LaharOLAP: Supporting OLAP Queries on Markovian Streams

April 13, 2009

Abstract

A huge amount of the world's data, is both *sequential* and *low-level*; many applications consume 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 streams that are imprecise. Common queries on this data include sequence-finding *event queries* and OLAP-style aggregates of these event queries (e.g. *"How many times do 2008 NPR podcasts use the phrase 'Barack Obama'?"*). These queries are difficult to support efficiently because of the large data volumes and rich semantics of imprecise data.

In this work, we introduce LaharOLAP, a warehousing system for a common type of imprecise, sequential model called a *Markovian stream*. As part of LaharOLAP, we propose semantics for OLAP-style event queries on Markovian streams and algorithms for processing these queries. To address performance, we study and exploit a variety of *lossy* compression techniques, noting that the applications that consume Markovian stream data are already accustomed to imprecision. Through experiments on real data collected from our building-wide RFID deployment, we find that, in practice, lossy compression techniques can yield significant performance gains with only a small loss in accuracy.

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 [23]. These information-rich archives are poised to revolutionize data-centric applications in diverse areas including patient and asset tracking in hospitals [45], activity monitoring for elder care [39], scientific environment observation [13], e-Learning [54], phone conversation mining, and multimedia search/retrieval.

While some applications can use raw sensor or multimedia streams directly [17, 38, 56], most rely on higher-level streams *inferred* from the low-level data. Search engines, for example, can index audio files by content only after they have been translated into text. 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"). An emerging framework for managing imprecise sequences is the *model-based* view [14], which exposes a high-level stream to applications while hiding the details of the raw, low-level data. Applications query such data as if it were precise, and receive query results scored with probabilities. Model-based views have been successfully applied to the management of imprecise location sequences [34, 44, 51] and environmental events [13, 52].

Many applications share a need for sophisticated queries on these model-based views, including sequence queries (e.g. *"Find all occurrences of the phrase 'launchers ready' in 2008 telephone conversations."*) and OLAP-style aggregates on sequence query results (e.g. *"How many of the phone conversations recorded from each household in Washington include the phrase 'launchers ready'?"*). Supporting these queries efficiently on imprecise streams is challenging for two reasons. First, relative to traditional (deterministic) data, imprecise data is extremely slow to process; and second, the scale of sensor and multimedia archives is staggering.

In this paper, we present LaharOLAP, a novel data warehousing system that supports sequence and aggregated-sequence queries on imprecise, sequential, materialized model-based views. Because these already-imprecise input sequences and the applications that use them can often tolerate some degree of additional imprecision, LaharOLAP addresses performance challenges using a battery of lossy compression techniques that allow it to trade accuracy for efficiency.

[1] Global Positioning System

[2] Radio Frequency Identification

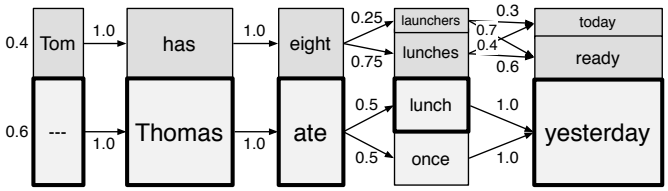


Figure 1: An example of a Markovian stream from an audio/speech domain.

1.1 Motivating Applications

We introduce two applications that we use as running examples throughout the paper:

[RFID-Based Hospital Support] RFID readers are increasingly deployed in hospitals to track the locations of medication, equipment, and even patients carrying RFID tags. This data is used in real-time to locate objects, but can also be queried in a historical context to streamline efficiency (*"How often does the ICU need to borrow a crash cart from neighboring units?"*) or to evaluate quality of care (*"How many times does a surgeon visit with each of his/her patients before performing surgery?"*).

Analysts asking these historical queries reason in terms of the locations that an object visits, (e.g., "Bob is in the ICU"), and not in terms of tag-detection tuples, (e.g., "Bob was sighted by reader 6"). This high-level location information is *inferred* from the low-level RFID stream, but due to noise or ambiguity the resulting location stream contains *uncertainty* about the object's true location (e.g., Bob is either in reception or in the neighboring ICU). This uncertainty is represented as a probability distribution over possible object paths and takes the form shown in Figure 2(b). This distribution contains temporal correlations, since an object's location at each instant is strongly related to its location at the previous instant. LaharOLAP allows analysts to query these uncertain location streams without reasoning explicitly about uncertainty. This abstracted view of location is important in domains beyond hospitals, including supply chains [18, 22] and office environments [55].

[Semantic Audio Search] Speech recognition systems have for decades used probabilistic models to infer the phoneme- and word-level content of raw audio signals [35, 40]. While today's systems work well, they still produce uncertain results even at the word level (though many interfaces hide this uncertainty by returning only the single most likely transcription). These text streams are both uncertain and temporally-correlated, as in Figure 1.

Semantic audio search leverages these uncertain transcriptions to do for audio files what search engines currently do for text—that is, to index/retrieve audio files based on their rich semantic content. Because traditional indexing algorithms are not directly applicable to uncertain text streams, semantic audio search engines must instead manage these streams in a warehouse like LaharOLAP. Semantic audio search is representative of a large set of applications that are made possible by LaharOLAP and transform existing data in rich and novel ways.

1.2 Approach and Contributions

LaharOLAP is a novel warehousing system designed to support the above applications. The input to LaharOLAP is a set of imprecise sequences called *Markovian streams* [28,34,44]. These streams are the result of probabilistic inference on common graphical models such as Hidden Markov Models [40] or chain-structured Conditional Random Fields [31]. They take the form of probabilistic, sequentially-correlated data streams like those illustrated in Figures 1 and 2(b).

LaharOLAP supports SELECT queries, event queries, and *event-OLAP* queries on Markovian streams. Event queries detect occurrences of fixed patterns in a single stream and are standard in deterministic stream processing [12, 57]. Example event queries include, *"Find all occurrences of the phrase 'launchers ready' in telephone conversations recorded in January 2008,"* and *"Find all times when Bob entered his office during the week of May 1."* *Event-OLAP* queries compute simple, OLAP-style aggregate statistics about the occurrences of events across a set of Markovian streams. These queries are important for performing analytics or data mining on a Markovian stream archive. Example aggregate queries include, *"How many people entered the databases lab on March 20, 2009?"* and *"How many times does the word 'launchers' appear across all telephone conversations recorded in 2008?"* LaharOLAP's support for event-OLAP aggregates on imprecise data is novel and a key contribution of this paper; we combine and extend prior work on OLAP for sequence databases [38] and probabilistic streaming algorithms [44] to develop semantics for these queries and algorithms for processing them.

The primary technical challenge of a Markovian stream warehouse is to process event and event-OLAP queries efficiently. This is challenging because 1) query processing algorithms for imprecise/correlated data are complex and slow, and 2)

Markovian stream warehouses are huge. These imprecise archives thus pose unique challenges, but also unique opportunities: Applications that leverage imprecise data are accustomed to imperfect results, and can in many cases tolerate some degree of additional imprecision. This flexibility can be exploited by trading small amounts of error for large performance gains.

A key contribution of this paper is the study of this performance/accuracy trade-off space in the context of Markovian stream warehouses. We study this space using LaharOLAP, which leverages *lossy* compression techniques to improve performance for a small loss in precision. Compression is a successful tool for improving OLAP workloads [1,21,41], although lossless compression techniques perform poorly on Markovian streams, which are dense with probabilities. Lossy compression techniques, however, can achieve good compression *and* exploit performance/accuracy trade-offs. In particular, lossy compression techniques judiciously chosen to reduce the uncertainty in a Markovian stream can simplify query processing algorithms dramatically. We show through experiments on real workloads that by leveraging lossy compression techniques, LaharOLAP can achieve performance improvements of orders of magnitude while producing query results whose probabilities differ from their precise values by, on average, less than 0.01.

In summary, the contributions of this paper are as follow:

- In Sec. 2, we develop a syntax and semantics for event-OLAP queries on Markovian streams, after introducing necessary background.
- In Sec. 3, we discuss LaharOLAP's core technical components: algorithms for event-OLAP queries and lossy compression techniques.
- In Sec. 4, we evaluate the performance and accuracy of LaharOLAP on a real-world, RFID-based Markovian stream archive.
- In Sec. 5, we discuss open problems highlighted by our experience building LaharOLAP and using it to process real-world data.

Exploiting the accuracy/performance tradeoffs of imprecise data using lossy compression is a promising technique for Markovian stream warehouses. Our study of the effects of this compression on query execution is a first step toward enabling such systems.

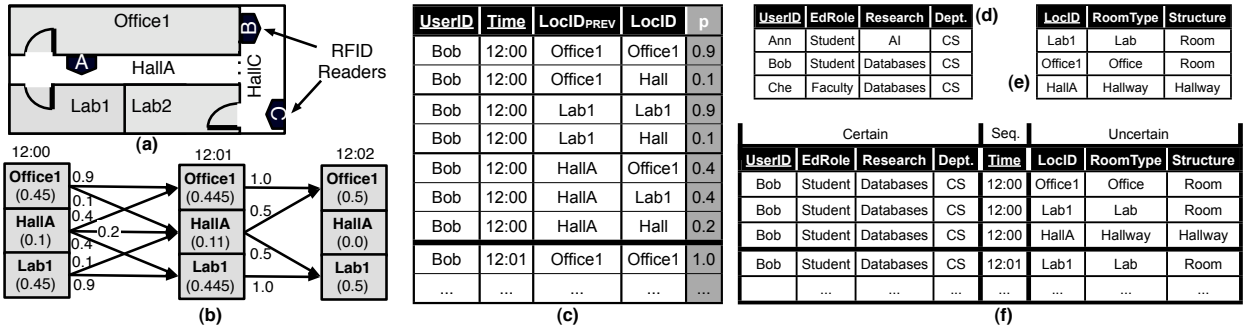
2 Markovian Stream Warehouses

In this section, we present the data model of Markovian stream warehouses (LaharOLAP's input) and develop semantics for OLAP on this model (LaharOLAP's output).

2.1 Data Model

In the building shown in Figure 2(a), RFID readers A, B, and C record the presence of nearby RFID tags using tuples of the form `(Tag_id, Reader_id, Time)`. The low-level tuples from each tag are used to infer a Markovian stream over the tag's location (Figure 2(b)). This Markovian stream is *uncertain*, both because of sensor noise (readers often fail to detect nearby tags) and because of inherent ambiguity (Office1 and Lab1 are *symmetric* with respect to the RFID readers—even on noise-free data it is impossible to tell which of the two rooms a tag is in). This uncertainty is similar to the uncertainty in a hospital deployment, where reader placement is restricted because of its interference with medical equipment. 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 it remained in that same room at time 12:02 with probability 1.0—but there is zero probability that it switched between rooms (Bob cannot teleport). If correlations are ignored, then the probability that Bob teleported between rooms becomes (incorrectly) equal to the probability that he stayed in place.

More precisely, a Markovian stream with uncertain attributes $A_1, \ldots, A_k$ (where A_i has domain D_i for $i \in [1,k]$) is a pair $(p_0, \vec{\mathsf{C}})$. Here p_0 is a distribution over $D_1 \times \cdots \times D_k$, representing the initial probability distribution of the stream; $\vec{\mathsf{C}}$ is a sequence of *conditional probability tables* (CPTs) $\mathsf{C}(A_1, \ldots, A_k, A'_1, \ldots, A'_k, T, P)$. Each tuple in C uniquely describes the probability of a specific state transition between time t and $t+1$. For example, a tuple $(a_1, \ldots, a_k, a'_1, ..., a'_k, t, p)$ in C 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. Markovian streams implement a standard possible worlds semantics in which each stream-length sequence is a distinct possible world [28,44].



UserID	Time	LocID$_{PREV}$	LocID	p
Bob	12:00	Office1	Office1	0.9
Bob	12:00	Office1	Hall	0.1
Bob	12:00	Lab1	Lab1	0.9
Bob	12:00	Lab1	Hall	0.1
Bob	12:00	HallA	Office1	0.4
Bob	12:00	HallA	Lab1	0.4
Bob	12:00	HallA	Hall	0.2
Bob	12:01	Office1	Office1	1.0
...	...	...	...	...

UserID	EdRole	Research	Dept.
Ann	Student	AI	CS
Bob	Student	Databases	CS
Che	Faculty	Databases	CS

LocID	RoomType	Structure
Lab1	Lab	Room
Office1	Office	Room
HallA	Hallway	Hallway

UserID	EdRole	Research	Dept.	Time	LocID	RoomType	Structure
Bob	Student	Databases	CS	12:00	Office1	Office	Room
Bob	Student	Databases	CS	12:00	Lab1	Lab	Room
Bob	Student	Databases	CS	12:00	HallA	Hallway	Hallway
Bob	Student	Databases	CS	12:01	Lab1	Lab	Room
...	...	...	...	...	...	...	...

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. (f) The Markovian stream view exposed by LaharOLAP. This view does not expose probabilities.

The relational schema of a location-based Markovian stream is shown in Figure 2(c). This relation may contain tuples from many different logical streams. 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 store 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 (Figure 2(d) and (e)). The Markovian stream relation is analogous to the facts table in a standard OLAP system.

LaharOLAP exposes to applications a Markovian stream view that hides uncertainty from queries (Figure 2(f)); the power of LaharOLAP is that it gives applications access to complex queries on uncertain data through this simple view. In order to simplify LaharOLAP's syntax, introduced shortly, this view is also pre-joined with all dimension tables. Clearly, this joined view is never materialized (join indexes are used at query time to process queries that leverage the implicit joins [34]).

2.2 Query Model

In this section, we introduce the queries that LaharOLAP supports on Markovian stream warehouses.

2.2.1 Select Queries

LaharOLAP's `SELECT` queries search for single timesteps in a Markovian stream (Q1 in Fig. 6) This is the only type of Markovian stream query on which OLAP-style aggregation has been explored in prior art [28]. An example `SELECT` query is:

```
SELECT INSTANTS
FROM Location WITHKEY 'Bob'
EVENT t WHERE t.locID = 'Office1'
```

This query returns a $\langle seqID, p \rangle$ tuple for each timestep in the stream, indicating the probability p with which the predicate was satisfied at time $seqID$.

A set of such tuples computed on a real RFID-derived Markovian stream is shown in Figure 3(b1). The Markovian stream correctly (within a 5 second window) identifies both times at which the predicate was true, each with probability around 0.25. It additionally yields a false positive around timestep 200. In contrast, the most likely deterministic sequence ("MAP" in Figure 3(b1)) assigns zero probability to all timesteps. The advantage LaharOLAP offers to applications is precisely its ability to leverage the increased recall of the Markovian stream, both on `SELECT` queries and on the more sophisticated event queries that we introduce next.

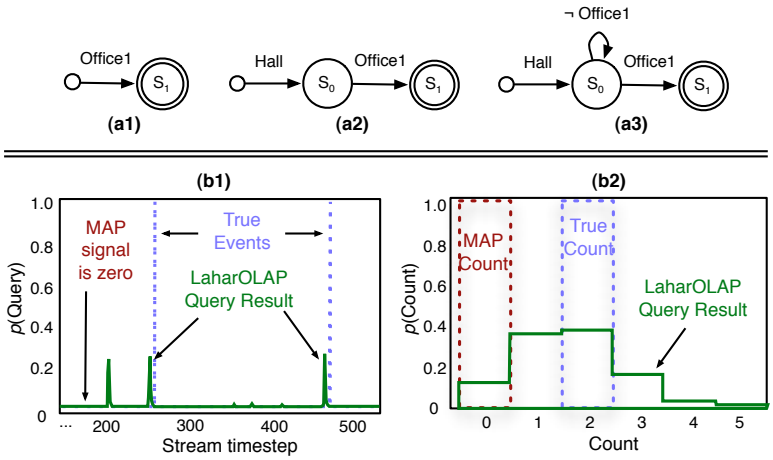


Figure 3: Markovian stream event queries [top] and sample query results [bottom]. (a1) A `SELECT` query on predicate `'Office1'`. (a2-a3) Event queries using `NEXT` (a2) and `BEFORE` (a3) sequence semantics. (b1) Event query output (unaggregated). (b2) Event query output aggregated temporally using `COUNT`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

2.2.2 Event Queries

LaharOLAP also supports previously-studied event queries [44] which search for patterns in a Markovian stream (Q2 or Q3 in Figure 6). A LaharOLAP event query 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 Figure 3(a1), (a2), and (a3). Single-link event queries are equivalent to `SELECT` queries (Figure 3(a1)).

The `EVENT` clause in LaharOLAP's syntax is used to define the structure of the query links. An example LaharOLAP query is shown in Figure 5; we develop this Figure throughout this section. 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 5).

Event queries produce the same $\langle seqID, p \rangle$ tuples as `SELECT` queries; however, event query probabilities are strongly affected by Markovian stream correlations, whereas `SELECT` results are not. To see this, recall the example from Section 2.1 in which assuming independence between timesteps causes Bob to appear to teleport between rooms. On the correlated stream in Figure 2(b), an event query looking for the sequence `Office1 NEXT Lab1` has zero probability in all timesteps; however, if correlations are ignored, then this query is satisfied at 12:01 with probability $0.45 * 0.445 = 0.20025$ and at 12:02 with probability $0.445 * 0.5 = 0.2225$. Thus correlation information is critical for correctly computing event query probabilities. As we will see in Section 3, processing correlations efficiently in support of event queries is a primary challenge of LaharOLAP.

2.2.3 Event-OLAP Queries

Inspired by traditional data cubes [19], LaharOLAP introduces several types of aggregation operators on top of event queries. These aggregations are highlighted in Figure 4 and include aggregations across a stream's uncertain attributes (analogous to relational rollup/drilldown operations and shown in Figure 4(a)), and temporal or cross-stream aggregations on the output of an event query (analogous to aggregations on a sliced/diced relational cuboid and shown in Figure 4(b-d) and Figure 4(e-i), respectively). We examine each type of aggregation and Figure 4 in more detail in this section.

[Rollup Aggregation] Rollup aggregation occurs in any event query whose predicates are potentially satisfied by multiple elements in the Markovian stream's uncertain domain (Q8 in Figure 6). Such predicates are defined by a warehouse's dimension tables and do not alter the format of event query results (Figure 4(a)).

[Temporal Aggregation] LaharOLAP supports two types of temporal aggregation on event query results. The first is `EXISTS`, which determines whether the event query was satisfied at any timestep (Q4 in Figure 6). `EXISTS` semantics compute a single probability from an event query signal (Figure 4(b)). The second type of temporal aggregation is `COUNT`, which determines the number of timesteps that satisfied the event query (Q5 in Figure 6). `COUNT` semantics compute a

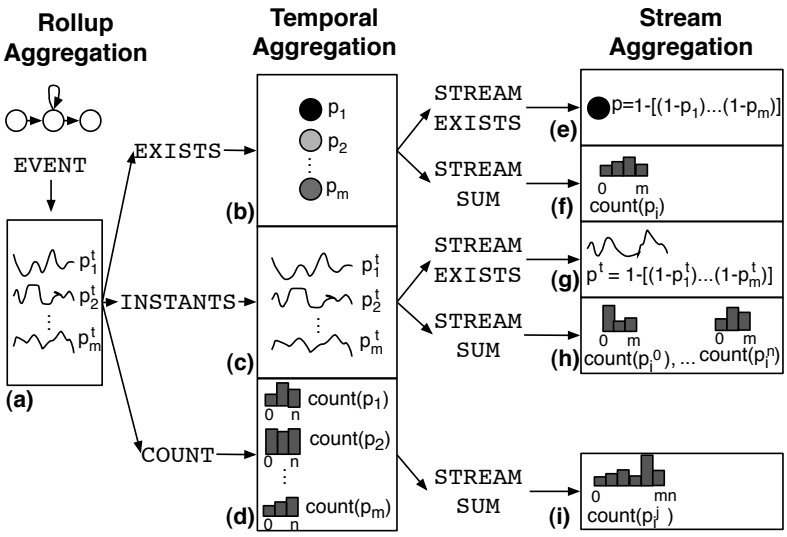


Figure 4: LaharOLAP query semantics. (a): Core event query patterns are defined using NFA-like link sequences. Rollup aggregation is specified using edge predicates. (b-d): Event query results are temporally aggregated using either `EXISTS` or `COUNT` semantics; alternately they are left un-aggregated (`INSTANTS` semantics). (g-k): Results are (optionally) further aggregated across streams using either `STREAMEXISTS` or `STREAMSUM` semantics to produce final query results.

distribution over count values from a query signal (Figure 4(d) and Figure 3(b2)). Temporal aggregation semantics are specified in LaharOLAP's syntax in the `SELECT` clause (see line 1 of Figure 5), or, alternately, the keyword `INSTANTS` is used to signal that LaharOLAP should not perform temporal aggregation (Figure 4(c)).

Temporal aggregation is a challenging operation because the result tuples of an event query are temporally correlated. Correctly computing temporal aggregations is a novel contribution of LaharOLAP which we discuss in more detail in Section 3. LaharOLAP currently supports only existence and count temporal aggregate functions; prior art shows that supporting richer aggregation functions may require some care [25, 43].

[Cross-Stream Aggregation] LaharOLAP supports two types of cross-stream aggregation. These mirror its temporal aggregations and are specified syntactically in the optional `GROUP BY` clause (lines 9-10 of Figure 6). The first is `STREAMEXISTS`, which determines whether any of its (probabilistic) Boolean inputs is true. `STREAMEXISTS` aggregations can be performed either on a per-stream basis, as in, *"Did anyone enter Office1 on Monday?"* (Figure 4(e)); or on a per-timestep basis, as in, ***"At each instant on Monday**, did anyone enter Office1?"* (Figure 4(g)). `STREAMEXISTS` aggregations over the output of temporal `COUNT` aggregations are meaningless and disallowed by LaharOLAP.

LaharOLAP's second type of cross-stream aggregation is `STREAMSUM`, which computes a distribution over the sum of the values of its imprecise inputs. As with `STREAMEXISTS`, `STREAMSUM` aggregations can be performed on a per-stream basis, as in, *"How many people entered Office1 on Monday?"* (Figure 4(f)); or on a per-timestep basis, as in, ***"As each instant on Monday**, how many people entered Office1?"* (Figure 4(h)). Additionally, `STREAMSUM` aggregations can be performed over temporally-aggregated count results, as in, *"How many people entered Office1 on Monday [including repeat visitors]?"* (Figure 4(i)).

```
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 5: LaharOLAP's query syntax. Lines 1-4 specify selection predicates and temporal aggregation semantics. Lines 5-8 specify the event query. Lines 9-10 specify the stream aggregation semantics and grouping criteria.

Audio Processing Domain	QID	Location Domain	Query Semantics
Find all occurreces of the word `economy' in Monday's NPR newscast.	**Q1**	*Find all instants on Monday when Bob was in Office1.*	SELECT
Find all occurrences of the phrase `economic downturn' in Monday's NPR newscast.	**Q2**	*Find all instants when Bob entered Office1 on Monday.*	EVENT-NEXT/ INSTANTS
Find all occurrences of the word `Obama' that appear any time after the word `economy' ...	**Q3**	*Find all times on Monday when Bob was in Office1 after visiting Lab1.*	EVENT-BEFORE/ INSTANTS
Does the phrase `economic downturn' appear in Monday's NPR newscast?	**Q4**	*Did Bob enter Office1 at all on Monday?*	EVENT-NEXT/ EXISTS
How many times does the phrase `economic downturn' appear in Monday's NPR newscast?	**Q5**	*How many times did Bob enter Office1 on Monday?*	EVENT-NEXT/ COUNT
Does any Monday newscast [from any network] contain the phrase `economic downturn'?	**Q6**	*Did anyone enter Office1 on Monday?*	EVENT-NEXT/ EXISTS/ STREAMEXISTS
How many different newscasts on Monday contained the phrase `economic downturn'?	**Q7**	*How many different people entered Office1 on Monday?*	EVENT-NEXT/ EXISTS/ STREAMSUM
*Find all occurrences of **any** economic buzzwords in Monday's NPR newscast.*	**Q8**	*Find all instants when Bob entered **any** office on Monday.*	EVENT-NEXT/ INSTANTS [+Rollup Aggregation]

Figure 6: Sample event-OLAP queries from an audio processing and a location tracking domain.

Although STREAMSUM aggregation is strictly richer than STREAMEXISTS—any STREAMEXISTS query can be answered from STREAMSUM results—we will see in Section 4 that the STREAMEXISTS semantics has performance advantages over STREAMSUM (as does EXISTS over COUNT).

3 The LaharOLAP System

The flow of data in LaharOLAP is shown in Figure 7(a). Markovian streams are generated outside of the system. Compressed stream views are computed and materialized (Section 3.2) during loading. At query time, LaharOLAP constructs a query plan (Section 3.1, Figure 7(b)) and executes it using the standard getNext() iterator model. Our current optimizer is trivial, a point we return to in Section 5. In this section we first discuss LaharOLAP's algorithms for processing precise (uncompressed) Markovian stream views. We then introduce the approximation techniques used to accelerate this processing.

3.1 Precise Query Processing

LaharOLAP query plans include a separate branch for each stream selected by the query (Figure 7)(b). Within each branch, the Ex (Extract) operator retrieves and decompresses Markovian stream timesteps. The Reg (RegularExpression) operator computes the event query on these timesteps and (optionally) performs temporal aggregation. The result tuples from each branch are then aggregated together by the root Agg (Aggregation) operator. We discuss each of these operators in turn.

3.1.1 Ex Operator

The Ex operator retrieves and decompresses Markovian stream timesteps. In our prior art we have studied optimization of the Ex operator using indexes [34]; however in this paper we implement Ex as a straightforward stream scan.

3.1.2 Reg Operator

The Reg operator processes an event query on a single Markovian stream and optionally performs temporal aggregation. The basic Reg algorithm (INSTANTS semantics) is prior art [44]. Here we briefly review this algorithm and introduce novel variations that compute EXISTS and COUNT aggregations in the presence of temporal correlations.

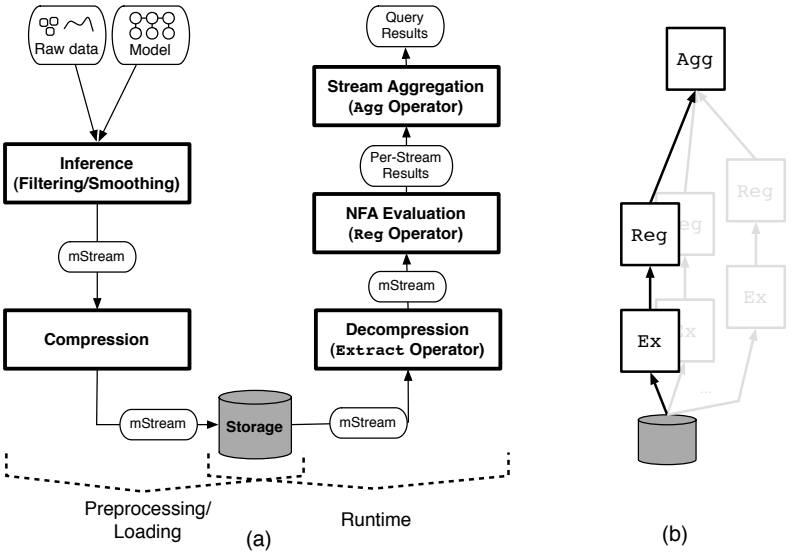


Figure 7: LaharOLAP data flow (a) and query plan (b).

[INSTANTS] The basic Reg algorithm uses standard NFA machinery adapted to handle uncertain, correlated input. This algorithm maintains a 2D state matrix M with a row for each possible *set* of NFA states (sets instead of individual states because the NFA is non-deterministic and can be in multiple states at once) and a column for each element in the Markovian stream's uncertain domain. Entry $M(i, j)$ represents the joint probability that the NFA is in the i^{th} state set *and* that the true value of the last input was the j^{th} element of the uncertain domain (e.g. Office1, Lab1, *etc.*). The entries of M sum to 1.0.

The Reg operator computes its updated state at each timestep into a temporary matrix M', which is then copied back into M before the next update. M' is computed as follows: For each entry $M(i, j)$ (with value p_{ij}) of the state matrix, and for each possible uncertain value j' of the current uncertain input, Reg computes two values. First, it determines the state set i' that results from a transition out of the NFA states in set i on input value j'. This value depends on the NFA structure and predicates. Second, Reg determines the probability $p(j'|j)$: the probability of input j' conditioned on the previous input being j. This probability can be read directly from the temporal correlations in the input. The Reg operator then adds probability p to the value of $M'(i', j')$, where $p = M(i, j) * p(j'|j)$.

On each input, computation of M' requires $N \times D \times D$ multiplications, where N is the number of NFA state sets and D is the size of the Markovian stream's uncertain domain. This cost is quadratic in the size of the uncertain domain (a direct result of manipulating temporal correlations) and exponential in the number q of query links (this cost is hidden in the fact that $N = 2^q$).

[EXISTS] As it receives each input, the EXISTS operator computes the probability p that either this or any previous input satisfies the event query. Intuition suggests that this operator might simply pass these inputs into the INSTANTS operator and combine the results. Such a computation would be incorrect, however, since event query result probabilities produced for different timesteps are correlated (this is a consequence of the temporal correlations in the Markovian input stream).

Fortunately, EXISTS semantics can easily be computed using INSTANTS machinery after a simple query rewrite. Specifically, during parsing of queries with EXISTS semantics, LaharOLAP adds a trivially-true self-loop edge to the final link of the event query NFA. Informally, this edge "captures" the probability mass added to the NFA's final state at every step where the query is satisfied. Correlations are automatically and correctly handled in this framework by the INSTANTS processing algorithm.

[COUNT] The COUNT operator computes a probability distribution over the number of timesteps that satisfy the event query. As with EXISTS semantics, this count cannot be post-computed on the output of an INSTANTS computation because this output does not make explicit the correlations between its values.

In order to correctly handle correlations, a COUNT operator implements the same computation as the INSTANTS operator, with one difference: while the INSTANTS state matrix entries are single probabilities, the COUNT matrix entries are histograms that represent a set of possible count values (and their probabilities). These histograms are *not* probability distributions: the sum of probabilities over all values in all of the matrix entries combined is 1.0.

One important characteristic of COUNT processing that is not shared by EXISTS or INSTANTS algorithms is that its perfor-

mance degrades with the length of the input stream. The cost of updating the counts (histograms) for each new input grows with the size of the count domain, which in turn can grow linearly (worst-case) with the number of inputs. This scaling problem 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 [28]. Currently LaharOLAP always computes an exact count distribution; we show in Section 4 that the performance consequences in practice are usually minor.

3.1.3 `Agg` Operator

The `Agg` operator combines event query results computed on independent Markovian streams. Its two semantics, `STREAMEXISTS` and `STREAMSUM`, are conceptually similar to the `EXISTS` and `COUNT` temporal aggregation semantics; however the `Agg` operators are simpler because their inputs are uncorrelated.

`[STREAMEXISTS]` The `STREAMEXISTS` operator computes the probability p that *any* of its inputs is true as one minus the probability that *none* of the M inputs are true: $p = 1 - \prod_{i=1}^{M} (1 - p_i)$. This computation is applied straightforwardly in the `Agg` operator. LaharOLAP assumes that each Markovian stream is independent from the others, so correlations are not a concern for cross-stream aggregation. It is illustrative to contrast this computation with `COUNT`, where correlations prevent us from writing such a simple formula.

`[STREAMSUM]` The `STREAMSUM` operator computes a distribution over the sum of its imprecise numeric inputs. These numeric inputs are usually Boolean values, which LaharOLAP treats as degenerate count distributions over the domain $\{0, 1\}$. The single case in which `STREAMSUM` inputs are non-Boolean is when they are the result of `COUNT` temporal aggregation. In either case, the `Agg` operator incorporates each input into its incrementally-computed sum using a simple pairwise sum between elements from the input and sum distributions.

As with `COUNT` aggregation, the latency of the `STREAMSUM` operator increases with the size of its distribution over the final sum. Growth of the `STREAMSUM` domain is potentially worse than the linear growth of the `COUNT` domain, since each single input to `STREAMSUM` may itself have a large domain (this occurs only on aggregations of `COUNT` results). In Section 4 we demonstrate that, in practice, `COUNT` aggregation is nevertheless more of a performance bottleneck because its cost recurs on a per-timestep basis, while the cost of a `STREAMSUM` aggregation over `COUNT` results is incurred only once per stream.

3.2 Markovian Stream Compression

The performance of all three `Reg` operators described in the previous section scales quadratically in the size of the input stream's active domain—in other words, event query processing on Markovian streams is slow. In this section we focus on compression techniques to improve performance, as is common in OLAP systems [1, 21, 41]. The performance benefits of compression are complementary to those of Markovian stream indexing, which has been studied in prior art [34].

While traditional databases use *lossless* compression to improve performance, *lossy* techniques are a more promising avenue for Markovian stream warehouses. In addition to reducing the number of bytes required to represent a Markovian stream, judiciously-chosen lossy compressions can reduce the cost of event query processing by shrinking the size of the uncertain domain, or by making quadratic algorithms unnecessary. Applications that leverage these streams are already accustomed to handling uncertainty, so they can often tolerate a small amount of additional imprecision incurred by the use of lossy techniques.

In this section we introduce the set of lossy and lossless compression techniques leveraged by LaharOLAP. Because LaharOLAP implements null suppression—meaning that it does not explicitly represent elements with zero probability—techniques that assign zero probability to elements are effectively pruning these elements from the stream. Because the effect of compression techniques on event query processing performance is more important than their effect on data size (by 1-2 orders of magnitude, as we demonstrate in Section 4), none of LaharOLAP's techniques produce compressed representations that require active decompression. We thus use the terms "lossy compression" and "approximation" interchangeably.

3.2.1 Numeric Compression

Numeric compressions do not consider the semantics of the stream values. These compressions include:

Thresholding compression uses a parameter T to prune/discard any temporal correlations (conditional probabilities) v such that $v < T$. The remaining values are normalized in order to maintain a consistent Markovian stream. Clearly, higher values of T produce more aggressively-compressed 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 4.

Independence compression simply removes all temporal correlations from a Markovian stream. The effect of this compression is dramatic (more precisely, quadratic), both in terms of disk storage and in terms of the effect on `Reg` performance.

Consider a Markovian stream on an uncertain domain D: the correlations between two adjacent timesteps of this stream form a matrix of size $|D| \times |D|$. After dropping these correlations, the independence-compressed stream view must store only the marginal distribution over the uncertain domain at each timestep. This marginal distribution has size $|D|$, yielding quadratic space savings.

Independence compression yields quadratic performance improvements in the `Reg` operator as well. Recall the state matrix M that is the core of the event query processing algorithms: this matrix has a separate column for each of the $|D|$ elements in the Markovian stream's uncertain domain. Conceptually, each column of M represents a different set of possible worlds; the j^{th} column of M represents the possible states of the query NFA *conditioned* on the fact that the j^{th} domain element was the previous input. `Reg` must keep track of the previous input in order to correctly compute the temporal correlations between this previous input and the next input to arrive. If the input stream is independent, then the $|D|$ columns of M are identical (at all times, on all inputs). Clearly in this case there is no need for `Reg` to maintain a full matrix, and M can be reduced to a single vector V. Thus, instead of the $N \times D \times D$ operations required to update M, the `Reg` operator update on an independence-compressed input requires only $N \times D$ operations. This is a quadratic performance improvement.

MAP compression identifies the single most likely deterministic sequence (called the **M**aximum **a P**osteriori sequence) in a Markovian stream. It saves this single, deterministic stream and discards everything else. MAP approximation thus eliminates not only correlations (like independence), but it eliminates all uncertainty. A MAP-compressed timestep requires constant space to store, since it contains only a single element from the stream's uncertain domain. A MAP-compressed timestep also requires constant time to process, since it is deterministic. Instead of a matrix or a vector, the internal state of the `Reg` operator on MAP input is simply a number that identifies the single *set* of states that the NFA is currently in. MAP approximation has a special significance since the MAP sequence is the most common way to deterministically represent an uncertain stream.

3.2.2 Semantic Compression

In contrast to numeric compression, semantic compression is applied with an awareness of the semantics of Markovian streams.

Rollup compression uses a concept hierarchy to produce materialized Markovian stream rollups. These rollups represent the uncertain domain elements at a coarser level of granularity. For example, consider the uncertain location domain shown in Figure 2(a): {Office1, Lab1, Lab2, HallA, HallC}. One possible rollup compression 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 compression has the domain: {Room, Hallway}.

Rollup views retain both the uncertainty and the temporal correlations from the underlying Markovian stream, so they require quadratic space to represent and quadratic time to process. However, these costs are quadratic in the size of a smaller domain. Rollups thus achieve performance gains according to the degree to which they collapse together the elements of the Markovian stream's uncertain domain.

A rollup view is equivalent to a Markovian stream produced by inference on a model (e.g. an HMM) defined at the granularity of the rollup. Thus rollup compression is lossless with respect to the coarse-grained underlying model. A rollup view is *not* lossless, however, with respect to the Markovian stream inferred using the original, fine-grained model.

To see this, consider the fine-grained office model in Figure 2(a), and suppose that the the dotted door between HallA and HallC is locked (Bob has no key). Suppose further that Bob's location in this model is uncertain: we do not know for sure which side of the door Bob was on. Any Markovian stream inferred using this model will assign zero probability to trajectories in which Bob visits both Lab1 and Lab2, since Bob cannot walk through walls or through locked doors. Now consider a rollup-compressed view in which all hallway locations (in this case, HallA and HallC) are collapsed into a single conceptual entity called Hallway. The trajectory Lab1-HallA-HallC-Lab2, which had zero probability in the uncompressed Markovian stream, is mapped to Lab1-Hall-Hall-Lab2 in the rollup-compressed stream. This latter trajectory may not have zero probability in the compressed stream, because the rollup compression discards correlations between entities that are collapsed together (e.g. it discards knowledge of the locked door between HallA and HallC that makes these locations negatively correlated in a single trajectory).

Rollup-compressed views are thus lossy with respect to fine-grained models, because some correlation information is discarded by the compression. However, rollup-compressed marginal values are not affected, and thus rollup compression can be used in conjunction with independence compression without incurring additional loss beyond that incurred by independence

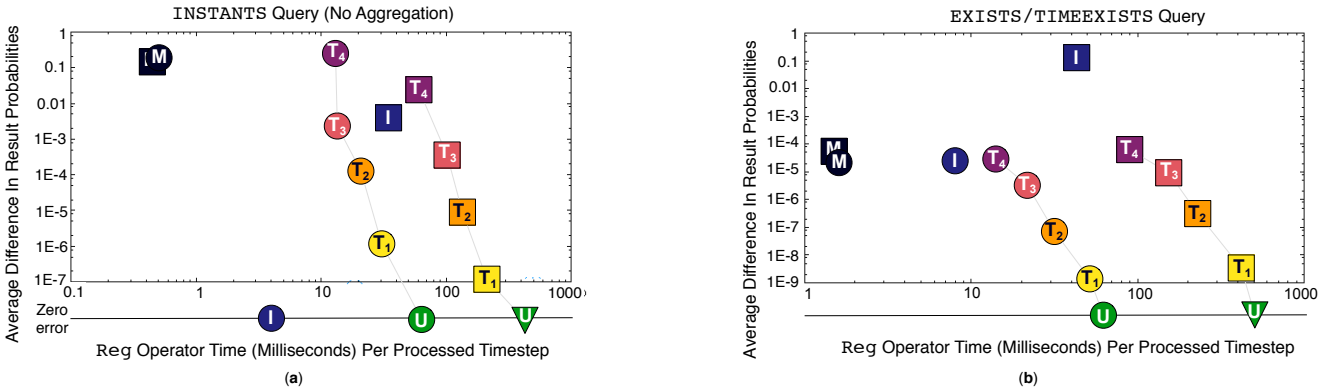


Figure 8: Accuracy vs. efficiency for two INSTANTS queries of length 1 (circles) and length 4 (triangles) on our ambiguous dataset. (a) shows unaggregated event query differences, while (b) shows differences in query results using a combination of EXISTS and STREAMEXISTS aggregation. The compression techniques shown are: U-Uncompressed, I-Independence, M-MAP, and T_1-T_4 are thresholded views using thresholds 1E-6, 1E-4, 1E-2, and 0.5, respectively.

alone. The combination of independence and rollup compression is particularly advantageous because it can significantly reduce the size of the uncertain domain.

4 Evaluation

In this section we demonstrate the performance of LaharOLAP on event-OLAP queries. We then examine the impact of lossy compression techniques on both performance and the precision of query results, demonstrating that these techniques can yield large performance gains while altering result probabilities only minimally.

[Data] Real-world Markovian streams are critical to our evaluation of LaharOLAP, 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. This deployment includes 160 RFID readers installed in the hallways (and only the hallways) of our 6-story office building, and 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 contains 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 and exiting various rooms for brief (1 minute) intervals. The Markovian streams inferred from the unambiguous dataset contain significant uncertainty, but identify a single most likely room during each in-room interval; in contrast, streams inferred from the ambiguous dataset generally identify 2-3 most likely rooms with roughly equal probability. Temporal correlations are thus stronger in the ambiguous data, which reflects the symmetry problem discussed in Section 2.1.

[Queries] We evaluate LaharOLAP on a set of queries designed to highlight the strengths and weaknesses of various compression techniques as well as their sensitivity to Markovian stream ambiguity/correlations. These queries search for the room-entry events present in our trace sets (specified using varying numbers of query links, according to the purpose of each experiment). For simplicity we restrict our evaluation to event queries that use NEXT sequence semantics—these queries are more common and also more sensitive to correlations.

[Overview of Performance/Accuracy Trade-Offs] We show a high-level view of LaharOLAP's trade-off space, on which we focus in the remainder of this section, in Figure 8(a) and (b). These figures plot the performance and result quality of two event queries (a 1-link query represented using circles, and a 4-link query represented using squares) computed on various compressed stream views. We show the trade-off space for both unaggregated event queries (Figure 8(a)) and for aggregated query results (Figure 8(b)). Note the log scale of all 4 axes. The x-axes represent the average, per-timestep latency of the Reg operator. The y-axes represent the average query error, computed as the absolute difference between the exact event query probability and its probability computed on an approximate view (for each timestep).

Throughout this section we define error in this way (with respect to the original Markovian stream); an equally valid and interesting evaluation would compare query results on different compressed views with respect to their fidelity to the underlying ground truth sequence. Such an evaluation confuses two sources of imprecision, however: imprecision from the

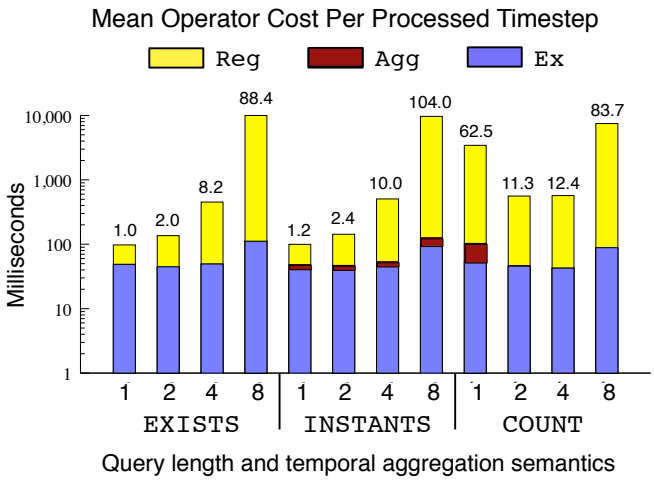


Figure 9: (a) Overview of baseline LaharOLAP performance on 5-stream `STREAMSUM` queries. Note the logscale y-axis; numbers above each query bar give the ratio of $\frac{\texttt{Reg}}{\texttt{Ex}}$ latency.

inference process and imprecision from the compression process. For clarity we choose the inferred Markovian streams as our exact baseline. As a point of reference, the error of randomly-chosen query probabilities in this framework is 0.5.

The baseline performance of LaharOLAP on an uncompressed stream is shown by the "U" points along the "zero error" lines in both plots in Figure 8. We discuss this performance in more detail in Section 4.1.

The two-dimensional portion of the plots in Figure 8 show LaharOLAP's performance/accuracy trade-offs. The spread of the points in these plots show that LaharOLAP's compressions achieve a flexible range of trade-offs, which we discuss in Section 4.2.

We first evaluate the baseline performance of the system when executing event and event-OLAP queries on uncompressed Markovian streams (Section 4.1). We then study the trade-offs between high performance and high accuracy when executing queries on compressed Markovian streams (Section 4.2).

4.1 Baseline Performance

In this section we first demonstrate LaharOLAP's baseline performance, and we then discuss several cases in which LaharOLAP can improve this performance while providing exact results.

4.1.1 Baseline Performance

We measure the baseline performance of each of LaharOLAP's three operators using a `STREAMSUM` aggregation over all five input streams from our ambiguous trace set, on representative event queries of varying lengths. This performance is shown in Figure 9 (note the logscale y-axis).

Figure 9 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 observation validates the intuition from Section 3 that event query processing, and not disk access, is LaharOLAP's performance bottleneck.

While the `Ex` operator's latency is a roughly-constant 50-100 milliseconds per timestep (equivalently, it processes 10-20 timesteps per second), the `Reg` operator's latency ranges from 50 milliseconds to nearly ten full seconds per stream timestep! Recall from Section 3.1 that `Reg` scales exponentially with the number of links in a query. This trend is most clearly visible on the `EXISTS` and `INSTANTS` queries in Figure 9. In practice, queries with length greater than 4 are not tenable (nor are they particularly useful).

The exponential scaling of the `Reg` operator on `COUNT` queries is less clear in Figure 9—this is because, as discussed in Section 3.1, the `COUNT` operator incurs additional latency as the size of its count domain grows. Comparing against `EXISTS` or `INSTANTS` performance as a baseline, Figure 9 shows that the 1- and 2-link `COUNT` queries incur a significant domain-induced latency, while the 4- and 8-link queries incur almost none. Indeed, the domain size of the `COUNT` estimate on the longest of the five input streams used in this experiment was 511, 101, 53, and 27, for 1-, 2-, 4-, and 8-link queries, respectively.

Milliseconds per stream timestep (INSTANTS query)

Optimization \ Query Length	1 Link (SELECT)	2 Links	4 Links	8 Links
Unoptimized	**57.37** ± 43.45	**96.80** ± 68.70	**391.4** ± 298.3	**9924** ± 7356
Rollup View ("uncertain")	**4.36** ± 1.80	**6.37** ± 3.10	**19.12** ± 10.39	**256.3** ± 203.1
Rollup View ("deterministic")	**3.60** ± 0.91	**4.86** ± 1.20	**12.64** ± 2.25	**212.7** ± 10.9
Independence	**4.10** ± 1.54	Results of EVENT queries are not exact when computed on an independence-compressed stream.		

Figure 10: Performance of LaharOLAP using compressed stream views to compute exact query results for INSTANTS queries of varying length.

Finally, Figure 9 shows that latency of the STREAMSUM operator is low compared to the latency of Reg. 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 9. The single-link COUNT query is an exception whose slow performance is due to the count domain size, which reaches 1536 elements (the ground truth count is 711). 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.

4.1.2 Leveraging Compression Losslessly

LaharOLAP can leverage compressed stream views to compute precise query results in two cases, summarized in Figure 10. First, SELECT queries without temporal aggregation can be processed precisely on an independence-compressed view (this explains the presence of the "I" circle on the zero-error line in Figure 8). Dropping temporal correlations (i.e. independence compression) doesn't affect SELECT queries because they examine timesteps in isolation; however, if temporal aggregation is applied then these correlations are again required to compute precise results.

Second, LaharOLAP can compute precise results (with respect to a coarse-grained data model, as explained in Section 3.2.2) using rollup-compressed views. Figure 10 shows LaharOLAP's performance on a rollup view that reduces the stream's uncertain domain from 966 discrete locations (individual rooms, halls, *etc.*.) down to 4 coarse-grained location types (offices, halls, stairs, and "other"). The 241-fold reduction in domain size yields speedups of 15-39% percent (Figure 10 rows 1-2)—the apparent mismatch between the magnitudes of these reductions reflects the fact that in our experiment, the Reg operator was optimized to scale quadratically in the size of the *active* uncertain domain (instead of the full uncertain domain). The active domain in the experiments in Figure 10 was roughly 20, which is consistent with the 15-39% speedups.

4.2 Performance and Accuracy Trade-Offs

In this section we examine in more detail the accuracy/performance tradeoffs highlighted in Figure 8. We look first at the performance benefits of compression (Section 4.2.1), then at the effects on result quality (Section 4.2.2).

4.2.1 Performance

The mean latency of LaharOLAP's Reg and Ex operators on various compressed stream views is shown in Figure 11, computed over a set of five 4-link INSTANTS queries. Queries with different lengths and temporal aggregation semantics show similar trends (with the exception of highly unselective COUNT queries, as discussed at the beginning of this section). Note that the y-axis is *not* in logscale, as small timing differences appear more clearly on linear axes.

Figure 11 clearly demonstrates that LaharOLAP can dramatically reduce processing time by leveraging compressed stream views. Thresholded views accelerate performance by a factor of 2-7, depending on the threshold. Independence- and MAP-compressed views achieve performance boosts of one and two orders of magnitude, respectively.

The performance benefits of the independence- and MAP-compressed views stem directly from their reductions to the dimensionality of the Reg operator's state matrix. To see this, consider in Figure 11 Reg performance on the thresholded

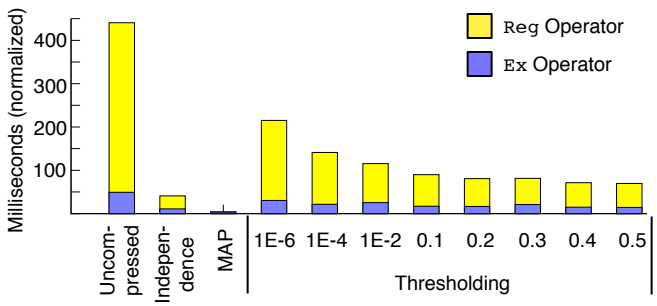


Figure 11: Unaggregated `Reg` and `Ex` operator performance on various compressed stream views. Performance on `EXISTS` and `COUNT` semantics is similar and we omit these plots.

stream with T=0.5; this stream is effectively deterministic. However, LaharOLAP is able to process the independence view—which maintains full uncertainty within each timestep—2.3 times faster than it processes the "deterministic" thresholded stream, despite the fact that 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, LaharOLAP 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.

The smaller physical size of both independent and aggressively-thresholded streams is only a secondary factor contributing to the lower latency of processing these views. Smaller streams proportionally reduce the cost of the `Ex` operator; for space reasons we omit graphs showing the compression ratios achieved by each of LaharOLAP's compression techniques.

4.2.2 Result Quality

LaharOLAP's performance-improving compression techniques are worthwhile only if they produce query results that can be used by applications. As we discuss in Section 4.2.3, compressed Markovian stream views can differ significantly from the original stream; however, applications care not about these stream-level differences but about their effects on query results. In this section we demonstrate that, empirically, LaharOLAP can compute high-quality query results on lossily-compressed Markovian streams. Our real-world RFID data is critical to this empirical evaluation.

We measure the quality of LaharOLAP's query results in terms of the absolute difference between result probabilities computed on approximate and precise Markovian streams.

Figure 12 shows the query-level probability differences on a single 4-link event query aggregated using `EXISTS` semantics (left column), `COUNT` semantics (right column), and using no temporal aggregation (middle column). The top and bottom row of the figure show query error on the unambiguous and ambiguous trace sets, respectively. For `EXISTS` and `INSTANTS` queries, the y-axis (error) is the same metric (average difference in query probability) used in Figure 8. For `COUNT` queries, which produce distributions over a count value, the error is computed as the Earth Mover's Distance (EMD) between these distributions [46]. Intuitively, an EMD value of n means that the count estimates of the two distributions differ by n. Finally, the quartiles of the `INSTANTS` plots (Figure 12 middle column) include only non-zero errors (this is also true in Figure 8), since the vast majority of timesteps produce zero error which, if plotted, obscure the magnitudes of the errors on the small number of timesteps when the query is actually satisfied with some probability.

Figure 12 shows that overall, differences in query results on compressed views are low. As thresholds increase, these differences naturally increase (on all types of data and temporal aggregation). The differences incurred by independence- and MAP-compressed views are less predictable, however, and since these are the views that yield the best performance, we look at these errors in more detail below.

Independence views ignore the temporal correlations in a Markovian stream. In our data sets, this generally results in overestimation of event query probabilities (recall the example from Section 2 in which Bob stays in either his Office or his Lab for an hour without moving). Although the magnitudes of these individual over-estimates are small (Figure 12(b), (e)), temporal aggregation of many of these over-estimated values can result in larger differences (Figure 12(a), (c), (d), and (f)).

In addition to computing potentially over-estimated event query values, independence also causes `COUNT` queries to over-count (recall again the example from Section 2 in which Bob's stream satisfies the event query zero times, but produces 60 false positives if correlations are ignored). We verified that the EMD error shown in Figure 12(c) and (f) is indeed caused by over-counting. The over-counting error is higher on the unambiguous trace set (Figure 12(c)) than on the ambiguous trace set

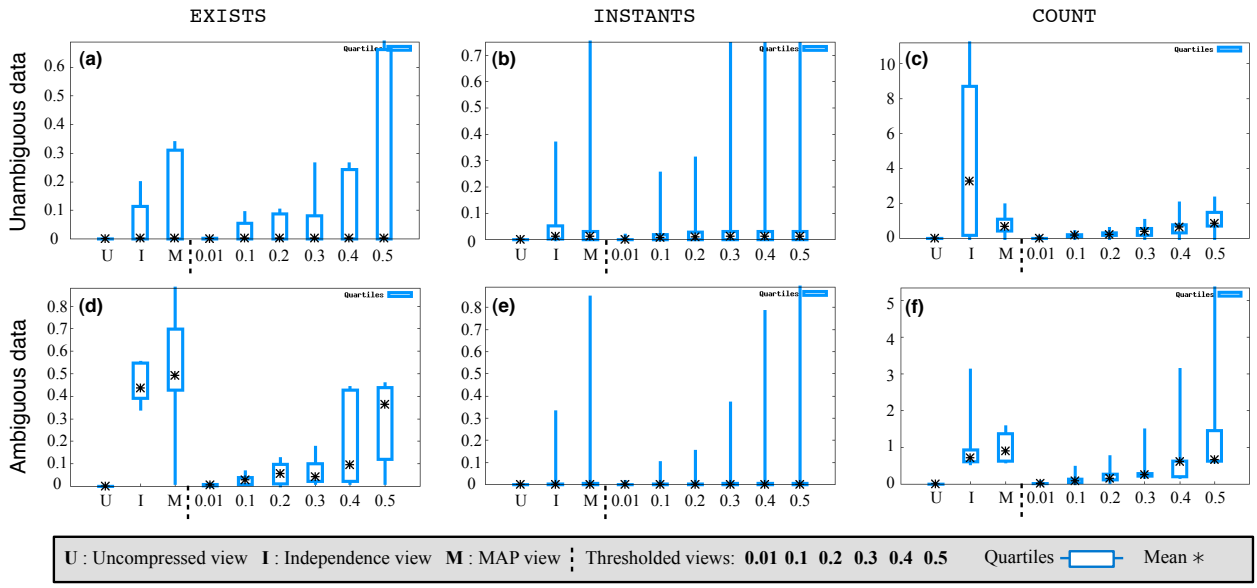


Figure 12: Differences in query result probabilities computed on exact and compressed Markovian streams. Differences are shown for each compression technique on a set of five 4-link queries with varying temporal aggregation semantics (Figure columns) and on two trace sets (Figure rows).

(Figure 12(f)) because the event query is satisfied in the unambiguous trace set with higher probability—causing LaharOLAP to have a higher confidence in its over-counted values.

MAP views replace a Markovian stream distribution with the single most likely (deterministic) sequence in the distribution. The quality of query results on these views is thus dependent on the level of uncertainty in the data; highly uncertain datasets are poorly represented by a single estimate. In practice we found that even our unambiguous trace set contained enough uncertainty that MAP error varied widely across a wide range of queries; thus we abstain from drawing further conclusions about MAP, except to note that the very unpredictability of its error is a potential liability. A LaharOLAP administrator must carefully identify the datasets on which MAP views can be accurately used (we discuss this selection in more detail in Section 5).

4.2.3 Stream-Level Error

As mentioned in the previous section, compressed Markovian stream views can differ significantly from the original stream. We quantify these differences at the distribution (stream) level by computing the EMD distance between the pairwise joint distributions of each stream. These differences, computed over our entire RFID archive, are shown in Figure 13. Interestingly, while these stream-level differences and the query-level differences shown in Figure 12 follow the same trends, the *magnitudes* of the stream-level differences are much greater (`COUNT` magnitudes are not comparable). We surmise that the reason these differences do not propagate up to the query/application level is that our techniques are often trimming noise from the stream (e.g. thresholding may drop errant tuples). This means that, perhaps counterintuitively, the approximations we study may in some cases improve Markovian stream quality (with respect to ground truth).

5 Discussion

In this section, we discuss some of the implications of the above findings and outline important research questions that remain to be addressed within the context of Markovian stream warehouses.

[Query optimization] As Section 4 demonstrates, lossy compression of Markovian streams enables interesting trade-offs between accuracy and performance. How best to leverage these trade-offs is an important question. One possible approach is for the system to isolate applications from the details of compression by materializing multiple compressed views and leveraging an optimizer to select an appropriate view for each query at runtime. Although such automated optimization

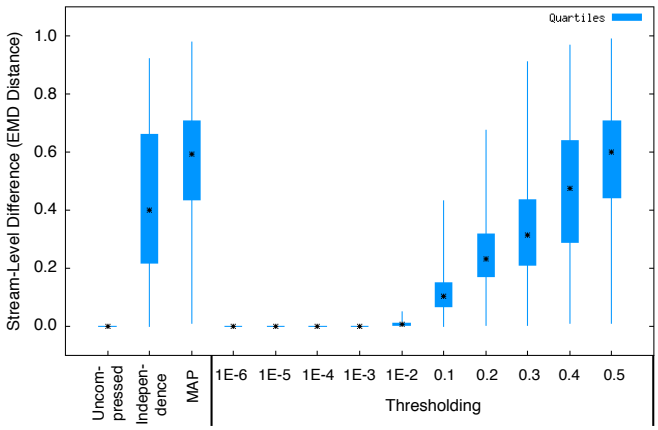


Figure 13: Distribution-level differences between Markovian streams and lossily-compressed views, computed over our entire RFID-based trace archive. Stream difference is computed using the Earth Mover's Distance on the joint distributions over each pair of consecutive stream timesteps.

is possible in some cases—for example, as we showed above, independence views are always preferable to uncompressed views for SELECT queries—in general, the contours of the accuracy/performance space are application-specific. For example, exploratory applications wishing to identify a set of candidate streams for closer analysis requires high performance but not high accuracy. In contrast, applications wishing to rank a set of similar events by probability require high accuracy but not high performance.

Thus a second approach might allow applications to set their desired precision/performance trade-off dynamically. In this approach, users would choose to execute their queries in either a *Maximum performance (MP)* or *Maximum accuracy (MA)* mode, similar to choosing a degree of isolation when executing a transaction. By choosing the MA mode, users signal to the system that precise results are required; however, by choosing the MP mode, the user signals that imprecise results are acceptable, allowing the system to leverage lossy compression. Related research opportunities include automatic selection of the appropriate set of compressed views to materialize for a given workload.

[Richer query and data models] A second related research problem is the question of how to extend LaharOLAP's warehousing and compression framework to other types of queries and more sophisticated data models. For example, richer, sequential probabilistic models such as *Dynamic Bayesian Networks* [36] exploit correlations between attributes within a timestep to specify distributions much more succinctly. The cost of succinct representation is that query answering may become more difficult. It would thus be interesting to study the interaction between these higher-level models and the compressions studied in this paper. Similarly, in this paper, we discuss queries that find complex patterns within a single stream, but the study of more sophisticated queries that search for complex patterns across several streams is also interesting in the context of Markovian stream warehouses [44].

6 Related Work

We briefly survey the work that LaharOLAP brings together, which encompasses classical database techniques such as OLAP, event processing, and compression, together with emerging database and AI techniques such as model-based views, particle filtering, and Markov Models.

Graphical models and probabilistic inference have long been used in the AI community to model and reason about uncertainty [26,32]. Dynamic Bayesian networks [36], of which Hidden Markov Models [40] are a simple example, are well-established models for temporally-correlated data. The particle filtering [4] and associated smoothing [10,30] algorithms used in this paper are only several of many well-established probablistic inference algorithms that infer unknown and/or high-level information from uncertain inputs like sensor data.

Many recent database systems allow users to pose relational queries on model-based views [14], whose values are computed at query time via probabilistic inference on an underlying graphical model using either exact inference [**?**,16,48,49,52] or sample-based inference [24, 27, 51]. LaharOLAP, in contrast, stores materialized versions of such views given as input, and thus assumes that all inference is performed offline.

A separate set of systems mange uncertainty by applying deterministic cleaning rules rather than probabilistic inference [6, 29, 42, 53]. These systems tend to target a specific data domain—usually RFID—and thus are less general than inference-based approaches.

Probabilistic relational DBMSs emerging in recent years handle uncertainty by imposing independence assumptions on the base data. Such systems include Mystiq [11] and Trio [5]. The query languages supported by these systems are richer than that of LaharOLAP, but their data models exclude the ordered, correlated tuples of Markovian streams.

Streaming event queries and NFA-based processing have been studied in the context of deterministic streams (e.g. the Cayuga [12] and SASE [**?**,2] systems) and also for probabilistic streams (e.g. a SASE extension [50] for uncorrelated streams, and previous work on Lahar [34,44] for Markovian streams). Although most of these systems are targeted at real-time stream processing, LaharOLAP's event query processing is built on similiar foundations. A separate set of related systems process streaming data—in many cases Markovian streams—using probabilistic inference inside the DBMS [**?**, 51], but as a storage manager LaharOLAP is solving a fundamentally different problem.

Work on probabilistic stream sketches [9, 25] is related to LaharOLAP, but focuses more on single-pass computation of stream statistics rather than relational/event query processing. Time-series databases [15, 58] and temporal databases [47] also share a natural connection to LaharOLAP because they handle temporal data, but they focus on very different problems (time-series databases focus on sequence similarity/matching, while temporal databases focus on managing the temporal evolution of relational datasets).

OLAP/warehousing systems and OLTP systems target very different workloads [7]. LaharOLAP is developed for the former, characterized by aggregate queries over large slices of historical data. Though the data cube [19] is the predominant data model for warehousing, its aggregation model is not directly applicable to the temporal, uncertain data of Markovian streams. LaharOLAP takes the same approach as other recent determinstic stream warehousing work [38] by answering each query directly on raw (non-cuboid) data. LaharOLAP's fundamental approach to trading efficiency against accuracy is similar in spirit to the iterative-refinement ideas in ????.

Because of RFID's popularity in commercial supply chains, much data warehousing work targets RFID data specifically [17, 18, 33]. This work exploits RFID-specific characteristics (e.g. shared path prefixes early in a supply chain) to support efficient query processing, but such systems require that queries be written at the semantic level of RFID tags/readers, ignoring noise issues and precluding the transformation of RFID data into higher-level information (e.g. location).

Compression within a DBMS is generally applied losslessly to relational data, whether in a row-store [41] or column-store [1] framework. In either case, the most successful techniques exploit the lack of tuple order to achieve effective, lightweight compression [20]; unfortunately the ordered nature of Markovian streams renders such techniques inapplicable in LaharOLAP. At a higher level, however, LaharOLAP is similar in spirit to the work of Apaydin et al. [3], who use approximate bitmap compression to create a tradeoff between query accuracy and efficiency; and to Chen et al. [8], who introduce(??) the notion of allowing a quer optimizer to choose between different types of compression.

Additional compression work has been done for XML data [37], but this work was targeted at reducing network bandwidth, and thus did not consider the effect of decompression time on query effciency.

Need to find a home for this discussion of Amol's ICDE work: LaharOLAP is able to process a superset of the queries processed by the Markovian stream operators proposed by Kanagal et al. [?] (LaharOLAP handles complex event queries and does not require that the final result stream be determinized in any way).

We need to either compare performance with Amol's system or clearly state why we have not.

7 Conclusion

In this paper, we presented LaharOLAP, a novel system for warehousing imprecise and temporally-correlated data sequences that we call Markovian streams. LaharOLAP is designed to support a broad class of applications from RFID tracking to speech processing and more. All of these applications need to query high-level data inferred using a model from low-level data like multimedia or sensor streams. Markovian streams are the result of this inference and the data managed by LaharOLAP.

LaharOLAP supports complex queries including `SELECT`, event, and event-OLAP queries that search for patterns in streams and, optionally, aggregate them. A key contribution of this paper is the study of tradeoffs between additional imprecision (via lossy compression techniques) and increased performance. Given the increased availability of large multimedia and sensor data archives, warehousing Markovian streams is an important problem and we view this paper as an important step toward enabling such systems.

References

[1] D. J. Abadi, S. R. Madden, and M. Ferreira. Integrating compression and execution in column-oriented database systems. In *SIGMOD*, pages 671–682, Chicago, IL, USA, 2006.

[2] J. Agrawal, Y. Diao, D. Gyllstrom, and N. Immerman. Efficient pattern matching over event streams. In *Proc. of the SIGMOD Conf.*, pages 147–160, New York, NY, USA, 2008. ACM.

[3] T. Apaydin, G. Canahuate, H. Ferhatosmanoglu, and A. S. Tosun. Approximate encoding for direct access and query processing over compressed bitmaps. In *Proc. of the 32nd VLDB Conf.*, pages 846–857. VLDB Endowment, 2006.

[4] 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.

[5] O. Benjelloun, A. D. Sarma, A. Halevy, and J. Widom. Uldbs: Databases with uncertainty and lineage. In *VLDB*, pages 953–964, 2006.

[6] J. Brusey, M. Harrison, C. Floerkemeier, and M. Fletcher. Reasoning about uncertainty in location identification with rfid. In *In Workshop on Reasoning with Uncertainty in Robotics at IJCAI-2003*, 2003.

[7] S. Chaudhuri and U. Dayal. An overview of data warehousing and olap technology. *SIGMOD Rec.*, 26(1):65–74, 1997.

[8] Z. Chen, J. Gehrke, and F. Korn. Query optimization in compressed database systems. In *Proc. of the SIGMOD Conf.*, pages 271–282, New York, NY, USA, 2001. ACM.

[9] G. Cormode and M. Garofalakis. Sketching probabilistic data streams. In *Proc. of the SIGMOD Conf.*, pages 281–292, New York, NY, USA, 2007. ACM.

[10] R. G. Cowell, S. L. Lauritzen, A. P. David, and D. J. Spiegelhalter. *Probabilistic Networks and Expert Systems*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1999.

[11] N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. In *Prov. of the 30th VLDB Conf*, 2004.

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

[13] 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.

[14] A. Deshpande and S. Madden. MauveDB: supporting model-based user views in database systems. In *Proc. of the SIGMOD Conf.*, pages 73–84, New York, NY, USA, 2006. ACM.

[15] M. Garofalakis, R. Rastogi, and K. Shim. Mining sequential patterns with regular expression constraints. *IEEE Trans. on Knowl. and Data Eng.*, 14(3):530–552, 2002.

[16] M. N. Garofalakis, K. P. Brown, M. J. Franklin, J. M. Hellerstein, D. Z. Wang, E. Michelakis, L. Tancau, E. W. 0002, S. R. Jeffery, and R. Aipperspach. Probabilistic data management for pervasive computing: The *data furnace* project. *IEEE Data Eng. Bull.*, 29(1):57–63, 2006.

[17] H. Gonzalez, J. Han, and X. Li. Flowcube: Constructing rfid flowcubes for multi-dimensional analysis of commodity flows. In *Proc. of the 32nd VLDB Conf.*, 2006.

[18] H. Gonzalez, J. Han, X. Li, and D. Klabjan. Warehousing and analyzing massive rfid data sets. In *Proc. of the 22nd ICDE Conf.*, page 83, Washington, DC, USA, 2006. IEEE Computer Society.

[19] 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.

[20] A. L. Holloway and D. J. DeWitt. Read-optimized databases, in depth. *Proc. VLDB Endow.*, 1(1):502–513, 2008.

[21] A. L. Holloway, V. Raman, G. Swart, and D. J. DeWitt. How to barter bits for chronons: compression and bandwidth trade offs for database scans. In C. Y. Chan, B. C. Ooi, and A. Zhou, editors, *SIGMOD Conference*, pages 389–400. ACM, 2007.

[22] Y. Hu, S. Sundara, T. Chorma, and J. Srinivasan. Supporting rfid-based item tracking applications in oracle dbms using a bitmap datatype. In *Proc. of the 31st VLDB Conf.*, pages 1140–1151. VLDB Endowment, 2005.

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

[24] R. Jampani, F. Xu, M. Wu, L. L. Perez, C. Jermaine, and P. J. Haas. Mcdb: a monte carlo approach to managing uncertain data. In *Proc. of the SIGMOD Conf.*, pages 687–700, New York, NY, USA, 2008. ACM.

[25] T. S. Jayram, A. McGregor, S. Muthukrishnan, and E. Vee. Estimating statistical aggregates on probabilistic data streams. *ACM Trans. Database Syst.*, 33(4):1–30, 2008.

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

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

[28] B. Kanagal and A. Deshpande. Efficient query evaluation over temporally correlated probabilistic streams. In *Proc. of the 25th ICDE Conf.*, Apr. 2009.

[29] N. Khoussainova, M. Balazinska, and D. Suciu. Probabilistic event extraction from rfid data. In *ICDE*, pages 1480–1482, 2008.

[30] 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.

[31] J. Lafferty, A. McCallum, and F. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *ICML*, 2001.

[32] S. L. Lauritzen. *Graphical Models*. Number 17 in Oxford Statistical Science Series. Clarendon Press, Oxford, 1996.

[33] C.-H. Lee and C.-W. Chung. Efficient storage scheme and query processing for supply chain management using rfid. In *Proc. of the SIGMOD Conf.*, pages 291–302, New York, NY, USA, 2008. ACM.

[34] J. Letchner, C. Ré, M. Balazinska, and M. Philipose. Access methods for markovian streams. In *Proc. of the 25th ICDE Conf.*, 2009.

[35] S. E. Levinson, L. R. Rabiner, and M. M. Sondhi. An introduction to the application of the theory of probabilistic functions of a Markov process to automatic speech recognition. *Bell Sys. Tech. J.*, 62:1035, 1983.

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

[37] H. Liefke and D. Suciu. Xmill: an efficient compressor for xml data. In *SIGMOD '00: Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, pages 153–164, New York, NY, USA, 2000. ACM.

[38] 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.

[39] 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. IEEE Computer Society, 2005.

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

[41] V. Raman and G. Swart. How to wring a table dry: entropy compression of relations and querying of compressed relations. In *Proc. of the 32nd VLDB Conf.*, pages 858–869. VLDB Endowment, 2006.

[42] J. Rao, S. Doraiswamy, H. Thakkar, and L. S. Colby. A deferred cleansing method for rfid data analytics. In *Proc. of the 32nd VLDB Conf.*, pages 175–186. VLDB Endowment, 2006.

[43] C. Ré and D.Suciu. Efficient evaluation of having queries on a probabilistic database. In *Proceedings of DBPL*, 2007.

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

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

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

[47] B. Salzberg and V. J. Tsotras. Comparison of access methods for time-evolving data. *ACM Comput. Surv.*, 31(2):158–221, 1999.

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

[49] P. Sen, A. Deshpande, and L. Getoor. Exploiting shared correlations in probabilistic databases. In *Proc. of the 34th VLDB Conf.*, pages 809–820, 2008.

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

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

[52] D. Z. Wang, E. Michelakis, M. N. Garofalakis, and J. M. Hellerstein. Bayesstore: managing large, uncertain data repositories with probabilistic graphical models. In *Proc. of the 34th VLDB Conf.*, pages 340–351, 2008.

[53] F. Wang and P. Liu. Temporal management of rfid data. In *Proc. of the 31st VLDB Conf.*, pages 1128–1139. VLDB Endowment, 2005.

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

[55] E. Welbourne, M. Balazinska, G. Borriello, and W. Brunette. Challenges for pervasive rfid-based infrastructures. In *PerCom Workshops*, pages 388–394. IEEE Computer Society, 2007.

[56] D. Woelk and W. Kim. Multimedia information management in an object-oriented database system. In *Proc. of the 13th VLDB Conf.*, pages 319–329, Brighton, England, 1987.

[57] E. Wu, Y. Diao, and S. Rizvi. High-performance complex event processing over streams. In *Proc. of the SIGMOD Conf.*, pages 407–418, New York, NY, USA, 2006. ACM Press.

[58] x. x. *x*, x:x, x.