Sensor Data Stream Exploration for Monitoring Applications

Shengliang Xu and Magdalena Balazinska
Department of Computer Science & Engineering
University of Washington, Seattle, WA, USA
{slxu, magda}@cs.washington.edu

ABSTRACT

This paper presents StreamXPlore, a system that enables users to *explore* historical stream data in order to determine what events to monitor in the future. At the heart of StreamXPlore is a new event modeling mechanism. StreamXPlore enables the specification, analysis, and mining of these new types of events. Event analysis enables event refinement using data-cube-style slice, dice, drill-down, and roll-up operations. Event data mining consists of event clustering for pattern identification. Preliminary experiments on a real stream of hydro-sensor data show that the proposed exploration mechanisms can indeed be helpful in identifying interesting patterns and are sufficiently fast for exploration of gigabyte-size stream repositories.

1. INTRODUCTION

Data stream management systems (DSMSs) (*e.g.*, [1, 6, 22, 24]) are effective tools for building sensing applications. DSMSs greatly simplify the development of monitoring applications because developers or end-users only need to declaratively express the events they are interested in monitoring and the DSMS takes care of the rest.

The challenge, however, is that, in a DSMS, users are expected to *precisely* specify their queries. A query specification includes precise patterns to extract and any parameter values including window sizes. Users can have difficulty in specifying such queries directly for several reasons:

Need for task- or subject-specific information. Many monitoring queries need subject- or task-specific information. Such queries cannot easily be constructed without information about the subject or task. For example, in an elder-care monitoring application, a nurse may want to monitor when an elder goes to the bathroom too frequently as this can be an indicator of poor sleep quality and a predictor for falls. The question here is what is the meaning of "frequently", *i.e.*, what should be the value of the threshold in the query. Different people have different habits at night. It is impossible for the nurses to set a threshold without familiarity with the monitored subject.

Vague information needs. A user may be aware that some conditions need to be monitored, but he may not be able to give an exact description. For example, a nurse may be aware that the actions of an elder at night can be indicators of health problems, but he may be unsure which actions to monitor: should there be an alert when the total number of activities at night exceeds a threshold or only the total number of bathroom visits or both?

Unknown patterns. No matter how profound a user's expertise of the monitored environment, a user may still miss important events that should be monitored. As an example, a nurse may have set up queries to monitor an elder's total level of activity, diet, and night actions, but she may be unaware that the elder sometimes forgets to shower.

To address the above challenges, we are building a stream data exploration system called StreamXPlore. The goal is to help users identify what monitoring queries to specify by examining historical stream data. At the heart of StreamXPlore is a new modeling mechanism for events. We define an event to be *a subset of a stream.* StreamXPlore allows users to specify events using either a time-window, or a tuple-window. While windows are known concepts in the stream-processing literature [2, 5], the novelty of our work lies in what we do with their contents. Instead of using windows as a mechanism for bounding stream aggregations, joins, or pattern matches, we consider the substream within a window as a first-class citizen. Each substream within a window becomes an event of its own.

StreamXPlore extracts series of such events and enables users to study them and refine them through *analysis* and *mining.* During analysis, the events are presented to users in the form of multidimensional cubes similar to OLAP data cubes [11] and the user can study them and refine them using standard slice/dice and roll-up/drill-down operations (*e.g.*, an event specified as "All activities between 10pm and 6am" can be refined to "Only bathroom activities between 11pm and 5am" or "Count of the number of bathroom activities every hour between 10pm and 6am"). Event mining can help identify patterns in a series of events. Currently, our system enables event clustering (*e.g.*, "There is one bathroom event on most days, sometimes none, and in a few cases two or more"), which can help identify groups of similar events. We leave additional mining algorithms for future work.

We evaluate StreamXPlore on a HydroSense dataset [9], which is a real data stream of water event records for different fixtures, such as shower, bath faucet, kitchen faucet, etc. collected in four houses over a 33 day period. Preliminary experimental results show that our data exploration mechanisms are able to help identify interesting patterns in

data and are sufficiently fast for a user to interactively explore stream warehouses on the scale of at least a gigabyte. While motivated by exploring sensor data streams, our approach is applicable to any stream warehouse.

2. RELATED WORK

Existing DSMSs (*e.g.*, [1, 2, 6, 22, 24]) enable users to ask continuous queries over data streams. Our goal, in this paper, is to help users explore stream archives in order to lower the barrier to articulating such queries.

Several systems extend DSMSs with stream warehouses [3, 4, 8, 10, 23]. We propose a specific method for exploring such archives. Liu *et al.* [19] proposed an E-Cube system to accomplish near real-time OLAP on stream data for decision making. Their proposed method shares some similarity with ours but the goal of our effort is to ease the query construction for the users, while their approach requires that users know precisely what patterns to extract from streams. Han *et al.* [12] and Lo *et al.* [20] also developed OLAP techniques for stream data. The former develop techniques to build a single large cube over an entire stream. The latter extended the data cube model by adding stream specific features such as pattern-based sequence summarization. In this paper, we take a radically different approach where we enable users to analyze and mine a new type of stream events all together. In earlier work [4, 17], we proposed the idea of clustering events on streams using complex context information. While the idea of clustering events is shared with this paper, our prior work used a different data model and had a different purpose, that of explaining newly observed events by showing similar past events.

Database usability has recently gained increased attention [14, 18]. Most closely related to our work, the QueRIE system [7] recommends to a user queries asked by other users. Similarly, in our prior work [16], we developed a system for auto-completing SQL queries. There also exists extensive work on visual query specification [25] and keyword queries [13]. While these systems help users articulate queries by offering suggestions or simplifying the query specification interface, we take a different approach in this paper: we help users explore the raw data in order to find interesting patterns that can motivate new queries.

3. DATA MODEL

In this section, we present StreamXPlore's data model.

Definition: Primitive event. A primitive event is a tuple e conforming to the schema `EventType(T,` A_1 `, ...,` A_n`)`. e represents some real world primitive event such as "Alice used the kitchen faucet at 10:05am". T is the time when the primitive event occurred and $A_1, \ldots, A_n$ are value attributes that describe the primitive event.

A stream of primitive events is an ordered set of primitive events of the same schema but distinct timestamps (T). Table 1 shows an example stream from the HydroSense data set. A stream database consists of several streams of primitive events and, optionally, standard relations.

Primitive events can be combined to form more complex events. In this paper, we define an event as follows:

Definition: Event. Given a stream S, an event E is a subset of primitive events in the stream. Let $e_1, e_2, \cdots, e_n$ be the sequence of primitive events in stream S, then $E \subseteq S$. That is $E \subseteq \{e_1, e_2, \cdots, e_n\}$.

Table 1: Example stream from the HydroSense database, where each primitive event corresponds to the activation of a water fixture in a house

timestamp	value attributes		
	fixture	duration	temper.
2010-02-01 08:09:59	Bathroom Sink	26.45097	Cold
2010-02-01 08:20:38	Bathroom Sink	25.30426	Cold
2010-02-01 08:27:25	Kitchen Sink	9.074509	Warm
2010-02-01 20:11:19	Kitchen Sink	7.922266	Cold
2010-02-01 21:50:05	Toilet	27.67681	Cold
2010-02-01 21:50:13	Bathroom Sink	22.66671	Warm
2010-02-01 21:58:01	Shower	927.2068	Warm

Our event definition is purposely general and one could write a selection query in any of the existing DSMSs (*e.g.*, [1, 2, 6, 22, 24]) to extract an event from a stream.

In StreamXPlore, we observe that, during stream exploration, users are often interested in extracting series of events, where the events in the series are related in some desired way. The novelty of our data model is precisely in capturing such *event series*, where each *event in a series is a substream*. For example, in order to study the nightly activities of an elder, a nurse may want to extract a series of events, each one showing the activity of the elder during one night. Events of interest do not always map onto specific time intervals. For example, a nurse wanting to examine the morning routine of an elder may want to see events that start with the first bathroom use in the morning and last until the last kitchen use that same morning.

In StreamXPlore, we provide two types of *event-series specifications*. The first one is based on sliding-windows [2, 6, 22]. Here, StreamXPlore identifies all primitive events that fall within each window and outputs them as one event (*e.g.*, all primitive events that fall within 12am and 5am on a given day form one event). The second specification defines a time window by specifying predicates on the primitive events that mark the begin and the end of the window [5] (*e.g.*, first bathroom to last kitchen use in the morning). We call the latter *tuple-windows*. Event-series specifications are currently the only way for users to extract events in StreamXPlore. We discuss both further when we present our language in Section 4.

Definition: Event Series. An *event series* is a sequence of events extracted from a stream using either a sliding-window or a tuple-window specification.

Given an event series, during stream exploration, a user needs the ability to inspect and compare events. To facilitate such exploration, StreamXPlore extracts, displays, and mines *event features*, which we define as follows:

Definition: Event Feature. A feature f of an event $E = \{e_1, e_2, ..., e_N\}$ is an aggregation γ on the primitive events, $\{e_i\}$, selected from E by a predicate p. $f = \gamma(\sigma_p(E))$.

Since the predicate space is infinite, it would clearly be infeasible to compute all features. To keep the feature-space tractable, StreamXPlore uses a constrained set of features based on hierarchies of primitive event attributes. We call these features, *semantic features*.

Hierarchy structures have been widely used in the data management community, such as in OLAP systems. Figure 1 shows an example hierarchy on the fixture attribute from the stream in Table 1. An attribute hierarchy can be seen as a hierarchy of predicates. For categorical attributes, each leaf node corresponds to a predicate of the form a_i

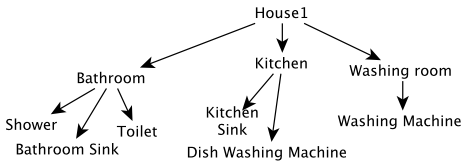

Figure 1: Example hierarchy for the fixture attribute from the stream in Table 1

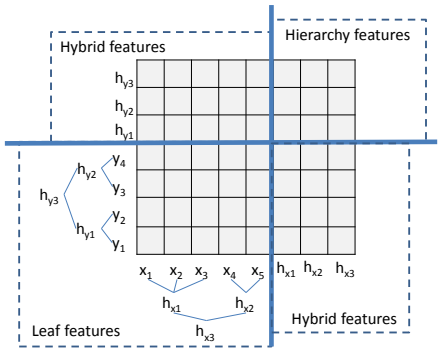

Figure 2: Example semantic feature space with two dimensions each with a three-level hierarchy.

='value', where a_i is an attribute in the stream and 'value' is a value from the active domain (*e.g.*, *fixture='Shower'*). Each inner node in the hierarchy corresponds to the predicate $p_{c1} \vee p_{c2} \cdots \vee p_{cn}$, where p_{ci} is the predicate of the i^{th} child node. For example, the predicate of the node *Bathroom* is *fixture='Shower'* $\vee$ *fixture='Toilet'* $\vee$ *fixture='Bathroom Sink'*. Similarly, for numerical attributes (*e.g.*, timestamp), predicates identify increasingly large ranges of values (*e.g.*, root node can cover a 24 hour period, inner nodes can cover groups of hours identifying the main periods of a day, and leaf nodes can map onto hours).

Definition: Semantic Feature Space. Given hierarchies $h_1, h_2, \cdots, h_H$ defined on the attributes of a stream S, let $\{p(h)\}$ be the set of predicates induced from each node in hierarchy h. The semantic feature space induced from the hierarchies is the Cartesian product of the hierarchy nodes, *i.e.* $\{p(h_1)\} \times \{p(h_2)\} \cdots \times \{p(h_n)\}$. Each feature in a semantic feature space is called a *semantic feature*.

If we treat each hierarchy as a dimension, a semantic feature space can be represented by a multidimensional cube, analogous to an OLAP data cube. Figure 2 illustrates a semantic feature space induced from two hierarchies.

The semantic feature space comprises three types of features: (1) *Leaf Features* are those semantic features where each predicate $p_i \in p(h_i)$ comes from a leaf of its corresponding hierarchy; (2) *Hierarchy Features* are those where each predicate p_i corresponds to a non-leaf node in its hierarchy; (3) *Hybrid Features* are all other features.

Let D be the number of dimensions defined on the stream, and L the number of leaf features, the number of hierarchy features is $O(L)$, but the number of hybrid features is $O(2^D L)$. This number thus quickly grows with D. To reduce the complexity, StreamXPlore only uses the leaf and hierarchy features as the features for the events. Although the hybrid features are dropped, their information is partly encoded in the leaf and hierarchy features.

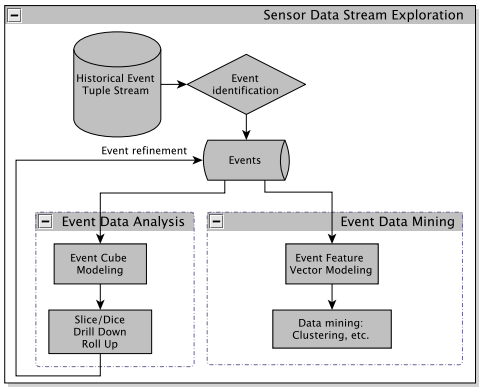

Figure 3: StreamXPlore Overview

EventSeries	:=	FILTER SQLPredicate
		SlidingWindowEvts | TupleWindowEvts
		[STREAM START SQLDatetimeExpression]
		[STREAM END SQLDatetimeExpression]
SlidingWindowEvts	:=	SIZE DurationExpression
		SLIDE DurationExpression
TupleWindowEvts	:=	W-START SQLPredicate
		W-END SQlPredicate [FIRST|LAST]
		WITHIN DurationExpression
		[ALLOW OVERLAP]
DurationExpression	:=	([0-9]+d)?([0-9]+h)?
		([0-9]+m)?([0-9]+s)?(\.[0-9]+)?

Figure 4: StreamXPlore's language

4. THE StreamXPlore SYSTEM

This section presents StreamXPlore's language and analysis and mining features. Figure 3 shows how these core components interact within the StreamXPlore system: event analysis and mining are mostly independant. While event mining is a one-way process, event analysis is iterative and produces the input to event mining.

4.1 Event Specification Language

StreamXPlore's language enables users to specify event series using either sliding or tuples windows as defined in Section 3. The language uses a subset of constructs from existing stream languages in a manner specialized for extracting event series, where each event is a substreams. Figure 4 shows the language grammar, where SQLDatetimeExpression and SQLPredicate represent the datetime type in SQL and the SQL where-clause predicate respectively.

We explain the key language features through two illustrative examples, one for each type of window. As first example, an event series, where each event comprises all primitive events between 10pm and 7am on a given day in house 'H2', can be expressed as follows.

FILTER	house='H2'
SIZE	9h
SLIDE	24h
STREAM START	2010-02-01 22:00:00
STREAM END	2010-03-05 07:00:00

In the language semantics, the FILTER clause is applied to the stream first and then the event-series is extracted as per the sliding-window specification. Optionally, the ex-

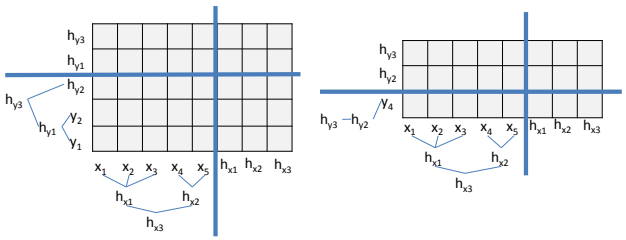

Figure 5: Illustration of roll-up and slicing for semantic feature space refinement

traction can be performed on some subset of the stream as indicated by the `STREAM START` and `STREAM END` clauses. In the absence of the `STREAM START` clause, sliding windows start at the first primitive event in the stream.

As a second example, a tuple window for morning routine events in house 'H2' can be defined as follows.

```
FILTER    hour(timestamp) >= 6 AND
          hour(timestamp) < 10 AND house='H2'
W-START   room='Bathroom'
W-END     room='Kitchen' LAST
WITHIN    4h
```

Here, `LAST` indicates that the last (rather than the first) Kitchen event that follows the Bathroom event within four hours closes the window. Our language also enables users to indicate whether tuple-windows are allowed to overlap or not with the ALLOW OVERLAP clause.

4.2 Event Analysis

To perform event analysis, a user submits an event series specification, a set of hierarchies on attributes of interest, and an aggregation function. StreamXPlore returns the list of events in the series together with the Leaf and Hierarchy features. The multi-dimensional semantic feature space defined by these hierarchies is analogous to an OLAP data cube and StreamXPlore provides the standard roll-up/drill-down and slice/dice operations for the user to further analyze the events in the series.

Roll-up/Drill-down. The roll-up/drill-down operations serve to refine the feature space of events in a series. Given a hierarchy on a stream attribute, the effect of roll-up is to remove some subtrees from the hierarchy. Figure 5 (A) illustrates a roll-up on the y-dimension of the event from Figure 2. Drill-down is the reverse. Roll-up and drill-down only change the semantic feature space and the output shown to the user. They do not change the events themselves.

Slice/Dice. As in OLAP, the effect of slicing or dicing events in a series is to drop some subtrees from the hierarchies. Figure 5 (B) illustrates a slice that drops the subtrees rooted at h_{y1} and y_3. Slice/Dice change both the semantic feature space and the events. They filter-out the primitive tuples from the events that satisfy the dropped-subtree predicates. As an example, for the hierarchy from Figure 1, if we slice away the 'Kitchen' node, all the primitive tuples satisfying *fixture='Kitchen Sink' or fixture='Dish Washing Machine'* will be filtered out from all the events in the series.

The above operations form the basis for interactive event browsing and refinement. In our current prototype, users perform these operations by specifying different attribute hierarchies and aggregation functions.

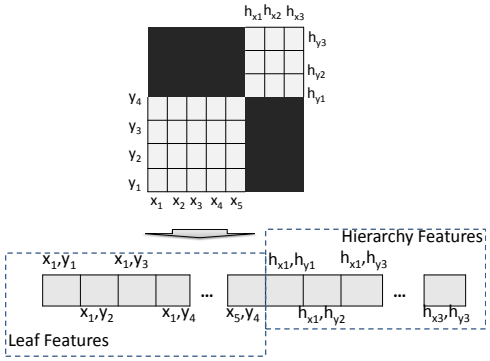

Figure 6: Feature vector representation of events.

4.3 Event Mining

OLAP-style analysis can help a user explore a series of events. Event mining can further help organize the events in the series. In particular, we propose the use of clustering to group events based on their similarity.

For event mining, StreamXPlore organizes the semantic features of an event into a feature vector as illustrated in Figure 6. Event feature vectors serve to calculate distances between events. Any distance measure in L^p space, *e.g.*, Manhattan or Euclidean distance, can be used.

While computing distances, features can be given different weights. To compute these weights, we propose to borrow techniques from information retrieval (IR), in particular their popular TF-IDF [15] weighting scheme. In our setting, for a feature f with predicate $p(f)$ on an event e in the series of events E, TF-IDF can be defined as:

$$TFIDF = count(\sigma_{p(f)}(e)) \times log(\frac{|E|}{|E_{match}|})$$

where $E_{match} = \{e' \in E | \text{count}(\sigma_{p(f)}(e')) > 0\}$

For each vector, this weighting scheme highlights the features that are most distinguishable: features that are prominent in this vector yet also occur rarely in other vectors have highest weights.

When data streams come from sensors, they typically contain noise, which affects StreamXPlore in two ways: (1) some tuple-windows may go undetected and (2) distances between some events may be incorrect, leading to noisy clustering. These inaccuracies, however, are in fact a system feature. StreamXPlore's goal is to help users explore a sensor data stream as it is. Exploration can thus help uncover and partly assess the extent and impact of sensor inaccuracies.

In summary, while a user could use existing tools to manually perform the extraction, analysis, and minig of an event series in a stream, StreamXPlore offers a specialized data-model and interface to facilitate this task.

5. EVALUATION

In this section, we present details of StreamXPlore's prototype implementation and report on preliminary results exploring a real sensor data stream.

System implementation. In our prototype, SQL Server 2008 serves as back-end stream store. Event identification is implemented by a Python script that executes a series of SQL queries. Algorithms 1 and 2 show the details of event extraction for the sliding- and tuple-windows respectively. The data-cube-based event refinement is performed by directly modifying the event specification, including the

Algorithm 1: SlidingWindow Event Extraction

Input: Event tuple stream S, filter predicate P^f, window size D_w, slide D_s, optional starting and ending timestamps T_s and T_e
Output: A list of sliding window events L

Let L be an empty list, let $T' = T_s$ or output of `SELECT MIN(timestamp) FROM` S, and let T_e be the output of `SELECT MAX(timestamp) FROM` S if T_e not given.
repeat
  Execute: `SELECT * FROM` S `WHERE timestamp>=` T' `AND timestamp<` $T' + D_w$ `and` P^f
  Assign output to new event E. Append E to L
  Let $T'=T' + D_s$
until $T' > T_e$;
Return L

Algorithm 2: TupleWindow Event Extraction.

Input: Event tuple stream S, filter predicate P^f, start predicate P^s, end predicate P^e, within size D_w, first-or-last flag F, optional start and end timestamps T_s and T_e
Output: A list of events L

If not given, initialize T_s and T_e to the beginning and end stream timestamps respectively;
Let L be an empty list; Let $T_{ee} = T_s$
Let L_s be the list of potential event start timestamps given by `SELECT timestamp FROM` S `WHERE` P^s `and` P^f `and timestamp` $> T_s$ `and timestamp` $< T_e$
foreach *timestamp* T_{es} *in* L_s *where* $T_{es} > T_{ee}$ **do**
  Let L_e be the output of `SELECT timestamp FROM` S `WHERE timestamp>=`T_{es} `and timestamp<`D_w`+`T_{es} `and timestamp<`T_e `and` P^f `and` P^e `order by timestamp`
  if *the first or last flag* F *is set to be first* **then**
    Set T_{ee} to be the first timestamp from L_e
  else
    Set T_{ee} to be the last timestamp from L_e
  Execute `SELECT * FROM` S `WHERE` P^f `and timestamp>=` T_{es} `and timestamp<`T_{ee}
  The resultset is an event E. Append E to L.
Return L

accompanying concept hierarchy. Vector distances are calculated using Euclidean distance. The clustering algorithm is K-means, which is borrowed from Scipy and Numpy.

Event Modeling Evaluation. We evaluate StreamXPlore using the HydroSense data stream [9]. This stream comprises water event records for different fixtures, such as showers, bath faucets, kitchen faucets, etc. collected in four houses (H1, H2, A1, A2) over a 33-day period from 2010-02-01 to 2010-03-05. The stream comprises 5956 primitive events with an average 42.5 events/day/house. In all experiments, events have in the order of hundreds of features.

The first question that we answer is **whether StreamXPlore can help identify different types of events in a series**, such as normal and abnormal events. We do not have any manually labeled ground-truth data. However, we observe that laundry days are one example of objectively defined abnormal days. These days are characterized by the use of the washing machine located in the laundry room. In the dataset, only data from H1, H2 and A2 have laundry events, 2 days for H2, 9 days for H1 and 4 days for A2. Therefore, we perform clustering only on the data for these three houses. We use a 24-hour sliding window as the event specification: (*i.e.*, `filter house='house name'; size 24h; slide 24h; start at 2010-02-01 00:00:00;`)

We set up separate hierarchies on the timestamp and fixture attributes. The timestamp hierarchy is the same for all the houses: it has individual hours at the leaves. Hours are grouped into time-periods of three to five hours, corresponding to the main periods of the day. The root is an entire 24 hour period. The fixture hierarchy is different for each house and it simply groups fixtures by room.

Table 2: Clustering Performance, the % numbers are the improvements comparing with the .base task

Task	Purity	F_2	RI	NMI
H1.base	0.73	0.65	0.48	0.05
H1.wk	0.88/21%	0.89/37%	0.56/17%	0.32/540%
H1.pd	0.85/16%	0.75/15%	0.61/27%	0.25/400%
H2.base	0.97	0.76	0.56	0.31
H2.wk	1.0/3%	0.4/-47%	0.94/68%	0.72/132%
H2.pd	1.0/3%	0.4/-47%	0.94/68%	0.72/132%
A2.base	0.88	0.69	0.55	0.06
A2.wk	0.88/0%	0.73/6%	0.46/-16%	0.13/117%
A2.pd	0.93/6%	0.82/19%	0.70/27%	0.30/400%

For all clustering tasks, we use K-means and set the number of clusters to four.[1] We use four because this number is larger than the two clusters we expect and there may be unexpected noise in the data.

For each house, we perform three clustering tasks to test the effect of event refinement (roll-up and slicing). '${house}.base' clusters events using the whole hierarchy structures. '${house}.wk' clusters events after removing the Kitchen sub-hierarchy (hierarchy slicing), thus dropping all features related to the Kitchen. We expect this operation to improve clustering quality, since we focus on laundry days, and thus kitchen events add noise to the clusters. Finally, '${house}.pd' is clustering after rolling-up the time hierarchy by one level, *i.e.*, the time hierarchy becomes: 0:24{ 0:5, 5:10, 10:14, 14:17, 17:20, 20:24}. This operation drops all the features that are the most detailed in time. We also expect this operation to improve cluster quality because laundry events are rare and sparsely distributed. The operation will not impact the laundry features significantly, while it will dramatically smooth other daily-routine features and thus reduce their impact during clustering.

Table 2 shows the clustering results. To measure the cluster quality, we use the following standard measures: purity, F_2, RI, and NMI [21]. As the table shows, in all three experiments, the clustering quality is high. As expected, refining events by rolling-up and ignoring unnecessary details or slicing and removing unnecessary features significantly improves the cluster quality overall.

The second question that we answer is **whether StreamXPlore can help identify appropriate parameters to use in a monitoring query**. As mentioned in Section 1, an important elder-care query is to identify abnormal night activities: "Raise an alert when the number of fixture utilizations between times T_1 and T_2 exceeds X". We use StreamXPlore to identify the three parameter values for house $H2$. For this, we find that a sliding-window specification for the [10pm,7am] time-frame every day, followed by a roll-up to get a total count of fixture utilizations every hour suffice to identify that most nights see no activity in the [2am,5am] time-period, three nights have only one event, and one night had an unusually higher three events. This scenario exemplifies StreamXPlore's ability to help a user quickly identify key query parameters ($X = 1$, $T_1 = 2$am, and $T_2 = 5$am in this example).

Finally, we **study the difference between sliding and**

[1]It is an open problem to select the optimum number of clusters in any clustering method both manually and automatically. We do not address this problem in this paper.

Table 3: System Performance

table size	duration of event extraction	#tuples	#answer events
4M	2s	18,028	32
100M	2s	546,584	999
1G	2m42s	5,408,400	9,899
10G	>32m	45,070,000	82,499

tuple windows using the morning routines of $H2$. A quick event extraction for [6am,10am] reveals that most mornings start with a bathroom event and are followed by bathroom and kitchen events. We thus define a tuple-window using the second sample query from Section 4. Scanning through the resulting 24 events is tedious. However, clustering the events readily reveals two high-level morning patterns: (1) 16 days see activity both in bathrooms and the kitchen within the first hour of the event, (2) the remaining 8 days comprise only bathroom activity for the first hour and then the kitchen starts being used. Interestingly, while StreamXPlore does not perfectly cluster events along these two patterns, it groups similar events together sufficiently to expose these key trends. Using a sliding- instead of a tuple-window returns an extra 9 days with no kitchen use in the morning (1 day with no morning activity), thus providing extra information. Events, however, become more difficult to compare because fixture uses occur at a variety of times rather than only within the first hour or two of the event start.

Performance. We assess whether our approach can support interactive exploration. Table 3 shows the time for our prototype to extract 24h sliding windows for H2 from an increasingly large, synthetically replicated HydroSense stream. Our prototype extracts events within a few seconds in streams up to 100 MB in size and also provides acceptable performance at 1 GB, which we posit should suffice for many applications. Performance drops for larger streams because StreamXPlore extracts each event using a separate SQL query. With a maximum of 1000 iterations and a 0.001 threshold, clustering takes 0.5s for 4MB, 18s for 100MB, 224s for 1GB, and 2724s for 10GB. However, k-means converges quickly in the initial rounds: 100 iterations produce acceptable results and reduce processing times to 1.65s for 100M, 21.98s for 1GB and 274.32s for 10GB.

6. CONCLUSION

We presented StreamXPlore, a system for data stream exploration. StreamXPlore's key contributions are (1) a new model for events on streams and (2) a system that enables the specification, extraction, analysis, and mining of these new types of events. Preliminary experiments on real data show that the approach is a promising step toward helping users identify relevant monitoring queries over sensor data.

Acknowledgments. We thank the anonymous reviewers for their helpful comments. This work is supported in part by the National Science Foundation through NSF grant IIS-0713123 and NSF CDI grant OIA-1028195.

7. REFERENCES

[1] Abadi et. al. . The design of the Borealis stream processing engine. In *Proc. of the Second CIDR Conf.*, Jan. 2005.

[2] Abadi et. al. Aurora: a new model and architecture for data stream management. *VLDB Journal*, 12:120–139, 2003.

[3] Ahuja et. al. Peta-scale data warehousing at Yahoo! In *Proc. of the SIGMOD Conf.*, pages 855–862, 2009.

[4] M. Balazinska, Y. Kwon, N. Kuchta, and D. Lee. Moirae: History-enhanced monitoring. In *Proc. of the Third CIDR Conf.*, Jan. 2007.

[5] Botan *et al.*. Extending XQuery with window functions. In *Proc. of the 33rd VLDB Conf.*, pages 75–86, 2007.

[6] Chandrasekaran et. al. TelegraphCQ: Continuous dataflow processing for an uncertain world. In *Proc. of the First CIDR Conf.*, Jan. 2003.

[7] G. Chatzopoulou, M. Eirinaki, and N. Polyzotis. Query recommendations for interactive database exploration. In *Proc. of the 21st SSDBM*, pages 3–18, 2009.

[8] Franklin et. al. Continuous analytics: Rethinking query processing in a network-effect world. In *Proc. of the Fourth CIDR Conf.*, 2009.

[9] Froehlich et. al. HydroSense: infrastructure-mediated single-point sensing of whole-home water activity. In *Proc. of the 11th Ubicomp Conf.*, pages 235–244, 2009.

[10] L. Golab, T. Johnson, J. S. Seidel, and V. Shkapenyuk. Stream warehousing with DataDepot. In *Proc. of the SIGMOD Conf.*, pages 847–854, 2009.

[11] 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 Mining and Knowledge Discovery*, 1:29–53.

[12] Han et. al. Stream cube: An architecture for multi-dimensional analysis of data streams. *Distrib. Parallel Databases*, 18:173–197, September 2005.

[13] V. Hristidis and Y. Papakonstantinou. Discover: keyword search in relational databases. In *Proc. of the 28th VLDB Conf.*, pages 670–681, 2002.

[14] Jagadish et. al. Making database systems usable. In *Proc. of the SIGMOD Conf.*, pages 13–24, 2007.

[15] K. S. Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation*, 28:11–21, 1972.

[16] N. Khoussainova, M. Balazinska, W. Gatterbauer, Y. Kwon, and D. Suciu. A case for a collaborative query management system. In *Fourth CIDR Conf. – Perspectives*, Jan. 2009.

[17] Y. Kwon, W. Y. Lee, M. Balazinska, and G. Xu. Clustering events on streams using complex context information. In *Proc. of the 4th MCD Workshop*, Dec. 2008.

[18] G. Li, J. Fan, H. Wu, J. Wang, and J. Feng. DBease: Making database user-friendly and easily accessible. In *Proc. of the Fifth CIDR Conf.*, pages 45–56, 2011.

[19] M. Liu, E. Rundensteiner, K. Greenfield, C. Gupta, S. Wang, I. Ari, and A. Mehta. E-cube: multi-dimensional event sequence analysis using hierarchical pattern query sharing. pages 889–900, 2011.

[20] Lo et. al. OLAP on sequence data. In *Proc. of the SIGMOD Conf.*, pages 649–660, 2008.

[21] C. D. Manning, P. Raghavan, and H. Schtze. *Introduction to Information Retrieval*. 2008.

[22] Motwani et. al. Query processing, approximation, and resource management in a data stream management system. In *Proc. of the First CIDR Conf.*, Jan. 2003.

[23] Tufte et. al. Travel time estimation using NiagaraST and latte. In *Proc. of the SIGMOD Conf.*, pages 1091–1093, 2007.

[24] E. Wu, Y. Diao, and S. Rizvi. High-performance complex event processing over streams. In *Proc. of the SIGMOD Conf.*, pages 407–418, 2006.

[25] M. M. Zloof. Query by example. In *Proc. of the May 19-22, 1975, nat. computer conf. and exp.*, AFIPS '75, pages 431–438, New York, NY, USA, 1975. ACM.