Towards Real-Time Data Stream Processing

Jessica Chang[*] and **Andrey Kolobov**[†]

1 Introduction

Many applications require the continuous tracking of the state of a system in order to detect the occurrence of a particular event. RFID sensors, in particular, have become an increasingly popular means of gathering tracking information about the objects of interest. The need to query these data has spurred research at the intersection of sensor networks and databases. There are a number of challenges in this effort. The extent to which an RFID ecosystem can monitor a particular location is limited by the number of antennas in the area of interest; usually, trackers do not provide exhaustive coverage. The accuracy of the sensors also leaves much to be desired. As a result, the networks produce data streams that are unstructured, incomplete, and highly inaccurate; querying such information directly is infeasible.

Instead, researchers have proposed building probabilistic inference models over the raw data, called *Markovian streams*. An example of a DBMS that uses Markovian streams to answers queries about the sensor data is Lahar [8]. Currently, the inference process to compute the marginals constituting Lahar's underlying Markovian streams from the raw data is carried out by a particle filter.

In many settings, the simple ability to answer queries about probabilistic events is not sufficient. DBMSs deployed for weather monitoring, surveillance, or intrusion-detection tasks are also required to be able to prepare the data for querying in real time; the speed of raw data processing is thus of paramount importance in these applications. Currently, the use of a particle filter prevents Lahar from satisfying the speed requirements of real-time data processing, taking over ten seconds to process each data point.

Our primary contribution rectifies this situation and turns Lahar into a real-time system by employing a Kalman filter in place of the particle filter. The Kalman filter relies on linear Gaussian process models to do inference, which allows it to do the computations very efficiently in closed form. At the same time, linear Gaussians may not model arbitrary processes well. However, we make the observation that users of probabilistic event databases are likely to be interested in only top K most probable answers to their query, ranked by probability. Even though the Kalman filter may miscalculate the exact probabilities of query answers, we hypothesize that their *ordering* will largely be maintained. Indeed, our experimental results indeed demonstrate that the Kalman filter gains a speedby a factor of over 3000 compared to the particle filter while, surprisingly, even improving on quality of the top K query results set.

[*]jschang@cs.washington.edu

[†]akolobov@cs.washington.edu

2 Background

Lahar [10] is a probabilistic event database that supports queries over temporal sensor data streams. Our work concerns improving Lahar's speed in answering a certain subclass of such queries, called *event queries*.

Event queries express questions about temporal sequences of states of an object or a person, e.g., *"When did John enter office 318?"*. Formally, this query asks for the time step t when the two-state sequence $Q = (In(John, Hallway, t); In(John, Office318, t+1))$ began. As an answer to it, Lahar can return, for every time step in the stream, the probability that the sequence started at that particular step. A possible response is in table 1.

Time step	0	1	2	3	4	...
Probability	0	0	0.17	0.49	0	...

Table 1: Lahar's possible response to an event query

In practice, users will likely want to retrieve only top K most probable time steps.

Note that the probability of Q starting at a given time t is $P(Q,t) = P(In(John, Office318, t+1)|In(John, Hallway, t))P(In(John, Hallway, t))$. Thus, Lahar needs to know the marginals $P(In(John, L, t))$ over locations L at each t given all evidence provided by the data stream up to the current moment. It also needs the *correlations* $P(In(John, L', t+1)|In(John, L, t))$ between John's locations at every two successive time steps. The marginals can be derived from the correlations if the prior $P(In(John, L, 0))$ is known.

Letting variable X_i denote the attribute $In(John, L, i)$, the sequence of distributions $(P(X_0), P(X_1|X_0), P(X_2|X_1)\ldots))$ is an example of a *Markovian stream*. More generally, a Markovian stream over value attributes $A_1, \ldots, A_n$ with respective domains $D_1, \ldots, D_n$ is a pair (p_0, C), where p_0 is a prior over attribute values in $D_1 \times \ldots \times D_n$, and C is a sequence of attribute correlation distributions for each

pair of successive time steps. Markovian streams serve as inputs to Lahar, based on which the latter does its query-processing.

Time step	1	2	3	4	...
Antenna IDs	104	100		112, 114	...

Table 2: Example data stream for a given RFID tag

Crucially, the raw data do not arrive in the form of a Markovian stream. While their exact format may differ between applications, in general they are composed of (noisy) sensor observations at various time steps. In our example, the data may come may come in the form of IDs of the RFID antenna that detected John's RFID tag in the vicinity at a given time. An the resulting data stream looks as in Table 2. Note that the readings are missing at some of the time steps. In practice, time intervals with missing data may be very long. This fact, coupled with noise in the observations and the typically incomplete coverage of the target area by the sensor network motivate why Lahar works not with the raw data but rather with Markovian streams the data's model-based view.

Thus, a critical question for Lahar's operation is how to turn the raw data stream into a Markovian stream, and do so efficiently. The transformation is performed by two related state estimation processes, *filtering* and *smoothing*. Filtering obtains a distribution over the state of an entity given all observations up to the current time step t_c, e.g. $P(In(John, L, t_c)|e_{t_c:1})$ where e_i denotes the locations of the antennas that detected John's RFID tag at time step i. Smoothing improves on the filtered estimates at previous time steps by correcting them with all the evidence up to the current time step. In our example, the result of smoothing are distributions $P(In(John, L, t)|e_{t_c:1})$ for all time steps $t \leq t_c$. Once these distributions are known for each pair of consecutive time steps, there are several ways of computing the correlations constituting the Markovian stream. We do not describe them here for the lack of space and employ our own method of computing them covered in detail later in the paper.

More precisely, filtering and smoothing try to obtain estimates of state variables $X_1, \ldots, X_{t_c}$ given the set of observations $e_1, \ldots, e_{t_c}$. Both estimators make the Markov assumption about the variable dependencies:

$$P(X_{i+1}|X_{i:1}) = P(X_{i+1}|X_i)$$
$$P(e_i|X_{i:1}) = P(e_i|X_i)$$

Assuming the prior $P(X_0)$, *transition model* $P(X_{t+1}|X_t)$, and the *observation model* $P(e_t|X_t)$ are known, we can recursively compute a filtered state estimate X_{t_c} using the newly arrived data point e_{t_c} by performing a *prediction step*

$$P(X_{t_c}|e_{t_c:1}) = \int P(X_{t_c}|X_{t_c-1})P(X_{t_c-1}|e_{t_c-1})dX_{t_c-1} \quad (1)$$

and an *update step*

$$P(X_{t_c}|e_{t_c}) = \frac{P(e_{t_c}|X_{t_c})P(X_{t_c}|e_{t_c-1})}{\int P(e_{t_c}|X_{t_c})P(X_{t_c}|e_{t_c-1})dX_{t_c}} \quad (2)$$

Having the results of update and prediction steps at all time steps up to $t_c - 1$, we can use the newly arrived data point e_{t_c} to recursively correct our state estimates for any of the past time steps $t \leq t_c$ starting at time $t_c - 1$ using the smoothing equation

$$P(X_t|e_{t_c:1}) = P(X_t|e_{t:1}) \int \frac{P(X_{t+1}|X_t)P(X_{t+1}|e_{t_c:1})}{P(X_{t+1}|e_{t:1})} dX_{t+1} \quad (3)$$

As they stand, equations 1, 2, and 3 are not directly usable, since the integrals in them may be hard to evaluate for general distributions. Below we briefly describe two approximation techniques that implement the ideas embodied in these equations.

Particle filter and smoother. Particle filters attempt to approximate the distributions $P(X_t|e_{t_c:1})$, $P(X_t|e_{t:1})$, and $P(X_{t+1}|e_{t:1})$ for all time steps $t \leq t_c$ by sets of weight samples, or *particles*. There are several variants of this approach, but all of them have to resample the particles at each time step. While an increasing number of particles allow the particle filter/smoother to approximate the target distributions arbitrarily well, they may also slow down the algorithm considerably.

Kalman filter and RTS smoother. At heart, a Kalman filter is restriction of equations 1 and 2 to linear Gaussian transition and observation models. Namely, the models have the form

$$P(X_{t+1}|X_t) = \mathcal{N}(FX_t, \Sigma_{X,t}))$$
$$P(e_t|X_t) = \mathcal{N}(HX_t, \Sigma_{e,t})$$

respectively, where F and H describe linear transformations of the current state X_t and $\Sigma_{X,t}, \Sigma_{e,t}$ denote the covariance matrices. Similarly, the prior $P(X_0)$ is also a Gaussian whose mean and variance depend on the problem context. Under these assumptions, it can be shown that the distributions $P(X_t|e_{t:1})$ and $P(X_t|e_{t_c:1})$ are Gaussian for all time steps t. Assuming the state distribution X_t at time t obeys $\mathcal{N}(\mu_t, \Sigma_{X,t})$, we can compute the state distribution at the next step by first calculating the optiman *Kalman gain matrix*

$$K_{t+1} = UH^T(HUH^T + \Sigma_{e,t})^{-1}$$

where

$$U = F\Sigma_{X,t}F^T + \Sigma_{X,t}$$

and performing the Gaussian update as follows:

$$\mu_{t+1} = F\mu_t + K_{t+1}(z_{t+1} - HF\mu_t)$$
$$\Sigma_{t+1} = (I - K_{t+1})(F\Sigma_t F^T + \Sigma_{X,t})$$

The *Rauch-Tung-Striebel (RTS)* smoother is an algorithm that lets us recursively obtain smoothed Gaussian position estimates via similar matrix manipulations. We do not cover it here for the lack of space.

The Kalman filter and RTS smoother's significant advantage is that it does all its computations in closed form. This makes them very fast and memory-efficient. At the same time, they only handle linear Gaussian models, which may approximate the process in a given application very poorly. Also, additional work is needed if the application involves discrete state distributions, as Gaussians are continuous.

3 Approach

3.1 Motivation

As pointed out previously, particle filtering (PF), currently used to generate Markovian streams for Lahar, has a major drawback. To accurately represent the distributions, it needs to do a lot of resampling at each time step. For instance, in our deployment scenario (described in the next subsection) the PF uses 50000 particles. As a result, each filtering step currently takes it over 10 seconds [5]— prohibitively slow for real-time data processing. Also, smoothing at that speed is practical only in data archival scenarios. Proposals have been made to decrease the number of particles to 1000, thereby greatly sacrificing accuracy for speed. However, even this change would result in a speedup by a factor of only about 50. The data arrival rate would still be limited to about 5 data points/sec (demands of real-time processing may be much higher). Also, running many instances of PF in parallel on the same machine would still be problematic due to memory requirements.

Employing the Kalman filter (KF) instead of the PF offers solutions to many of the above issues. In KF, performing one filtering (or smoothing) step constitutes only a few manipulations with low-dimensional matrices. This makes KF very fast and its speed independent of the desired approximation accuracy. For the same reason, its memory requirements are very low, allowing tracking of a large number of streams on one machine. The KF renders smoothing feasible in real time; each data point can be used immediately to adjust estimates at past time steps.

Certainly, Gaussians may not approximate arbitrary distributions well, leading to some loss of accuracy. The crucial observation that we make is that Lahar's users are likely to be interested only in top K event query results, ranked by their probability. Therefore, the accuracy of probability values is important only insofar as the "good" results end up high in the ranking. Hence, the approximation we make by using Gaussians may have little or no influence on the quality of the result set returned by Lahar.

Applying the Kalman filter in our scenario faces several challeges. Below, after discussing our Lahar deployment we discuss the each of the following issues and our solutions to them: expressing the process model in terms of linear Gaussians, control of model variance growth, variable-lag smoothing, and discretization of Gaussians.

3.2 Setup

In our scenario, Lahar is used to pose event queries about an RFID data stream gathered in our department building. More concretely, the RFID tag sightings are provided by an RFID ecosystem whose antennas are scattered throughout the building's hallways but not other inner spaces, e.g. offices or labs. The range of each antenna is approximately 3 meters.

Since Lahar's users are interested in information about discrete locations (e.g. hallways, offices, etc), the structure of building's inner space is abstracted into a Voronoi graph. Voronoi graph can be viewed as a "skeleton" of the building. Its nodes are located in offices, office entrances, at hallway intersections, and some other places. Figure 1 serves as an example. Each node is labeled with a type of place in which it is located, and

Figure 1: Two offices and the hallway with Voronoi graph in light blue. The grey boxes are RFID antennas.

the queries to Lahar are formulated in terms of the node IDs and RFID tags IDs. For instance, the state sequence $(In(John, Hallway, t); In(John, Office318, t+1))$ corresponding to our running example query *"When did John enter office 318?"* would be expressed as $(In(48580232, 316521, t); In(48580232, 369383, t+1))$ assuming that 48580232 is the ID of John's tag, 316521 is the ID of the node located at the entrance to office 318, and 369383 is the ID of the node inside office 318.

The data available to us include:

- Voronoi graph in the form of node IDs and node adjacency lists
- GPS coordinates of the Voronoi nodes
- GPS coordinates of the RFID antenna locations
- For each antenna, the endpoints of the Voronoi edge segments that are within 3 meters of the antenna

Every second, the ecosystem provides us with the list of antennas that sighted a particular tag, as in Table 2. Lahar takes as input a Markovian stream of (discrete) distributions over the Voronoi nodes given this evidence.

3.3 Transition and Sensor Models

The transition model currently implemented by the PF postulates that if an RFID tag is at a given Voronoi node at time t, it first picks a direction to move from among the adjacent Voronoi edges uniformly at random. Its position is then translated by 1 meter along the chosen edge to obtain a temporary position. The true new position at time $t+1$ is sampled from a Gaussian with variance of 0.3 that isoriented along the edge and centered at the temporary position, and then "snapped" to the nearest Voronoi node.

Imitating this behavior with a linear Gaussian is nontrivial for two reasons. First, the PF model forces the tag's position to always remain on the Voronoi graph (which is not necessarily realistic but greatly simplifies the problem of assigning the tag's position to the nearest Voronoi node). Second, the transition distribution is very nonlinear and has several peaks, whereas a Gaussian has only one. After analyzing the alternatives, the model we constructed for the KF simply says that the position at time step $t+1$ is sampled from a Gaussian with variance of 1.7 in all directions, centered at the position at time

t. It embodies the intuition that, in the absence of observations, our belief about the tag's position will tend to remain around the tag's current position. The drawback is that such a model spreads the probability mass poorly and in the wrong directions (not just along the Voronoi edges).

The PF observation model is even trickier to emulate with a sinlge linear Gaussian. The PF model's underlying idea is that a tag is detected with 90% probability by each of the antennas that are within 3 meters of the tag and with 10% probability by antennas that are further away. This scheme's salient feature is that the "locations" of the evidence at each time step (i.e., the locations of antenna IDs) are a highly nonlinear function of the tag's true position. Additionally, the antennas are not located on the Voronoi graph, and their positions aren't always very indicative of the tag's position.

In the case of KF, since our transition model spreads the probability mass in all directions, the observation model is our only way to concentrate the tag's predicted position on the Voronoi graph. Therefore, we constructed a non-stationary model (i.e. one that changes with time) as follows. For each antenna that observed the tag at a given time, we retrieve the set of all Voronoi edge segments within 3 meters of that antenna (we remind that this information was given to us as an input). Each segment is characterized by its endpoint coordinates. We then intersect all these sets to find the endpoints of the segments that are in the range of *all* relevant antennas at the given time step. These segments contstitue the part of the Voronoi graph where the tag currenly is with very high probability. We take their endpoints and fit a Gaussian to the latter by computing their mean and covariance. The resulting model is usually sucecssful at accumulating belief around the Voronoi graph.

3.4 Control of Undue Variance Growth

During a time interval in which no evidence is observed, the variance of the marginals over position computed by the Kalman filter keeps increasing due to the transition model. Since our transition model spreads the probability mass in all directions, the variance growth soon becomes too unrealistic, despite being mathematically justified. The PF implementation solves a similar problem by a hack whereby the spread of the probability mass is explicitly blocked after 30 seconds of seeing no evidence. In the spirit of maintaining consistency with the PF, at each time step with no evidence we scale down the covariance matrix of our transition model Gaussian by a factor determined by the following function:

$$f(t) = 1 - \frac{1}{1 + e^{-\frac{t}{3}+6}}$$

Here t represents the amount of time that has passed since the latest observation. Intuitively, this adjustment means that the longer we see no evidence, the more we hypothesize that the tag is not moving at all. Importantly, as t approaches 30, this function drops off as shown in Figure 1, shrinking the transition covariance to 0. Once we see a new piece of evidence, the transition covariance is reset back to its starting value of 1.7.

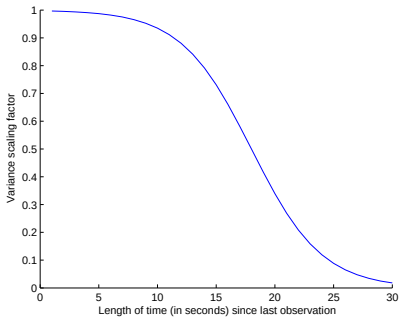


Figure 2: Variance scaling function.

3.5 Smoothing with Variable Lag

Filters are typically used with fixed-lag smoothing, allowing them to correct past state estimates with the current evidence. Unfortunately, the magnitude of lag needs to be determined empirically and is therefore not very robust. Furthermore, different evidence may require different amounts of smoothing. For example, the first piece of evidence after a long period of seeing no observations at all will likely help to greatly reduce variance at all the preceding time steps with no evidence. On the other hand, an observation that arrived only one second after the previous one will probably correct the distributions at only a few previous time steps, and only by a little. We therefore propose a very simple idea. Suppose the current estimate of state at time t is a Gaussian $\mathcal{N}(\mu_t, \Sigma_t)$. Then the RTS smoothing process should continue backwards past time step t only if smoothing yields a Gaussian $\mathcal{N}(\overline{\mu_t}, \overline{\Sigma_t})$ that is significantly different from $\mathcal{N}(\mu_t, \Sigma_t)$. Namely, the estimates should satisfy either $||\overline{\mu_t} - \mu_t|| \geq \epsilon$ or $||\overline{\Sigma_t} - \Sigma_t||_F \geq \delta$ for some ϵ and δ. Implementing this stopping condition requires only a minor change to the RTS algorithm, and we adopted it for our Markovian stream generation process.

3.6 Discretization

Since Gaussians are continuous and Lahar expects state distributions over the the discrete Voronoi nodes, we have devised a procedure to discretize the Gaussians into such distributions. It operates at each time step by iterating over all the Voronoi nodes and selecting the nodes at which the probability density d_i of the Gaussian state estimate is above some threshold fraction (we chose 0.05) of the probability density at that Gaussian's mean. Each "eligible" Voronoi node is then assigned a probability $\frac{d_i}{\sum_j d_j}$ where j ranges over all such Voronoi nodes. If no more than one Voronoi nodes pass the threshold, the threshold is halved and the procedure is repeated from the start (this is done to avoid assigning all of the Gaussian's probability density to just one node). It is easy to see that the sum of probabilities at all nodes computed this way is 1, hence we indeed obtain a valid distribution. We acknowledge that this algorithm is very unsophisticated. In particular, the iteration over all Voronoi nodes could potentially be avoided. However, our experiments show that even with some such inefficiencies our implementation vastly outperforms the PF, so due to time pressure we did not optimize it.

3.7 Generating Correlations

After constucting the discrete distributions over the tag's location at each of a pair of adjacent time steps, we need to generate the correlations between them. I.e., if P_t and P_{t+1} are location distributions at two consecutive time steps, we want to find the transition probabilities $P(n_{t+1}|n_t)$, for each pair of Voronoi nodes n_t, n_{t+1} s.t. $n_t \in Support(P_t)$ and $n_{t+1} \in Support(P_{t+1})$. The PF implementation uses a rather sophisticated technique for this purpose. The technique relies on the transition model being discrete, which is not true for the KF. Due to the lack of time, we could not adapt it to our model. Instead, we designed a different, very simple way of generating the desired probabilities. For all pairs n_t, n_{t+1}, we set $P(n_{t+1}|n_t) = P_{t+1}(n_{t+1})$. It can be verified that this assignment of probabilities yields a valid conditional probability distribution. Its weakness is in the transition model it yields — a model under which, by its definition, the tag's present location is entirely dictated by P_{t+1} and does not depend on the location at the previous time step. While clearly unrealisitic, this is just a stopgap solution to test our main ideas.

3.8 Implementation: Putting It All Together

To test our ideas, we implemented a Kalman filter, our variable-lag smoother based on the RTS smoother, and the discretization procedure in Java, the language of the PF implementation. Writing the Kalman filter and smoother from scratch was necessary because no off-the-shelf Java package we could find would let us both specify a non-stationary observation model and use our variable-length smoothing algorihm. A fair amount of our code generates the transitiion and observation models and performs various data preprocessing tasks, e.g. converting the GPS coordinates of antennas and Voronoi nodes to metric xy-coordinates, and outputs the generated Markovian stream.

4 Experimental Evaluation

4.1 Setup and Demonstration Goals

The goal of our experiments was to demonstrate that

- KF indeed outperforms PF in terms of speed in spite of many implementation inefficiencies.
- Variable-lag smoothing on average saves smoothing steps and therefore time when compared to fixed-lag smoothing.
- The multiple crude approximations and inefficiencies that we allowed don't damage the query performance too much.

During the experiments, we had KF and PF process 700-800s long sections of several data streams with 1-second time steps to compare the processing times. As mentioned previously, the data was supplied by an RFID ecosystem deployed in our building. Each data stream section tracked a person who typically walked along the hallways and occasionally entered offices. When the data streams were processed, we posed event queries of the type *"When did John enter office X?"* and noted the precision and accuracy of Lahar's responses using several metrics.

As a foreword to the results presentation, we report that our approach not only managed to confirm the above three hypotheses but also showed significantly better precision/recall characteristics than PF.

4.2 Timing Experiments

We start with *raison d'etre* of our approach, the speed performance. To measure it, we ran PF, KF with a fixed-lag smoothing, and KF with variable-lag smoothing on sections of several data streams. On each section, PF did smoothing only once, after filtering the entire section. Both flavors of KF did smoothing at every time step with evidence. For KF with fixed-lag smoothing applied to a given stream, the lag was set to the longest contiguous part of the stream during which no evidence had arrived. These "gaps" normally happen when the person enters an office, where the RFID ecosystem has no coverage.

Table 3 summarizes the times it took each of the three methods to process one time step in a given data stream. For KF with fixed-lag smoothing, the number in parentheses denotes the size of the lag. For KF with variable-lag, it specifies the average lag over all time steps. Unfortunately, we could not obtain more precise timings for PF, since we did not have the implementation available to us and had to rely on someone else to run it.

The results clearly show the vast speedup factor of at least 3000 achieved by both flavors of KF over the current PF implementations. Moreover, they demonstrate an approximately 25% speed improvement of variable-lag smoothing over fixed-lag thanks to a noticeably smaller average lag in the former case. In fact, the speed achieved by KF indicates that it is capable of processing the data at the arrival rate of about 300 points per second, more than sufficient to meet the requirements of most real-time applications.

4.3 Query Performance

We pose event queries over data streams of two types: ambiguous and unambiguous. The ambiguous ones trace a tag entering one of the two offices located right across the hallway from each other. Since the RFID ecosystem has no coverage in the offices, the evidence makes it hard to determine which of the two offices the tag actually entered. In unambiguous traces, identifying the office visited by the tag presents less of a problem.

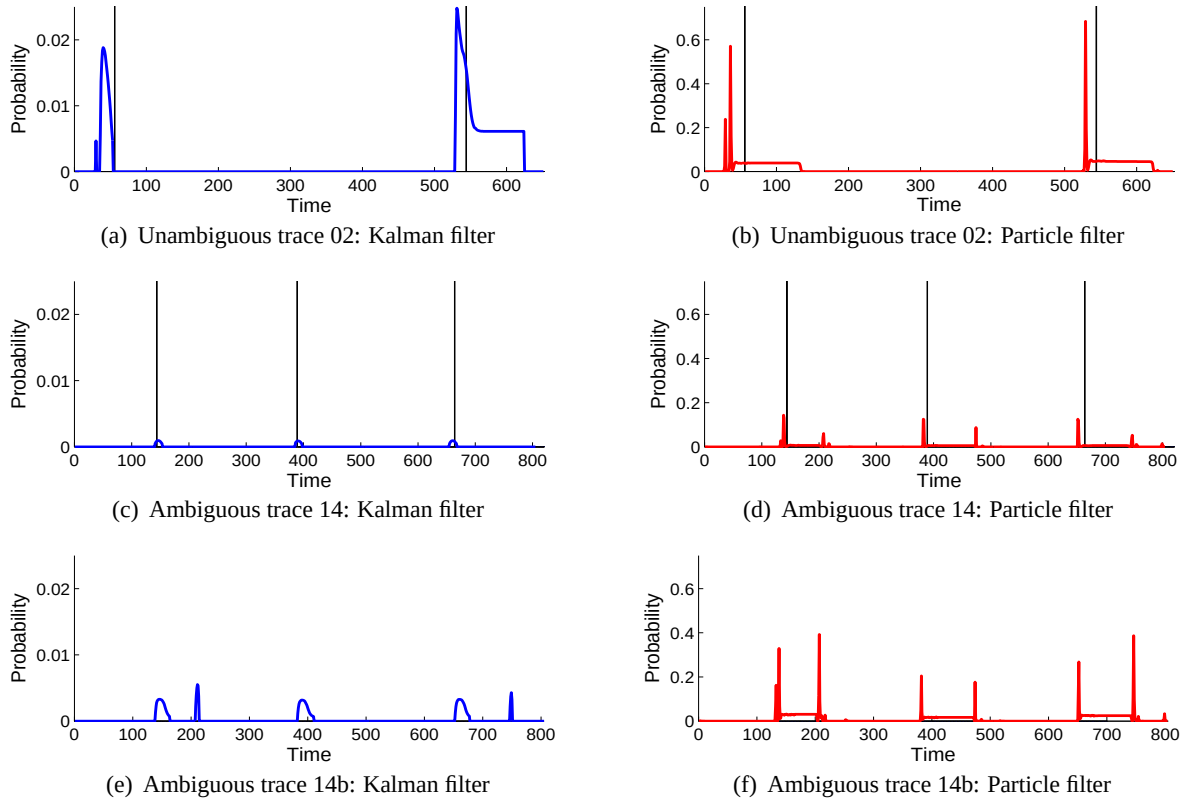
As an example of an unambiguous trace (see Figures 3(c) and 3(d)) Trace 03 involves the subject walking along the hallways of the third floor, and entering room 314 twice, at timesteps 46 and 387. Ambiguous traces are analyzed on two queries: one querying for the room that was actually entered and one for the room across the hall. We refer to the latter query as "Query B", as in Figures 3(e) and 3(f). On all these subfigures, for each time step we plotted the probability of the query event happening, according to the corresponding Markovian stream. The "ground truth" time steps are marked with vertical lines. Note their lack for Query B since its event never actually happened.

One thing to note is that the probability estimates returned by the KF stream tend to be unrealistically low. This inaccuracy comes, as we prdicted, from the apprximations made by the KF and, in particular, from the KF distributing probability mass in all directions instead of along the edges of the Voronoi graph.

Nevertheless, the KF streams have several advantages over those of the PF. The former do not involve as much noise as the latter. In addition, although the PF streams'

	KF+VL smoothing	KF+FL smoothing	PF+smoothing
Stream 2	0.00314(13)	0.00364(100)	> 10
Stream 3	0.00319(13)	0.00433(116)	> 10
Stream 13	0.00363(12)	0.00411(90)	> 10
Stream 14	0.00453(12)	0.00538(91)	> 10

Table 3: Comparative processing speed in sec/datapoint



(a) Unambiguous trace 02: Kalman filter

(b) Unambiguous trace 02: Particle filter

(c) Ambiguous trace 14: Kalman filter

(d) Ambiguous trace 14: Particle filter

(e) Ambiguous trace 14b: Kalman filter

(f) Ambiguous trace 14b: Particle filter

Figure 3: Probability estimates of the Kalman filter stream and the particle filter stream for an unambiguous and an ambiguous trace. Note that in Query B, there is no ground truth because the subject never enters the room of interest.

peaks are both high and narrow (implying a fair degree of certainty), they tend to miss the ground truth completely. The KF stream is more robust in this aspect; while its peak may not correspond with ground truth exactly, the probability estimates corresponding to the ground truth times are significantly higher than its noise level.

On ambiguous traces, both streams returned by the KF and PF suffer from sharp decrease in probability estimates. At first glance, it appears that the estimates returned by the KF are relatively much lower than those returned by the PF (see Figures 3(c) and 3(d)). However, it should be noted that on Query B, the PF stream is also relatively more certain than the KF that the tag entered the room across the hall, whereas in reality it did not. Thus, we suspect that PF streams' tendency to exhibit higher degrees of certainty is a double-edged sword; it serves as an advantage for queries on rooms that the subject actually entered, but also results in a high degree of false certainty for queries on rooms across the hall from the room actually entered.

It is also interesting to note that both the KF and PF streams display higher degrees of certainty on Query B, contrary to what one would expect the optimal inference model to return (i.e. an equal certainty for both queries). We suspect that this is a result of the length of the Voronoi edges in the corresponding offices. The "false" office is much smaller than the one the tag entered. Because the marginal obtained by the KF is essentially a distribution centered in the hallway whose variance increases equally in all directions at each time step with no observations, there will be more probability mass (after discretization) at the center of the smaller office than in the larger office. Similarly, since the PF spreads the probability along the edges of the graph, it will favor the node in the smaller office since it is closer to the point of last observation.

4.4 Threshold Accuracy

The simplest way to eliminate noisy query results involves setting a threshold and returning a positive result

(e.g. declaring the room entered) at any timestep where the stream's query event probability estimate exceeds the threshold. Since the magnitues of the probabilies of various events differ, it makes sense to define the threshold to be not an absolute value but a *fraction* of the maximum probability estimate seen so far for a given query event. The same threshold value can be shared by muplitple streams and queries.

Naturally, it is not clear a-priori what fraction the theshold should be, and setting it may be an error-prone process if the the stream has a lot of high-magnitude noise. Thus measuring precision and recall while varying the threshold fraction from 0 to 100% quantifies how robust and therefore noise-free the given stream is. We conducted such experiments and plotted the results in Figure 4.

It should be noted that the PF streams' precision peaking around a 20%-threshold is a due to Trace 13, in which the narrow spike of the particle filter hits the ground truth exactly. Instances like this are very uncommon for the PF in our experience. Neither precision curves are monotone because the threshold is held constant across queries.

Note that the PF's recall curve witnesses a steeper and faster decline than that of the KF; this is due to the fact that in the PFs stream, ground truth often happens at the noisy 'plateau'. Also, note that, contrary to most recall curves, ours are plotted against an increasing threshold and therefore are monotone non-increasing.

Importantly, the KF streams maintains reasonably high precision and recall for a much wider range of values than PF streams. This indicates a greater robustness of the KF streams to errors in setting the threshold.

4.5 Top K Accuracy

Another way of exhibiting precision-recall characteristics is to evaluate the quality of the top K query results (ranked by probability) returned by Lahar based on Markovian streams generated with KF and PF. Retrieving only top K results models a very natural scenario of Lahar use. In this situation, the users would like to retrieve as few results as possible, but at the same time be sure that the retrieved set contains as many "ground truth" results as possible with very few false positives. In this experiment, we posed a query for each of four different streams, accumulated the results for different values of K and plotted the corresponding precision and recall curves in Figure 5.

As we see, the KF-generated Markovian translates into reasonably high recall for much smaller values of K. Moreover, while the maximum precision of both methods is approximately equal, the KF-generated stream achieves high recall and high precision for approximately the same low value of K. This means that the user can request only a few top results from Lahar without giving up much of either recall or precision, which is not the case with PF-generated stream. This supports our hypothesis about KF's probability estimation inaccuracies not affecting the ordering of the probability values.

5 Related Work

This project extends the Lahar system [7], [8], [10]. However, there are also other systems with similar goals; they take a variety of approaches to the problem of real-time data processing .

For example, the authors of [6] also implement particle filtering (with smoothing) to infer state values given the evidence. Whereas Lahar splits the inference process and the query response into two separate components, [6] merges both processes into one large system. They consider Hidden Markov Model and Kalman filter variants whose transition and sensor models are learned as queries are processed.

Tran et. al. [11] also consider location inference from noisy sensor data, but they additionally consider the context in which the RFID readers are mobile. (In this sense, Lahar is not directly comparable with [11]'s system.) Their system implements a particle filter, enhanced by particle factorization, spatial indexing and belief compression. These modifications address the issue of scalability in the sense that multiple, high-volume streams can be processed simultaneously. For enhanced accuracy over more complex domains, the number of required particles increases, but the authors show that the combination of particle factorization and indexing scaling allows using few particles to achieve an acceptable of accuracy. While their memory requirements are modest for a particle filter, they still do not compare to those required by the Kalman filter.

6 Future Directions

Considering the promising results demonstrated by KF so far, we outline three directions for future research in the area of real-time Markovian stream generation.

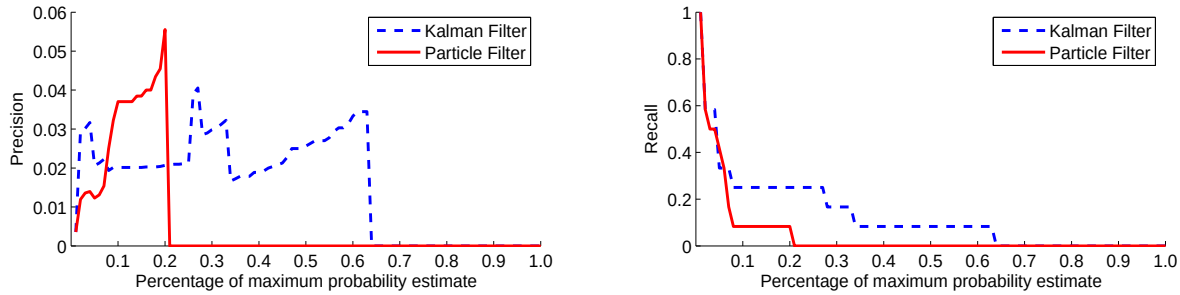
While KF performance is impressive as it is, we believe it can be improved further. Our linear Gaussian formulation of transition and observation models is rather crude. (In fact, it may be worth adopting a Gaussian sum filter a variant of KF that allows state beliefs to be represented with *mixtures* of Gaussians. We have considered this alternative but had to abandon it due to the lack of time.) The pipeline for generating the Markovian stream with the help of KF has a number of inefficiencies, e.g. the discretization step. Last but not least, the implementation is in Java, which is likely not the fastest alternative. Rectifying these deficiencies may result in significant speedups.

On the other hand, the current setup may not be giving PF a full justice. Its implementation also has some inefficiencies. Most importantly, however, the number of particles currently used, 50000, is almost certainly an overshot in terms of accuracy. Optimizations and acceptable sacrifices in accuracy could decrease the number of needed particles down to 5000 or even a 1000, which would in turn bring the speed within the range of real-time processing requirements.

Additionally, the KF and PF are by far not the only alternatives for generating Markovian streams. The process could also be carried out by the Conditional Random Fields (CRFs), for which efficient inference algorithms exist. Their speed would likely fall between that of PF and KF, possibly hitting just the right balance of accuracy and performance.

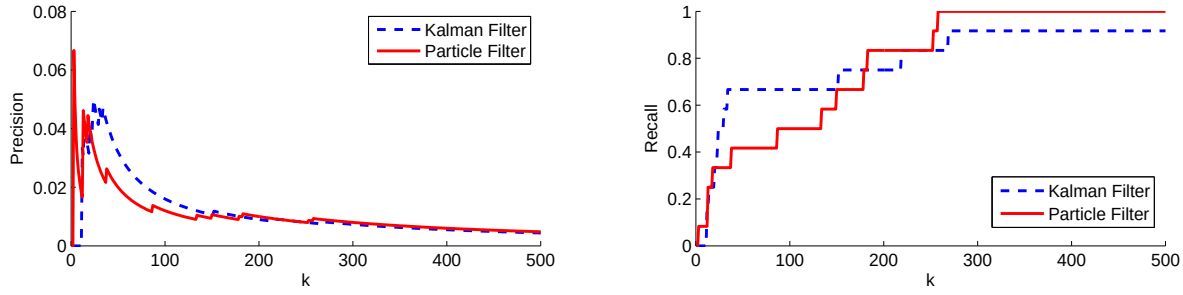
7 Conclusions

Our work has enabled Lahar, a probabilistic event database, to process data in real time by employing the Kalman filter instead of the particle filter for generating Markovian streams. While the Kalman filter sacrifices



(a) Threshold precision (b) Threshold recall

Figure 4: Threshold precision and recall computed over five traces, including Trial 13. In Trial 13, the particle filter stream peaks exactly at ground truth. One hundred uniformly distributed thresholds are chosen between the range of 0 and the maximum probability estimate achieved by the Kalman streams (the particle streams, respectively).



(a) Top-k precision (b) Top-k recall

Figure 5: Precision and recall computed over the top k timesteps, ranked by the estimates of the Kalman filter (particle filter, respectively).

some probability estimation accuracy, the streams it generates manage to improve on the particle filter-generated streams in terms of precision and recall of the top K query results. Most importantly, the Kalman filter proves to be about 3000 times faster than the current particle stream implementation despite performing smoothing at every time step. A variable-length smoothing technique contributes to the success of Kalman filter by letting it do only as many smoothing steps as necessary. We expect the performance of the Kalman filter can be improved further by designing a more accurate process model and optimizing the implementation.

Acknowledgements. We would like to thank Julie Letchner and Chris Ré for experimental data pertaining to particle filtering and in-depth discussions of Lahar.

References

[1] Cormode, G. and Garofalakis, M. Sketching Probabilistic Data Streams. In *Proceedings of the 6th SIGMOD Conference*, 2007.

[2] Dalvi, N. and Suciu, D. Efficient query evaluation on probabilistic databases. in *VLDB*, 2004.

[3] Garofalakis, M.N., Rastogi, R. and Shim. Mining sequential patterns with regular expression constraints. In *IEEE TKDE. Knowl. Data Eng.*, 14(3):530-552, 2002.

[4] Jayram, T.S., Kale, S. and Vee, E. Efficient aggregation algorithms for probabilistic data. In *Proceedings of SODA*, 2007.

[5] Letchner, J. Personal communication, 2009.

[6] Kanagal, B. and Deshpande, A. Online filtering, smoothing and probabilistic modeling of streaming data. Technical Report CS-TR-4867, University of Maryland, May 2007.

[7] Letchner, J., Ré, C., Balazinska, M. and Philipose, M. Access Methods for Markovian Streams. To appear in *Proceedings of the* 25^{th} *International Conference on Data Engineering*, 2009.

[8] Letchner, J., Ré, C., Balazinska, M. and Philipose, M. Challenges for Event Queries over Markovian Streams. In *IEEE Internet Computing, Special Issue on Data Management*, 2008.

[9] University of Washington. RFID Ecosystem. http://rfid.cs.washington.edu/.

[10] Ré, C., Letchner, J., Balazinska, and M., Suciu, D. Event Queries on Correlated Probabilistic Streams. In *Proceedings of the 7th SIGMOD Conference*, 2008.

[11] Tran, T., Sutton, C., Cocci, R., Nie, Y., Diao, Y. and Shenoy, P. Probabilistic inference over RFID streams in mobile environments. In *ICDE*, 2007.