PODS: A New Model and Processing Algorithms for Uncertain Data Streams *

Thanh T. L. Tran, Liping Peng, Boduo Li, Yanlei Diao, Anna Liu†
Department of Computer Science †Department of Mathematics and Statistics
University of Massachusetts, Amherst
{ttran, lppeng,boduo,yanlei}@cs.umass.edu †anna@math.umass.edu

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 PODS system that supports stream processing for uncertain data naturally captured using *continuous random variables*. PODS 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 significantly outperform a state-of-the-art sampling method. 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.

Categories and Subject Descriptors

H.2 [**Database Management**]: Systems

General Terms

Algorithms, Design, Performance, Theory

1. 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 [9, 14], radio frequency identification (RFID) networks [17, 29], GPS systems [18], and weather radar networks [20]. 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 [29]. 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 detect, for instance, 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 [20], 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 through the entire stream system, making tornado detection results error-prone. Given the potential social impact of such a system, it is absolutely vital that the system capture the quality of its 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*, allow the visualization of data uncertainty in any step of the processing. They also allow the stream system to feed the tuples output from one operator as input to another and characterize the results of further processing—evidently, having only statistics such as mean and variance for the tuples output from the previous operator 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 to be designed if an integral-based solution does not exist. Second, such computation must be performed 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 stringent performance requirements.

Despite a flurry of recent work on uncertain data management, the two challenges stated above have not been adequately addressed. Most probabilistic databases [2, 3, 6, 24, 30] and stream systems [7, 16, 19] 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.

The state-of-the-art techniques for continuous random variables

*This work has been supported in part by the National Science Foundation under the grants IIS-0746939, IIS-0812347, and EEC-0313747, and by the grant NSA H98230-09-1-0044.

SIGMOD'10, June 6–11, 2010, Indianapolis, Indiana, USA.

employ either multivariate integrals or Monte Carlo simulation. The integral-based approach to aggregation [6] performs n-1 integrals to compute the sum of n tuples. While the result is exact, its computation is too slow for stream processing, as we shall show later in this paper. The Monte Carlo approach [12, 15, 25] samples from the input tuple distributions and computes the result tuple distribution from the samples. For real-world data, however, it is difficult, sometimes impossible, to know right the number of samples needed to guarantee both accuracy and efficiency for complex relational operations, as we also show in our performance study.

Our work presented in this paper originates from our belief that for a significant fragment of relational algebra, faster and more accurate algorithms are possible. Then these algorithms can be used to improve existing Monte Carlo systems when queries involve the set of common relational operations that these algorithms support.

Scope and contributions. In this paper, we present a PrObabilistic Data Stream system, which we call PODS, that supports relational processing of uncertain data streams modeled using continuous random variables. The architectural design of PODS, as described in [11], is to extend the box-arrow paradigm for stream processing [4] 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*. [1] These operators are crucial to our target applications but have not been sufficiently addressed. Further in the streaming context, our goal is to perform such complex operations at high speed, e.g., thousands of tuples per second or higher. More specifically, our contributions include:

Data model. The foundation of PODS is a unique data model based on Gaussian Mixture distributions. This model is highly flexible as it subsumes Gaussian distributions and can model arbitrary real-world distributions [21]. It further allows efficient computation by exploiting Gaussian properties and powerful statistical methods for continuous variables. Most importantly, this model allows a *closed-form* solution for a range of relational operations, among which this paper focuses on aggregates and joins, and the result distributions of these operations still obey Gaussian Mixture distributions. Our model stands in contrast to those based on histograms [12] and weighted particles [18], which indicate the use of samples in computation.

Aggregates. Our data model empowers us to design novel algorithms for aggregates, such as `sum` and `avg`, that are grounded in statistical theory. Our first algorithm obtains exact result distributions of aggregates while completely eliminating the use of integrals (in contrast to using multiple integrals in [6]). However, the formulas for result distributions produced by the exact algorithm 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 accuracy requirements while achieving high speed in stream processing.

Joins. Our data model also enables efficient, accurate evaluation techniques for joins. We propose two types of joins to suit different application semantics. The first type models equi-joins on continuous-valued uncertain attributes as a join of an input stream and a probabilistic view. PODS supports such joins with efficient regression techniques to construct the view and given the view, a closed-form solution in Gaussian Mixture Models to represent result distributions. The second (traditional) type of join pairs tuples from two inputs for inequality comparison and is modeled by a cross-product followed by a selection. PODS supports such joins with exact result distributions and methods for pruning tuples with low existence probabilities.

Evaluation. We perform a thorough evaluation of our techniques for joins and aggregates, and compare them with sampling-based methods ([12] 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 of tornado detection by feeding a trace collected from a real tornadic event into the PODS system. Our results show that fitting the data to the PODS model and using its processing techniques to characterize uncertainty enables the tornado detection algorithm (for the first time) to produce detection results at stream speed with much improved quality.

2. MOTIVATING APPLICATIONS

In this section, we present two motivating applications.

2.1 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 [17, 29] 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. Recent work such as [29] 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. Query 1 shows 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, $Temp$). 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 were precise.

```
Q1:  Select Rstream(R.Tag_id, R.X, R.Y, T.Temp)
     From   FlammableObject [Now] As R,
            Temperature [Partition By sensor_id
                         Rows 1] As T
     Where  T.Temp > 60 °C and
            R.X = T.X and R.Y = T.Y
```

Query 2 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
```

2.2 Hazardous Weather Monitoring

The CASA Research Center is developing weather radar networks to detect hazardous weather events such as tornados and storms [20]. 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 [20].

A CASA radar node rotates to scan. It sends around 2000 pulses per second, alternating between 54 high frequency pulses and 40

[1] We note that our model and algorithms can be generally applied to both probabilistic databases and data stream systems where uncertain attributes are modeled by continuous random variables.

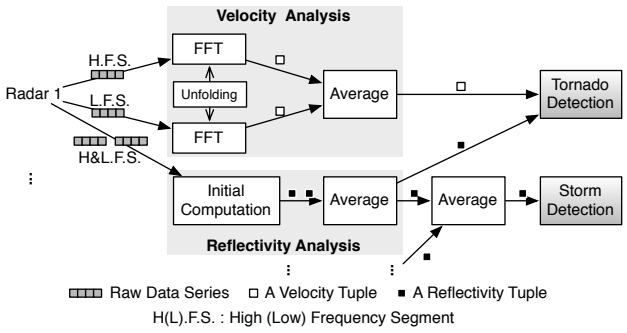


Figure 1: Simplified stream processing in the CASA radar system

low frequency ones. The raw data stream is partitioned into high and low frequency stream segments accordingly, and further partitioned based on the distance to the radar. Overall, the raw data is generated at 175 Mb 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 data is fed into a stream processing system for real-time weather event detection. The current CASA system addresses both data volume and noise issues by means of taking the 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 a Fast Fourier Transform (FFT) to each stream segment for signal processing. It then outputs a single velocity value for each stream segment and averages the values for adjacent high and low frequency stream 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 §6.3), 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 ($Time$, $Azimuth$, $Distance$, $Velocity^p$ or $Reflectivity^p$). We also study the transformation of these distributions through CASA operations, e.g., the frequently used aggregation operations. By doing so, we expect to gain better tornado detection results yet with lower running time.

3. THE PODS DATA MODEL

The foundation of the PODS system is a data model based on Gaussian Mixture distributions that can capture a variety of uncertainties for continuous-valued attributes and further allow fast relational processing. We describe the PODS data model in this section.

3.1 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:*

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

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

where $0 \le p_i \le 1$, $\sum_{i=1}^{m} p_i = 1$, and each mixture component is a Gaussian distribution with mean μ_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:*

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

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

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

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

Second, theoretical results have shown that GMMs can approximate any continuous distribution arbitrarily well [21]. Hence, they are suitable for modeling complex real-world distributions. In the tornado detection application, a detected bimodal distribution of velocity at the boundary between a positive velocity area and a negative velocity area is shown in Fig. 2(a). In contrast, Fig. 2(b) shows a velocity distribution in a positive velocity area, where one Gaussian component captures the high concentration of velocity and the other captures the noise widely spread across the entire spectrum. In the RFID application, Fig. 2(c) shows the inferred location distribution of a recently moved object [29]. 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.

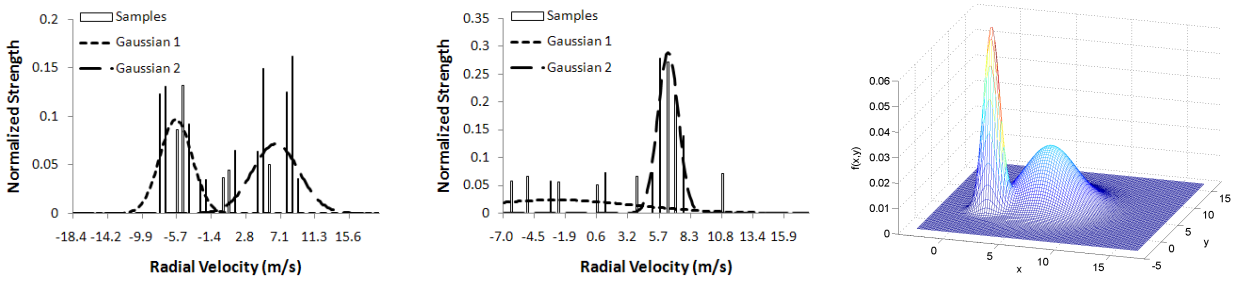
The third benefit of GMMs is efficient computation based on Gaussian properties and advanced statistical theory. First, the mean and variance of GMMs can be computed directly from those of the mixture components:

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

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

Furthermore, the cumulative distribution function (cdf) of a GMM with a single variable has an analytic expression based on a known error function. Values of the error function are precomputed in any numerical library. Hence, computing $\int_a^b f_X(x)dx = F_X(b)\text{-}F_X(a)$ using the cdf incurs little cost. Other computational benefits of GMMs, such as the characteristic functions, product distributions, and linear transformation, are described in the later relevant sections.

Gaussian Mixture Models can be generated from real-world data in a variety of ways. Recent studies [18, 29] have employed *graphical models* to infer distributions from noisy raw data. Since these distributions are often represented using weighted samples, GMMs can be generated from these samples using standard density estimation or function fitting methods. *Time series* techniques can also be used to generate GMMs from temporally correlated input data . In our case study of tornado detection, a Fast Fourier Transform (FFT) is used to translate 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, as will be described in §6.3. (See [11] for a discussion of other techniques that can be employed to generate GMMs.)



(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

Figure 2: Gaussian Mixture Models for real-world data collected from our target applications

3.2 PODS Data Model

We now present the complete PODS data model for relational processing. An uncertain data stream is an infinite sequence of tuples that conform to the schema $\mathbf{A}^d \cup \mathbf{A}^p$. The attributes in $\mathbf{A}^d$ are deterministic attributes, such as the tuple id and the fixed x-y location of a sensor. The attributes in $\mathbf{A}^p$ are continuous-valued uncertain attributes, such as the temperature of a location and the wind velocity in an area. In each tuple, the attributes in $\mathbf{A}^p$ are modeled by a vector of *continuous random variables* $\mathbf{X}$. If the schema further has that the attributes in $\mathbf{A}^p$ can be partitioned into independent attributes, A_i^p, and groups of correlated attributes, $\mathbf{A}_j^p$, we can model A_i^p in a tuple using a Gaussian Mixture distribution, denoted by $f_i(x_i)$, and model $\mathbf{A}_j^p$ in the tuple using a multivariate Gaussian Mixture distribution, denoted by $f_j(\mathbf{x}_j)$. Then the *tuple distribution* can be written as:

$$f_{\mathbf{X}}(\mathbf{x}) = \prod_i f_i(x_i) \prod_j f_j(\mathbf{x}_j),$$

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 [3]. Our current model does not include such correlations for two reasons: First, while raw data is often temporally correlated, the methods that our system employs to transform raw data to tuples with distributions, such as graphical models and FFT, 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 tornado detection system ignores spatial correlations in any data processing before the final tornado detection, and existing probabilistic stream systems such as [18] ignore inter-tuple correlations, all for performance reasons. A thorough treatment of tuple correlations in stream processing is a focus of our future work.

4. AGGREGATION OF UNCERTAIN TUPLES

We first address aggregation of uncertain tuples under the PODS data model. In this work, we focus on `sum` and `avg` because they are crucial to our target applications but have not been sufficiently addressed for continuous random variables in the literature.[2]

Aggregates such as `sum` and `avg` are known to be hard problems in probabilistic databases that employ discrete random variables to model data uncertainty and the possible worlds semantics (PWS) for query processing [8]. For instance, computing the distribution of the average of n discrete random variables may require the enumeration of an exponential number of possible worlds, as depicted in Fig. 3(a). A recent technique that employs Markov sequences to compute `sum` and `avg` [19] has a time complexity $O(nD^3)$ where D is the size of the domain of each random variable, which becomes inefficient for large domains. Aggregates of continuous random variables have related complexities but are naturally modeled by multivariate integrals. Fig. 3(b) illustrates an average of four continuous random variables, $Y = \frac{1}{4}(X_1 + \cdots + X_4)$. The probability that Y is in the range of U is defined by the multivariate integral in the figure. This definition when applied to the discrete setting is consistent with PWS: in each possible world, we take a value from X_1, a value from X_2, and so on, and then add up the probabilities of those possible worlds where the average is in the range of U.

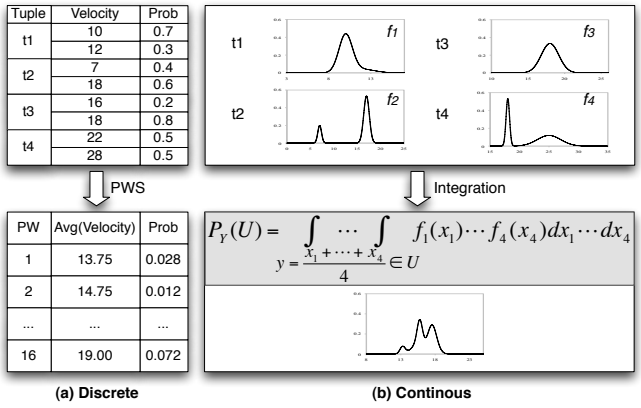


Figure 3: Aggregation in the discrete setting (using PWS) and in continuous setting (using integration).

Given that multivariate integration is a prohibitively expensive operation, the state of the art *integral-based approach* [6] integrates two continuous random variables at a time, resulting in the use of n-1 integrals to aggregate n variables. As we shall show in §4.1, the performance of this technique is not suitable for stream processing.

The *sampling-based approach* [12, 25] 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 provide knowledge

[2] Our technique for `min` and `max` is similar to that in [6], hence omitted in this paper.

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 efficiency.

4.1 A Basic Algorithm

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, [5]):

$$\Phi_U(t) = E[e^{iUt}], \tag{3}$$

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:

$$f_U(x) = \frac{1}{2\pi}\int_{-\infty}^{+\infty} e^{-itx}\Phi_U(t)dt. \tag{4}$$

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:

$$\begin{aligned}\Phi_U(t) &= Ee^{iUt} = Ee^{i(X_1+X_2+...+X_n)t} \\ &= \Phi_{X_1(t)}\Phi_{X_2(t)}...\Phi_{X_n(t)}\end{aligned} \tag{5}$$

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 and take the product of these functions according to Eq. 5. (2) For a given value x, apply the inverse transformation at x to yield $f_U(x)$ according to Eq. 4. In particular, we call the inverse transformation in the second step a *parameterized integral* because it takes an argument x.

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:

$$\Phi_X(t) = p_1e^{i\mu_1 t-\frac{1}{2}\sigma_1^2t^2} + p_2e^{i\mu_2 t-\frac{1}{2}\sigma_2^2t^2}.$$

Thus, Step 1 of the above algorithm does not involve any integration. The only integral required is the one for inverse transformation in Step 2. This analysis holds for all common distributions whose characteristic functions are known. This gives a boost in performance compared to the two-variable convolution method, which requires n-1 parameterized integrals [6].

The main drawback 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 [23] to compute integrals. The task is to average over 10 tuples and compute the pdf values for 20 points. Even with manual optimizations, 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.

4.2 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 4.1 *Let each $X_i, (i = 1..n)$ be a mixture of i_m components identified by the parameters $(p_{i_j}, \mu_{i_j}, \sigma_{i_j}), (j = 1..i_m)$. The result distribution for $U = \sum_{i=1}^{n} X_i$ is a Gaussian 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):*

$$p_k = \prod_{i=1}^{n} p_{i_j};\ \mu_k = \sum_{i=1}^{n} \mu_{i_j};\ \sigma_k = \sqrt{\sum_{i=1}^{n} \sigma_{i_j}^2}. \tag{6}$$

The theorem can be proved by mathematical manipulation of the inverse transformation formula. Due to space constraints, proofs of all the theorems presented in this paper are omitted. The result subsumes the well-known linear property of Gaussian distributions. However, in the context of GMMs, we are not aware of any state-of-the-art books on mixture models [21, 22] that show 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. 6; thus, the time complexity is $O(NM^N)$. As such, 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.

4.3 Approximation using Sort-Group

Our first approximation technique simplifies the result distribution formula while satisfying the accuracy requirement. This is important for the cases such as when we are summing 10 tuples and the result distribution has more than 1000 components. The resulting approximate distribution, with fewer components, is easier for users to read and incurs lower cost in subsequent processing.

Our algorithm, referred to as **sort-group**, is based on the idea that we can group adjacent Gaussian peaks in the exact result formula and approximate each group using a single Gaussian component. Algorithm 1, sketched below, searches for a GMM with K components that approximates the exact result distribution while meeting the application accuracy requirement. (An example of simplified approximate distributions is shown in Figure 7(c) in §6.) There are four main steps in Algorithm 1 as follows.

The algorithm first generates the result GMM based on Theorem 4.1 in Step 1. The result GMM is exact but has an exponential number of Gaussian mixture components. The algorithm then sorts these components by their means in Step 2. Next, it creates K groups of consecutive components in Step 3, normalizes the components

in each group into a proper Gaussian mixture distribution, so that mean and variance can be computed, and then approximates each group of components 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 the last step, in particular, the algorithm checks if the current K-component GMM is close enough to the exact distribution. The point-based Variation Distance (VD), similar in idea to the VD in [12], is used as the distance metric for two continuous distributions $D_1(x)$ and $D_2(x)$:

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

where the values of x are evenly spaced in the range where $D_1(x)$ and $D_2(x)$ have most of their density mass, and Δx is the distance between two consecutive x points. The constant $\frac{1}{2}$ ensures that VD is in [0,1]. For efficiency, we use only a small number of points to compute the VD between the exact distribution and an approximation distribution. It is experimentally shown that when the sort-group algorithm determines the value of K using a 30-point VD, the resulting K-component GMM successfully meets the application accuracy requirement, which we validated using a 1000-point precise VD.

Algorithm 1 Sketch of the **sort-group** algorithm for approximation

1: Compute all T components of the result distribution based on Theorem 4.1. 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 the last that may replace less.
4: Calculate the VD against the exact distribution. If the K-component mixture satisfies a given accuracy constraint, return it; otherwise, increase K and go to step 3.

We next consider the time complexity of this algorithm. Since Step 1 is the same as the algorithm for exact derivation, its cost is $O(NM^N)$. In Step 2, sorting has a complexity of $O(TlogT)$, with $T = M^N$, which yields $O(NM^N)$. In Steps 3 and 4, grouping enumerates the M^N components K times and performs VD calculation, which adds a cost $O(KM^N)$. As this analysis shows, the sort-group algorithm results in a simplified formula of size K, but still has an exponential time cost $O((N+K)M^N)$. In fact, it incurs a somewhat higher cost than exact derivation (due to the costs of Steps 2-4) to obtain a simplified distribution with bounded error.

4.4 Approximation using CF Fitting

The previous approximation algorithm becomes inefficient when the number of tuples to aggregate is large. We next propose to approximate the exact result distribution 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. 5), rather than an exponential number of components (Eq. 6). Our goal is to find some Gaussian mixture distribution whose CF best fits this product function.

We devise an approximation algorithm, named **Characteristic Function (CF) fitting**. which is sketched in Algorithm 2. 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, it returns the fitted parameters; otherwise it increases the number of components and repeats fitting. Note that the objective function for fitting contains

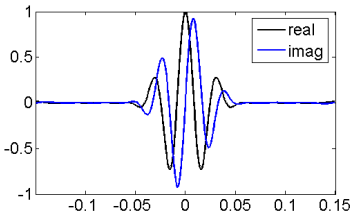


Figure 4: Example characteristic function for `sum` of 10 tuples.

both *real* and *imaginary* parts since the CFs are complex functions and both parts contribute to the pdf via inverse transformation.

We further employ a suite of optimizations based on statistical theory to improve performance and accuracy. The first optimization regards the choice of an appropriate range in the domain of the CF, $\Phi_{\texttt{sum}}(t)$, for fitting. Statistical theory shows that $\Phi(t)$ approaches 0 fast as t moves from the center 0. Figure 4 shows an example CF for `sum` of 10 tuples, with both the real and imaginary parts of the CF. Given this observation, we set the range for fitting to be a small region centered around 0, which directly affects the fitting quality.

The second optimization concerns the initial guess of the parameters of a K-component Gaussian mixture. Due to the oscillating behavior of the CF, the fitting result is quite sensitive to the initial guess of its parameters and can get stuck in local optima. Finding a good initial guess for fitting can be as hard as fitting itself. Luckily, Theorem 4.1 provides insight into choosing such an initial guess. Specifically, we precompute a small number of result components whose means are widely separated, capturing different regions of the result distribution. We group these components into a k-component mixture, and then use the parameters of this mixture as the initial guess for fitting.

Finally, the fitting residual ϵ needs to be chosen to guarantee the VD requirement. We have analyzed the connection between ϵ and VD through mathematical manipulation of the inverse transformation of CF: Given our choice of other parameters such as the range of CF fitting, our analysis shows that $\epsilon \leq 6 \cdot 10^{-5} \cdot VD^2$ is approximate bound for the fitted distribution to meet the application VD requirement, and this bound is insensitive to the characteristics of input distributions.

Algorithm 2 Sketch of the **CF fitting** algorithm for approximation

1: Obtain the expression of the CF of the `sum`, $\Phi_{\texttt{sum}}(t) = \prod_{i=1}^{n} \Phi_{X_i}(t)$. This is a complex function.
2: Take P points $\{t_i\}, (i = 1..P)$ from the domain of $\Phi_{\texttt{sum}}(t)$, and compute $\{\Phi_{\texttt{sum}}(t_i)\}, (i = 1..P)$.
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: $\sum_{i=1}^{P} \left[(Re(\overline{\Phi}(t_i) - \Phi_{\texttt{sum}}(t_i)))^2 + (Im(\overline{\Phi}(t_i) - \Phi_{\texttt{sum}}(t_i)))^2\right]$.
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.

The complexity of this algorithm is dominated by two steps. In Step 2, computing P points, each of which is a product of N complex terms with M components, has a cost $O(PMN)$. For Step 4, the complexity of the least squares fitting algorithm we use is $O(p^3)$, where p is the number of parameters [31], and $p = 3k$ in our case. Since fitting is repeated K times, its total cost is approximately $O(K^4)$. While this result is only approximate (due to the difficulty of bounding the fitting cost precisely), it has eliminated the exponential cost as in sort-group. However, K^4 is a non-trivial cost: when the result distribution has a complex shape, we need many Gaussian components (e.g., K=10) to approximate it. On the other hand, we

observe empirically that when the number of tuples N is large, the result distribution becomes smoother so we can use a small value of K. Therefore, we expect CF fitting to 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 [5]. This gives an asymptotic result but our algorithm dynamically determines when this result can apply. For example, the CASA system sometimes requires a small number of stream segments to be averaged, for which our algorithm determines that the CLT does not apply, whereas when the number of tuples is sufficiently large (e.g., greater than 20), the result distribution starts to become a smooth single Gaussian.

4.5 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 §6. This implies that once the hybrid solution is configured with those switching points, it can be applied to different workloads.

Our hybrid solution also supports the use of windows. It can be directly applied to stream systems using tumbling windows such as in CASA [20] and XStream [13]. When sliding windows are used, we employ incremental computation. First, generating the mixture formulas for exact derivation and sort-group (Eq. 6) can be incremental by factoring out the old tuples and adding the new ones. When the window size is large and the CLT is known to apply, the computation can also be implemented in an incremental fashion. We report on the efficiency of the resulting hybrid solution in §6.

5. JOINS OF UNCERTAIN TUPLES

In this section we consider efficient evaluation of joins under the PODS data model. The evaluation strategies of joins vary significantly with the nature of the join attributes. Recent research on probabilistic databases [2, 3, 19, 30] 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 hardly work because we cannot enumerate the possible values of a continuous random variable—the number of such possible values is infinite and each possible value has probability 0.

Below, we propose two types of joins of continuous random attributes to suit different application semantics. The first type deals with the complexities associated with equijoins on attributes modeled by continuous random variables. Our system employs a *probabilistic view* to facilitate such joins and offers a closed-form solution in Gaussian Mixture Models (GMMs) to represent join result distributions. The other (traditional) type of join pairs tuples from two inputs for inequality comparison and is modeled by a cross-product followed by a selection [25]. Our system supports such joins with exact result distributions in GMMs and efficient methods for pruning tuples with low existence probabilities.

5.1 Joins using Probabilistic Views

Let us first consider Query 1 in §2.1. Given each object location, it retrieves the corresponding temperature for this location. Most notably, the two common join attributes (X, Y) in the object location stream are uncertain and modeled by continuous random variables (in contrast, the join attributes in the temperature stream are the fixed sensor locations and hence deterministic). This kind of join is inherently difficult to support for two reasons. First, given a location tuple i, it is not possible to enumerate the values of (X_i, Y_i) modeled by continuous random variables. Second, since each value of (X_i, Y_i) has probability 0, any join result that pairs a specific value of the location tuple and a temperature tuple also has probability 0. These issues are inherent in equijoins involving continuous random variables. For instance, if we change the (X, Y) attributes in the temperature stream to also be probabilistic attributes (e.g., returned by a mobile sensor), the equijoin between the location stream and the temperature stream on (X, Y) still has the above two issues.

To attain proper result distributions for equijoins involving continuous random variables, we introduce the notion of a **probabilistic view**. In the example of Query 1, a probabilistic view on the temperature stream is defined to be the distribution of *Temp* given (x, y), denoted by $p_{Temp}(t|x, y)$. Then, for each tuple i in the location stream, the process of iterating all possible values of (X_i, Y_i) and retrieving the corresponding temperature distribution can be compactly represented by $f_{X_i,Y_i}(x, y)p_{Temp}(t|x, y)$, yielding a joint distribution $f_{X_i,Y_i,Temp}(x, y, t)$. Below we give formal definitions of such equi-joins using a probabilistic view.

Definition 3 *We denote a stream by $S(\mathbf{A}^*, \mathbf{B}^*, \bar{\mathbf{S}})$, where $\mathbf{A}$ is a vector of attributes that can be deterministic or probabilistic (denoted by $\mathbf{A}^* = \mathbf{A}$ or $\mathbf{A}^p$), $\mathbf{B}$ is another vector of deterministic or probabilistic attributes that are related to attributes in $\mathbf{A}$, and $\bar{\mathbf{S}}$ is a vector for the rest of the attributes. The probabilistic view of $\mathbf{B}$ as a function of $\mathbf{A}$, denoted by $V_{\mathbf{B}|\mathbf{A}}(S)$, is a distribution of $\mathbf{B}$ for a given value of $\mathbf{A}$, characterized by $p_{\mathbf{B}}(\mathbf{b}|\mathbf{A} = \mathbf{a})$.*

Following the definition, we call the $\mathbf{B}$ attributes that the view returns the *view attributes*, and say that they view-depend on the $\mathbf{A}$ attributes. In the temperature stream, the view attribute is *Temp* and it view-depends on the attributes X and Y. Given specific arguments, e.g., (X=1, Y=2), the view gives a distribution of temperature $p_{Temp}(t|X = 1, Y = 2)$.

Given an equi-join query, the suitable probabilistic view can be recognized by the compiler based on the following observation: As in traditional databases, equi-joins only include one copy of the join attributes, say $R.(X, Y)$, in the output; the other copy of the join attributes, $S.(X, Y)$, is used only for comparison but suppressed from output. This implies a probabilistic view for the other input, S, and the view is defined for the S attributes included in the output, e.g., $S.Temp$, dependent on the join attributes.

Definition 4 *Given two independent streams $R(\mathbf{A}^p, \bar{\mathbf{R}})$ and $S(\mathbf{A}^*, \mathbf{B}^*, \bar{\mathbf{S}})$ with the probabilistic view $V_{\mathbf{B}|\mathbf{A}}(S)$, an equi-join of R and S on A using the probabilistic view, denoted by $R \underset{A}{\bowtie^v} S$, is a join of R and $V_{\mathbf{B}|\mathbf{A}}(S)$: For any tuple i in R, denoted by $(\mathbf{A_i}, \bar{\mathbf{R}}_\mathbf{i})$, the join combines the tuple with the view $V_{\mathbf{B}|\mathbf{A}}(S)$ and outputs a tuple $(\mathbf{A_i}, \bar{\mathbf{R}}_\mathbf{i}, \mathbf{B})$ with the joint distribution defined as:*

$$f_{\mathbf{A_i},\bar{\mathbf{R}}_\mathbf{i},\mathbf{B}}(\mathbf{a}, \bar{\mathbf{r}}, \mathbf{b}) \equiv f_{\mathbf{A_i},\bar{\mathbf{R}}_\mathbf{i}}(\mathbf{a}, \bar{\mathbf{r}}) \cdot p_{\mathbf{B}}(\mathbf{b}|\mathbf{S}.\mathbf{A} = \mathbf{a}).$$

In this definition, the join preserves each tuple in R and extends it with the attributes in $\mathbf{B}$ from S. It is evident that this definition gives a proper joint distribution for the result tuple because $f_{\mathbf{A_i},\bar{\mathbf{R}}_\mathbf{i},\mathbf{B}}(\mathbf{a}, \bar{\mathbf{r}}, \mathbf{b})$ integrates to 1 over $\mathbf{a}, \bar{\mathbf{r}}, \mathbf{b}$.

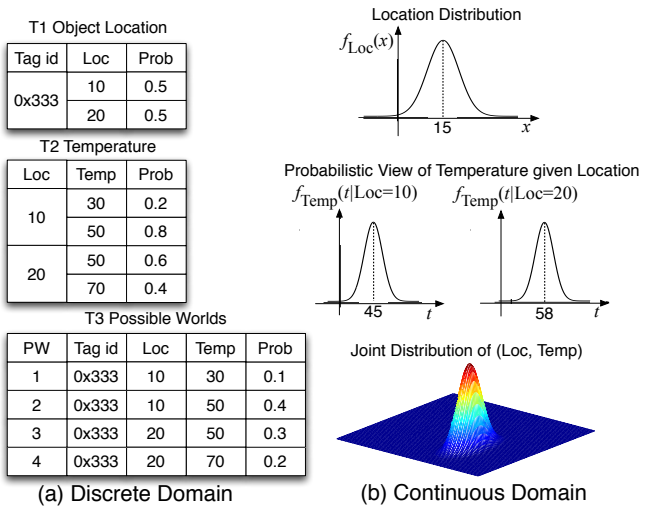


Figure 5: Compare equi-joins in the discrete domain (using PWS) and in the continuous domain (using a probabilistic view).

Given Definition 4, an important observation is that our equijoins of *continuous* random variables using a probabilistic view are consistent with the possible worlds semantics for equijoins of *discrete* random variables. Fig. 5 illustrates the connection: In Fig. 5(a), T_1 shows a discrete distribution of an object location, T_2 shows discrete distributions of temperature given two specific locations, and T_3 shows a join result for each possible world comprising a unique (location, temperature) pair. Fig. 5(b) shows the continuous version of these tables. In particular, the probabilistic view serves the same purpose as T_2, i.e., a distribution of temperature for each location. Then the joint distribution of $(Loc, Temp)$ is equivalent to the distribution of all possible worlds computed by the product of the location distribution and the temperature distribution given a specific location.

Closed-form result distributions in GMMs. Given the above definitions, we seek a closed-form solution to join result distributions in our data model. Recall that the PODS data model describes uncertain attributes 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 GMMs. While the assumption of order-1 linearity may seem restrictive, it actually can be applied to the view at either a global or local scale, allowing implementation choices for both accuracy and efficiency.

Our theorem below offers join result distributions in GMMs when both the join attributes $\mathbf{A}$ and the view attributes $\mathbf{B}$ are deterministic in the input S on which the view $V_{\mathbf{B}|\mathbf{A}}(S)$ is defined. It also offers a foundation for extending our solution to other types of attributes used to define the view.

Theorem 5.1 *Given $R(\mathbf{A}^p, \bar{\mathbf{R}})$, $S(\mathbf{A}, \mathbf{B}, \bar{\mathbf{S}})$ with the view $V_{\mathbf{B}|\mathbf{A}}(S)$, and $R \underset{A}{\bowtie^v} S$, assume order-1 linear regression to model the view:*

$$\mathbf{B} = \mathbf{A}\boldsymbol{\beta} + \mathbf{E} \tag{7}$$

where $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{E}$ are row vectors, $\boldsymbol{\beta}$ is a parameter matrix, and $\mathbf{E}$ is normally distributed with mean $\mathbf{0}$ and covariance matrix Σ_E. For each tuple i in R, if $(\mathbf{A_i}, \bar{\mathbf{R}}_\mathbf{i})$ follows a GMM, a mixture of m components identified by the parameters $(p_c, \boldsymbol{\mu}_c, \Sigma_c)$, $(c = 1..m)$, then based on Definition 4, the join of tuple i and $V_{\mathbf{B}|\mathbf{A}}(S)$ yields a distribution of $(\mathbf{A_i}, \bar{\mathbf{R}}_\mathbf{i}, \mathbf{B})$ which also follows a GMM, a mixture of m components with parameters $(p_c, \boldsymbol{\mu}'_c, \Sigma'_c)$, $(c = 1..m)$:

$$\boldsymbol{\mu}'_c = (\boldsymbol{\mu}_c, \boldsymbol{\mu}_c\bar{\boldsymbol{\beta}}), \bar{\boldsymbol{\beta}} = \begin{pmatrix} \boldsymbol{\beta} \\ \mathbf{0} \end{pmatrix}, \Sigma'_c = \begin{pmatrix} \Sigma_c & \Sigma_c\bar{\boldsymbol{\beta}} \\ \bar{\boldsymbol{\beta}}^T\Sigma_c & \Sigma_E + \bar{\boldsymbol{\beta}}^T\Sigma_c\bar{\boldsymbol{\beta}} \end{pmatrix}.$$

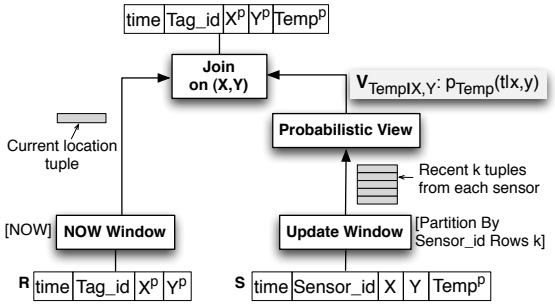


Figure 6: Query plan for the join using a probabilistic view (Q1).

Theorem 5.1 fully characterizes each join result distribution with all its parameters. In practice, $\boldsymbol{\beta}$ and Σ_E are unknown. They can be estimated using regression over the tuples in S, denoted by $(\mathbf{A_j}, \mathbf{B_j}, \bar{\mathbf{S}}_\mathbf{j})$, $(j = 1 \cdots n)$. The least squares estimates of $\boldsymbol{\beta}$ and Σ_E are:

$$\boldsymbol{\beta} = (\tilde{\mathbf{A}}^T\tilde{\mathbf{A}})^{-1}\tilde{\mathbf{A}}^T\tilde{\mathbf{B}} \tag{8}$$

$$\Sigma_E = \tilde{\mathbf{B}}^T(\mathbf{I}_n - \tilde{\mathbf{A}}(\tilde{\mathbf{A}}^T\tilde{\mathbf{A}})^{-1}\tilde{\mathbf{A}}^T)\tilde{\mathbf{B}}/(n-k), \tag{9}$$

where $\tilde{\mathbf{A}} = (\mathbf{A}_1^T, \cdots, \mathbf{A}_n^T)^T$, $\tilde{\mathbf{B}} = (\mathbf{B}_1^T, \cdots, \mathbf{B}_n^T)^T$, $\mathbf{I}_n$ is the identity matrix of size n, and k is the size of any vector $\mathbf{A}_i$.

We next consider other types of attributes used to define the view; that is, at least one of the join and view attributes in the view input S is probabilistic. Many smoothing techniques are available for creating the view. In our work, we opt to sample from the distributions of those probabilistic attributes, collect the samples into a new input S', and apply Theorem 5.1 to the original input R and the new view input S'. This method allows us to obtain a closed-form solution to the join result distribution and guarantees that it is also in the form of GMMs. Note that sampling here is on a Gaussian Mixture distribution, hence easy and fast, and the samples are used to build an order-1 regression model. This is different from a sampling approach that constructs the full join result distribution directly from the samples—the lack of a model-based view and the need to sample over the joint space makes this approach less desirable, as we shall show in §6.2.

Evaluation Techniques. The query plan for Query 1 with a join and a probabilistic view is depicted in Fig. 6. The plan first applies the query-specified windows to the inputs: A *Now* window feeds each location tuple as the left input to the join. An *update* window, [Partition By sensor_id Rows k], contains the most recent k temperature tuples from each sensor; the probabilistic view $V_{temp|x,y}$ is maintained over the update window and is the right input to the join. The join extends each location tuple with the temperature 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 for sensor data [14, 10], our work differs by exploring the tradeoffs of applying order-1 linear regression at a global versus local scale.

Global regression applies regression equations (Eq. 8 and Eq. 9) to all S tuples in the current update window to construct the view. As proposed in recent work [14], the view can be maintained incrementally by updating intermediate matrices for $\boldsymbol{\beta}$, e.g., $\tilde{\mathbf{A}}^T\tilde{\mathbf{A}}$ and $\tilde{\mathbf{A}}^T\tilde{\mathbf{B}}$ in Eq. 8. 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 1, the temperature may not be a linear function of the location but rather, say, a quadratic function. In that case, 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. We design a local regression method as follows: Given each R tuple following a GMM, use the means and the variances of the components of the GMM to 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 method is that it does not require the assumption of global linearity, hence allowing more accurate view construction. However, a very small set of tuples in the LRR 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 (as shown experimentally in §6.2). Computation-wise, regression is applied to a small set of tuples, hence with a low cost.

5.2 Joins using the Cross-Product

Depending on the application, a join can also be modeled using a cross-product followed by a selection [25]—in this case, only inequality predicates are allowed to avoid join results of zero probability. An example is Query 2 in §2.1: it compares every pair of objects for proximity in location. Our system supports such joins with result distributions in GMMs. Our system can further filter the results of the cross-product that have low existence probabilities. Since computing the tuple existence probability based on the join condition requires an expensive integral, we devise *linear transformation* for GMMs to reduce the dimensionality of integration and hence improve efficiency. Due to space constraints, details of these techniques are omitted here, but can be found in [28].

6. PERFORMANCE EVALUATION

We now evaluate our techniques for joins and aggregates, and compare them to sampling-based methods ([12] for aggregates and our own method for joins) to demonstrate our performance benefits. We further perform a case study in a real-world tornado detection application [20], and show that our techniques have enabled better detection results and stream speed computation.

6.1 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 uncertain attribute. Each tuple is modeled by a mixture of two Gaussian components. The means of the two components are uniformly sampled from [0, 5] and [5, 50] respectively to model complex real-world distributions from asymmetric to bimodal (the resulting workload is much harder than those using Gaussian distributions only such as in [25]). The standard deviation of each Gaussian component is within [0.5, 1] and the coefficient is uniform from [0, 1].

We evaluated `avg` over the above tuple stream. The windows used can be tumbling (default) or sliding, and the sizes are measured by the number of tuples N. The accuracy metric is the variation distance (VD) defined in §4.3. The default accuracy requirement is VD ≤ 0.1. Throughput numbers were obtained from a server using a 3Ghz dual-core xeon processor with 1GB memory for use in Java.

Expt 1: Compare our algorithms. We first compare the three algorithms, exact derivation, sort-group for approximation, and CF fitting for approximation, that constitute our hybrid solution. We varied the window size N, since it directly affects the result distribution and the computation needed.

Fig. 7(a) shows the throughput results. As expected, the throughput of exact derivation and sort-group is high when N is small, e.g., up to 10, but deteriorates quickly afterwards because the exact result formulas generated grow exponentially in N. Sort-group has an increased cost to simplify these result formulas. 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.

Fig. 7(b) shows that most algorithms satisfy the requirement of VD ≤ 0.1. This is because the approximation algorithms compare with the true distributions through either direct VD comparison or fitting with a small residual. We observe that the hardest range is 5 to 10 tuples, where the result distributions 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. 7(c). From 15 tuples onwards, the result distributions become smoother with fewer peaks. CF fitting accuracy is low for less than 10 tuples since we limited the number of components in fitting to gain some performance.

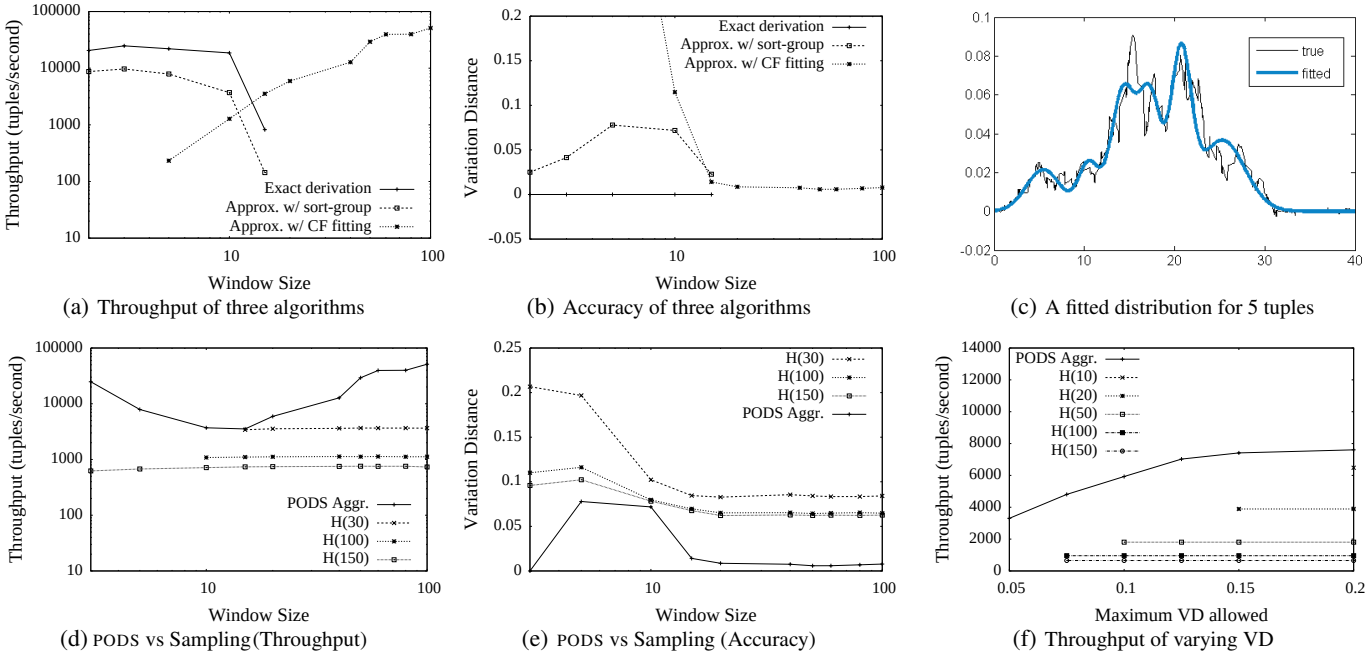
We further evaluated our algorithms under different data characteristics, e.g., different ranges of means and variances. In all cases, we observe the same trends for both accuracy and throughput, and the crossing points among the three algorithms stay the same. We also note that our workload is already 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 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 [12]. Given N tuples, this algorithm (1) generates $k \cdot \mu$ samples for each tuple, (2) performs aggregation over them to get $k \cdot \mu$ result samples, and (3) sorts the result samples and builds a histogram with k buckets and μ samples for each bucket. Since we found the accuracy of this algorithm to be more sensitive to k, we varied k among 30, 100, and 150 while fixing μ to 50.

Figs. 7(d) and 7(e) show the results. Our hybrid algorithm outperforms all settings of the histogram algorithm in both throughput and accuracy. For accuracy, only the histogram with $k = 150$ ensures VD ≤ 0.1. The other two violate this in the "hard" range of 5 to 15 tuples (hence their throughput numbers are omitted). These results confirm the advantages of our algorithm over sampling since we can adapt to the accuracy requirement while maximizing the throughput.

Expt 3: Vary the VD requirement. To further study our adaptivity to accuracy requirements, we varied VD from 0.05 to 0.2. We used sliding windows with the size N gradually increasing from 5 to 50, so we can examine different ranges of the hybrid solution. The window slides by 10 tuples or N, whichever is smaller. We also made the histogram algorithm incremental by maintaining necessary samples from the previous window. Fig. 7(f) shows the throughput results (when the VD requirement is met). Our algorithm is shown to outperform the histogram algorithm for all values of VD. Moreover, we can achieve better throughput under a relaxed condition.



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

(d) PODS vs Sampling(Throughput) (e) PODS vs Sampling (Accuracy) (f) Throughput of varying VD

Figure 7: Experimental results for aggregation using our algorithms and histogram-based sampling $H(k)$ (with μ=50)

Similar results are observed for tumbling windows and different slide sizes and hence omitted here.

6.2 Evaluation of Joins

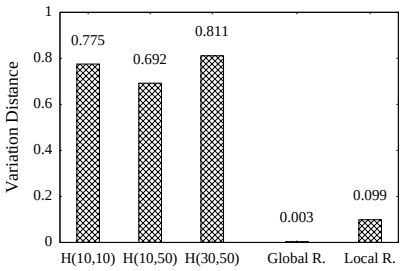
We next evaluate our join techniques. We focus on the equi-joins between R=(A^p) and S=(A, B^*), where B can be deterministic or probabilistic. In our discussion below, we first consider B being deterministic and then summarize similar results observed when B is probabilistic. Besides our join technique based on the probabilistic view, we also develop a sampling technique that constructs the full join result distribution directly from samples, without using a model-based view or our closed form solution to the join result distribution. We compare these techniques in both accuracy and throughput.

Histogram-based sampling. For each R tuple, this method takes samples from the distribution of $R.A^p$. It then attempts to extend each sample a for $R.A^p$ with a sample b for $S.B$ (as the join). To do so, the method searches all S tuples in the update window for the two S tuples whose $S.A$ values are closest to the given sample a. It then applies linear interpolation to the $S.B$ values in these two tuples, with added random noise to facilitate later histogram construction, to obtain a sample b. Finally it uses all the samples (a, b) to construct an equi-depth 2-dimensional histogram as an approximate distribution for each join result (A^p, B^p). The histogram setting $\mathrm{H}(k, \mu)$ depends on the number of buckets per dimension, k, and the number of samples per bucket, μ. To build a 2-dimensional histogram for each join result, we create $k^2\mu$ samples for the attribute A^p of each R tuple, resulting in $k^2\mu$ samples for the each join result (A^p, B^p).

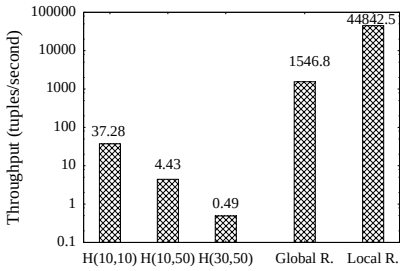
In our experiments, the R stream is an object location stream from an RFID inference system [29] where each tuple has a Gaussian distribution (using a mixture distribution will not incur more cost given our closed-form solution). The S stream is produced by our temperature simulator, which generates 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* (UW) 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 join.

Expt 4: Sampling versus regression. We compare the sampling method with our join method using global or local regression for view construction. We first generate the temperature stream by assuming that temperature is a linear function of location with added noise. The local regression region (LRR) is set to be 18. As seen in Fig. 8(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). This is because the interpolation-based sampling only takes two points into consideration, thus is more sensitive to the temperature reading error. In contrast, our regression-based view uses more points to better estimate the underlying function and the random noise. 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. 8(b) shows that the sampling method is already very slow due to the use of a large number of samples: 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.

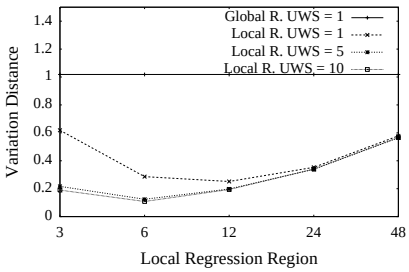
Expt 5: Global versus local regression. We next use a quadratic function to generate the temperature stream and compare global versus local regression. Since the sampling method performs poorly again, we omit its results here. Since local regression is sensitive to the number of data points available, we vary its local regression region LRR in a wide range (which does not affect global regression). As Fig. 8(c) shows, global regression has poor accuracy since its



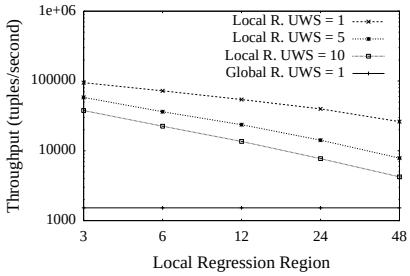
(a) Sampling vs regression in VD (linear function).



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

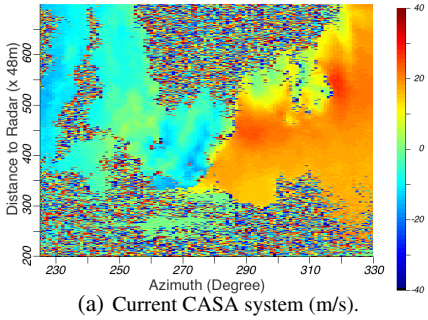


(c) Global vs local regression in VD (quadratic function)

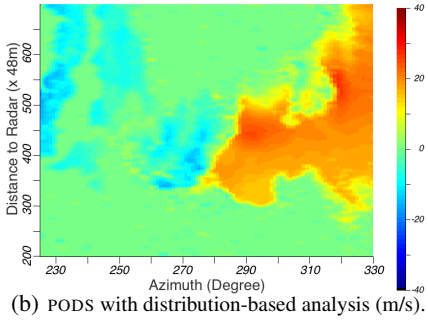


(d) Global vs local regression in Throughput (quadratic function)

Figure 8: Results of joins using sampling $H(k, \mu)$, global regression (Global R.), and local regression (Local R.).



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



(b) PODS with distribution-based analysis (m/s).

Figure 9: Radial velocity maps of a true tornadic region from CASA and PODS.

global linearity assumption is not true any more. The VD of local regression (UW=1) first improves as the increased region gives more points for regression. Then it degrades 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 the update window UW, e.g., using the most recent 5 readings from each sensor. The rationale behind this is that the underlying function usually changes slowly, and using several old tuples from the past few seconds will not add much stale information. Fig. 8(c) shows such improved VD with UW=5 and 10. Fig. 8(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 UW=5, local regression can gain both high accuracy and efficiency.

In another set of experiments, we generated the stream S=(A, B^p) where the attribute B is probabilistic. As stated in §5, we sample each S tuple on the attribute B, collect all the samples into a new input S', handle the rest as if S' has a deterministic attribute B. We made similar observations about the advantages of our join method over sampling and the tradeoffs between global and local regression.

6.3 Case Study: Tornado Detection

We now demonstrate the effectiveness of capturing uncertainty using distributions in a real-world tornado detection system [20]. We first modified the velocity analysis module in Fig. 1 to generate velocity distributions in GMMs. 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: **Step 1 Strength Filter.** If the radar signal strength is below a threshold, we output zero and skip Step 2. **Step 2 GMM Fitting.** Create a Gaussian distribution from the mean and variance of f_F. Return the distribution for output if it passes the goodness test against f_F, for both the Gaussian shape and high concentration around the mean. If the goodness test fails, fit a mixture of two Gaussians from f_F and remove any component with a large variance as noise. **Step 3 Smoothing.** We average the distributions of high and low frequency stream segments and across neighboring regions. For `avg` over GMMs, we apply the techniques in §4.2 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 detection algorithm.

Table 1: Result of a real tonadic dataset of 947s from 84 scans.

	Analysis Time	Detection Time	False Positives
CASA	182.1 s	4486 s	2137
Step1	180.06 s	640 s	1125
Step3	170.78 s	956 s	1650
Step1+3	176.01 s	441 s	313
Step1+2+3 (PODS)	**581.9 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. The comparison between Fig. 9(a) and 9(b) shows that our techniques yield much smoother velocity maps. Specifically, the Strength Filter removes most colorful dots (i.e., noise) produced from the regions with weak signals (indicating the lack of interesting weather events); the GMM Fitting smoothes data by removing noise in the regions with strong signals; the Smoothing step finally smoothes data across regions, which is especially important for the boundaries between weak-signal regions and strong-signal regions.

We measure the analysis speed, detection speed and detection result quality. To explore the effect of each step, we show the breakdown of these measurements in Table 1. As shown in rows 1-3, both Step 1 and Step 3 can significantly reduce the detection time because data is smoother, but have only a limited effect on

false positives. We further combine Step 1 and 3 as shown in row 4, resulting in further reduction of detection time and false positives. While tornado detection can now be performed at stream speed, the remaining 313 false positives still result in a poor quality of detection results. When we turn on Step 2 for model fitting and model-based analysis, the number of false positives drops to 9 across all 84 scans as shown in row 5. The reason for this remarkable effect is that Step 2 removes noise in the regions with strong radar signals on which the detection algorithm focuses. Although the analysis time increases due to model fitting, given pipeline parallelism, the overall system can still run at stream speed since each of the analysis phase and the detection phase is faster than the radar sensing speed. As such, our model fitting and model analysis approach is shown to provide high-quality detection results while enabling stream-speed data analysis and tornado detection.

7. 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 [7, 16, 19] adopts the finite and discrete tuple model as in probabilistic databases. As stated previously, most of the techniques proposed for discrete variables cannot be applied to problems with continuous variables. Furthermore, most of these techniques compute the mean or a few higher moments of result distributions [7, 16]. In contrast, PODS computes the full result distribution that enables subsequent operators to compute their result distributions as well as visualization in scientific applications.

Models and views of sensor data. Recent work on sensor networks [9, 14] builds statistical models to capture correlations among attributes. Given a query, such models enable reduced costs of data acquisition and communication. FunctionDB [27] transforms discrete sensor observations into continuous functions and supports querying over these functions. More relevant to our work, in particular, our equi-join technique, is the work that supports views over uncertain data and uses inference to evaluate queries defined on a view [10, 18]. In contrast, our work uses a closed-form solution to derive join result distributions once the view is constructed, hence gaining much higher throughput than the inference-based approach.

Probabilistic databases with continuous uncertainty. Two recent workshop papers [1, 26] consider the extension of probabilistic databases with continuous-valued attributes. While they mainly present the motivation or initial design, they made similar arguments as in our paper for a suitable model for continuous random variables and the need to compute distributions, not just a few moments.

8. CONCLUSIONS

In this paper, we presented the PODS system for uncertain data stream processing, in particular, its unique data model based on GMMs and advanced techniques for aggregates and joins under the model. These techniques are grounded in statistical theory to ensure correctness, and also amenable to approximation and optimization for improved performance. Our techniques further produce result distributions in GMMs, hence confirming the practicality of our data model. Our results show that PODS outperforms sampling methods in accuracy and speed in stream processing. A case study further reveals that PODS can improve a real tornado detection system with better quality of results and stream-speed processing.

In the future, we plan to extend our work in a few directions including the support of a broader set of relational operators, query optimization, inter-tuple correlations using advanced techniques such as lineage, and a hybrid system that explore the combination of both discrete and continuous random variables.

Acknowledgements. The authors would like to thank Peter Haas for useful conversations, Divesh Srivastava for comments on an earlier draft of this paper, and the anonymous reviewers for their helpful feedback.

9. REFERENCES

[1] P. Agrawal and J. Widom. Continuous uncertainty in Trio. In *MUD*, 2009.
[2] L. Antova, et al. Fast and simple relational processing of uncertain data. In *ICDE*, 983–992, 2008.
[3] O. Benjelloun, et al. Uldbs: Databases with uncertainty and lineage. In *VLDB*, 953–964, 2006.
[4] D. Carney, et al. Monitoring streams: a new class of data management applications. In *VLDB*, 215–226, 2002.
[5] C. Casella and R. Berger. *Statistical Inference*. Duxbury, 2001.
[6] R. Cheng, et al. Evaluating probabilistic queries over imprecise data. In *SIGMOD*, 551–562, 2003.
[7] G. Cormode and M. Garofalakis. Sketching probabilistic data streams. In *SIGMOD*, 281–292, 2007.
[8] N. N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. *VLDB J.*, 16(4):523–544, 2007.
[9] A. Deshpande, et al. Model-driven data acquisition in sensor networks. In *VLDB*, 588–599, 2004.
[10] A. Deshpande and S. Madden. MauveDB: supporting model-based user views in database systems. In *SIGMOD*, 73–84, 2006.
[11] Y. Diao, et al. Capturing data uncertainty in high-volume stream processing. In *CIDR*, 2009.
[12] T. Ge and S. B. Zdonik. Handling uncertain data in array database systems. In *ICDE*, 1140–1149, 2008.
[13] L. Girod, et al. Xstream: a signal-oriented data stream management system. In *ICDE*, 1180–1189, 2008.
[14] C. Guestrin, et al. Distributed regression: an efficient framework for modeling sensor network data. In *IPSN*, 1–10, 2004.
[15] R. Jampani, et al. Mcdb: a monte carlo approach to managing uncertain data. In *SIGMOD*, 687–700, 2008.
[16] T. S. Jayram, et al. Estimating statistical aggregates on probabilistic data streams. In *PODS*, 243–252, 2007.
[17] S. R. Jeffery, et al. Adaptive cleaning for RFID data streams. In *VLDB*, 163–174, 2006.
[18] B. Kanagal and A. Deshpande. Online filtering, smoothing and probabilistic modeling of streaming data. In *ICDE*, 2008.
[19] B. Kanagal and A. Deshpande. Efficient query evaluation over temporally correlated probabilistic streams. In *ICDE*, 2009.
[20] J. F. Kurose, et al. An end-user-responsive sensor network architecture for hazardous weather detection, prediction and response. In *AINTEC*, 2006.
[21] G. McLachlan and D. Peel. *Finite Mixture Models*. Wiley-Interscience, 2000.
[22] S. Fruhwirth-Schnatter. *Finite Mixture and Markov Switching Models*. Springer, 2006.
[23] T. Sauer. *Numerical Analysis*. Addison Wesley, 2005.
[24] P. Sen, et al. Exploiting shared correlations in probabilistic databases. In *VLDB*, 809–820, 2008.
[25] S. Singh, et al. Database support for probabilistic attributes and tuples. In *ICDE*, 1053–1061, 2008.
[26] D. Suciu, et al. Embracing uncertainty in large-scale computational astrophysics. In *MUD*, 2009.
[27] A. Thiagarajan and S. Madden. Querying continuous functions in a database system. In *SIGMOD*, 791–804, 2008.
[28] T. Tran, et al. PODS: A new model and processing algorithms for uncertain data streams. Technical report, UMass Amherst, 2009. `http://www.cs.umass.edu/~ttran/pubs/pods.pdf`.
[29] T. Tran, et al. Probabilistic inference over rfid streams in mobile environments. In *ICDE*, 1096–1107, 2009.
[30] D. Z. Wang, et al. Bayesstore: Managing large, uncertain data repositories with probabilistic graphical models. In *VLDB*, 2008.
[31] N. Ye, editor. *The Handbook of Data Mining*. Lawrence Earlbaum Associates, 2003.