian Mixture Models (GMMs). Next, we propose a special model for the probabilistic view, which we call *order-1 linear regression*, that allows us to obtain join result distributions also in the form of GMMs. While the assumption of order-1 linearity may sound restrictive, it actually can be applied to the view at either a global or local scale, hence allowing implementation choices for both accuracy and efficiency.

The following two theorems offer closed-form join result distributions. They differ based on the nature of the attributes used to construct the probabilistic view.

Theorem V.1 *Given $R = \{A_1^p, \cdots, A_k^p, \bar{R}\}$, $S = \{A_1, \cdots, A_k, B_1, \cdots, B_l, \bar{S}\}$, assume order-1 linear regression:*

$$\tilde{B} = \tilde{A}\boldsymbol{\beta} + \mathbf{E} \tag{6}$$

where $\tilde{B} = (B_1, \cdots, B_l)$, $\tilde{A} = (A_1, \cdots, A_k)$, $\boldsymbol{\beta}$ is a parameter matrix of $k \times l$, and $\mathbf{E}$ is an error vector of length l, which is normally distributed with mean $\mathbf{0}$ and covariance matrix Σ_E. If $\tilde{A}$ follows a GMM, then the outer join result distribution $(\tilde{A}, \tilde{B})$ follows a multivariate GMM.

Proof sketch: According to (6), we have

$$\tilde{B}|\tilde{A} \sim N(\tilde{A}\boldsymbol{\beta}, \Sigma_E). \tag{7}$$

Given $\tilde{A} \sim$ GMM, there exists a random variable C that captures the mixing distribution of $\tilde{A}$ such that

$$C \sim Multinomial(m, p_1, \cdots, p_m) \tag{8}$$

$$\tilde{A}|(C = c) \sim N(\boldsymbol{\mu}_{\tilde{A},c}, \Sigma_{\tilde{A},c}) \tag{9}$$

Based on (7) and (9), matrix manipulation can show that

$$(\tilde{A}, \tilde{B})|(C = c) \sim N\left((\boldsymbol{\mu}_{\tilde{A},c}, \boldsymbol{\mu}_{\tilde{A},c}\boldsymbol{\beta}), \begin{pmatrix} \Sigma_{\tilde{A},c} & \Sigma_{\tilde{A},c}\boldsymbol{\beta} \\ \boldsymbol{\beta}^T\Sigma_{\tilde{A},c} & \Sigma_E + \boldsymbol{\beta}^T\Sigma_{\tilde{A},c}\boldsymbol{\beta} \end{pmatrix}\right).$$

Together with (8), we have that $(\tilde{A}, \tilde{B})$ follows a multivariate GMM. ∎

The above proof sketch fully characterizes the join result distribution, including all its parameters. In practice, $\boldsymbol{\beta}$ and Σ_E are unknown. They can be estimated using regression over the S (view input) tuples. The least squares estimates of these parameters are:

$$\boldsymbol{\beta} = (\tilde{\mathbf{A}}^T\tilde{\mathbf{A}})^{-1}\tilde{\mathbf{A}}^T\tilde{\mathbf{B}} \tag{10}$$

$$\Sigma_E = \tilde{\mathbf{B}}^T(\mathbf{I}_n - \tilde{\mathbf{A}}(\tilde{\mathbf{A}}^T\tilde{\mathbf{A}})^{-1}\tilde{\mathbf{A}}^T)\tilde{\mathbf{B}}/(n-k), \tag{11}$$

where $\tilde{\mathbf{A}} = (\tilde{A}_1^T, \cdots, \tilde{A}_n^T)^T$ and $\tilde{\mathbf{B}} = (\tilde{B}_1^T, \cdots, \tilde{B}_n^T)^T$.

Theorem V.2 *Given $R = \{A_1^p, \cdots, A_k^p, \bar{R}\}$, $S = \{A_1, \cdots, A_k, B_1^p, \cdots, B_l^p, \bar{S}\}$, let $\tilde{B}^p = (B_1^p, \cdots, B_l^p)$ and assume it follows a GMM with r components, each with mean $\boldsymbol{\mu}_{\tilde{B},j}$ and covariance $\Sigma_{\tilde{B},j}$. Further assume the order-1 linear regression:*

$$\boldsymbol{\mu}_{\tilde{B},j} = \tilde{A}\boldsymbol{\beta}_j^p + \mathbf{E}_j^p, \quad j = 1, \cdots, r \tag{12}$$

where $\boldsymbol{\beta}_j^p$ and $\mathbf{E}_j^p$ are similarly defined as $\boldsymbol{\beta}$ and $\mathbf{E}$, and we denote the covariance matrix of $\mathbf{E}_j^p$ as $\Sigma_{E,j}^p$. Then if $\tilde{A}^p$ follows

a GMM, we conclude that $(\tilde{A}^p, \tilde{B}^p)$ follows a multivariate GMM as well.

Proof sketch: Given $\tilde{B}^p \sim GMM$, there exists a random variable $C_{\tilde{B}}$ that capures the mixing distribution of $\tilde{B}^p$ such that

$$C_{\tilde{B}} \sim Multinomial(r, q_1, \cdots, q_r) \quad (13)$$

$$\tilde{B}^p | (C_{\tilde{B}} = j, \boldsymbol{\mu}_{\tilde{B},j}, \Sigma_{\tilde{B},j}) \sim N(\boldsymbol{\mu}_{\tilde{B},j}, \Sigma_{\tilde{B},j}) \quad (14)$$

According to assumption (12), we have

$$\boldsymbol{\mu}_{\tilde{B},j} | \tilde{A} \sim N(\tilde{A}\boldsymbol{\beta}_j^p, \Sigma_{E,j}^p) \quad (15)$$

Combining (14) and (15), we obtain

$$\tilde{B}^p | (C_{\tilde{B}} = j, \tilde{A}) \sim N(\tilde{A}\boldsymbol{\beta}_j^p, \Sigma_{E,j}^p + \Sigma_{\tilde{B},j}). \quad (16)$$

Given $\tilde{A}^p \sim$ GMM, there exists a random variable $C_{\tilde{A}}$ that captures the mixing distribution of $\tilde{A}^p$ such that

$$C_{\tilde{A}} \sim Multinomial(m, p_1, \cdots, p_m) \quad (17)$$

$$\tilde{A}^p | (C_{\tilde{A}} = i) \sim N(\boldsymbol{\mu}_{\tilde{A},i}, \Sigma_{\tilde{A},i}) \quad (18)$$

Combining (16) and (18), matrix manipulation can show that

$$(\tilde{A}^p, \tilde{B}^p) | (C_{\tilde{A}} = i, C_{\tilde{B}} = j) \sim N\left((\boldsymbol{\mu}_{\tilde{A},i}, \boldsymbol{\mu}_{\tilde{A},i}\boldsymbol{\beta}_j^p), \begin{pmatrix} \Sigma_{\tilde{A},i} & \Sigma_{\tilde{A},i}\boldsymbol{\beta}_j^p \\ \boldsymbol{\beta}_j^{pT}\Sigma_{\tilde{A},i} & \Sigma_{E,j}^p + \Sigma_{\tilde{B},j} + \boldsymbol{\beta}_j^{pT}\Sigma_{\tilde{A},i}\boldsymbol{\beta}_j^p \end{pmatrix} \right).$$

Define $C = (C_{\tilde{A}}, C_{\tilde{B}})$, then based on (13) and (17), we have

$$C \sim Multinomial(mr, p_i q_j, i = 1, \cdots, m, \quad j = 1, \cdots, r).$$

The above two equations show that $(\tilde{A}^p, \tilde{B}^p)$ follows a GMM. ∎

In practice, $\boldsymbol{\beta}_j^p$ and $\Sigma_{E,j}^p$ are unknown. They can be estimated using regression over the tuples in the S input, denoted by S_i=$\{\tilde{A}_i, \tilde{B}_i^p\}$, $i = 1, \cdots, n$. Also, the distribution of $\tilde{B}_i^p$ is given, which is a GMM with mean $\boldsymbol{\mu}_{\tilde{B},j}$ and covariance $\Sigma_{\tilde{B},j}$, $j = 1, \cdots, r$. The least square estimates of these parameters are:

$$\boldsymbol{\beta}_j^p = (\tilde{\mathbf{A}}^T\tilde{\mathbf{A}})^{-1}\tilde{\mathbf{A}}^T\tilde{\boldsymbol{\mu}} \quad (19)$$

$$\Sigma_{E,j}^p = \tilde{\boldsymbol{\mu}}^T(\mathbf{I}_n - \tilde{\mathbf{A}}(\tilde{\mathbf{A}}^T\tilde{\mathbf{A}})^{-1}\tilde{\mathbf{A}}^T)\tilde{\boldsymbol{\mu}}/(n-k), \quad (20)$$

where $\tilde{\mathbf{A}} = (\tilde{A}_1^T, \cdots, \tilde{A}_n^T)^T$ and $\tilde{\boldsymbol{\mu}} = (\boldsymbol{\mu}_{\tilde{B},1}^T, \cdots, \boldsymbol{\mu}_{\tilde{B}_n}^T)^T$.

Evaluation Techniques. The query plan for query Q1 with a left outer join and a probabilistic view is illustrated in Fig. 4. It first applies the query-specified windows to the two inputs: A *Now* window feeds each arriving location tuple as the left input to the join; An *update* window, [Partition By sensor_id Rows k], selects the most recent k temperature tuples from each sensor into the window. The probabilistic view $\mathbf{V}_{temp^p|x,y}$ is maintained over the update window and is the right input to the join. The join extends each location tuple with the attributes presented by the view, and returns a joint distribution.

Given the closed-form result distribution, the main implementation issue is the view construction using regression. While recent research has applied regression to build models and views over sensor data [17], [11], our work differs by exploring the tradeoffs of applying order-1 linear regression at a global versus local scale.

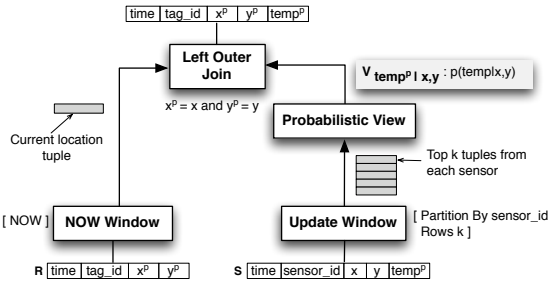


Fig. 4. Query plan for the join using a probabilistic view (Q1).

Global regression applies regression equations (Eq. 10 and Eq. 11) to all S tuples in the current update window to construct the view defined by Eq. 7. The view can be maintained incrementally by updating intermediate matrices for $\boldsymbol{\beta}$ [17], e.g., $\tilde{\mathbf{A}}^T\tilde{\mathbf{A}}$ and $\tilde{\mathbf{A}}^T\tilde{\mathbf{B}}$ in Eq. 10. Then, when an R tuple arrives, the view is refreshed by completing the matrix operations for $\boldsymbol{\beta}$ and Σ_E.

A fundamental limitation of global regression is that the order-1 linear assumption may not hold over the entire view. For query Q1, the temperature may not be a linear function of the location but rather, say, a quadratic function. Hence, global regression may result in severe error when its assumption fails to hold.

Local regression is motivated by the statistical theory that a smooth function can be approximated by a low degree polynomial, e.g., a linear function, in the neighborhood of any point. Hence, we design a local regression technique as follows: Given each R tuple that follows a GMM, use the means and the variances of the components of the GMM to dynamically define a sufficient *local regression region* (LRR). As a simple example, the LRR for an R tuple that follows $N(\mu, \sigma)$ can be $[\mu - m\sigma, \mu + m\sigma]$ with $m \geq 2$. Then, retrieve the subset of S tuples that reside in the LRR and apply regression to these tuples.

A key advantage of this technique is that it does not require the assumption of global linearity, hence allowing more accurate view construction. However, when given a very small set of tuples in the LRR, it may not have enough data points to achieve the accuracy (which is a data problem, not a model problem). When this problem occurs, we can collect more data points by adjusting the LRR appropriately. Computation-wise, regression is applied to a small set of tuples, hence with a low cost.

B. Joins using the Cross-Product

Depending on the application, a join can also be modeled using a cross-product followed by a selection [29]. An example is query Q2 in § II-A: it compares every pair of objects for proximity in location. Our second technique supports such joins with result distributions in GMMs. More specifically, when join attributes in both streams are probabilistic, the result distribution of a cross-product is the product of the two input distributions, i.e., a multivariate GMM in our model. The subsequent selection defines a region over the GMM, and integrating the GMM over this region gives the existence probability

of the result tuple. Here we note that integrating a continuous distribution over the region defined by equality predicates always yields 0. This will cause all join results to be filtered. To retain tuples of potential interest, we rewrite the equality predicates into inequality ones using a small bound, i.e., rewrite $R.x = S.x$ to $|R.x - S.x| < \delta$. With small modification, the above discussion also applies to joins between probabilistic and deterministic attributes based on the cross-product semantics.

A well-known problem with the cross-product is that it can create a large number of intermediate tuples. Hence, it is important to prune results with low existence probabilities. However, computing the existence probability requires an expensive integration operation. In our work, we devise *linear transformation* for GMMs to expedite processing when the predicates take a linear form, e.g., $|R.x - S.x| < \delta$, which is a common case in practice.

Take the join of tuple r from R and s from S in query Q2 as an example. For simplicity, assume $(r.x, r.y) \sim N(\mu_r, \Sigma_r)$, $(s.x, s.y) \sim N(\mu_s, \Sigma_s)$. Then the result of cross-product follows $N(\mu_r, \Sigma_r)N(\mu_s, \Sigma_s)$. Let $f_{\mu,\Sigma}(x)$ denote the *pdf* of a multivariate Gaussian $N(\mu, \Sigma)$, the existence probability of the join result is

$$\iiiint_{|r.x-s.x|<3, |r.y-s.y|<3} f_{\mu_r,\Sigma_r}(r.x, r.y) f_{\mu_s,\Sigma_s}(s.x, s.y)\, dr.x\, dr.y\, ds.x\, ds.y.$$

If we let $z_1 = r.x - s.x$ and $z_2 = r.y - s.y$, then it equals:

$$\iint_{-3<z_1<3, -3<z_2<3} f_{\mu,\Sigma}(z_1, z_2)\, dz_1\, dz_2,$$

where $\mu = \mu_r - \mu_s$ and $\Sigma = \Sigma_r + \Sigma_s$. As is shown, this technique reduces the dimensionality of integration by half. It is straightforward to extend the above example to the GMM, as it is a linear combination of Gaussians.

VI. Performance Evaluation

In this section, we evaluate our techniques for joins and aggregates, and compare them with sampling-based methods ([14] for aggregates and a home-grown method for joins) to demonstrate our performance benefits. We further perform a case study in the real-world application of tornado detection, and show that our techniques can improve the existing system with better detection results and faster computation.

A. Evaluation of Aggregation

We first use synthetic streams with controlled properties to evaluate our techniques for aggregates. Our data generator produces a tuple stream with one continuous random attribute. Each tuple is modeled by a mixture of two Gaussian components. The mean of the first component is uniformly sampled from [0, 5] and the mean of the second is from [5, 50]. This way, we can model a variety of real-world distributions, including bimodal when the two means are far apart, asymmetric when they are somewhat close, and almost Gaussian when they are very close (a much harder workload than using Gaussian distributions only as in [29]). The standard deviation of each Gaussian component is within [0.5, 1] by default. The coefficient of each component is uniform from [0, 1].

We evaluated `avg` over the above tuple stream. The window used is a tumbling (default) or sliding window, and its size is measured by the number of tuples N. in Section IV-C. The default accuracy requirement is $VD \leq 0.1$ for each algorithm. The performance metric is throughput. All experiments were run on a 3Ghz dual-core xeon processor with 1GB memory for use in Java.

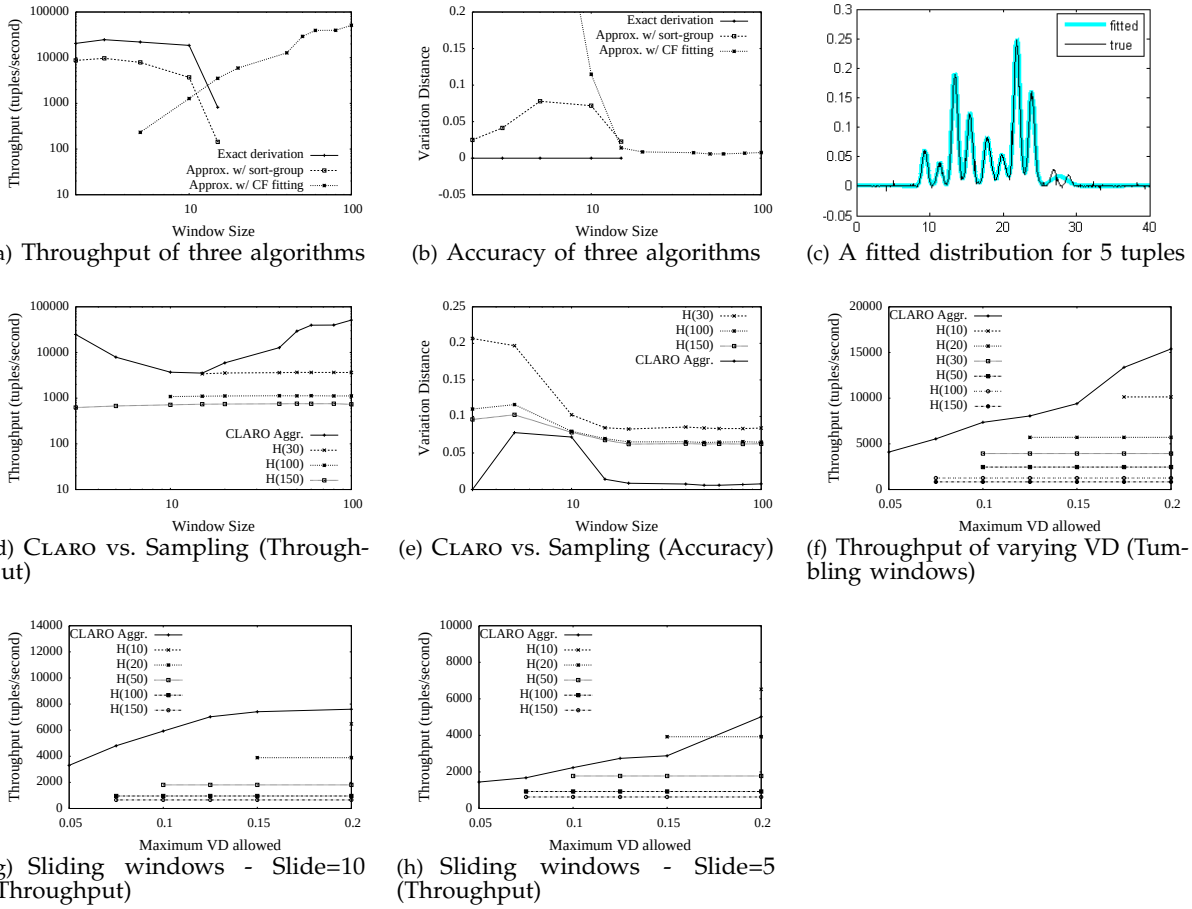
Expt 1: Compare our algorithms. We first compare the three algorithms that we proposed: exact derivation, approximation using sort-group, and approximation using characteristic function (CF) fitting. We varied the number of tuples in each window, N, since it directly affects the result distribution and the computation needed. The throughput and accuracy results are shown in the Fig. 5(a) and Fig. 5(b).

We observe that the throughout using exact derivation is high due to being a closed-form solution. However, we can only apply it to the small values of N because its formulas grow exponentially in N, hence becoming less and less useful. Sort-group is an approximation of the exact derivation, yielding simplified formulas. It achieves high throughput when N is small, e.g., up to 10, and deteriorates quickly after that. This is again because the size of the formula grows exponentially in N. In contrast, CF-fitting works well for large numbers of N, e.g., after 10. This is due to the smoother result distributions in this range, hence easier to fit, and the one-time fitting cost being amortized over more tuples.

For accuracy, most algorithms satisfy the requirement of VD $\leq$ 0.1. This is because both approximation algorithms compare with the true distributions through either direct VD comparison or function fitting with a small residual. We observe that the hardest range for approximation is 5 to 10 tuples. Result distributions in this range are complex and require a mixture of many components to fit. An example of the true and fitted distributions for 5 tuples is shown in Fig. 5(c). From 15 tuples onwards, the result distribution becomes smoother with fewer peaks. The only case of poor accuracy is CF fitting for less than 10 tuples. This is because we limited the maximum number of components it can search for to gain some performance.

We further evaluated the performance of our algorithms under different data characteristics. We ran the above experiment with traces generated with different ranges of means and variances. Specifically, the means of the two components were chosen from the same range or from two separate ranges. The variances were sampled from different ranges, [0.3, 0.5] and [1, 3]. In all cases, we observed the same trends for both accuracy and throughput, and the crossing points among the three algorithms stay the same. We also note that our workload is already hard by involving an asymmetric or bimodal distribution in each tuple. If most distributions are Gaussians instead, sort-group and CF fitting both improve performance but with a similar crossing point.

The above results suggest the configuration for the



(a) Throughput of three algorithms (b) Accuracy of three algorithms (c) A fitted distribution for 5 tuples

(d) CLARO vs. Sampling (Throughput) (e) CLARO vs. Sampling (Accuracy) (f) Throughput of varying VD (Tumbling windows)

(g) Sliding windows - Slide=10 (Throughput) (h) Sliding windows - Slide=5 (Throughput)

Fig. 5. Experimental results for aggregation using our algorithms and histogram-based sampling

hybrid solution. When the number of tuples N is smaller than 5, we use exact derivation. For the range of [5, 10], we use the sort-group algorithm. After that, we switch to CF fitting. In addition, when N is large enough (e.g. > 20), the result distributions are mostly a smooth Gaussian. These distributions can be computed directly using the Central Limit Theorem (CLT). Hence, we can use CLT as an optimization when $N \geq 30$ (e.g., in Expt 3 below).

Expt 2: Compare to histogram-based sampling. Next, we compare our hybrid solution with the histogram based sampling algorithm [14]. Given N tuples, this algorithm (1) generates $k * \mu$ samples for each tuple, (2) performs aggregation over them to get $k * \mu$ result samples, and (3) sorts the result samples and builds a histogram with k buckets and μ samples for each bucket. k and μ are the parameters of this algorithm. Since we found the accuracy of this algorithm to be more sensitive to k, we used three settings in this experiment: k= 30, 100, or 150 while μ is fixed to 50.

Fig. 5(d) and Fig. 5(e) show the results. As observed, our hybrid algorithm outperforms the three settings of the histogram algorithm in both throughput and accuracy. In terms of accuracy, only the histogram with $k = 150$ ensures VD ≤ 0.1. The other two violate this in the "hard" range between 5 and 15 tuples (hence these throughput numbers are removed from Fig. 5(d)). These results show the advantages of our algorithm over the sampling based approach. Our algorithm can adapt to the accuracy requirement while maximizing the throughput. Using sampling, one has to manually choose the parameters to meet the requirement, and the optimal parameter setting varies with workloads.

Expt 3: Vary the VD requirement. To further study our adaptivity to accuracy requirements, we varied VD from 0.05 to 0.2. In the first setting, we used tumbling windows with N randomly chosen from [2, 50] for each stream segment. Fig. 5(f) shows the throughput results (when the VD requirement is satisfied). As observed, our algorithm outperforms the histogram algorithm for all values of VD used. It also shows that we can achieve better throughput under a relaxed condition.

In the second setting, we repeated the experiment using sliding windows. The window size N gradually increases from 5 to 50 in the increment of 5. This way, we can examine different ranges of the hybrid solution. The window slides by 10 tuples or the window size, whichever is smaller. We also made the histogram algorithm incremental by maintaining necessary samples from the previous window. Fig. 5(g) shows the throughput results (when the VD requirement is met). The same observations as for tumbling windows hold. We also note that our algorithm uses less memory than the histogram algorithm

for incremental computation. We further examined the behavior of sliding windows with a small slide size (i.e. 5). As shown in Fig. 5(h), the throughput of our algorithm slightly decreases compared to the larger slide size. This is because some non-incremental components in our algorithm such as function fitting need to be repeated more often. The performance of our algorithm is still comparable with those of the histogram algorithm in this setting.

B. Evaluation of Joins

We next evaluate our join techniques. We focus on the join between R=$\{a^p\}$ and S=$\{a, b\}$ where b's value depends on a. We first design a sampling-based method for the join using a probabilistic view. We use it as a baseline to compare to our regression-based methods.

Histogram-based sampling. For each R tuple, this method takes samples from the distribution of $R.a^p$. It then extends each sample for a with a sample for b (as the join). To do so, the method searches all S tuples in the update window for the closest two points based on the a value. It then applies linear interpolation with added random noise (for later histogram construction) to obtain a b sample. Finally it uses all the (a, b) samples to construct an equi-depth 2-dimensional histogram as an approximate result distribution $f(a^p, b^p)$. The setting of the histogram H(k, μ) depends on the number of buckets per dimension, k, and number of samples per bucket, μ.

In our experiments, the R stream is an object location stream from an RFID inference system [31] where each tuple has a Gaussian distribution (using a mixture distribution will not incur more cost given our closed-form solution). The S stream is produced by our temperature simulator, which generates tuples by adding random noise to the underlying function between temperature and location. This function can be linear or quadratic in this study. The query-specified *update window size* (UWS) on S is 1, i.e., containing the most recent temperature reading from each sensor. R and S tuples arrive at the same rate. Throughput measures the number of R tuples pipelined through the left outer join.

Expt 4: Sampling versus regression. We first compare the sampling method with our regression methods (global and local). We use a linear function to generate the temperature stream. The local regression region (LRR) is set to be 18. As seen in Fig. 6(a), the sampling method produces results far from the true result distributions (VD $>$ 0.7) while regression methods are much more accurate (VD $<$ 0.1). The VD of sampling improves as μ increases, e.g., from H(10,10) to H(10,50), because it uses more samples to construct the histogram. However, the VD worsens when k increases, e.g., from H(10,50) to H(30,50). This is because when k is too large, the area of each bucket is very small, and the samples in each bucket mostly fit the noise added during interpolation. While it is possible to keep increasing μ, Fig. 6(b) shows that the sampling method is already very slow: the throughput of H(10,10) is 37 tuples/sec and that of H(10,50) is 4. On the other hand, our global regression gains a throughput of 1547 and local regression gains 44843, outperforming sampling by 2-4 orders of magnitude.

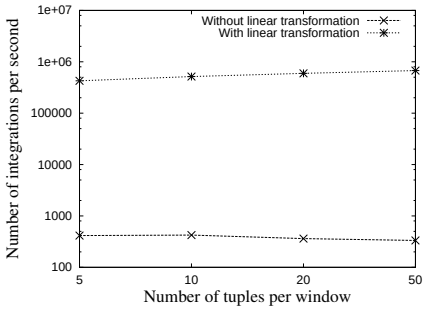


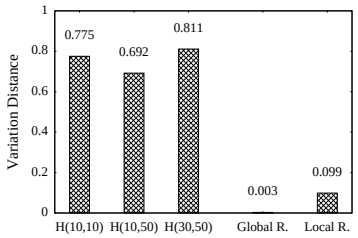
Fig. 8. Joins using cross-product with and without linear transformation

Expt 5: Global versus local regression. We next use a quadratic function to generate the temperature stream and compare global versus local regression. Since local regression is sensitive to the number of data points available, we vary its LRR (which has no effect on global regression). As Fig. 6(c) shows, global regression has poor accuracy since its assumption of global linearity does not hold. The VD of local regression (UWS=1) first decreases because the increased region has more points for regression. Then it increases because the region is too large to meet the local linearity assumption—the local regression is becoming more like global regression. A further optimization for local regression is to enlarge UWS, e.g., using the most recent 5 readings from each sensor. The rationale behind this is that the underlying function usually changes slowly. Hence using old tuples from the past few seconds will not add stale information. Fig. 6(c) shows such improved VD with UWS=5 and 10.

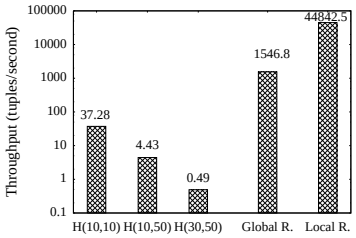
Fig. 6(d) shows that increasing the LRR reduces the throughput as the regression uses more points. Despite that, local regression outperforms global regression by a wide margin. In practice, if we choose a reasonable setting, e.g., LRR=6 and UWS=5, local regression can gain both high accuracy and efficiency.

To further understand the effect of UWS on local regressiong, we then fix the LRR as 6 and use the method of enlarging the UWS, from 1 to 20, to feed more data points to local regression (while global regression still uses UWS=1). Fig. 6(e) and Fig. 6(f) show that local regression outperforms global regression in both accuracy and speed. The increase of the UWS improves the VD of local regression significantly due to the use of more points. Its throughput decreases for the same reason. However, even when local regression uses a window 20 times larger, its throughput is still much better than global regression due to the benefit of local computation.

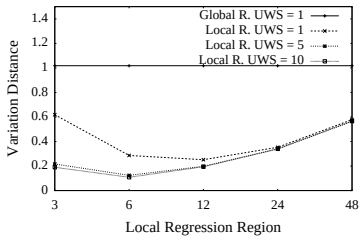
Expt 6: Joins using cross-product Next we evaluate joins between R=$\{a^p\}$ and S=$\{a^p\}$ using cross-product semantics with the computation of tuple existence probability. We compare such computation for join results with and without linear transformation. Both R and S are object location streams used above. Each location tuple has a Gaussian distribution. The R and S tuples arrive at the same rate. For each input stream, there is a tumbling



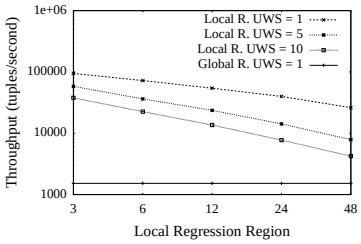
(a) Sampling vs regression in VD (linear function).



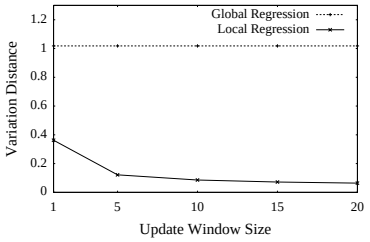
(b) Sampling vs regression in Throughput (linear function).



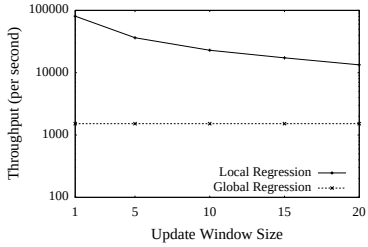
(c) Global vs local regression in VD (quadratic function)



(d) Global vs local regression in Throughput (quadratic function)

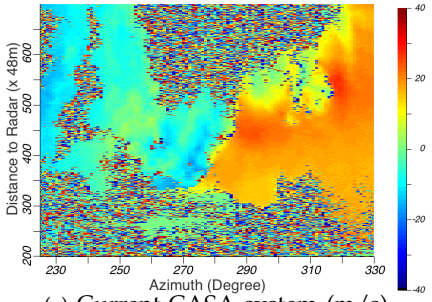


(e) Global vs local regression in VD, with varying update window (quadratic function)

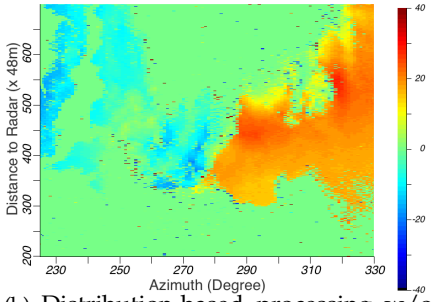


(f) Global vs local regression in Throughput, with varying update window (quadratic function)

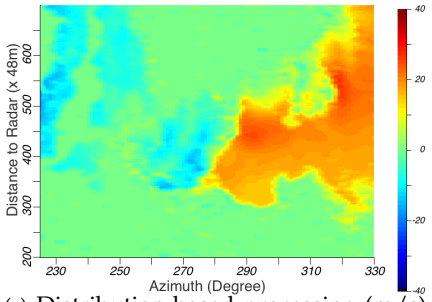
Fig. 6. Results of joins using sampling $H(k, \mu)$, global and local regression (Global/Local R.).



(a) Current CASA system (m/s).



(b) Distribution-based processing w/o Smoothing Step (m/s).



(c) Distribution-based processing (m/s).

Fig. 7. Radial velocity of a true tornadic region.

window and we vary the number of tuples per window. Fig. 8 shows that by applying linear transformation to reduce the dimensionality of integrations, we get a 3-order-of-magnitude improvement.

C. Case Study: Tornado Detection

We now demonstrate the effectiveness of capturing uncertainty using distributions in a real-world tornado detection system [23]. We first modified the velocity analysis module in Fig. 1 to generate velocity distributions in GMM. In the FFT module, the current system takes a weighted average from the discrete FFT distribution f_F. Instead, we apply a model-based analysis: **(1) Strength Filter**: If the radar signal strength is below a threshold, we output zero and skip the following steps. **(2) Gaussian Estimation**: Create a Gaussian distribution from the mean and variance of f_F. Return it for output if it passes the goodness test based on the domain knowledge. **(3) GMM Fitting**: Otherwise, fit a mixture of two Gaussians from f_F and remove any component with a large variance as noise. After the FFT module, we apply **Smoothing** by averaging distributions of high and low frequency segments and across neighboring regions. For `avg` over GMMs, we apply the techniques in Section IV-B to compute the result distribution. Since the current tornado detection algorithm does not take distributions as input, we feed the mean of each result distribution to the algorithm.

TABLE I
PROCESSING RESULT OF A REAL TONADIC DATASET OF 947S, 84 SCANS.

	Analysis Time	Detection Time	False Positives
CASA	197.76 s	4486 s	2137
Distribution	578.7 s	392 s	9

Our case study used a real tornadic dataset collected in Oklahoma on May 8, 2007, containing raw data of 84 radar scans in 947 seconds. As true velocity changes gradually in space and the tornado detection algorithm expects smooth input, we first examine the spatial smoothness of velocity. Fig. 7(a) shows the velocity map of a sector scan of a real tornadic region generated by current CASA system. For comparison, we show the mean of velocity distributions of the same sector scan after processed by Strength Filter, Gaussian Estimation and GMM Fitting in Fig. 7(b). We also give the final output velocity map in Fig. 7(c). As shown, our techniques yield much smoother velocity maps. Specifically, the Strength Filter removes many colorful dots (i.e., noise); the GMM Fitting smoothes by further removing noise (Fig. 2(b) shows a detailed

example of the noise removal effect of GMM Fitting from Section III. The velocity calculated in CASA is 5.6m/s. After dropping the Gaussian component of noise, the mean becomes 6.4m/s, which is closer to the true value visually observed); the Smoothing step smoothes the remaining sharp velocity changes.

We further measure the analysis speed, detection speed and result quality. As shown in Table I, our distribution-based method reduces the detection time from 4486s to 392s, and produces much fewer false positives (which mainly come from noise). Although our method costs three times the data analysis time, it is still faster than the radar sensing speed. As such, our distribution-based method improves the detection algorithm to be stream-speed, and significantly improves the detection quality.

VII. Related Work

Previous sections have discussed closely related work. Below, we survey several broader areas.

Probabilistic stream processing has gained research attention very recently. Existing work [6], [18], [21] adopts the finite and discrete tuple model as in probabilistic databases. As stated previously, many techniques for discrete variables cannot be applied for problems with continuous variables. Furthermore, existing work such as [6] produces mean, variance, and higher moments to describe result distributions, which cannot be easily used to compute the distributions of subsequent operators.

Models and views of sensor data. Recent work on sensor networks [10], [9] builds statistical models to capture correlations among attributes. Given a query, such models enable reduced costs of data acquisition and communication. FunctionDB [30] transforms discrete sensor observations into continuous functions and supports querying over these functions. More relevant to us is supporting views over uncertain data [11], [20]. However, the throughput of stream queries on the views can be limited, e.g., 50 tuples/sec, due to high inference cost.

Wavelets-based methods [7], [15], [22] build a single summary (distribution) over a relational table or a data stream, and compute aggregates over the summary. In contrast, each tuple includes a distribution in our work and hence aggregation of tuples requires the use of integrals (or their approximations) of the tuple distributions.

VIII. Conclusions

In this paper, we presented the Claro system for uncertain data stream processing, in particular, its unique data model and advanced techniques for aggregates and joins under the model. Our results show that Claro outperforms sampling-based methods in accuracy and speed in stream processing. A case study further reveals that Claro can improve a real tornado detection system with better quality of results and stream-speed processing.

The work presented in this paper provides a foundation for us to explore more advanced issues.

Relational algebra. Our first task is to support a larger set of relational operators. Projections in our system are marginalization of multivariate Gaussian mixtures, which is very fast. Selections add conditions to tuple distributions and can cause tuple existence probabilities to be less than 1, which may complicate the evaluation of some other operators. This leads to interesting optimizations such as using commutativity of operators and approximation of distributions, so that we can apply current fast techniques early while delaying computation such as Monte Carlo simulation later in a query plan.

Tuple correlations. We will also address inter-tuple correlations and investigate the strengths and limitations of advanced techniques such as lineage in stream processing.

A hybrid system. A third direction is to explore the combination of histograms and Gaussian mixture models to support both discrete and continuous random variables and to improve overall performance.

References

[1] L. Antova, T. Jansen et al. Fast and simple relational processing of uncertain data. In *ICDE*, 983–992, 2008.
[2] O. Benjelloun, A. D. Sarma et al. Uldbs: Databases with uncertainty and lineage. In *VLDB*, 953–964, 2006.
[3] D. Carney, U. Çetintemel et al. Monitoring streams: a new class of data management applications. In *VLDB*, 215–226, 2002.
[4] G. Casella and R. L. Berger. *Statistical Inference*. Duxbury, 2001.
[5] R. Cheng, D. V. Kalashnikov et al. Evaluating probabilistic queries over imprecise data. In *SIGMOD*, 551–562, 2003.
[6] G. Cormode and M. Garofalakis. Sketching probabilistic data streams. In *SIGMOD*, 281–292, 2007.
[7] G. Cormode, M. Garofalakis. Histograms and Wavelets on Probabilistic Data. In *ICDE*, 2009.
[8] N. N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. *VLDB J.*, 16(4):523–544, 2007.
[9] A. Deshpande, C. Guestrin et al. Exploiting correlated attributes in acquisitional query processing. In *ICDE*, 2005.
[10] A. Deshpande, C. Guestrin et al. Model-driven data acquisition in sensor networks. In *VLDB*, 588–599, 2004.
[11] A. Deshpande et al. MauveDB: supporting model-based user views in database systems. In *SIGMOD*, 2006.
[12] Y. Diao, B. Li et al. Capturing data uncertainty in high-volume stream processing. In *CIDR*, 2009.
[13] S. Frühwirth-Schnatter. *Finite Mixture and Markov Switching Models*. Springer, 2006.
[14] T. Ge and S. B. Zdonik. Handling uncertain data in array database systems. In *ICDE*, 1140–1149, 2008.
[15] A. Gilbert, Y. Kotidis et al. Surfing wavelets on streams: one-pass summaries for approximate aggregate queries. In *VLDB*, 2001.
[16] L. Girod, Y. Mei et al. Xstream: a signal-oriented data stream management system. In *ICDE*, 1180–1189, 2008.
[17] C. Guestrin, P. Bodi et al. Distributed regression: an efficient framework for modeling sensor network data. In *IPSN*, 2004.
[18] T. S. Jayram, A. McGregor et al. Estimating statistical aggregates on probabilistic data streams. In *PODS*, 2007.
[19] S. Jeffery, M. J. Franklin et al. An adaptive RFID middleware for supporting metaphysical data independence. *VLDB J.*, 17(2), 2008.
[20] B. Kanagal and A. Deshpande. Online filtering, smoothing and probabilistic modeling of streaming data. In *ICDE*, 2008.
[21] B. Kanagal and A. Deshpande. Efficient query evaluation over temporally correlated probabilistic streams. In *ICDE*, 2009.
[22] P. Karras, N. Mamoulis. One-pass wavelet synopses for maximum-error metrics. In *VLDB*, 2004.
[23] J. F. Kurose, E. Lyons et al. An end-user-responsive sensor network architecture for hazardous weather detection, prediction and response. In *AINTEC*, 1–15, 2006.
[24] G. McLachlan et al. *Finite Mixture Models*. Wiley-Interscience, 2000.
[25] C. Ré, J. Letchner et al. Event queries on correlated probabilistic streams. In *SIGMOD*, 715–728, 2008.
[26] C. Ré and D. Suciu. Approximate lineage for probabilistic databases. In *VLDB*, 2008.
[27] T. Sauer. *Numerical Analysis*. Addison Wesley, 2005.
[28] P. Sen, A. Deshpande et al. Exploiting shared correlations in probabilistic databases. In *VLDB*, 2008.

[29] S. Singh, C. Mayfield et al. Database support for probabilistic attributes and tuples. In *ICDE*, 1053–1061, 2008.
[30] A. Thiagarajan and S. Madden. Querying continuous functions in a database system. In *SIGMOD*, 791–804, 2008.
[31] T. Tran, C. Sutton et al. Probabilistic inference over rfid streams in mobile environments. In *ICDE*, 2009.
[32] D. Wang, E. Michelakis et al. Bayesstore: Managing large, uncertain data repositories with probabilistic graphical models. *VLDB*, 2008.
[33] N. Ye. *The Handbook of Data Mining*. Lawrence Earlbaum Associates, 2003.

Appendix

Proof of Theorem IV.1.

We first consider the sum of two variables, $S = X_1 + X_2$, X_1 and X_2 be mixtures of m_1 and m_2 components. That is:

$$\begin{aligned} f_1(x) &= p_{11}N(\mu_{11},\sigma_{11}) + ... + p_{1m_1}N(\mu_{1m_1},\sigma_{1m_1}) \\ f_2(x) &= p_{21}N(\mu_{21},\sigma_{21}) + ... + p_{2m_2}N(\mu_{2m_2},\sigma_{2m_2}) \end{aligned}$$

The pdf of the sum S can be written as:

$$\begin{aligned} f_S(s) &= \int_{x_1}\int_{x_2:(x_1+x_2=s)} f_1(x_1)f_2(x_2)dx_2dx_1 \\ &= \int_{-\infty}^{+\infty} f_1(x)f_2(s-x)dx \\ f_1(x)f_2(s-x) &= \left[p_{11}N(\mu_{11},\sigma_{11}) + ... + p_{1m_1}N(\mu_{1m_1},\sigma_{1m_1})\right] \\ &\quad \left[p_{21}N(\mu_{21},\sigma_{21}) + ... + p_{2m_2}N(\mu_{2m_2},\sigma_{2m_2})\right] \\ &= \sum_{i=1,j=1}^{m_1,m_2} p_{1i}p_{2j}\frac{1}{2\pi\sigma_{1i}\sigma_{2j}}e^{-\frac{1}{2}\left(\frac{(x-\mu_{1i})^2}{\sigma_{11}^2}+\frac{(x-\mu_{2j})^2}{\sigma_{2j}^2}\right)} \end{aligned}$$

Now consider the integral of one term of the sum:

$$A = \int_{-\infty}^{+\infty} p_{1i}p_{2j}\frac{1}{2\pi\sigma_{1i}\sigma_{2j}}e^{-\frac{1}{2}\left(\frac{(x-\mu_{1i})^2}{\sigma_{1i}^2}+\frac{(x-\mu_{2j})^2}{\sigma_{2j}^2}\right)}$$

Let B be the term in the exponent:

$$\begin{aligned} B &= \frac{(x-\mu_{1i})^2}{\sigma_{1i}^2} + \frac{(x-\mu_{2j})^2}{\sigma_{2j}^2} \\ &= \frac{1}{\sigma_{11}^2\sigma_{21}^2}\left[(\sigma_{1i}^2+\sigma_{2j}^2)x^2 - 2\sigma_{2j}^2x\mu_{1i} + 2\sigma_{1i}^2x(\mu_{2j}-s) + \sigma_{2j}^2\mu_{1i}^2 + \sigma_{1i}^2(\mu_{2j}-s)^2\right] \\ &= \frac{\sigma_{1i}^2+\sigma_{2j}^2}{\sigma_{11}^2\sigma_{21}^2}\left[\left(x - \frac{\sigma_{2j}^2\mu_{1i}+\sigma_{1i}^2(s-\mu_{2j})}{\sigma_{1i}^2+\sigma_{2j}^2}\right)^2 - \left(\frac{\sigma_{2j}^2\mu_{1i}+\sigma_{1i}^2(s-\mu_{2j})}{\sigma_{1i}^2+\sigma_{2j}^2}\right)^2 + \frac{\sigma_{2j}^2\mu_{1i}^2+\sigma_{1i}^2(s-\mu_{2j})^2}{\sigma_{1i}^2+\sigma_{2j}^2}\right] \\ &= \frac{\sigma_{1i}^2+\sigma_{2j}^2}{\sigma_{1i}^2\sigma_{2j}^2}\left[\left(x - \frac{\sigma_{2j}^2\mu_{1i}+\sigma_{1i}^2(s-\mu_{2j})}{\sigma_{1i}^2+\sigma_{2j}^2}\right)^2 + \frac{\sigma_{1i}^2+\sigma_{2j}^2(s-\mu_{1i}-\mu_{2j})^2}{(\sigma_{1i}^2\sigma_{2j}^2)^2}\right] \end{aligned}$$

Substitute into A, we have:

$$\begin{aligned} A &= \int_{-\infty}^{+\infty} p_{1i}p_{2j}\frac{1}{2\pi\sigma_{1i}\sigma_{2j}}e^{\frac{1}{2}\frac{\sigma_{1i}^2+\sigma_{2j}^2}{\sigma_{1i}^2\sigma_{2j}^2}\left(x-\frac{\sigma_{2j}^2\mu_{1i}+\sigma_{1i}^2(s-\mu_{2j})}{\sigma_{1i}^2+\sigma_{2j}^2}\right)^2}e^{\frac{(s-\mu_{1i}-\mu_{2j})^2}{\sigma_{1i}^2+\sigma_{2j}^2}}dx \\ &= \frac{p_{1i}p_{2j}}{\sqrt{2\pi}\sqrt{\sigma_{1i}^2+\sigma_{2j}^2}}e^{\frac{(s-\mu_{1i}-\mu_{2j})^2}{\sigma_{1i}^2+\sigma_{2j}^2}}\int_{-\infty}^{+\infty}\frac{1}{\sqrt{2\pi}\frac{\sigma_{1i}\sigma_{2j}}{\sqrt{\sigma_{1i}^2+\sigma_{2j}^2}}}e^{\frac{1}{2}\frac{\sigma_{1i}^2+\sigma_{2j}^2}{\sigma_{1i}^2\sigma_{2j}^2}\left(x-\frac{\sigma_{2j}^2\mu_{1i}+\sigma_{1i}^2(s-\mu_{2j})}{\sigma_{1i}^2+\sigma_{2j}^2}\right)^2}dx \end{aligned}$$

The integral is equal to 1 since it is the integral of the pdf of a Gaussian distribution. Hence:

$$A = \frac{p_{1i}p_{2j}}{\sqrt{2\pi}\sqrt{\sigma_{1i}^2+\sigma_{2j}^2}}e^{\frac{(s-\mu_{1i}-\mu_{2j})^2}{\sigma_{1i}^2+\sigma_{2j}^2}}$$

This is one component of a Gaussian mixture with mean $(\mu_{1i}+\mu_{2j})$, variance $(\sigma_{1i}^2+\sigma_{2j}^2)$ and coefficient $p_{1i}p_{2j}$. Therefore, the theorem is proved for the case of $N = 2$.

The theorem can also be proved by manipulating the inverse transformation formula of the characteristic function of the sum S.

The generalization to an arbitrary N is straightforward since we can do the sum for two distributions at a time by getting the result of previous sum and summing it with the next distribution.