Challenges for Event Queries over Markovian Streams

Building applications on top of sensor data streams is challenging because sensor data is noisy. A model-based view can reduce noise by transforming raw sensor streams into streams of probabilistic state estimates, which smooth out errors and gaps. The authors propose a novel model-based view, the Markovian stream, to represent correlated probabilistic sequences. Applications interested in evaluating event queries — extracting sophisticated state sequences — can improve robustness by querying a Markovian stream view instead of querying raw data directly. The primary challenge is to properly handle the Markovian stream's correlations.

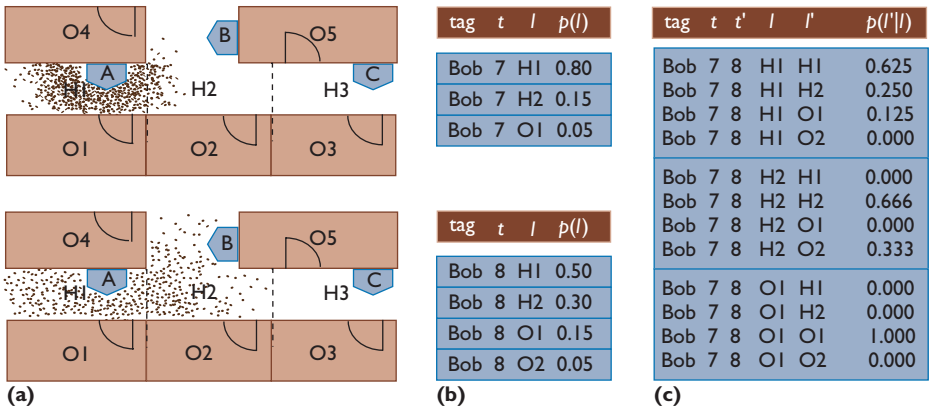
Julie Letchner,
Christopher Ré,
and Magdalena Balazinska
University of Washington

Matthai Philipose
Intel Research

Advances in sensing technologies are making large-scale sensor deployments increasingly common. These deployments transform existing applications, such as supply-chain management,[1] and enable various new ones, such as elder care[2] and activity recognition.[3] Building applications on top of sensor data, however, remains challenging for three reasons. First, sensor data is often inaccurate due to noise, miscalibration, or malfunction. Second, in real-world deployments, sensors are often unevenly distributed and don't cover all areas of interest to applications. Third, sensors often produce data that's difficult to use directly.[3,4] For example, in an activity-recognition context, sensor readings measuring location, acceleration, and so on aren't useful until they're transformed into higher-level state information (that the person wearing the sensor is currently on the bus, for instance).

For these reasons, building applications directly on raw sensor data is untenable in all but the simplest settings. A common alternative uses a model of the monitored environment to transform raw data into probabilistic state estimates.[5,6] A location-tracking model, for example, might take as input sightings of a person's RFID badge and produce as output a sequence of location distributions describing the person's trajectory: she walked down the hallway and then entered either her office or an adjacent lab. These models

 1089-7801/08/$25.00



tag	t	l	p(l)
Bob	7	H1	0.80
Bob	7	H2	0.15
Bob	7	O1	0.05

tag	t	l	p(l)
Bob	8	H1	0.50
Bob	8	H2	0.30
Bob	8	O1	0.15
Bob	8	O2	0.05

tag	t	t'	l	l'	p(l'|l)
Bob	7	8	H1	H1	0.625
Bob	7	8	H1	H2	0.250
Bob	7	8	H1	O1	0.125
Bob	7	8	H1	O2	0.000
Bob	7	8	H2	H1	0.000
Bob	7	8	H2	H2	0.666
Bob	7	8	H2	O1	0.000
Bob	7	8	H2	O2	0.333
Bob	7	8	O1	H1	0.000
Bob	7	8	O1	H2	0.000
Bob	7	8	O1	O1	1.000
Bob	7	8	O1	O2	0.000

Figure 1. Uncertain location estimates and an associated Markovian stream. (a) Particle snapshots show Bob's location distribution at two time steps. (b) We can see the corresponding marginal distributions over Bob's location, as well as (c) correlations between Bob's location at times 7 and 8.

are specified by trained administrators familiar with the sensor deployment rather than by applications, which instead interact only with the model's output (that is, the model-based view). Sensor noise and the inference of high-level information are thus hidden from applications, making them more robust and simpler to write.

The model-based approach to dealing with sensor data has a long history in the artificial intelligence community; however, only recently has the database community adopted this approach.[6–8] Here, we introduce a specific, materialized model-based view called a Markovian stream, used to represent correlated sequences. We discuss our approach to processing event queries on such streams and identify related, open research challenges. We look first at two motivating applications.

Motivating Applications

We highlight the utility of Markovian streams in the context of two applications, both of which are difficult or impossible to write by accessing raw data directly.

Fine-Grained RFID-Based Location Tracking

Commercial[1] and user-centered (http://rfid.cs.washington.edu.) applications increasingly use RFID technology to track the locations of people and objects. Typical deployments attach RFID tags to people or objects of interest. RFID readers placed throughout an environment then detect and log the presence of nearby tags. RFID applications traditionally query tables of raw tag sightings, which are imprecise because readers frequently fail to detect tags and generally don't provide full coverage over all areas of interest.

A model-based view can shield applications from this imprecision by inferring a tag's location using a probabilistic model, even when antennas fail or aren't present. Applications in this case query a view that exposes a sequence of correlated distributions over a tag's location. Figures 1b and 1c illustrate a view in which, at time 7, Bob is either in the hallway (probability 0.95) or in his office (probability 0.05). Bob also moves from the hallway to his office at time 8 with probability 0.125. We'll return to this figure in the next section.

Applications such as friend or equipment trackers, activity "diaries," and so on can query this view directly—without regard to imprecision in the underlying data—to answer queries such as, "Alert me when Bob enters his office after getting coffee," or "When, yesterday, did Bob enter the coffee room?" or "Did Bob attend a lecture in Room 400 within the past week?"

Sensor-Based Activity Recognition

An activity-recognition system for elder care infers a patient's activity (making tea or taking medication, for example) from a suite of environmental sensors (accelerometer, temperature, and so on). Such a system lets caregivers monitor patients in real time and query historical records for behavioral changes that often indicate cognitive decline.[2] Such an application is impossible to write directly on a raw data stream. A model-based view is required to infer from low-level sensor data the (probabilistic, correlated) sequence of activities a patient performs. Health care applications can then query

this view directly to answer questions such as, "Did Sally take her medication yesterday?" and, "Send an alert if Sally leaves her home without first taking her medication."

Markovian Streams

A Markovian stream is a specific type of model-based view derived from the most commonly used temporal graphical model, the hidden Markov model (HMM). HMMs are simple but powerful models that capture the time-dependent relationship between a "hidden" state (for example, a tag's location) and a set of sensor readings (such as the sightings of a tag by an RFID reader). Given a sensor stream as input, inference on an HMM produces two related streams:

- distributions over the hidden state (one distribution per time step) and
- correlations (matrices giving the hidden state transitions at adjacent time steps).

These two elements together define a Markovian stream. In other words, we define a Markovian stream as the output of inference on an HMM. Figures 1b and 1c illustrate an example Markovian stream.

Inferring Markovian Streams

Although Markovian stream generation is independent of any specific inference technique (there are many), for simplicity, here we describe a popular technique called particle filtering.[9] This approach uses samples (called particles) to represent the distribution over a tag's location. At time 7 (Figure 1a, top), antenna A sights Bob, so his location distribution is tightly concentrated. To update Bob's location for time 8, the particle filter first predicts each particle's location at the next time step based on its current location (but independent from the other particles' locations). It then resamples the particles, choosing with a higher probability those whose locations are more consistent with the sensor readings (for example, those within the read range of an antenna that sighted the tag). Because no antennas sight Bob at time 8, his location distribution is correspondingly diffuse (Figure 1a, bottom). The HMM defines both the particle motion and the notion of sensor-location consistency.

To compute a marginal probability distribution over a tag's location at a given time, we divide the number of particles in each discrete location by the total number of particles. Figures 1b (top) and 1b (bottom) show tables representing Bob's location distribution at times 7 and 8, respectively.

We can apply this filtering process in real time to create a Markovian stream from raw sensor readings. However, for applications interested in historical queries, we can apply an additional postprocessing step called *smoothing*[5] to produce a more refined Markovian stream. Suppose antenna C sights Bob at time step 9. Given this sighting, it's highly unlikely in retrospect that Bob was in O1 at time 8. Smoothing revises past distributions (such as Bob's location at time 8) to make them consistent with future sensor readings (sighting of Bob at C at time 9 [not pictured]) — in this case, by decreasing the probability that Bob was in O1 at time 8. For simplicity, the filtered and smoothed marginals in Figure 1 are identical, but in general, smoothing changes marginal values as described.

The smoothing process also extracts a set of correlations between distributions at adjacent time steps, capturing the likelihood of transitions from one state to another. The smoothing process captures these correlations using probabilistic constraints encoded in the HMM. The correlations shown in Figure 1c, for example, reflect the fact that Bob can't walk through walls: the bottom entry in the table states that the probability of Bob entering O2 at time 8 given that he was in O1 at time 7 is zero. Similarly, given that Bob is in H2 at time 7, he's most likely to remain there at time 8 because he's unlikely to abruptly switch direction and head back to O1. Thus, smoothing's overall result is a Markovian stream that contains nontrivial correlations and reflects reality more accurately than a filtered stream alone.

Markovian Stream Trajectories

A Markovian stream represents a distribution over an exponential number of state sequences, each of which must be considered during query processing. Consider an event query ("When did Bob go from his office, O1, to the hallway, H1?" for example) over Bob's location stream in Figures 1b and 1c. If his location distributions at times 7, 8, and 9 include 3, 4, and 5 locations, respectively, then we must evaluate the query on each of Bob's 3 * 4 * 5 = 60 possible paths. We must also compute the probability that Bob took each of these paths. In Figure 1, the path

(H1, H2, O2) has probability 0.8 * 0.25 * x, where x is the conditional probability that Bob went from H2 to O2 at time 9. Although we can easily process Bob's 60 paths, in general, the number of possible state sequences is exponential in the number of time steps in a stream. An hour-long Markovian stream updated once per second contains more than $2^{3,600}$ individual paths; processing an event query by enumerating such paths is prohibitively inefficient.

Event Queries over Markovian Streams

Next, let's look at our approach to processing event queries over Markovian streams. First, we'll outline the class of queries that we address.

Query Space

At the highest level, we can split queries over Markovian streams into two classes: unordered and ordered. Unordered queries don't rely on any ordering among the stream's time steps. This set includes traditional select/project/join/aggregate queries. Examples include, "Where was Bob at noon yesterday?" and "In what location did Bob spend the most time yesterday?"[7] In contrast, we focus on ordered, or event-style queries that search for sophisticated event sequences within a *single* Markovian stream. Thus the query, "When did Bob enter his lab through the north door?" is within our scope, while, "When did Bob enter his lab and put down his mug?" isn't because it joins together two Markovian streams. We identify this class of single-stream, ordered queries in our prior work as *regular* queries.[8] We revisit non-regular queries when we discuss open challenges.

Regular queries are analogous to regular expressions, or, equivalently, *nondeterministic finite automata* (NFA). Figure 2a shows the NFA representation of the query, "Did Bob enter O2 from H2?" Just as a regular expression detects the pattern matches in a string, a regular query detects pattern matches in a temporal state sequence. Intuitively, we label the edges of a regular query NFA with the states that satisfy the edge transition. Any time step at which the NFA reaches a final state is considered a satisfying time step. Thus, the regular query processing task is to compute the probability with which each time step in the (single) Markovian stream satisfies the query. The primary challenge here is the exponential number of state sequences that contribute to this probability at a single time step.

Processing Regular Event Queries

Processing event-style queries on deterministic streams is straightforward using the NFA machinery just described. One simple but intractable way to adapt this machinery to Markovian streams is to separately process each of the exponentially-many deterministic trajectories contained in a single Markovian stream. A more realistic approach is to sample a subset of trajectories, but even this approach is inefficient due to the numerous samples required to achieve reasonable accuracy. Our analytic approach[8] not only produces exact results but also yields an order of magnitude speedup over sampling. Our key observation is that we can combine processing for the exponentially many trajectories in a Markovian stream by tracking a single distribution over sets of NFA states, instead of tracking a separate NFA for each trajectory. Example 1 demonstrates this combined computation using Figure 2 as a reference:

Example 1. Consider the query NFA and Markovian stream shown in Figures 2a and 2b. After processing the marginal distribution representing the stream's first time step (ts7), a 0.15 probability exists that Bob was in H2 (shaded row of the input marginal). All other probability mass remains in the start state. This is reflected in the matrix ts7, which shows a 0.15 probability that the NFA is in state s1 only, with the last-seen symbol being H2. Correlations between ts7 and ts8 define a 0.333 probability that, given that Bob was in H2 at ts7, he moved to O2 in ts8 (shaded row). Thus, the query is satisfied at ts8 with probability 0.15 * 0.333 = 0.05. The probability mass of 0.3 in state s1 at ts8 comes from the summation of the probabilities of the two trajectories that end in H2 at ts8 — namely, (H1, H2) with probability 0.8 * 0.25 = 0.2, and (H2, H2) with probability 0.15 * 0.666 = 0.1.

Experimental validation of our approach on real and synthetic RFID data[8] demonstrates that it can perform simultaneous, real-time event detection on thousands of Markovian streams, each updating once per second, on a single desktop machine. In contrast, a naive sampling approach can handle only several hundred tags. The time our analytic approach requires to process a single update depends on two factors. First, the update time scales exponentially with the query size (the number of states in the query NFA), as is true for standard relational queries. Our reported timing results

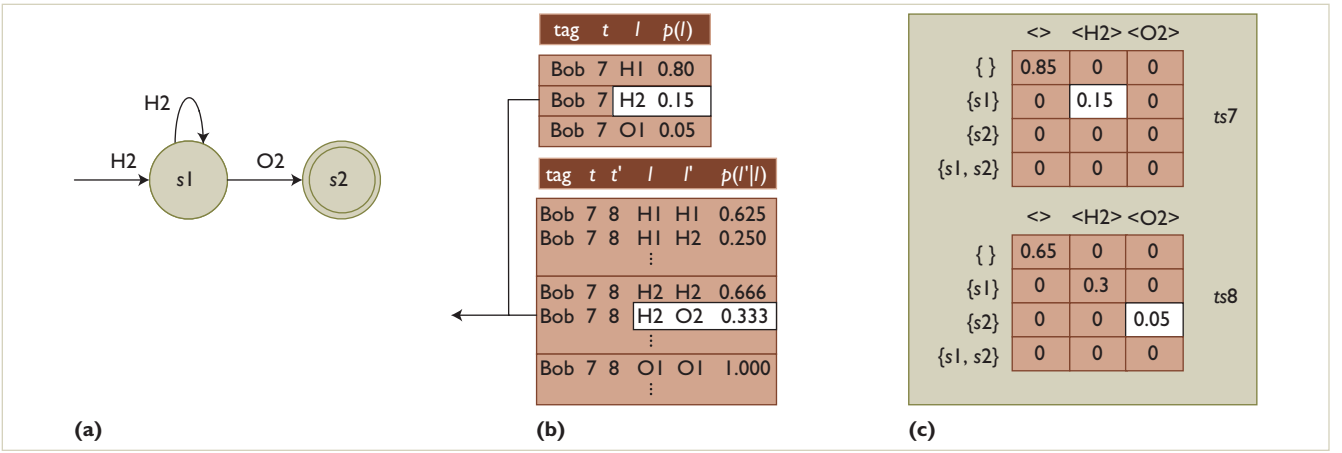


Figure 2. Processing regular event queries. (a) NFA query encoding "Entered O2 from H2." (b) Two time steps of an input Markovian stream. The query processor initializes the NFA with a marginal distribution (here for time 7) and updates it thereafter with conditional distributions. (c) The query-processing state after each input in (b). The state is a joint probability distribution over sets of NFA states (rows) and the latest input symbol (columns).

are on two-state queries. Second, the update time scales linearly with the size of the time steps' marginal distributions; distributions in our test set of location-based streams contained roughly 10 to 20 entries.

In terms of accuracy, queries on the Markovian streams in our experiments show a roughly 20 percent improvement in precision/recall over queries run on a deterministic stream representing the same data (such as the single most likely trajectory in the Markovian stream). This improvement is due largely to the Markovian stream's ability to represent multiple, conflicting trajectories. A deterministic stream, on the other hand, can represent only one possible trajectory, which in uncertain situations is often incorrect.

Challenges

Thus far, we've described a basic algorithm for processing regular event queries on Markovian streams; however, we have identified several research challenges that must be addressed before a complete data management system for Markovian streams is feasible.

Accuracy/Latency Trade-Offs

Recall that smoothing can improve Markovian streams' quality. Unfortunately, real-time applications, such as RFID-based theft detection, can't leverage this technique. Such applications consume time-step data immediately, while a processing system can't apply smoothing until the entire raw input stream is available. Thus, applications requiring low latency can't afford smoothing, whereas applications requiring high accuracy can't afford to run in live settings. In between these extremes lie partial smoothing techniques[10] that apply smoothing to only small, recent stream intervals (as a sliding window, for instance). This lets the newest data receive some smoothing benefits while still becoming available for querying with only a short delay — increased delays allow for increased smoothing quality. Open problems include identification of the ideal trade-off (different for each application) between accuracy and latency, as well as development of algorithms to achieve these trade-offs.

Disk Layout

Historical applications (such as behavioral analysis on personal-activity archives) must read Markovian streams from disk before processing them. In such situations, the organization of the on-disk data can significantly affect query efficiency. Because it's processed sequentially, the data comprising a single Markovian stream is most naturally stored in chronological order. However, a stream's marginal and correlation data might be stored interleaved (as in Figure 3) or separately (all marginal information first, followed by all correlation information). Preliminary experiments using our prototype implementation show that the separated layout can outperform the interleaved layout by roughly a factor of two on location-based Markovian stream queries.[11] This performance is workload-dependent, however, and thus quantitatively evaluating the workloads for which each layout is optimal would be a valuable step toward generalized Markovian stream management.

Indexing

As previously mentioned, NFA-based query processors must read an archived stream's entire history from disk. This is costly and even unnecessary if the processor can use indices to identify the (presumably small) subsets of a stream that are relevant to a particular query. Unlike relational indexing, Markovian stream indexing must properly handle sequential, probabilistic data.

Consider the query in Figure 2. Only times at which Bob has a nonzero probability of being in location O2 or H2 are relevant to this query. Figure 3 shows one possible index for efficiently identifying this relevant set. This B+ tree sorts its entries first by location and then within each location, chronologically. One possible evaluation strategy using this index is to intersect the index entries of each query predicate (in this case, H2 and O2). Here, the intersection of the potential query start times ($t1$, $t2$ matching H2) with the single potential query end time ($t1$ matching O2) reveals, without touching any stream data, that this stream segment can't possibly satisfy the query.

A preliminary evaluation of this indexing approach on location-based Markovian streams demonstrates a query speedup of two to five orders of magnitude.[11] Open challenges include developing more sophisticated indices (over common query subsequences, for instance) and techniques for efficiently and continuously updating such indexes with real-time data. We must also develop cost models and optimizers for query processors that leverage these indexes.

Approximation/Compression

Historical queries over weeks' or years' worth of data will inevitably run slowly, even when leveraging sophisticated indexing techniques. In such cases, the ability to trade time for accuracy is particularly important. Markovian streams' sequential, probabilistic nature identifies a huge space of approximation and compression techniques ripe for exploration in this context. Approximation-based techniques for truncating or pruning probability distributions are potentially applicable, as are algorithms for compressing temporal or otherwise-sequential data. Algorithms that deliver successive sets of results with increasing accuracy are also highly desirable in this context because they let users explicitly control the desired accuracy/latency trade-off.

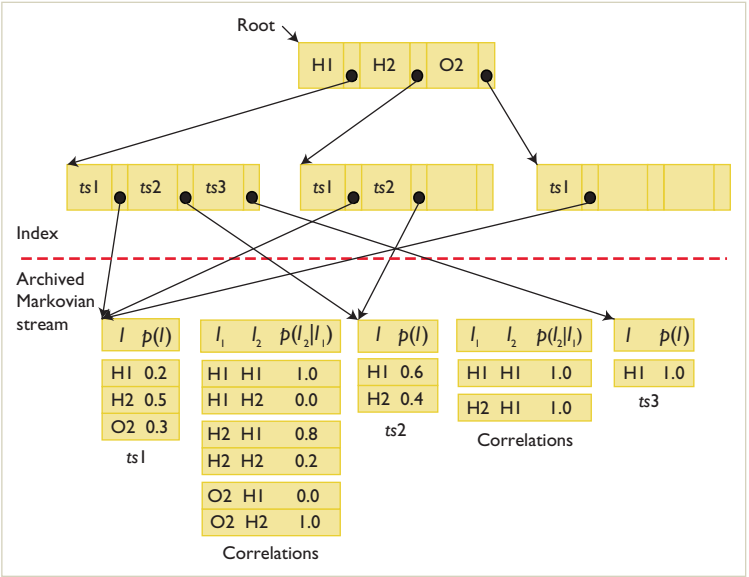


Figure 3. Markovian stream and B+ tree index. This time-step-interleaved layout of a Markovian stream (bottom) is analogous to that introduced in Figure 1. The correlation information for a pair of time steps is stored between the marginal distributions for those time steps. The B-Tree index (top) on (location, time) allows efficient lookup of time steps in which the indexed location has non-zero probability.

More Powerful Queries

Many applications require support for stream queries richer than the regular queries we've discussed. Examples include, "When did Bob and Anna last attend a lecture together?" which requires a join over two Markovian streams, or, "How many students attended Wednesday's lecture?" which requires aggregation over a large set of Markovian streams. Identifying the classes of such queries that can be answered efficiently, languages for specifying these queries, and algorithms for processing them remain open challenges.

We believe that the management of massive archives of uncertain, temporal data is quickly becoming a primary challenge for the database community. Model-based views are a promising approach to dealing with imprecise sensor data, and Markovian streams in particular provide an abstraction that captures the sequential, correlated nature of sensor and other noisy temporal data while still providing efficient support for a large class of event queries.

When we integrate these views into robust query-processing systems, many important challenges arise. Our current work focuses on the development of Markovian stream warehousing techniques (see http://mstreams.cs.washington.edu). Many of the challenges in this work are the same challenges outlined here. By addressing these challenges, we hope not only to produce a practical, robust system for managing Markovian streams but also to advance state-of-the-art techniques for uncertain data modeling and management.

Acknowledgments

This work is partially supported by the US National Science Foundation (NSF) grants IIS-0713123, IIS-0454425, and CRI-0454394. Julie Letchner's work is also supported by an NSF graduate research fellowship.

References

1. F. Wang and P. Liu, "Temporal Management of RFID Data," *Proc. 31st Very Large Database Conf.*, ACM Press, 2005 pp. 1128–1139.
2. T.S. Barger, D.E. Brown, and M. Alwan, "Health-Status Monitoring through Analysis of Behavioral Patterns," *IEEE Trans. Systems, Man, and Cybernetics, Part A*, vol. 35, no. 1, 2005, pp. 22–27.
3. T. Choudhury et al., "Towards Activity Databases: Using Sensors and Statistical Models to Summarize People's Lives," *IEEE Data Eng. Bulletin*, vol. 29, no. 1, 2006, pp. 49–58.
4. L. Liao, D. Fox, and H. Kautz, "Location-Based Activity Recognition Using Relational Markov Networks," *Proc. 19th Int'l Joint Conf. Artificial Intelligence* (IJCAI 05), 2005.
5. L.R. Rabiner, "A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition," *Proc. IEEE*, vol. 77, 1990, pp. 257–286.
6. A. Deshpande and S. Madden, "MauveDB: Supporting Model-Based User Views in Database Systems," *Proc. 2006 ACM SIGMOD Int'l Conf. Management of Data*, ACM Press, 2006, pp. 73–84.
7. B. Kanagal and A. Deshpande, "Online Filtering, Smoothing and Probabilistic Modeling of Streaming Data," *Proc. IEEE 24th Int'l Conf. Data Eng*, IEEE Press, 2008, pp. 1160–1169.
8. C. Re et al., "Event Queries on Correlated Probabilistic Streams," *Proc. 2008 SIGMOD Int'l Conf. Management of Data*, ACM Press, 2008, pp. 715–728.
9. A. Doucet, N. De Freitas, and N. Gordon, eds. *Sequential Monte Carlo Methods in Practice.* Springer-Verlag, 2001.
10. T. Clapp and S. Godsill, "Fixed-Lag Smoothing Using Sequential Importance Sampling," In J.M. Bernardo, J.O. Berger, A.P. Dawid, and A.F.M. Smith, editors, *Bayesian Statistics VI*, Oxford University Press, 1999, pp. 743-752.
11. J. Letchner et al., *Access Methods for Markovian Streams*, tech. report, Univ. of Washington, 2008.

Julie Letchner is a graduate student in the University of Washington's computer science and engineering department. Her current work explores the intersection of machine learning and large-scale data management techniques. Letchner has a BS and MS in computer science from Stanford University with a focus in artificial intelligence and robotics. She is a US National Science Foundation (NSF) graduate research fellow and has received National Defense Science and Engineering (NDSEG), Achievement Rewards for College Students (ARCS), and Google Anita Borg graduate fellowships. Contact her at letchner@cs.washington.edu.

Christopher (Chris) Ré is a graduate student in the University of Washington's computer science and engineering department. His thesis work on probabilistic data management has produced two systems: Mystiq, a system to manage relational probabilistic data, and Lahar, a streaming probabilistic database. Ré has BAs in math and computer science from Cornell University and an MS in computer science from the University of Washington. Contact him at chrisre@cs.washington.edu.

Magdalena Balazinska is an assistant professor in the University of Washington's computer science and engineering department. Her research interests focus on data management systems for distributed data streams, uncertain sensor data, and large-scale scientific data. Balazinska has a PhD in computer science from the Massachusetts Institute of Technology. She is a Microsoft Research New Faculty Fellow and received the Rogel Faculty Support Award and a Microsoft Research Graduate Fellowship. Contact her at magda@cs.washington.edu.

Matthai Philipose leads the Everyday Sensing and Perception (ESP) project at Intel Research, where he builds sensor-based systems that allow computers to understand and act on human state. He has a strong interest in applying such systems to the long-term care of the elderly. To this end, he has collaborated with Intel product groups, universities, and government organizations to build and field-test novel telecare systems in the Pacific Northwest. Philipose has a PhD in computer science from the University of Washington. Contact him at matthai.philipose@intel.com