Approximation Trade-Offs in a Markovian Stream Warehouse: An Empirical Study
UW TR: #UW-CSE-09-07-03

Julie Letchner [#1] Christopher Ré [+2]
Magdalena Balazinska [#3]
Matthai Philipose [*4]

[#]University of Washington, Seattle, WA
{[1]letchner, [3]magda} @cs.washington.edu

[+]University of Wisconsin, Madison, WI
[2]chrisre@cs.wisc.edu

[*]Intel Research, Seattle, WA
[4]matthai.philipose@intel.com

September 24, 2009

Abstract

A large amount of the world's data is both *sequential* and *low-level*. Many applications need to query higher-level information (e.g., words and sentences) that is inferred from these low-level sequences (e.g., raw audio signals) using a model (e.g., a hidden Markov model). This inference process is typically statistical, resulting in high-level sequences that are imprecise. Once archived, these imprecise streams are difficult to query efficiently because of their rich semantics and large volumes, forcing applications to sacrifice either performance or accuracy. There exists little work, however, that characterizes this trade-off space and helps applications make an appropriate choice.

In this paper, we study the effects—on both efficiency and accuracy—of various stream approximations such as ignoring correlations, ignoring low-probability states, or retaining only the single most likely sequence of events. Through experiments on a real-world RFID data set, we identify conditions under which various approximations can improve performance by several orders of magnitude, with only minimal effects on query results. We also identify cases when the full rich semantics are necessary. This study is the first to evaluate the cost vs. quality trade-off of imprecise stream models.

We perform this study using Lahar, a prototype Markovian stream warehouse. A secondary contribution of this paper is the development of query semantics and

algorithms for processing aggregation queries on the output of pattern queries—we develop these queries in order to more fully understand the effects of approximation on a wider set of imprecise stream queries.

1 Introduction

People and computers worldwide generate exabytes of audio, video, text, GPS[1], RFID[2], and other types of multimedia and sensor data—and because disk storage is cheap, most of this data is archived for future use [14]. These information-rich archives are poised to revolutionize data-centric applications in diverse areas including patient and asset tracking in hospitals [33], activity monitoring for elder care [29], scientific environment observation [10], e-Learning [43], phone conversation mining, and multimedia search/retrieval.

While some applications can use raw sensor or multimedia streams directly [12,27], most rely on higher-level streams *inferred* from the low-level data. Search engines, for example, can index audio files by content only after these files have been translated into text. Similarly, location tracking or activity monitoring applications require that raw sensor streams be transformed into location or activity sequences, respectively, before they are processed. Due to noise in the data or ambiguity in the inference process (or both), these inferred, high-level streams are *imprecise* (e.g., a spoken word might be either "eight" or "ate"; while an RFID reading might only narrow a person's location down to one of several adjacent rooms).

The current state of the art for supporting imprecise sequences is the model-based view [11]. A model-based view allows applications to query data as if it were deterministic; internally, however, the DBMS answers the query on the imprecise sequence and returns results annotated with appropriate confidence scores. Model-based sequence views are most commonly used to represent imprecise location streams, typically inferred from GPS [26] or RFID data [20, 24, 32, 39]. They are also used to model environmental statistics (temperature, light levels, *etc.*) inferred from distributed sensor networks [10,18,42], wildlife population counts inferred from sparsely-deployed habitat sensors [19], and structured language information inferred from written text (i.e. information extraction) [20].

A model-based view decouples the queried view from the model used to represent the underlying imprecise sequences, giving a DBMS designer considerable flexibility in choosing an appropriate model. The simplest model, called MAP in the AI literature [35], represents the imprecise stream using only a single deterministic sequence (e.g. the most likely path a person took through a building) [1, 9]. A slightly richer model might represent uncertainty within each individual sequence element (e.g. distributions over a person's uncertain location at each timestep) [35, 38], while an even richer model might additionally represent correlations between these uncertain values (e.g. distributions over entire paths through a building) [19, 24, 32]. An orthogonal design question involves the level of detail at which the chosen model is expressed. In the location domain, for instance, imprecision might reflect uncertain values at the

[1]Global Positioning System
[2]Radio Frequency Identification

level of every room, or might instead model many rooms as the same entity, using a single label (e.g. "Office" or "Lab").

In general, model choice has a significant impact on DBMS quality and performance: increased complexity yields higher accuracy, but incurs additional computational and I/O costs. The appropriate model choice is complicated by the fact that applications generally require support for complex queries, including *event queries* [1,9] (e.g. *"Find all times in May when Bob entered the coffee room."*), and *aggregated event queries* (e.g. *"How many people entered the coffee room each day in May?"*) [32]. Such queries are expensive to compute on sequential, imprecise data, where the utility of scalability techniques like indexing and compression is limited. These high processing costs naturally raise the question of whether rich imprecise sequence models are worthwhile. Would applications notice a difference in result quality if rich, imprecise streams were approximated using simple, deterministic ones? What performance benefits could be gained from such an approximation, which would allow any of several high-performance, deterministic stream processing engines [1,9] to be leveraged? How might a system achieve a flexible trade-off between the accuracy and efficiency of imprecise sequence processing?

In this paper, we address these questions using an empirical study of several common Markovian stream approximations. We show using examples and a brief theoretical analysis that worst-case error bounds on these approximations are too large to be of practical consequence; however, we demonstrate that in practice, errors are often orders of magnitude smaller than these bounds would indicate. We report both performance and accuracy results on real-world location sequences inferred from an office-building RFID deployment. We provide heuristics to guide model choice for various types of query, and finally we generalize our results beyond our application domain by identifying the underlying properties of our data that are responsible for the effects that we measure.

We perform our study using the Lahar Markovian stream warehouse prototype [24, 32]. Lahar natively supports Markovian model-based sequence views, which are the richest of the models used in practice [17,19,22,24,31,32]. With simple modifications, Lahar also supports the approximations that we study here. As part of this study, we additionally augment Lahar to support novel aggregated-event queries. The semantics and processing algorithms for these queries are a secondary contribution of this paper, developed to support our primary goal of studying the effects of approximation on a wide variety of imprecise stream queries.

Through our study, we find that the accuracy/performance trade-off space is rich. A sample of the trade-offs achieved using approximation are shown in Figure 1 (we return to this Figure in Section 5). In our study we identify two models—called independence and MAP—that accelerate performance by 1 and 2 orders of magnitude respectively, incurring typically-low but occasionally-high errors. These models are best suited for performance-critical applications. We identify two additional models—thresholding and rollups—that are best suited for applications that prioritize accuracy. These models accelerate performance only moderately, but consistently return errors that are either zero or so close as to be negligible. Interestingly, we find that richer stream models do not always produce better accuracy than simple models. In particular, the accuracy of the two high-performance models varies significantly based on query characteristics:

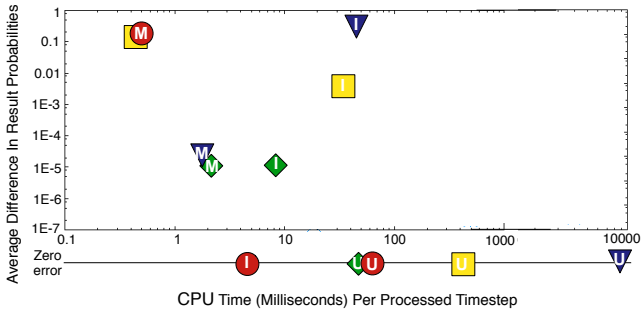

Figure 1: Accuracy/efficiency trade-offs achieved on four queries (identified by shape/color) on three different approximations (identified as 'U' (unapproximated), 'I' (independence), and 'M' (MAP)). Note that the relative accuracies of independence and MAP change depending on the query.

the simpler (and faster) of the two models consistently outperforms the more complex model on some types of aggregate-event query, while on other query types the more complex approximation achieves better accuracy, as one would expect.

The remainder of this paper is structured as follows: In Section 2, we introduce the Markovian stream model and our query models in more detail. In Section 3, we introduce the approximate stream models that form the trade-off space that we study. In Section 4, we briefly overview Lahar, the prototype used to perform our study. In Section 5, we present our study, and in Section 7, we conclude.

2 Data & Query Model

In this section, we present the data model and queries used in our study.

2.1 Data Model: Markovian Stream Warehouses

Our study focuses on an imprecise, sequential data model called a *Markovian stream*. Markovian streams are the most popular type of imprecise stream model [17, 22, 31]; this model or a simplified version has been adopted in nearly all imprecise-sequence management systems [18–20, 24, 32, 38, 39, 42].

The real-world Markovian streams in our study are derived from an office-building RFID deployment in which RFID antennas are mounted in hallways. Figure 2(a) shows a small portion of this office environment, in which RFID readers A, B, and C are highlighted. When a mobile RFID tag (attached to a person or an object) moves within the read range of an antenna, it records the time and presence of the tag using tuples of the form `(Tag_id, Reader_id, Time)`.

The low-level tuples recorded for each tag are used to infer a Markovian stream over the tag's location. This Markovian stream represents a probability distribution over the different paths the tag may have taken through the building. This Markovian stream is *imprecise*, both because of sensor noise (readers often fail to detect nearby tags) and

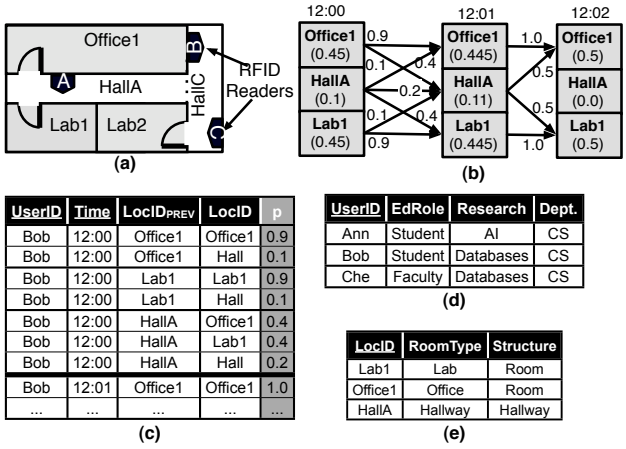

Figure 2: Markovian stream warehouse (RFID/location domain). (a) A schematic view of an RFID deployment. (b) A schematic view of a distribution over location (Markovian stream). The number on an arrow $a \rightarrow b$ indicates the conditional probability $p(b \mid a)$; the number inside each box represents the box's marginal probability. (c) A relational representation of the Markovian stream in (b). (d-e) Dimension tables.

because of *inherent ambiguity* (even on noise-free data it is impossible to distinguish between Office1 and Lab1). The stream in Figure 2(b), for example, states that Bob was in one of three locations at time 12:01, but at 12:02 was in either Office1 or Lab1. The Markovian stream is also *temporally correlated*: the distribution over the tag's location at time $t+1$ depends on its uncertain location at time t. Temporal correlations in Figure 2(b), for example, indicate that if the tag (i.e., Bob) was in some room at time 12:01, then he remained in that same room at time 12:02 with probability 1.0—but there is zero probability that he switched between rooms (Bob cannot teleport). Temporal correlations also encode "soft" constraints, such as the fact that Bob was "probably" (with probability 0.9) in Office1 at 12:01 if he was in Office1 at 12:00.

More precisely, a Markovian stream of length N with uncertain attributes $A_1, \ldots, A_k$ (where A_i has domain D_i for $i \in [1,k]$) is a pair $(p_0, \vec{\mathbb{C}})$. Here $p_0 : (D_1 \times \cdots \times D_k) \rightarrow [0,1]$, such that the elements of p_0 sum to 1.0, is a probability distribution over the initial element of the stream. $\vec{\mathbb{C}}$ is a sequence of *conditional probability tables* (CPTs) $\mathbb{C}^{(t)} : (A'_1, \ldots, A'_k \times A_1, \ldots, A_k) \rightarrow [0,1]$ for $t \in [1, N-1]$, represented as a relation. Each entry in $\mathbb{C}$ uniquely describes the probability of a specific state transition between time t and $t+1$. For example, a tuple $\mathbb{C}^{(t)}(a'_1, \ldots, a'_k | a_1, ..., a_k) = p$ in $\mathbb{C}^{(t)}$ (written using a bar to suggest conditional probability functions) indicates that p is the conditional probability that the state of the stream is $(A_1 = a'_1, \ldots, A_k = a'_k)$ at time $t+1$, given that the state at time t was $(A_1 = a_1, \ldots, A_k = a_k)$. The set of tuples sharing a value of $T = t$ together define the entire stream transition (CPT) between t and $t+1$.

A Markovian stream is thus a compact representation of a probability distribution over an exponential number of deterministic sequences of length N. Markovian streams implement a standard possible worlds semantics in which each stream-length sequence is a distinct possible world [19, 32]. Without loss of generality, consider a Markovian

stream over a single uncertain attribute A with domain D. A path $\vec{x}$ in this stream is an element of D^N. Element x^i of $\vec{x}$ represents the imprecise value at time $t = i$ (e.g. a tag's location at a specific time i). The probability of path $\vec{x}$ is thus $p(\vec{x}) = p_o(x_1) \prod_{i=1...N-1} \mathbb{C}^{(i)}(x_{i+1}|x_i)$.

The relational schema of the location-based Markovian streams used in this study is shown in Figure 2(c). This relation may contain tuples from many different logical streams (i.e. from different RFID tags). Each logical stream is identified by its *key attributes* (`UserID` in the example). Within each logical stream, element ordering is determined by a *sequence attribute* (`Time` in Figure 2(c)). The stream's uncertain domain is represented using *uncertain attributes* (`LocID` in the example). Each uncertain attribute is replicated ($\texttt{LocID}_{PREV}$ is the replication of `LocID` in the example) in order to represent conditional probability distributions (e.g. $p(loc_{t+1}|loc_t)$); whose values are stored in the `p` column.

A Markovian stream warehouse is a Markovian stream relation (Figure 2(c)) and, optionally, a set of *dimension tables* on the Markovian stream attributes (Figures 2(d) and (e)). The Markovian stream relation is analogous to the facts table in a standard OLAP system. In the Lahar system, the uncertainty in a Markovian stream warehouse is hidden from applications, which instead query a deterministic view schema. In order to simplify Lahar's syntax (introduced in Section 4.1), this view schema is also prejoined with all dimension tables. This joined view is of course never materialized. It serves only as a conceptual model for applications to query.

In general, Markovian stream warehouses over any domain share the structure described above. The Markovian stream relation is inferred via *probabilistic inference* [17], regardless of the domain of either the raw input data or the inferred Markovian stream. Any of many well-established inference algorithms can be used to generate Markovian streams; once they are materialized, query processing algorithms are agnostic to the type of inference used to produce them.

2.2 Query Model: Event and Event-OLAP Queries

In this section, we introduce the different types of Markovian stream queries (event and event-OLAP) that we use to study the effects of approximation. We also briefly introduce Lahar's algorithms for evaluating these queries, since familiarity with these algorithms is important for understanding the effects that approximation have on their performance.

2.2.1 Event Queries

At the heart of Markovian stream processing are *event queries*, which search for patterns in a Markovian stream (e.g. *"Find all times when Bob entered the coffee room last week."*). An event query in Lahar is defined as an ordered sequence of *query links*, which are simply NFA states with 1) a single incoming edge and 2) an optional self-loop edge. The link structure and associated edge predicates together define the event query pattern. Examples are shown in Figures 3(a1), (a2), and (a3). Single-link event queries are equivalent to traditional select queries (Figure 3(a1)), which in this context simply select the subset of timesteps that satisfy a given predicate with non-zero

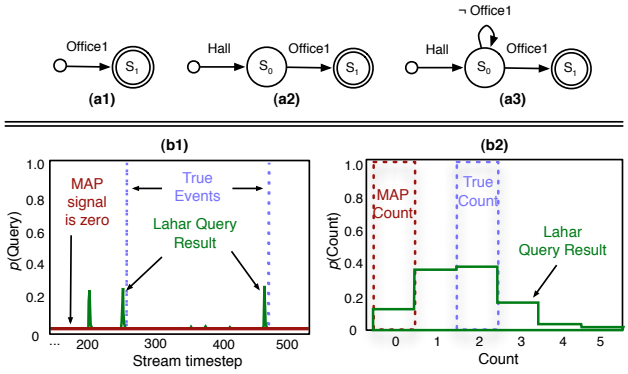

Figure 3: Markovian stream event queries [top] and sample query results [bottom]. (a1) A `SELECT` query on predicate `'Office1'`. (a2-a3) Event queries using `NEXT` (a2) and `BEFORE` (a3) sequence semantics. (b1) Event query output (unaggregated) on a real-world Markovian stream. (b2) Event query output aggregated temporally using `COUNT` semantics.

probability.

An event query result comprises a $\langle seqID, p \rangle$ tuple for each timestep in the stream, indicating the probability p with which the pattern was satisfied at stream element $seqID$. A set of such tuples computed on a real RFID-derived Markovian stream is shown in Figure 3(b1). In this example, the query pattern is correctly detected twice on the Markovian stream, with probability around 0.25. In contrast, the most likely deterministic sequence ("MAP") does not contain any query matches—a fact we explore further in Section 5.

Lahar processes event queries on Markovian streams using an NFA-based algorithm developed in prior art [32]. This algorithm maintains a joint distribution over ⟨ current NFA state(s), last input ⟩. Updating this distribution for a single Markovian stream timestep, to process a query with k links, requires $O(2^k \times D^2)$ time: Each of the 2^k *sets* of possible NFA states (sets instead of individual states because an NFA can be in multiple states simultaneously) must be updated with each of the D^2 entries of the incoming timestep's CPT (correlation matrix). Although D^2 may potentially be large, it is usually tractable: In our study, the number D of discrete office locations was 966. The above algorithm can be easily optimized to operate on only the *active* domain, which is the set of elements in a given timestep that have non-zero probability. This domain changes with each timestep and its size D ranged between 5 and 30 in our study.

2.2.2 Event-OLAP Queries

Many applications additionally require OLAP-style analytics on top of event queries. In order to study the effect of model approximation on aggregated event query results, we introduce here several classes of Markovian stream aggregations inspired by traditional data cubes [13]. These queries have not been previously studied and Lahar's

algorithms for processing them are novel. These aggregations include temporal and cross-stream aggregations; we first introduce temporal aggregation.

Lahar supports two types of temporal aggregation. The first type is `EXISTS`, which determines whether the event query was satisfied at any timestep (e.g. *"Did Bob enter the coffee room at any time last Monday?"*). The second type is `COUNT`, which determines the number of timesteps that satisfied the event query (e.g. *"How many times did Bob enter the coffee room last Monday?"*). Additional temporal aggregation functions (e.g. MIN/MAX) are an area of ongoing work; prior art shows that supporting richer aggregations may require some care [15].

Intuition suggests that temporal aggregation might simply be computed by post-processing event query results. Such a naïve aggregation would be incorrect, however, since event query results produced for different timesteps are correlated. In the Markovian stream in Figure 2(b), for example, the pattern (HallA, Office1) is satisfied at timesteps 12:01 and 12:02, but these matches are mutually exclusive because they depend on Bob being in two different locations at time 12:01.

In order to properly compute temporal aggregations, Lahar maintains the joint distribution over ⟨ current NFA state(s), last input, aggregated-value ⟩. Updating this distribution with a single Markovian stream timestep requires $O(2^k \times D^2 \times A)$ time, where A is the size of the aggregated value's *domain*.

For `EXISTS` aggregation, $A \in \{0, 1\}$; however, for `COUNT` aggregation, A grows linearly in the length of the Markovian input stream (in the worst case). This linear scaling of count domains on imprecise inputs has been noted in prior art and can be avoided in cases where only summary statistics (e.g. the expected value) about the final count are required [19]. For the purposes of this study, Lahar always computes an exact count distribution. Importantly, we show in Section 5 that the performance consequences in practice are usually minor.

The second class of aggregation introduced for this study is cross-stream aggregation. Lahar supports two types of cross-stream aggregation whose semantics—`STREAMEXISTS` and `STREAMSUM`—mirror its `EXISTS` and `COUNT` semantics. However, unlike temporal aggregation, cross-stream aggregation is not complicated by correlations because Lahar assumes independence between different Markovian streams. We refer readers to the full version of this paper for a detailed discussion of cross-stream aggregation [25].

3 Markovian Stream Approximations

In this section we introduce the Markovian stream approximations whose performance and accuracy we study in Section 5. We defer our discussion of error bounds to Section 5.

1) MAP: MAP approximation represents a Markovian stream with its single most likely deterministic sequence (called the **M**aximum **a** **P**osteriori sequence). The MAP estimate is an important approximation because it is the AI community's standard technique for obtaining a deterministic representative of a distribution (e.g. using the Viterbi algorithm [35]). MAP achieves quadratic space and computational savings by reducing the size of a single timestep from D^2 to a single value.

A secondary reason for our inclusion of MAP in this study is the fact that MAP streams can be directly processed by deterministic stream processing systems like SASE [1] or Cayuga [9]. Our characterization in Section 5 of the accuracy/performance trade-off of MAP approximation is thus equivalently a study of the performance/error achieved by applications that determinize their imprecise inputs in order to leverage these deterministic systems.

2) Independence: Independence approximation reduces the Markovian (i.e. first-order) stream to a zero-order stream in which each timestep is independent of the others; that is, independence simply discards the temporal correlations of the Markovian stream. The the probability of a path $\vec{x}$ in an independent stream is thus $p(\vec{x}) = p_o(x_1) \prod_{i=2...N} \mathtt{M}^{(i)}(x_i)$, where marginal distributions $\mathtt{M}^{(i)}$ have replaced the conditional distributions $\mathtt{C}^{(i)}$ of the full Markovian stream. Assuming independence between values that are correlated in reality is a common practice in the AI community. One of the most widely-used classifiers (naïve Bayes) makes independence assumptions which are usually known to be false in the data, but which work well and efficiently in practice [35]. Independence approximation reduces the size of a single timestep from D^2 to D. Our study of the trade-offs achieved using independence sheds light directly on the performance/accuracy of systems that dispense with Markovian correlations and use independent streams as their input [38]. Of the four approximations that we study, independence is the only one that is not *consistent*: it can assign non-zero probability to paths that had zero probability in the original Markovian stream.

3) Thresholding: Thresholding approximation uses a parameter T to prune/discard any temporal correlations (conditional probabilities) v such that $v < T$. The remaining values are normalized. Pruning is used in the AI community—as it is here—to eliminate highly unlikely possible worlds in order to produce a more tractable distribution over the remaining worlds [16]. Clearly, higher values of T produce more aggressively-approximated streams. T values of 0.5 or higher produce deterministic streams in which the most likely element at each timestep has a normalized probability of 1.0. We discuss the trade-offs of various choices of T in Section 5.

4) Rollups: In contrast to the previous approximations, rollup approximations are constructed with an awareness of the semantics of Markovian streams—much in the same way that OLAP aggregations are semantically informed [13]. Rollup approximation is inspired by and similar to "rollup" aggregation in a data cube [13]. Markovian stream rollups represent uncertain domain elements at a coarser level of granularity than the original Markovian stream. For example, consider the uncertain location domain shown in Figure 2(a): {Office1, Lab1, Lab2, HallA, HallC}. One possible rollup on this domain is defined by a "Floor Plan" concept hierarchy that groups all room locations into a single concept and all hallway locations into another. The resulting rollup has the smaller domain: {Room, Hallway}.

Rollups retain temporal correlations, so each timestep requires quadratic space to represent. However, this space is quadratic in the size of a domain of size C which is generally much smaller than the original domain size of D. A rollup over a domain C is equivalent to a Markovian stream produced by inference on a model (e.g. an HMM) defined over C. The resulting rollup is lossless with respect to C, but is *not* lossless with respect to the original Markovian stream over domain D (for an example see the full version of this paper [25]).

4 The Lahar Warehousing System

In this section we briefly introduce Lahar's query syntax and architecture.

4.1 Query Syntax

An example of Lahar's query syntax is shown in Figure 4. In this figure, lines 2-4 identify the stream schema and filtering predicates that are applied to the stream before event query processing begins. These predicates include key-based selection predicates, and temporal windowing predicates, identified by `WITHKEY` and `WINDOW` keywords, respectively.

Every Lahar query contains an event query specification, identified in an `EVENT` clause. In Figure 4, line 5 specifies the structure of the query NFA. Link sequence definitions use either the `NEXT` or `BEFORE` keywords to specify links lacking or containing self-loop edges, respectively. The `WHERE` clause includes additional conjuncts to specify the predicates on each link edge (e.g. lines 6-8 in Figure 4).

Temporal aggregation semantics are specified in Lahar's syntax in the `SELECT` clause (line 1 of Figure 4); alternately, the keyword `INSTANTS` is used to signal that Lahar should not perform temporal aggregation. Cross-stream aggregation is specified syntactically in the optional `GROUP BY` clause (lines 9-10 of Figure 4).

4.2 System Architecture

The flow of data in Lahar is shown in Figure 5(a). Markovian streams are generated outside of the system. Approximate stream views are computed and materialized during loading. At query time, Lahar constructs a query plan (Figure 5(b)) and executes it using the standard getNext() iterator model. Our current optimizer is trivial, a point we revisit in Section 5.

Lahar query plans (Figure 5(b)) include a separate branch for each stream selected by the query. Within each branch, the `Ex` (`Extract`) operator retrieves Markovian stream timesteps from disk. In our study, we refer to `Ex` operator latency and I/O latency interchangeably. Indexing methods to accelerate the `Ex` operator have been

```
1. SELECT <EXISTS | INSTANTS | COUNT>
2. FROM Location L
3.     WITHKEY L.researchArea = 'Databases'
4.     WINDOW L.seqID=12:00 TO L.seqID=16:00
5. EVENT E1 BEFORE E2 NEXT E3
6.     WHERE E1.LocID = 'Room_609'
7.     AND E2.RoomType = 'Hallway'
8.     AND E3.RoomType = 'Office'
9. GROUP BY L.EdRole
10.    USING <STREAMEXISTS | STREAMSUM>
```

Figure 4: Lahar's query syntax. Lines 1-4 specify selection predicates and temporal aggregation semantics, applied before every query processing begins. Lines 5-8 specify the event query. Lines 9-10 specify the stream aggregation semantics and grouping criteria.

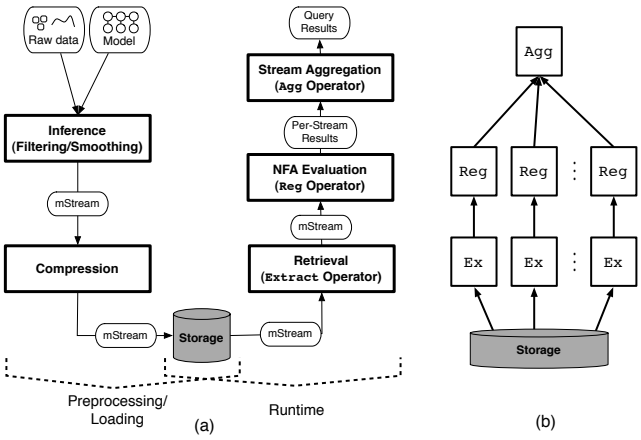

Figure 5: Lahar data flow (a) and query plan (b).

studied in prior art [24]; however, we implement `Ex` as a sequential scan in this study to better isolate the effects of approximation on performance.

Lahar's `Reg` (`RegularExpression`) operator processes the event query on timesteps received from `Ex`, and (optionally) performs temporal aggregation. The result tuples from each branch (stream) are then aggregated together by the root `Agg` (`Aggregation`) operator.

5 Empirical Study

In this section, we study the impact of various approximations on both performance and query accuracy, using real-world data from an RFID-based location tracking domain. In Section 5.2, we study performance and find that event query processing is heavily CPU-bound: approximations accelerate performance in proportion to the amount by which they reduce the complexity of the `Reg` operator, with data size having only a secondary effect. In Section 5.3, we study accuracy and find that threshold approximations can be tuned to achieve effectively zero error, while MAP and independence errors vary considerably based on the type of query. Interestingly, MAP achieves higher accuracy than the richer independence approximation on count-based temporally-aggregated queries. In Section 5.4 we offer practical heuristics for leveraging trade-offs: performance-critical applications can use a combination of MAP and independent models to achieve high performance and low error, while applications that value accuracy highly can use non-aggressive thresholding or rollup approximations to achieve effectively zero error with moderate query acceleration. We begin our study in Section 5.1 with a detailed explanation of our study setup.

5.1 Study Setup

5.1.1 Data

Real-world Markovian streams are critical to our study, which focuses on empirical rather than worst-case accuracy. Our streams are inferred using a particle filter from RFID readings collected by a real deployment [40]. Our deployment includes 160 RFID readers installed in the hallways (and only the hallways) of our 6-story office building. This hallways-only setup is similar to RFID deployments in hospitals, which are becoming increasingly common [28, 30, 33, 41] and which restrict RFID reader placement because of interference with medical equipment. The readers in our deployment record the locations of over 300 RFID tags attached to books, laptop computers, and even to people. From over 6.6 million tag sighting events, we have curated two data sets which we label *unambiguous* and *ambiguous*. Each set comprises five distinct RFID traces manually annotated with detailed ground-truth location information, for a total of 2.2 hours of Markovian streams (sampled at 1Hz).

The traces in both sets reflect a person walking around an office environment like the one shown in Figure 2(a), entering various rooms for 1-minute visits. The Markovian streams inferred from the unambiguous data set contain significant uncertainty, but identify a single most likely room during each in-room interval; in contrast, streams inferred from the ambiguous data set generally identify 2-3 most likely rooms with roughly equal probability. The ambiguous data set thus reflects the inherent ambiguity problem discussed in Section 2.1, where temporal correlations are important because they encode the constraint that objects cannot teleport.

5.1.2 Queries

We evaluate Lahar on a set of queries designed to highlight the strengths and weaknesses of various approximations as well as their sensitivity to Markovian stream ambiguity/correlations. These queries are specified using varying numbers of query links, according to the purpose of each experiment. We chose event query link predicates that search for room-entry events, although this particular choice is not critical to any of the performance or error trends that we study. For simplicity we restrict our study to event queries that use `NEXT` sequence semantics (equivalently, *fixed-length* queries [24]). Although the approximations and aggregations that we study apply equally to queries using `BEFORE` semantics (equivalently, *variable-length* queries), the effects of approximation are in general stronger and more clearly observed on fixed-length queries.

5.1.3 Defining Error

Throughout this study, we measure and present error with respect to a full, unapproximated Markovian stream. For Boolean queries, this error is $e = |p_m - p_a|$, where p_m and p_a are the probabilities with which the query is satisfied in the Markovian and approximate streams, respectively. For count-based queries, the error is defined as the Earth Mover's Distance (EMD) [34] between the count distributions D_m and D_a computed on the Markovian and approximate streams. We chose EMD over other distance metrics

because it is well-defined over distributions with different support, which are common in our measurements.

An alternative definition of error compares query results to the actual underlying sequence of events represented by a Markovian stream. We studied accuracy using both metrics, but restrict our discussion to the definitions above because they isolate approximation-induced error from inference error (even an unapproximated stream contains errors with respect to the actual events it represents). Approximation error is important because it is the only error that a query optimizer can control.

In this section, we measure error at the level of query results because this is the error felt by applications. In Section 5.4, we briefly discuss the effects of approximation on Markovian streams themselves (i.e. independent of any queries).

5.2 Performance Study

In this section we study the following questions about the performance of event and event-OLAP processing:

1. **What are the performance bottlenecks of event and event-OLAP query processing?** We find that NFA processing dominates performance, and has a stronger effect than both disk I/O costs and the theoretically-unbounded costs of aggregation, which we find to be small in practice. The performance of NFA processing degrades exponentially with event query length.

2. **Which approximations yield the best performance, and why?** We find that approximations accelerate performance in proportion to the amount by which they reduce the dimensionality of the state tracked by the `Reg` operator. Thus independence and MAP perform best, accelerating processing by one and two orders of magnitude, respectively, while thresholding and rollup approximations yield only moderate speedups.

We address these two questions in detail below.

5.2.1 Performance Bottlenecks

We identified query processing bottlenecks by measuring the baseline (no approximation) performance of each type of event processing operator on a set of representative queries. These queries include all combinations of four basic event specifications (NFAs), each processed using three temporal aggregation semantics and two cross-stream aggregation semantics, for a total of $4 \times 3 \times 2 = 24$ queries. The four basic event specifications include queries of length 1, 2, 4, and 8 links. The three temporal aggregation semantics and two cross-stream aggregation semantics are of course `EXISTS`, `INSTANTS`, and `COUNT`; and `STREAMEXISTS` and `STREAMSUM`, respectively. For simplicity, we show only the results of the twelve queries processed using `STREAMSUM` cross-stream aggregation semantics. These results appear in Figure 6 (note the logscale y-axis). The performance of the other twelve `STREAMEXISTS`-aggregated queries are identical except that the `Agg` costs are smaller. Each query was aggregated over five

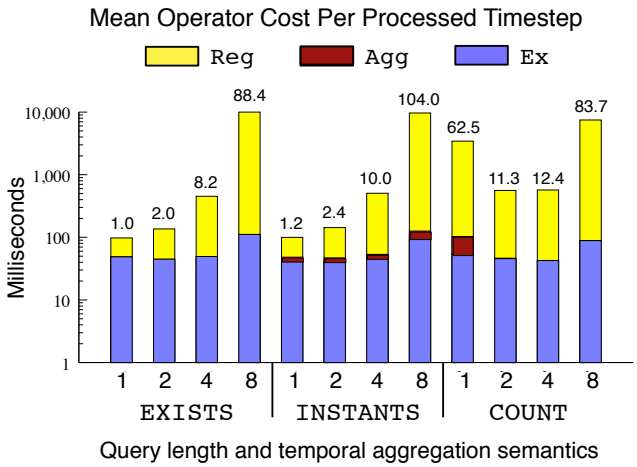

Figure 6: (a) Overview of operator performance on STREAMSUM queries aggregating together 5 streams. Note the logscale y-axis. Numbers above each query bar give the ratio of $\frac{\texttt{Reg}}{\texttt{Ex}}$ latency.

Markovian streams, and Figure 6 reports the average timing results over five separate evaluations of each query.

The key points characterizing the baseline performance of Markovian stream warehousing are as follow:

B1. Reg dominates performance: Figure 6 shows that the Reg operator is consistently the most expensive, sometimes by several orders of magnitude (the ratio between the time spent in the Reg and Ex operators is shown above each bar in the plot). This feature sets Markovian stream processing apart from standard data warehousing where disk I/O costs, rather than CPU costs, would traditionally dominate in an untuned prototype equivalent to Lahar.

B2. Reg slows exponentially with query length: The exponential scaling of the Reg's operator's state matrix with the number of query links, and the direct effect of this matrix size on performance, is clearly visible on the EXISTS and INSTANTS queries in Figure 6. The Reg operator's latency ranges from 50 milliseconds on 1-link queries to nearly ten full seconds per stream timestep for 8-link queries! The exponential trend is obscured in the COUNT queries by the cost of the COUNT aggregation (included in the Reg time), which we discuss shortly. Figure 6 demonstrates clearly that exact processing of 4-link or longer queries is simply not scalable.

B3. Aggregation latency is minimal: Recall from Section 2.2 the incremental update cost of count-based aggregations (COUNT, STREAMSUM) slows with the size of the count domain. This unbounded update cost could potentially grind Lahar to a halt; however, in practice we find that count-based aggregations in our domain scale well.

Figure 6 shows that the latency of the STREAMSUM operator (shown as the Agg cost) is low compared to the latency of the Reg operator. The amortized costs of one-time cross-stream aggregations (i.e. over EXISTS and COUNT results) are all below 25 milliseconds and are not even visible in Figure 6. The single-link COUNT query is

an exception whose slow performance is due to the large STREAMSUM domain, which reaches 1536 elements (the ground truth sum is 711 by comparison). The amortized cost of per-timestep cross-stream aggregations (i.e. over INSTANTS output) is naturally somewhat higher (6-35 milliseconds per timestep), but is still dominated by the cost of the Reg operator.

The cost of count-based temporal aggregation (i.e. COUNT) is included in Figure 6 in the performance of the Reg operator on the four COUNT queries. These queries exhibit a somewhat-constant latency. Short event queries are faster to process but are satisfied relatively frequently, producing larger count domains (511 for the 1-link COUNT query in Figure 6). Longer event queries are slower to process but are satisfied infrequently, meaning that count domains remain small (27 for the 8-link COUNT query in Figure 6). Thus only queries satisfied frequently incur non-trivial COUNT costs. Addressing this problem using approximations—for example, returning the expected value instead of the full count distribution—is an area of ongoing work in Lahar.

5.2.2 Performance on Approximate Streams

In this section we study the performance benefits of approximation, and find that the dimensionality of approximated timesteps is is the primary factor determining the performance benefits of each approximation. The effect of approximation on the imprecise domain has a second-order effect. We develop and illustrate these findings using the twenty-four representative queries described in Section 5.2.1. The *relative* effects of each approximation technique are consistent across all query lengths and all types of aggregation, so for clarity we present results for a single 4-link query that uses no aggregation. Figure 7 shows the latency of this query, averaged over five trials on each of five different Markovian streams. The y-axis of Figure 7 is *not* in logscale, as small timing differences appear more clearly on linear axes.

P1. Reg state dimensionality dominates efficiency: Figure 7 shows clearly that the MAP and independence yield the largest performance gains, of two and one order(s) of magnitude, respectively. Interestingly, these speedups stem almost exclusively from acceleration of the Reg operator. The reduction in Reg latency is a direct result of the fact that MAP and independence reduce the Reg operator's state matrix dimensionality as discussed in Section 3. In contrast, although thresholding accelerates performance proportionally as thresholds increase, threshold approximations do not reduce state dimensionality and thus do not yield the performance benefits achieved by independence or MAP.

The smaller physical size of both independent and MAP streams is only a secondary factor in the higher performance achieved on these views. Smaller timesteps proportionally reduce the cost of the Ex operator; for space reasons we omit graphs showing the data size reduction ratios achieved by each approximation. The dramatic effect of the dimensionality of the Reg operator's state on latency underscores the fact that CPU cost, not disk I/O, is the performance bottleneck of event query processing.

P2. Approximation-aware optimizations are important: In order to reduce the dimensionality of the Reg operator's state, a DBMS must be optimized to leverage the simplified structure of MAP or independence approximations. To underscore this point, consider in Figure 7 the Reg latency on the stream approximated with thresh-

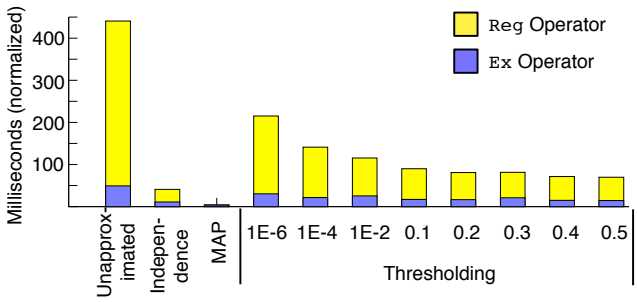

Figure 7: Unaggregated `Reg` and `Ex` operator performance on various approximate stream views. Performance on `EXISTS` and `COUNT` semantics is similar and we omit these plots.

olding using T=0.5. This stream is effectively deterministic. However, Lahar was able to process the independence view—which maintains full uncertainty within each timestep—2.3 times faster than it processed the "deterministic" thresholded stream, despite the fact that the independence view is 1.45 times larger than the thresholded view! Because the determinism of the thresholded view is a side-effect, but not a guaranteed property, of the thresholded view, Lahar must still process it using a 2D state matrix that is an order of magnitude slower to update than the 1D state vector used to process the independence view.

P3. Domain size has a second-order effect: A secondary factor in the performance of the `Reg` operator is the size of the uncertain domain, which affects both the size of the `Reg` operator's state matrix and the size of each Markovian stream timestep. Recall that rollup approximation works precisely by reducing the size of this domain from D to a smaller value C. The performance of `Reg` on two rollups is shown in Figure 8.

The performance numbers in the "4 Link" column of Figure 8 are directly comparable to the `Reg` latencies in Figure 7. Not surprisingly, rollups do not achieve the performance of MAP; in this example, however, they do surpass the average `Reg` latency (26.8 milliseconds per timestep) on independent streams. On highly aggressive rollups like the ones used to generate Figure 8, the comparatively lower cost of C^2 (1 or 16) vs. D (966) can result in superior performance on the rollup stream. In general, however, the order-of-magnitude acceleration achieved by independence is likely to outperform a rollup approximation.

Milliseconds per stream timestep (`INSTANTS` query)

Rollup \ Query Length	1 Link	2 Links	4 Links	8 Links
Original (D=966)	**57.37** ± 43.45	**96.80** ± 68.70	**391.4** ± 298.3	**9924** ± 7356
Rollup View (C=4)	**4.36** ± 1.80	**6.37** ± 3.10	**19.12** ± 10.39	**256.3** ± 203.1
Rollup View (C=1)	**3.60** ± 0.91	**4.86** ± 1.20	**12.64** ± 2.25	**212.7** ± 10.9

Figure 8: Performance of Lahar using two rollup approximations.

5.3 Accuracy Study

In this section we examine the effect in practice of approximation on the accuracy of query results. We omit rollup approximations from this discussion because rollups are lossless with respect to the errors we measure. We continue to omit cross-stream aggregation from our analysis, since `STREAMEXISTS` and `STREAMSUM` errors derive directly from errors in event and temporally-aggregated results, whose errors we characterize shortly. Specifically, in this section we study the following question:

1. **In practice, what are the effects of MAP, independence, and thresholding on the results of event and event-OLAP queries, and what characteristics of Markovian stream data contribute to these effects?** We find that MAP and independence errors vary dramatically based on the type of temporal aggregation used in a query, while thresholding errors increase predictably with the threshold. We find that the deterministic MAP model produces higher-quality results than the richer independence model on some types of aggregation, due to *temporal uncertainty* in our test data.

Our discussion of accuracy is guided by Figure 9, which shows the error incurred by various stream approximations on two types of data (Figure rows) and using different (or no) temporal aggregations (Figure columns). Errors incurred by event queries of different lengths were similar, and for brevity we show only the results on 4-link queries. Figures 9(a-f) plot this error using the metrics described in Section 5.1.3. All plots show errors computed on five Markovian streams.

We now characterize the errors of approximation.

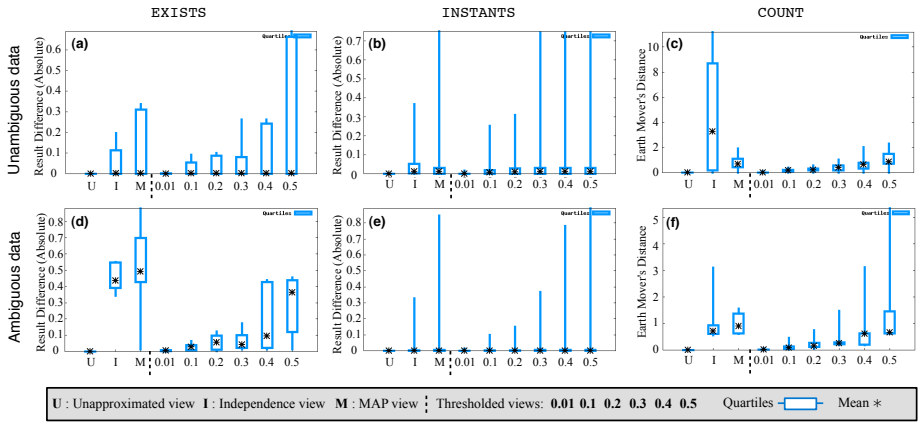

Figure 9: Differences in query results computed on full vs. approximate Markovian streams. Differences are shown for each approximation on a set of five 4-link queries with varying temporal aggregation semantics (Figure columns) and on two trace sets (Figure rows). Differences in result probabilities are reported for `EXISTS` and `INSTANTS` queries, while `COUNT` differences are reported as the Earth Mover's Distance between distributions over the count value.

A1: MAP error is high on `EXISTS` queries, bimodal on `INSTANTS` queries, and low on `COUNT` queries in our location domain. Recall that a MAP stream is deterministic and thus produces event query results with probability 0.0 or 1.0. Figures 3(b1) and (b2) show examples of query results computed on a real-world MAP stream. When the MAP estimate omits a pattern match that is present in the original Markovian stream with probability p, the error incurred by the false negative is p. In our data, p generally ranges between 0.0 and 0.3 with values heavily clustered in the lower end of this range, so MAP errors tend to be small in magnitude. In cases when the MAP estimate *does* include a pattern, however, the query is satisfied with probability 1.0 and the error incurred by the true positive is $1.0 - p$. Thus, although MAP errors are generally low, they can approach 1.0 when the MAP estimate is a poor representative of the original Markovian stream. The bimodal behavior of MAP error on `INSTANTS` queries is visible in Figures 9(b) and (e), where the mean errors are close to zero but the maximum errors are close to 1.0. Contrast this behavior with the theoretical error bound for MAP on an `INSTANTS` query, whose worst-case example stream is shown in Figure 10(a). This bound of $1.0 - (0.5^k)$ quickly approaches 1.0 as the query length k increases. For 4-link queries like those studied in Figure 9, the theoretical bound is $1.0 - 0.0625 = 0.9375$. In our study, MAP error is far lower than this worst-case bound in almost all cases.

MAP streams exhibit very different error behaviors on temporally-aggregated queries. On `EXISTS` queries, which are Boolean, MAP's deterministic result of 0.0 or 1.0 is generally a poor estimate of the true value, which almost always falls somewhere in the middle. MAP returns 1.0 if at least one pattern match occurs anywhere in the MAP estimate, causing it to overestimate `EXISTS` probabilities in our study. We manually verified that the MAP errors in Figures 9(a) and (d) resulted from this type of overestimation. Again, however, even these overestimated results do not approach the theoretical error bound of 1.0, as described in Figure 10(a).

On `COUNT` queries, on the other hand, the single count computed with probability 1.0 on a MAP stream was surprisingly accurate in our study. Although MAP's count is inaccurate in proportion to the number of false positives and false negatives in the MAP stream, it is at least in the right order of magnitude (Figures 9(c) and (f)) in our location domain. In other domains, however, the MAP count could be nearly perfect (e.g. on clearly-spoken audio with little uncertainty) or nearly meaningless (e.g. on noisy location data in which many paths are possibly correct, and MAP has little chance of representing the correct one). The quality of MAP results in general is highly dependent on the fidelity of the MAP estimate to the original Markovian stream, which is dependent in turn on the amount of uncertainty in a data set.

A2: Independence error is high on `COUNT` queries, moderate on `EXISTS` queries, and low on `INSTANTS` queries in our domain. Ignoring temporal correlations in location-based data tends to yield slightly *under*estimated event query probabilities—we manually verified that the errors incurred on the independent streams in Figures 9(b) and (e) were the result of underestimation (note that these two plots include *only non-zero errors*, since the vast majority of timesteps incur zero error and hence obscure the behavior of the remaining errors). In general, these underestimation errors are not large: 3/4 of the non-zero errors on both data sets were of magnitude less than 0.05, while the maximum error was less than 0.4. Contrast these errors with

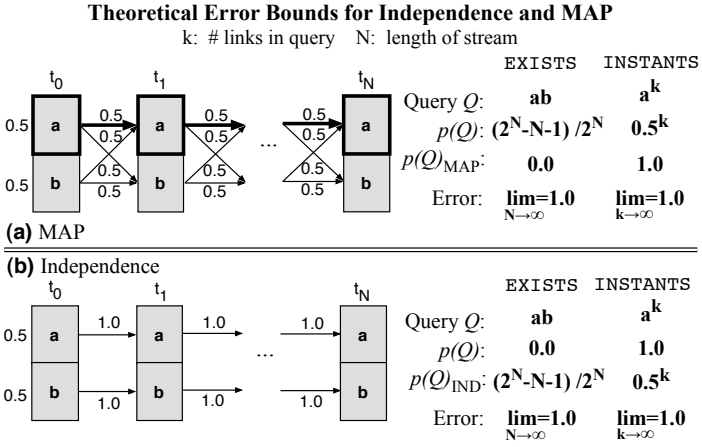

Figure 10: Worst-case Markovian streams and associated theoretical error bounds. *MAP:* (a) shows the worst-case scenario for MAP approximation, in which all paths have equal probability. Assume that the MAP estimate is the path a^N (highlighted). On the original stream, the EXISTS query 'ab' is satisfied with probability approaching 1.0 as $N \rightarrow \infty$, because there exist only $N + 1$ paths that do *not* satisfy 'ab'. These paths are the paths $b^i a^{N-i}$, for $0 \leq i \leq N$. However, the query 'ab' has a probability of zero on the MAP estimate. Similarly, the INSTANTS query 'a^k' (which searches for k consecutive 'a' values) is satisfied with probability 1.0 in the MAP estimate, but has probability 0.5^k (0.0 as $k \rightarrow \infty$) in the original stream. *Independence:* Worst-case independence errors are derived using similar intuition, using the stream shown in (b); its independence approximation is the stream in (a). Note that for both independence and MAP, COUNT error can be generalized from EXISTS, which effectively computes the probability that the count is greater than zero (the true count in these examples is zero). In fact, much worse bounds can be demonstrated for COUNT, but we omit them for brevity.

the theoretical error bound as described in Figure 10(b), which for a four-link query is 0.9375.

In contrast, assuming independence on temporally-aggregated queries tends to yield results that *over*estimate the true values, often significantly—we manually verified that the errors incurred on the independent streams in Figures 9(a), (c), (d), and (f), were overwhelmingly due to overestimation. Even these errors, however, did not approach the theoretical worst-case limit of 1.0, described in Figure 10(b).

The overestimation problem is a direct result of the *temporal uncertainty* in our location data. Temporal uncertainty appears when probabilistic inference cannot determine the precise time at which an event occurred, and thus produces large Markovian stream intervals (30-60 seconds in our data) that contain event pattern matches. Temporal correlations are the only indication of consistent versus inconsistent paths through these intervals. Independent streams lose the distinction between consistent and inconsistent paths, and overestimate the aggregated query result by counting both types in the aggregation.

We expect that other temporally-based application domains (e.g. location inferred from RFID in hospitals or retail stores, or activity sequences inferred from wearable sensors or smart homes) will exhibit similar temporal uncertainty. Applications in these domains will thus observe similar behaviors to the ones identified here, for all queries. In other domains, however, such as audio, streams will contain little temporal uncertainty. We expect that applications in these domains will observe lower errors on independence approximations for temporally-aggregated queries, with `INSTANTS` query errors remaining largely the same.

A3: Thresholding error increases smoothly with the threshold T on all types of query in our domain. Not surprisingly, Figure 9 shows that the error incurred by threshold-based approximations increases with the threshold T. Thus the error incurred by thresholding is controllable in a way that MAP and independence errors are not. In our study, error is negligible for low values of T (e.g. 0.01 and lower). In general, thresholding error is proportional to the amount of probability mass pruned from the original Markovian stream: In Figure 9, for example, thresholding errors are somewhat higher on the "ambiguous" data set because distribution values are lower (a result of the ambiguity). Thus the same threshold T prunes away more information from an "ambiguous" Markovian stream than from an "unambiguous" one.

5.4 Performance/Accuracy Trade-Offs

In this section we synthesize our separate studies of performance and accuracy into a discussion of the trade-offs between the two. We first outline heuristics to help applications choose an appropriate model. Next we discuss the higher-level question of whether the performance costs of full Markovian models are justified by improved accuracy. Finally, we close our study with a brief discussion of the challenges to developing query optimization algorithms to exploit the trade-offs studied in this paper.

5.4.1 Leveraging Tradeoffs in Practice

Thus far we have not addressed the key question for applications, which is: **What model will best satisfy my performance & accuracy needs?** Based on our study results, we offer the following guidelines:

- Applications for which performance is the top priority should use a combination of independence and MAP approximations, which accelerate processing by one and two orders of magnitude, respectively. MAP is preferred for `COUNT` queries because it avoids the overcounting exhibited by independence, while independence is preferred for `INSTANTS` and `EXISTS` queries because it better captures uncertainty over Boolean query results. Event the most aggressive thresholding or rollup-based approximations do not accelerate processing competitively enough for use in a performance-critical application.
- Applications for which accuracy is the top priority should use rollup or threshold approximations, whose errors can be controlled via approximation parameters (e.g. the threshold parameter or rollup concept hierarchy). Threshold approximations can be tuned to provide effectively zero error (e.g. using a threshold of

$T = 0.01$ for our data, as in Figure 9), while still yielding moderate processing speedups (e.g. by a factor of roughly 4, as in Figure 7). Rollup approximations are lossless with respect to the rollup model, and when used aggressively can process scores of timesteps per second 8, achieving performance close to that of independence.

These characterizations are only guidelines, of course, and counterexamples exist even within our study. In light of the above, re-consider Figure 1. The circle, diamond, square, and triangle points correspond to 1-link `INSTANTS`, 1-link `EXISTS`, 4-link `INSTANTS`, and 4-link `EXISTS` queries, respectively. Here, on the 4-link `EXISTS` (triangle) query, independence error is several orders of magnitude greater than the error incurred by MAP. This counterexample underscores the need for more refined, data-specific model optimization, a point that we revisit shortly.

5.4.2 The Markovian Model

The final question addressed by this study, in light of the above analyses, is the following: **Is the complexity of the Markovian stream model justified?** Specifically, is the improved accuracy of full Markovian stream processing, worth the orders-of-magnitude higher processing costs, relative to the lower costs of approximations?

The short answer is clearly 'yes'. In theory as well as in practice, there exist queries (e.g. `EXISTS` aggregations on ambiguous location data, as in Figure 9(d)) for which approximations yield highly inaccurate results. The full Markovian stream model is the model of choice for obtaining accurate answers in these cases. Furthermore, there is no single approximation that *guarantees* high accuracy across all types of data and query—only the full Markovian stream can do this. *In short, any system that does* not *support the Markovian stream model risks serious inaccuracies.*

On the other hand, this study demonstrates equally strongly that many queries can be processed with high accuracy on approximate models (e.g. `INSTANTS` queries using independence or moderate thresholding, as in Figure 9(e)). *Thus, any system that supports* only *the Markovian stream model risks serious and unnecessary inefficiencies*.

In summary, this study demonstrates that the Markovian stream model *is* justified, but DBMS's should take advantage of the many opportunities for optimization that approximations can provide. We demonstrate that the accuracy/performance trade-off space is rich and is indeed worth exploiting.

5.4.3 Toward Query Optimization

The findings in this paper evoke a natural follow-on question, which is how to develop a query optimizer to automatically optimize the trade-off space for a given data set, query load, and performance or accuracy requirement. While we provide general guidelines above, fine-tuned optimization is beyond the scope of this study. A formal cost model for optimization—and identification of the statistics that will allow an optimizer to leverage such a cost model in practice—are important areas for future work.

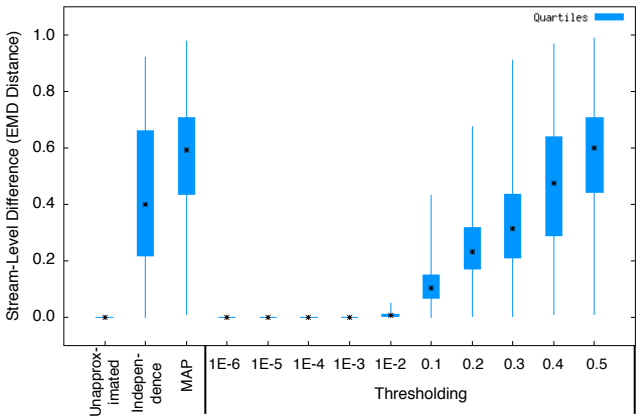

Figure 11: Statistical, query-independent differences between our full Markovian streams and their approximations. Differences are computed for each pair of timesteps separately, using the Earth Mover's Distance. Such quantifications of the difference between a stream and its approximation might yield tighter, data-specific bounds on query errors.

One possible tool for addressing this question is the approximated streams themselves: structural or statistical differences between a Markovian stream and an approximation may yield insights about (or limits on) the query-level errors incurred by the approximation. There are many ways to quantify the differences between two Markovian streams (or approximations). One possible measure is the EMD distance between two streams, viewed as probability distributions over possible sequences. Computation of this distance between two Markovian stream distributions is intractable, but can be approximated using the EMD distance between every k-length substream, in a sliding-window fashion. Figure 11 shows this difference, computed on our RFID/location data for each type of stream approximation, using subsequences of length k=2.

Interestingly, Figure 11 resembles the plots in Figure 9(a-f), even though the data in Figure 11 is independent of any query. The similarity between the plots suggests that stream-level differences might be used to compute tighter, data-specific bounds on query-level errors. Consider a simple example using MAP: if the MAP stream is very similar to the original Markovian stream, then the original stream contains little uncertainty and MAP will produce high-quality results. The quality of the MAP estimate can be computed inside a DBMS, directly on the data. Of course, stream-level differences do not capture the differences in error that result from different types of queries (e.g. different temporal aggregations). Methods for leveraging distribution-level errors in an effort to better characterize query-level error is an area for future work.

6 Related Work

We briefly survey the work synthesized in Lahar; for a more complete related work, please see our technical report [25].

Graphical models and probabilistic inference have long been used in the AI community to reason about uncertainty [7, 17]. Dynamic Bayesian networks [35], of which Hidden Markov Models [31] are a simple example, are a well-established model for temporally-correlated data [19, 24, 32]. The particle filtering [5] and associated smoothing [7,21] algorithms used to generate this study's data are only several of many commonly-used probabilistic inference algorithms that can be applied to these models to generate Markovian streams.

Approximation techniques for probability distributions, including the MAP, independence, and thresholding approximations used in this study, are well-established in the AI literature [16, 35]. Approximation has been applied in deterministic DBMS indexes to create an accuracy/performance trade-off similar in spirit to the one studied here [4].

Model-based views [11] are a popular way to expose uncertain relations to applications. Many systems based on model-based views perform probabilistic inference at query time [19, 36, 37, 39, 42]; Lahar and its predecessors [24, 32] take an alternate approach in which the output of inference (i.e. Markovian streams) are materialized. This eliminates the cost of online inference and allows for materialization of different, approximated versions of each stream.

The OLAP warehousing model [13] is well-studied for deterministic data, but is not directly applicable to imprecise sequences. Recent work focuses separately on cube models for deterministic sequences [27] and for imprecise relations [6]. Kanagal & Deshpande [18, 19] study OLAP-style Markovian stream aggregations similar to those introduced here; however this work does not support aggregations on event queries with self-loops. Much data warehousing work targets RFID data specifically [12, 23]. In contrast, Lahar supports more general types of data such as speech/audio, GPS, *etc.*

Event queries are standard in stream processing. Several systems, including SASE [1] and Cayuga [9], process these queries on deterministic streams. One system by Shen et al. [38] processes event queries on independent, imprecise streams, while still others support these queries on a full Markovian model [19, 20, 24, 32]. Lahar is the only of these systems to exploit the tradeoff space created by the use of different models to approximate the same data.

Probabilistic relational DBMSs handle uncertainty scalably by imposing independence assumptions on the base data. Such systems include Mystiq [8], Trio [2], and MayBMS [3]. These systems support standard SQL, which is richer than Lahar's language, but their data models exclude the ordered, correlated tuples of Markovian streams.

7 Conclusion

In this paper, we studied the performance/accuracy trade-offs of several standard approximations of Markovian streams, in an RFID-based location tracking domain. This study is the first to perform a cost/benefit analysis on imprecise stream models. We found that the trade-off space is rich, affording many opportunities for query acceleration with minimal impact on query error. We characterized query error on event queries and novel event-OLAP queries, including count- and existence-based temporal

and cross-stream aggregations. A detailed analysis of both performance and accuracy revealed that high-performance applications are well-served by a combination of MAP and independence models, while the needs of high-accuracy applications are best met by threshold- or rollup-based model. We generalized beyond our application domain by identifying data characteristics such as temporal uncertainty that are responsible for observed error behaviors.

References

[1] J. Agrawal, Y. Diao, D. Gyllstrom, and N. Immerman. Efficient pattern matching over event streams. In *SIGMOD '08: Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 147–160, New York, NY, USA, 2008. ACM.

[2] P. Agrawal, O. Benjelloun, A. D. Sarma, C. Hayworth, S. U. Nabar, T. Sugihara, and J. Widom. Trio: A system for data, uncertainty, and lineage. In *VLDB*, pages 1151–1154, 2006.

[3] L. Antova, C. Koch, and D. Olteanu. MayBMS: Managing incomplete information with probabilistic world-set decompositions (demonstration). In *ICDE*, 2007.

[4] T. Apaydin, G. Canahuate, H. Ferhatosmanoglu, and A. S. Tosun. Approximate encoding for direct access and query processing over compressed bitmaps. In *VLDB '06: Proceedings of the 32nd international conference on Very large data bases*, pages 846–857. VLDB Endowment, 2006.

[5] S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp. A tutorial on particle filters for on-line non-linear/non-gaussian bayesian tracking. *IEEE Transactions on Signal Processing*, 50(2):174–188, Feb. 2002.

[6] D. Burdick, P. M. Deshpande, T. S. Jayram, R. Ramakrishnan, and S. Vaithyanathan. Olap over uncertain and imprecise data. *VLDB J.*, 16(1):123–144, 2007.

[7] R. G. Cowell, A. P. Dawid, S. L. Lauritzen, and D. J. Spiegelhalter. *Probabilistic Networks and Expert Systems*. Springer, 1999.

[8] N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. In *VLDB*, pages 864–875, 2004.

[9] A. Demers, J. Gehrke, M. Hong, M. Riedewald, and W. White. Towards expressive publish/subscribe systems. In *In Proc. EDBT*, pages 627–644, 2006.

[10] A. Deshpande, C. Guestrin, S. Madden, J. M. Hellerstein, and W. Hong. Model-based approximate querying in sensor networks. *VLDB J.*, 14(4):417–443, 2005.

[11] A. Deshpande and S. Madden. MauveDB: supporting model-based user views in database systems. In *SIGMOD '06: Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pages 73–84, New York, NY, USA, 2006. ACM.

[12] H. Gonzalez, J. Han, and X. Li. Flowcube: Constructing rfid flowcubes for multi-dimensional analysis of commodity flows. In *In: VLDB 2006*, 2006.

[13] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, and H. Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. *Data Min. Knowl. Discov.*, 1(1):29–53, 1997.

[14] IDC. The expanding digital universe: A forecast of worldwide information growth through 2010. An IDC White Paper sponsored by EMC., March 2007.

[15] T. S. Jayram, A. McGregor, S. Muthukrishnan, and E. Vee. Estimating statistical aggregates on probabilistic data streams. In *PODS*, pages 243–252, 2007.

[16] F. Jensen and S. K. Andersen. Approximations in bayesian beleif universes for knowledge-based systems. In *Proc. of the 6th Conference on Uncertainty in Artificial Intelligence*, pages 162–169, 1990.

[17] M. I. Jordan, editor. *Learning in graphical models*. MIT Press, Cambridge, MA, USA, 1999.

[18] B. Kanagal and A. Deshpande. Online filtering, smoothing and probabilistic modeling of streaming data. In *ICDE*, pages 1160–1169, 2008.

[19] B. Kanagal and A. Deshpande. Efficient query evaluation over temporally correlated probabilistic streams. In *ICDE*, pages 1315–1318, 2009.

[20] B. Kanagal and A. Deshpande. Indexing correlated probabilistic databases. In *SIGMOD*, 2009.

[21] M. Klaas, M. Briers, N. de Freitas, A. Doucet, S. Maskell, and D. Lang. Fast particle smoothing: if I had a million particles. In *Proc. of the 23rd ICML*, pages 481–488, New York, NY, USA, 2006. ACM.

[22] J. D. Lafferty, A. McCallum, and F. C. N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *ICML*, pages 282–289, 2001.

[23] C.-H. Lee and C.-W. Chung. Efficient storage scheme and query processing for supply chain management using rfid. In *SIGMOD '08: Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 291–302, New York, NY, USA, 2008. ACM.

[24] J. Letchner, C. Ré, M. Balazinska, and M. Philipose. Access methods for markovian streams. In *25th International Conference on Data Engineering*, 2009.

[25] J. Letchner, C. Ré, M. Balazinska, and M. Philipose. LaharOLAP: Supporting OLAP queries on Markovian streams. Technical Report #CSE-09-03-03, University of Washington, March 2009.

[26] L. Liao, D. J. Patterson, D. Fox, and H. A. Kautz. Learning and inferring transportation routines. *Artif. Intell*, 171(5-6):311–331, 2007.

[27] E. Lo, B. Kao, W.-S. Ho, S. D. Lee, C. K. Chui, and D. W. Cheung. Olap on sequence data. In J. T.-L. Wang, editor, *SIGMOD Conference*, pages 649–660. ACM, 2008.

[28] New Oregon Hospital Adopts IR-RFID Hybrid System. http://www.rfidjournal.com/article/view/4846/1, May 2009.

[29] D. J. Patterson, D. Fox, H. A. Kautz, and M. Philipose. Fine-grained activity recognition by aggregating abstract object usage. In *ISWC*, pages 44–51, 2005.

[30] Philly Hospital Uses RTLS to Track Patient Flow, Care and Training. http://www.rfidjournal.com/article/view/4934/1, May 2009.

[31] L. R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[32] C. Ré, J. Letchner, M. Balazinska, and D. Suciu. Event queries on correlated probabilistic streams. In *SIGMOD Conference*, pages 715–728, 2008.

[33] RFID Journal. Hospital gets ultra-wideband RFID. `http://www.rfidjournal.com/article/view/1088/1/1`, Aug. 2004.

[34] Y. Rubner, C. Tomasi, and L. J. Guibas. A metric for distributions with applications to image databases. In *ICCV*, pages 59–66, 1998.

[35] S. J. Russell and P. Norvig. *Artificial intelligence : a modern approach*. Prentice Hall, 2nd edition, 2003.

[36] P. Sen and A. Deshpande. Representing and querying correlated tuples in probabilistic databases. In *ICDE*, pages 596–605, 2007.

[37] P. Sen, A. Deshpande, and L. Getoor. Exploiting shared correlations in probabilistic databases. *PVLDB*, 1(1):809–820, 2008.

[38] Z. Shen, H. Kawashima, and H. Kitagawa. Probabilistic event stream processing with lineage. In *Proceedings of Data Engineering Workshop*, 2008.

[39] T. Tran, C. Sutton, R. Cocci, Y. Nie, Y. Diao, and P. Shenoy. Probabilistic inference over rfid streams in mobile environments. In *ICDE*, 2009.

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

[41] R. van der Togt, E. J. van Lieshout, R. Hensbroek, E. Beinat, J. M. Binnekade, and P. J. M. Bakker. Electromagnetic interference from RFID inducing potentially hazardous incidents in critical care medical equipment. *Journal of the American Medical Association*, 299:2884–2890, 2008.

[42] D. Z. Wang, E. Michelakis, M. N. Garofalakis, and J. M. Hellerstein. BayesStore: managing large, uncertain data repositories with probabilistic graphical models. *PVLDB*, 1(1):340–351, 2008.

[43] `http://hr.dop.wa.gov/eln/`.