XStream: Explaining Anomalies in Event Stream Monitoring

Haopeng Zhang, Yanlei Diao, Alexandra Meliou
College of Information and Computer Sciences, University of Massachusetts Amherst
{haopeng, yanlei, ameli}@cs.umass.edu

ABSTRACT

In this paper, we present the XStream system that provides high-quality explanations for anomalous behaviors that users annotate on CEP-based monitoring results. Given the new requirements for explanations, namely, *conciseness*, *consistency with human interpretation*, and *prediction power*, most existing techniques cannot produce explanations that satisfy all three of them. The key technical contributions of this work include a formal definition of optimally explaining anomalies in CEP monitoring, and three key techniques for generating sufficient feature space, characterizing the contribution of each feature to the explanation, and selecting a small subset of features as the optimal explanation, respectively. Evaluation using two real-world use cases shows that XStream can outperform existing techniques significantly in conciseness and consistency while achieving comparable high prediction power and retaining a highly efficient implementation of a data stream system.

1. INTRODUCTION

Complex Event Processing (CEP) extracts useful information from large-volume event streams in real-time. Users define interesting patterns in a CEP query language (e.g,. [3, 4]). With expressive query languages and high performance processing power, CEP technology is now at the core of real-time monitoring in a variety of areas, including the Internet of Things [18], financial market analysis [18], and cluster monitoring [28].

However, today's CEP technology supports only *passive monitoring* by requesting the monitoring application (or user) to explicitly define patterns of interest. There is a recent realization that many real-world applications demand a new service beyond passive monitoring, that is, the ability of the monitoring system to identify interesting patterns (including anomalous behaviors), produce a concrete explanation from the raw data, and based on the explanation enable a user action to prevent or remedy the effect of an anomaly. We broadly refer to this new service as *proactive monitoring.* We present two motivating applications as follows.

1.1 Motivating Applications

Production Cluster Monitoring. Cluster monitoring is crucial to many enterprise businesses. For a concrete example, consider a production Hadoop cluster that executes a mix of Extract-Transform-Load (ETL) workloads, SQL queries, and data stream tasks. The programming model of Hadoop is MapReduce, where a MapReduce job is composed of a `map` function that performs data transformation and filtering, and a `reduce` function that performs aggregation or more complex analytics for all the data sharing the same key. During job execution, the map tasks (called mappers) read raw data and generate intermediate results, and the reduce tasks (reducers) read the output of mappers and generate final output. Many of the Hadoop jobs have deadlines because any delay in these jobs will affect the entire daily operations of the enterprise business. As a result, monitoring of the progress of these Hadoop jobs has become a crucial component of the business operations.

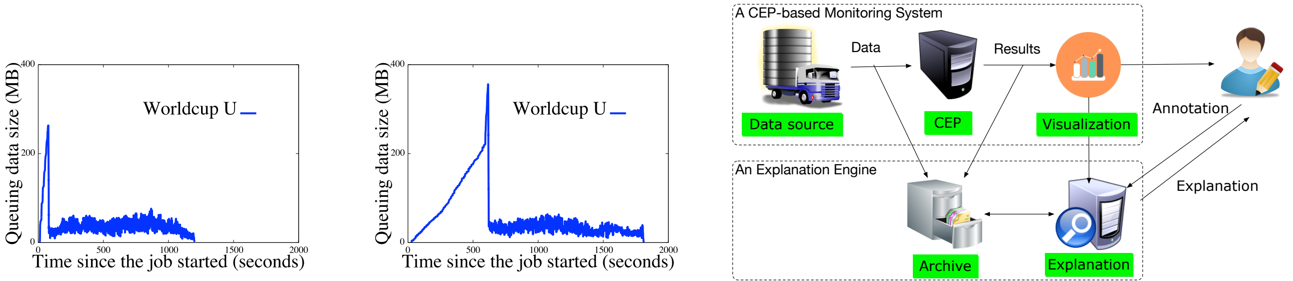
However, the Hadoop system does not provide sufficient monitoring functionality by itself. CEP technology has been shown to be efficient and effective for monitoring a variety of measures [28]. By utilizing the event logs generated by Hadoop and system metrics collected by Ganglia, CEP queries can be used to monitor Hadoop job progress; to find tasks that cause cluster imbalance; to find data pull stragglers; and to compute the statistics of lifetime of mappers and reducers. Consider a concrete example below, where the CEP query monitors the size of intermediate results that have been queued between mappers and reducers.

Example 1.1 (Data Queuing Monitoring)**.** Collect all the events capturing intermediate data generation/consumption for each Hadoop job. Return the accumulative intermediate data size calculated from those events (Q1).

Figure 1(a) shows the data queuing size of a monitored Hadoop job. The X-axis stands for the time elapsed since the beginning of the job, while the Y-axis represents the size of queued data. In this case, the job progress turns out to be normal: the intermediate results output by the mappers start to queue at the beginning and reach a peak after a short period of time. This is because a number of mappers have completed in this period while the reducers have not been scheduled to consume the map output. Afterwards, the queued data size decreases and then stabilizes for a long period of time, meaning that the mappers and reducers are producing and consuming data at constant rates, until the queued data reduces to zero at the end of the job.

Suppose that a Hadoop user sees a different progress plot, as shown Figure 1(b), for the same job on another day: there is a long initial period where the data queuing size increases gradually but continually, and this phase causes the job completion time to be delayed by more than 500 seconds. When the user sees the job with an odd shape in Figure 1(b), he may start considering the following questions:

- ▶ What is happening with the submitted job?
- ▶ Should I wait for the job to complete or re-submit it?



(a) Data queuing size of a normal Hadoop job

(b) Data queuing size of an abnormal Hadoop job

(c) Architecture of XStream

Figure 1: Hadoop cluster monitoring: examples and system architecture.

- ▶ Is the phenomenon caused by the bugs in the code or some system anomalies?
- ▶ What should I do to bring the job progress back to normal?

Today's CEP technology, unfortunately, does not provide any additional information that helps answer the above questions. The best practice is manual exploration by the Hadoop user: he can dig into the complex Hadoop logs and manually correlate the Hadoop events with the system metrics such as CPU and memory usage returned by a cluster monitoring tool like Ganglia [13]. If he is lucky to get help from the cluster administrator, he may collect additional information such as the number of jobs executed concurrently with his job and the resources consumed by those jobs.

For our example query, it turns out that the odd shape in Figure 1(b) is due to high memory usage of other programs in the Hadoop cluster. However, this fact is not obvious from the visualization of the user's monitoring query, Q1. It requires analyzing additional data beyond what is used to compute Q1 (which used data relevant only to the user's Hadoop job, but not all the jobs in the system). Furthermore, the discovery of the fact requires new tools that can automatically generate explanations for the anomalies in monitoring results such that these explanations can be understood by the human and lead to corrective / preventive actions in the future.

Supply Chain Management. The second use case is derived from an aerospace company with a global supply chain. By talking with the experts in supply chain management, we identified an initial set of issues in the company's complex production process which may lead to imperfect or faulty products. For instance, in the manufacturing process of a certain product the environmental features must to be strictly controlled because they affect the quality of production. For example, the temperature and humidity need to be controlled in a certain range, and they are recorded by the sensors deployed in the environment. However, if some sensors stop working, the environmental features may not be controlled properly and hence the products manufactured during that period can have quality issues. When such anomalies arise, it is a huge amount of work to investigate the claims from customers given the complexity of manufacturing process and to analyze a large set of historical data to find explanations that are meaningful and actionable.

1.2 Problem Statement and Contributions

The overall goal of XStream is to provide good explanations for anomalous behaviors that users annotate on CEP-based monitoring results. We assume that an enterprise information system has basic CEP monitoring functionality: a CEP monitoring system offers a dashboard to illustrate high-level metrics computed by a CEP query, such as job progress, network traffic, and data queuing. When a user observes an abnormal value in the monitoring results, he annotates the value in the dashboard and requests XStream to search for an explanation from the archived raw data streams. XStream generates an optimal explanation by quickly replaying a fraction of the archived data streams. Then the explanation can be encoded into the system for proactive monitoring for similar anomalies in the future.

Challenges. The challenges in the design of XStream arise from the requirements for such explanations. Informed by the two real-world applications mentioned above, we consider three requirements in this work: (a) *Conciseness*: The system should favor smaller explanations, which are easier for humans to understand. (b) *Consistency*: The system should produce explanations that are consistent with human interpretation. In practice, this means that explanations should match the true reasons for an anomaly (*ground truth*). (c) *Prediction power*: We prefer explanations that have predictive value for future anomalies.

It is difficult for existing techniques to meet all three requirements. In particular, prediction techniques such as logistic regression and decision trees [2] suffer severely in conciseness or consistency as shown in our evaluation results. This is because these techniques were designed for prediction, but not for explanations with conciseness and consistency requirements. Recent database research [27, 22] seeks to explain outliners in SQL query answers. This line of work assumes that explanations can be found by searching through various subsets of the tuples that were used to compute the query answers. This assumption does not suit real-world stream systems for two reasons: As shown for our example, Q1, the explanation of memory usage contention among different jobs cannot be generated from only those events that produced the monitoring results of Q1. Furthermore, the stream execution model does not allow us to repeat query execution over different subsets of events or perform any precomputation in a given database [22].

Contributions. In this work, we take an important step towards discovering high-quality explanations for anomalies observed in monitoring results. Toward this goal, we make the following contributions:

1) Formalizing explanations (Section 2): We provide a formal definition of optimally explaining anomalies in CEP monitoring as a problem that maximizes the information reward provided by the explanation.

2) Sufficient feature space (Section 3): A key insight in our work is that discovering explanations first requires a sufficient feature space that includes all necessary features for explaining observed anomalies. XStream includes a new module that automatically transforms raw data streams into a richer feature space, $\mathbf{F}$, to enable explanations.

3) Entropy-based, single-feature reward (Section 4): As a basis for building the information reward of an explanation, we model the reward that each feature, $f \in \mathbf{F}$, may contribute using a new entropy-based distance function.

4) Optimal explanations via submodular optimization (Section 5): We next model the problem of find an optimal explanation from the feature space, $\mathbf{F}$, as a submodular maximization problem. Since submodular optimization is NP-hard, we design a heuristic algorithm that ranks and filters features efficiently and effectively.

5) Implementation and evaluation (Section 6): We have implemented XStream on top of the SASE stream engine [3, 28]. Experiments using two real-world use cases show very promising results: (1) Our entropy distance function outperforms state-of-the-art distance functions on time series by reducing the features considered by 94.6%. (2) XStream significantly outperforms logistic regression [2], decision tree [2], majority voting [17] and data fusion [21] in consistency and conciseness of explanations while achieving comparable, high predication accuracy. Specifically, it outperforms others in consistency by 3201% on average, and reduces 90.5% of features on average to ensure conciseness. (3) Our implementation is very efficient: with 2000 concurrent monitoring queries, the triggered explanation analysis returns explanations within a minute and affects the performance only slightly, delaying events processing by less than 1 second.

2. EXPLAINING CEP ANOMALIES

The goal of XStream is to provide good explanations for anomalous behaviors that users annotate on CEP-based monitoring results. We first describe the system setup, and give examples of monitoring queries and anomalous observations that a user may annotate. We then discuss the requirements for providing explanations for such anomalies, and examine whether some existing approaches can derive explanations that fit these requirements. Finally, we define the problem of optimally explaining anomalies in our setting.

2.1 CEP Monitoring System and Queries

In this section, we describe the system setup for our problem setting. The overall architecture of XStream is shown in Figure 1(c). We consider a data source S, generating events of n types, $\mathbf{E} = \{E_1, E_2, \ldots, E_n\}$. Events of these types are received by the CEP-based monitoring system continuously. Each event type follows a schema, comprised of a set of attributes; all event schemas share a common timestamp attribute. The timestamp attribute records the occurrence time of each event. Figure 2 shows some example event types in the Hadoop cluster monitoring use case [28].

We consider a CEP engine that monitors these events using user-defined queries. For the purposes of this paper, monitoring queries are defined in the SASE query language [3], but this is not a restriction of our framework, and our results extend to other CEP query languages. Figure 3 shows the general syntax of CEP queries in SASE, and an example query, Q_1, from the Hadoop cluster monitoring use case. Q_1 collects all data-relevant events during the lifetime of a Hadoop job. We now explain the main components of a SASE query.

Sequence. A query Q may specify a sequence using the `SEQ` operator, which requires components in the sequence to occur in the specified order. One component is either a single event or the Kleene closure of events. For example, Q_1 specifies three components: the first component is a single event of the type *JobStart*; the second component is a Kleene closure of a set of events of the type *DataIO*; and the third component is a single event of type *JobEnd*.

Predicates. Q can also specify a set of predicates in its `Where` clause. One special predicate among these is the bracketed *partitionAttribute*. The brackets apply an equivalence test on the attribute inside, which requires all selected events to have the same value for this attribute. The *partitionAttribute* tells the CEP engine which attribute to partition by. In Q_1, *jobId* is the partition attribute.

Return matches. Q specifies the matches to return in the `Return` clause. Matches comprise a series of events with raw or derived attributes; we assume *timestamp* and the *partitionAttribute* are included in the returned events. We denote with m a match on one partition and with M_Q the set of all matches. Q_1 returns a series of events based on selected *DataIO* events, and the returned attributes include *timestamp*, *jobId*, and a derived attribute— the total size for all selected *DataIO* events. In order to visualize results in real time, matches will be sent to the visualization module as events are collected.

Visualizations and feedback. Our system visualizes matches from monitoring queries on a dashboard that users can interact with. The visualizations typically display the (relative) occurrence time on the X-axis. The Y-axis represents one of the derived attributes in returned events. Users can specify simple filters to focus on particular partitions. All returned events of M_Q are stored in a relational table T_{M_Q}, and the data to be visualized for a particular partition is specified as $\pi_{t,attr\_i}(\sigma_{partitionAttribute=v}(M))$. Figure 1(a) shows the visualization of a partition, which corresponds to a Hadoop job for this query. In this visualization, the X-axis displays the time elapsed since the job started, and the Y-axis shows the derived sum over the "DataSize" attribute.

Users can interact with the visualizations by annotating anomalies. For example, the visualization of Figure 1(b) demonstrates an unexpected behavior, with the queueing data size growing slowly. A user can drag and draw rectangles on the visualization, to annotate the abnormal component, as well as reference intervals that demonstrate normal behavior. We show an example of these annotations in Figure 4. A user may also annotate an entire period as abnormal, and choose a reference interval in a different partition. We use I_A to denote the annotated abnormal interval in a partition P_A: $I_A = (Q, [lower, upper], P_A)$. We use I_R to denote the reference interval, which can be explicitly annotated by the user, or inferred by XStream as the non-annotated parts of the partition. We write $I_R = (Q, [lower, upper], P_R)$, where P_R and P_A might be the same or different partitions.

Event type	**Meaning**	**Schema**
JobStart	Recording a Hadoop job starts	(timestamp, eventType, eventId, jobId, clusterNodeNumber)
JobEnd	Recording a Hadoop job finishes	(timestamp, eventType, eventId, jobId, clusterNodeNumber)
DataIO	Recording the activities of generation (positive values) / consumption (negative values) of intermediate data	(timestamp, eventType, eventId, jobId, taskId, attemptId, clusterNodeNumber, dataSize)
CPUUsage	Recording the CPU usage for a node in the cluster	(timestamp, eventType, eventId, clusterNodeNumber, CPUUsage)
MemUsage	Recording the memory usage for a node in the cluster	(timestamp, eventType, eventId, clusterNodeNumber, memUsage)

Figure 2: Example event types in Hadoop cluster monitoring. Event types can be specific to the Hadoop job (e.g., JobStart, DataIO, JobEnd), or they may report system metrics (e.g., CPUUsage, FreeMemory).

Q	Pattern $\textsc{seq}(Component_1,\ Component_2\ ,\ \ldots)$ Where $[partitionAttribute] \wedge Pred_1 \wedge Pred_2 \wedge \ldots$ Return $(timestamp,\ partitionAttribute,\ derivedA_1,$ $derivedA_2,\ \ldots)[]$

Q_1	Pattern $\textsc{seq}(\text{JobStart } a, \text{ DataIO+ } b[], \text{ JobEnd } c)$ Where [jobId] Return $(b[i]$.timestamp, a.jobId, $sum(b[1 \cdots \text{i}]$.dataSize))[]

Figure 3: Syntax of a query in SASE (on the left), and an example query for monitoring data activity (on the right).

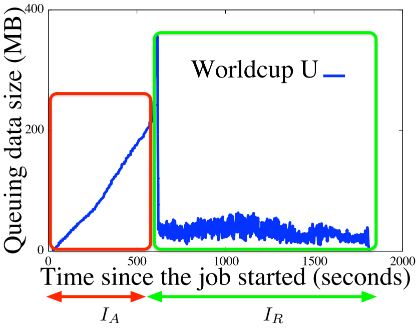


Figure 4: Abnormal (I_A) and reference (I_R) intervals.

2.2 Explaining Anomalies

Monitoring visualizations allow users to observe the evolution of various performance metrics in the system. While the visualizations help indicate that something may be unusual (when an anomaly is observed), they do not offer clues that point to the reasons for the unexpected behavior. In our example from Figure 4, there are two underlying reasons for the abnormal behavior: (1) the free memory is lower than normal, and (2) the free swap space is lower than normal. However, these reasons are not obvious from the visualization; rather, a Hadoop expert had to manually check a large volume of logs to derive this explanation. Our goal is to automate this process, by designing a system that seamlessly integrates with CEP monitoring visualizations, and which can produce explanations for surprising observations.

We define three desirable criteria for producing explanations in XStream:

1. **Conciseness**: The system should favor smaller, and thus simpler explanations. Conciseness follows the Occam's razor principle, and produces explanations that are easier for humans to understand.

2. **Consistency**: The system should produce explanations that are consistent with human interpretation. In practice, this means that explanations should match the true reasons for an anomaly (*ground truth*).

3. **Prediction power**: We prefer explanations that have predictive value for future anomalies. Such explanations can be used to perform *proactive monitoring.*

Explanations through predictive models. The first step of our study explored the viability of existing prediction techniques for the task of producing explanations for CEP monitoring anomalies. Prediction techniques typically learn a model from training data; by using the anomaly and reference annotations as the training data, the produced model can be perceived as an explanation. For now, we will assume that a sufficient set of features is provided for training (we discuss how to construct the feature space in Section 3), and evaluate the explanations produced by two standard prediction techniques for the example of Figure 4.

Logistic regression [2] produces models as weights over a set of features. The algorithm processes events from the two annotated intervals as training data, and the trained prediction model — a classifier between abnormal and reference classes — can be considered an explanation to the anomaly. The resulting logistic regression model for this example is shown in Figure 5. While the model has good predictive power, it is too complex, and cannot facilitate human understanding of the reported anomaly. The model assigns non-zero weights to 30 out of 345 input features, and while the two ground truth explanations identified by the human expert are among these features (23 and 24), their weights in this model are low. This model is too noisy to be of use, and it is not helpful as an explanation.

Decision tree [2] builds a tree for prediction. Each non-leaf node of the tree is a predicate while leaf nodes are prediction decisions. Figure 6 shows the resulting tree for our example. The decision tree algorithm selects three features for the non-leaf nodes, and only one of them is part of the ground truth determined by our expert. The other two features happen to be *coincidentally* correlated with the two intervals, as revealed in our profiling. This model is more concise than the result of logistic regression, but it is not consistent with the ground truth.

The above analyses showed that prediction techniques are not suitable for producing explanations in our setting. While the produced models have good predictive power (as this is what the techniques are designed for), they make poor explanations, as they suffer in consistency and conciseness.

No.	Feature	Weight
1	DataIOFrequency	-0.01376
2	CPUIdleMean	0.0089
3	PullFinishFrequency	-0.00708
4	ProcTotalMean	0.00085
5	PullStartTaskIdMean	-0.00046
...	...	...
20	MemBuffersMean	2.97E-05
21	PktsOutMean	-1.88E-05
22	MapPeriodMean	0.00001
23	SwapFreeMean	-4.79E-07
24	MemFreeMean	-3.28E-07
25	MemCachedMean	-2.42E-07
26	BytesOutMean	-1.02E-07
27	SwapTotalMean	-8.96E-08
28	BytesInMean	-3.85E-08
29	MemTotalMean	-3.69E-08
30	BoottimeMean	2.61E-10

Figure 5: Model generated by logistic regression for the annotated anomaly of Figure 4.

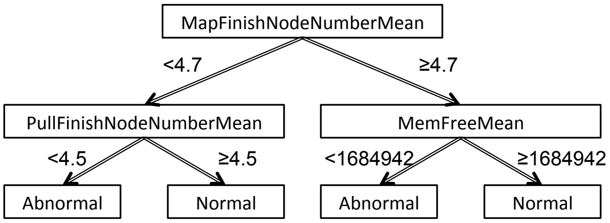


Figure 6: Model Generated by Decision Tree

Our goal is to design a method for deriving explanations that satisfies all three criteria (Figure 7).

2.3 Formalizing Explanations

Explanations need to be understandable to human users, and thus need to have a simple format. XStream builds explanations as a conjunction of predicates. In their general format, explanations are defined as follows.

Definition 2.1 (Explanation)**.** An explanation is a boolean expression in Conjunctive Normal Form (CNF). It contains a conjunction of clauses, each clause is a disjunction of predicates, and each predicate is of the form $\{v\ o\ c\}$, where v is a variable value, c is a constant, and o is one of five operators: $o \in \{>, \geq, =, \leq, <\}$.

Example 2.1. The formal form of the true explanations for the anomaly annotated in Figure 4 is ($MemFreeMean < 1978482 \wedge SwapFreeMean < 361462$), which is a conjunction of two predicates. It means that the average available memory is less than 1.9GB and free swap space is less than 360MB. The two predicates indicate that the memory usage is high in the system (due to resource contention), thus the job runs slower than normal.

Arriving at the explanation of Example 2.1 requires two non-trivial components. First, we need to identify important features for the annotated intervals (e.g., $MemFreeMean$, $SwapFreeMean$); these features will be the basis of forming meaningful predicates for the explanations. Second, we have to derive the best explanation given a metric of optimality. For example, the explanation ($MemFreeMean < 1978482$) is worse than ($MemFreeMean < 1978482 \wedge SwapFreeMean < 361462$), because, while it is smaller, it does not cover all issues that contribute to the anomaly, and is thus less consistent with the ground truth.

Algorithm	Conciseness	Consistency	Prediction quality
Logistic regression	Bad	Bad	Good
Decision tree	Ok	Bad	Good
Goal	Good	Good	Good

Figure 7: Performance of prediction methods on our three criteria for explanations.

Ultimately, explanations need to balance two somewhat conflicting goals: simplicity, which pushes explanations to smaller sizes, and informativeness, which pushes explanations to larger sizes to increase the information content. We model these goals through a reward function that models the information that an explanation carries, and we define the problem of deriving optimal explanations as the problem of maximizing this reward function.

Definition 2.2 (Optimal Explanation)**.** Given an archive of data streams D for CEP, a user-annotated abnormal interval I_A and a user-annotated reference interval I_R, an optimal explanation e is one that maximizes a non-monotone, submodular information reward R over the annotated intervals: $\text{argmax}_e\, R_{I_A,I_R}(e)$

The reward function in Definition 2.2 is governed by an important property: rewards are not additive, but *submodular*. This means that the sum of the reward of two explanations is greater than or equal to the reward of their union: $R_{I_A,I_R}(e_1) + R_{I_A,I_R}(e_2) \geq R_{I_A,I_R}(e_1 \cup e_2)$. The intuition for the submodularity property is based on the observation that adding predicates to an explanation offers diminishing returns: the larger an explanation already is, the lower the reward of adding a new predicate will be. Moreover, R is non-monotone. This means that adding predicates to an explanation *could decrease the reward*. This is due to the conciseness requirement that penalizes big explanations. The optimal explanation problem (Definition 2.2) is therefore a submodular maximization problem, which is known to be NP-hard [12].

2.4 Existing Approximation Methods

Submodular optimization problems are commonly addressed with greedy approximation techniques. We next investigate the viability of these methods for our problem setting.

For this analysis, we assume a reward function for explanations based on mutual information. Mutual information is a measure of mutual dependence between features. This is important in our problem setting, as features are often correlated. For example, $PullStartFrequency$ and $PullFinishFrequency$ are highly correlated, because they always appear together for every pull operation. For this precise reason, Definition 2.2 demands a submodular reward function. Mutual information satisfies the submodularity property. Greedy algorithms are often used in mutual information maximization problems. The way they would work in this setting is the following: given an explanation e, which is initially empty, at each greedy step, we select the feature f that maximizes the mutual information of $e \cup f$.

Figure 8 shows the performance of the greedy algorithm for maximization of mutual information (with two variants that select a different feature in case of ties), with a few

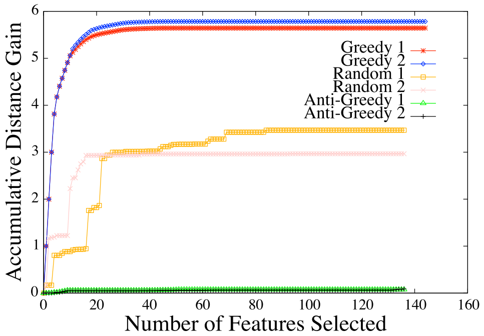


Figure 8: Accumulative mutual information gain under different strategies.

other strawman alternatives. The greedy strategy clearly outperforms the alternatives by reaching higher mutual information gains with fewer features, but it still selects a large number of features (around 20-30 features before it levels off). This means that this method produces explanations that are too large, and unlikely to be useful for human understanding.

2.5 Overview of the XStream Approach

Since standard approaches for solving the optimal explanation problem are insufficient for our problem setting, we develop a new heuristic method based on good intuitions to address the problem. We next provide a high-level overview of our approach in building XStream.

1. Sufficient feature space (Section 3): A key insight in our work is that discovering optimal explanations first requires a sufficient feature space that includes all necessary features for explaining observed anomalies. Our work differs fundamentally from existing work on discovering explanations from databases [27, 22]: First, XStream operates on raw data streams, as opposed to the data carefully curated and stored in a relational database. Second, XStream does not assume that the raw data streams carry all necessary features for explaining anomalous behaviors. In our above example, the feature, $SwapFreeMean$, captures average free swap space and it does not exist in Hadoop event logs or Ganglia output. Our system includes a module that automatically transforms raw data streams into a richer feature space, $\mathbf{F}$, to enable the discovery of optimal explanations.

2. Entropy-based, single-feature reward (Section 4): As a basis for building the information reward defined in Definition 2.2, we consider the reward that each feature, $f \in \mathbf{F}$, may contribute. To capture the reward in such a base case, we propose a new, entropy-based distance function that is defined on a single feature across the abnormal interval, I_A, and the reference interval, I_R. The larger the distance, the more differentiating power over the two intervals that the feature contributes, and hence more reward produced.

3. Optimal explanations via submodular optimization (Section 5): The next task is to find an optimal explanation from the feature space, $\mathbf{F}$, that maximizes the information reward provided by the explanation. The reward function in Definition 2.2 is non-monotone and submodular, resulting in a submodular maximization problem. Since submodular optimization is NP-hard, our goal is to design a heuristic to solves this problem. Our heuristic algorithm first uses the entropy-based, single-feature reward to rank features, subsequently identifies a cut-off to reject features with low reward, and finally uses correlation-based filtering to eliminate features with information overlap (emulating the submodularity property). Our evaluation shows that our heuristic method is extremely effective in practice.

timestamp	node	usagePercent
4	2	35
5	5	49
6	8	99
7	1	86
8	2	61
9	6	43

Figure 9: Sample events in the type of $CPUUsage$.

3. DISCOVERING USEFUL FEATURES

Explanations comprise of predicates on measurable properties of the CEP system. We call such properties *features*. Some features for our running example are $DiskFreeMean$, $MemFreeMean$, $DataIOFrequency$, etc. In most existing work on explanations, features are typically determined by the query or the schema of the data (e.g., the query predicates in Scorpion [27]). In CEP monitoring, using as features the query predicates or data attributes is not sufficient, because many factors that impact the observed performance are due to other events and changes in the system. This poses an additional challenge in our problem setting, as the set of relevant features is non-trivial. In this section, we discuss how XStream derives the space of features as a first step to producing explanations.

In an explanation problem, we are given an anomaly interval I_A and a reference interval I_R; the relevant features for this explanation problem are built from events that occurred during these two intervals. To support the functionality of providing explanations, the CEP system has to maintain an archive of the streaming data. The system has the ability to purge archived data after the relevant monitoring queries terminate, but maintaining the data for longer can be useful, as the reference interval can be specified on any past data.

Formally, the events arriving in a CEP system in input streams and the generated matches compose the input to the feature space construction problem. We assume that the CEP system maintains a table for each event type, such as the one depicted in Figure 9. That is, for each event type E_i, logically there is a relational table $R(E_i)$ to store all events of this type in temporal order. There is also a table $R(M)$ to archive all match events in type M. Let D denote the database for XStream, which is composed of those tables. So D is defined as $D = \{R(E_i)|1 \leq i \leq n\} \cup R(M)$.

Each attribute in event type E_i, except the timestamp, forms a time series in a given interval (which can be an anomaly interval I_A or a reference interval I_R). Such time series as features are called *raw features*.

Example 3.1. The table of Figure 9 records events of type $CPUUsage$ in a given time interval [4, 9], and forms two raw features, from two time series. The first one is $CPUUsage.Node$, and its values are ((4, 2), (5,5), (6,8), (7,1), (8,2), (9,6)); the other is $CPUUsage.UsagePercent$ with values ((4,35), (5,49), (6,99), (7,86), (8,61), (9,43)).

We found that the raw feature space is not good for deriving explanations due to noise. Instead, we need higher-level

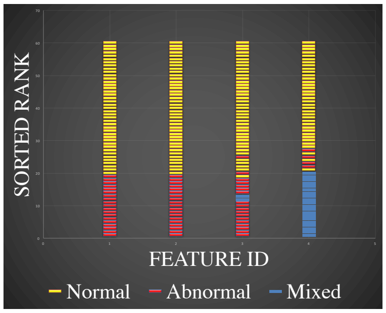


Figure 10: Visualization of the separating power of four features: (1) free memory size, (2) idle CPU percentage, (3) CPU percentage used by IO, and (4) system load. This visualization is not part of XStream, but we show it here for exposition purposes.

features, which we construct by applying aggregation functions to features at different granularities. We apply sliding windows over the time series features and over each window, aggregate functions including *count* and *avg* to generate new time serious features. The XStream system has an open architecture that allows any window size and any new aggregate functions to be used in the feature generation process. Features produced this way are “smoothed” time series; they demonstrate more general trends than raw features, and outliers are smoothed. Example high-level features that we produce by applying aggregations over windows on the raw features are $DataIOFrequency$ and $MemFreeMean$.

4. SINGLE-FEATURE REWARD

In this section, we present the core of our technique: an entropy-based distance function that models the reward of a single feature. We first discuss the intuition and requirements for this function, we then discuss existing, state-of-the-art distance functions and explain why they are not effective in this setting, and, finally, we present our new entropy-based distance metric.

4.1 Motivation and Insights

In seeking explanations for CEP monitoring anomalies, users contrast an anomaly interval with a reference interval. An intuitive way to think about the different behaviors in the two intervals is to consider the differences in the events that occur within each interval. We can measure this difference per feature: how different is each feature between the reference and the anomaly. Each feature is a vector of values, a time series, and our goal is to measure the distance between the time series of a feature during the abnormal interval and the time series of the same feature during the normal interval.

To explain one of the desirable properties of the distance function, we visualize a feature as follows: We order the values of a feature in increasing order and assign a color to each value; red for values that appear in the abnormal interval only, yellow for values that appear in the normal interval only, and blue for values that appear in both normal and abnormal intervals. Figure 10 shows this visualization for 4 different features. In this figure, we note that the first 2 features show a clear separation of values between the normal and abnormal periods. The third feature has less clear separation, but still shows the trend that lower values are more likely to be abnormal. Finally, the fourth feature is mixed for a significant portion of values.

Intuitively, the first two features in Figure 10 are better explanations for the anomaly, and thus have higher reward. The first feature means when the anomalies occur, the free memory size is relatively low, while during the reference interval the free memory size is relatively high. The second feature means that during the abnormal interval, idle CPU percentage is low while it is high during the reference interval. The unclear separation of the other two features, in particular the blue segments, indicate randomness between the two intervals, making them less suitable to explain the annotated anomalies.

This example provides insights on the properties that we need from the distance function: it should favor clear separation of normal and abnormal values, and it should penalize features with mixed segments (values that appear in both normal and abnormal periods). Therefore, the reward of a feature is high if the feature has good separating power, and it is lower with more segmentation in its values.

4.2 Existing State of the Art

Distance functions measuring similarities of time series have been well studied [26], and there is over a dozen distance functions in the literature. However, these metrics were designed with different goals in mind, and they do not fit our explanation problem well. We discuss this issue for the two major categories of distance functions [26].

Lock-step measure. In the comparison of two time series, lock-step measures compare the ith point in one time series to exactly the ith point in another. Such measures include the Manhattan distance (L_1), Euclidean distance (L_2) [10], other L_p-norms distances and approximation based $DISSIM$ distance. Those distance functions treat each pair of points independently, but in our case, we need to compare the time series holistically. For example, assume four simple time series: $TS_1 = (1, 1, 1)$, $TS_2 = (0, 0, 0)$, $TS_3 = (1, 0, 1)$ and $TS_4 = (0, 1, 0)$. Based on our separating power criterion, $D(TS_1, TS_2)$ should be larger than $D(TS_3, TS_4)$ because there is a clear separation between the values of TS_1 and TS_2, while the values of TS_3 and TS_4 are conflicting. However, applying any of the lock-step measures produces $D(TS_1, TS_2) = D(TS_3, TS_4)$.

Elastic measure. Elastic measures allow comparison of one-to-many points to find the minimum difference between two time series. These measures try to compare time series on overall patterns. For example, Dynamic Time Warping (DTW) tries to stretch or compress one time series to better match another time series; while Longest Common SubSequence(LCSS) is based on the longest common subsequence model. Although these measures also take value difference into account, the additional emphasis on pattern matching makes them ill-suited for our problem.

Both lock-step and elastic measures fall in the category of sequence-based metrics. This means that they consider the order of values. Lock-step functions perform strict step-by-step, or event-by-event comparisons; such rigid measures cannot find similarities in the flexible event series of our problem setting. Elastic measures allow more flexibility, but the emphasis on matching the microstructure of sequences introduces too much randomness in the metric.

In our case, temporal ordering is not important, because we assume the sample points in time series are independent.

This makes set-based functions a better fit (as opposed to sequence-based). Set-based functions measure the macro trend while smoothing low-level details.

4.3 Entropy-Based Single-Feature Reward

Since existing distance functions are not suitable to model single-feature rewards, we design a new distance function that emphasizes the separation of feature values between normal and abnormal intervals (Section 4.1). Our distance function is inspired by an entropy-based discretization technique [11], which cuts continuous values into value intervals by minimizing the class information entropy. The segmentation visualized in Figure 10, shows an intuitive connection with entropy: The more mixed the color segments are, the higher the entropy (i.e., more bits are needed to describe the distribution). We continue with some background definitions, and then define our entropy-based distance function, which we will use to model single-feature rewards.

Definition 4.1 (Class Entropy)**.** Class entropy is the information needed to describe the class distributions between two time series. Given a pair of time series, TS_A and TS_R, belonging to the abnormal and reference classes, respectively. Let $|TS_A|$ and $|TS_R|$ denote the number of points in the two time series, let $p_A = \frac{|TS_A|}{|TS_A|+|TS_R|}$, and let $p_R = \frac{|TS_R|}{|TS_A|+|TS_R|}$. Then, the entropy of the class distribution is:

$$H_{Class}(f) = p_A * log(\frac{1}{p_A}) + p_R * log(\frac{1}{p_R}) \tag{1}$$

Definition 4.2 (Segmentation Entropy)**.** Segmentation entropy is the information needed to describe how merged points are segmented by class labels. If there are n segmentations, and p_i represents the ratio of data points included in the ith segmentation, the segmentation entropy is:

$$H_{Segmentation} = \sum_{i=1}^{n} p_i * log(\frac{1}{p_i}) \tag{2}$$

Complicated segmentations in a feature result in more entropy. When there is a clear separation of the two classes, as in the first two features of Figure 10, then the segmentation entropy is the same as the class entropy. If there are more segments, the segmentation entropy is more than the class entropy.

Penalizing for mixed segments. Segmentation entropy captures the segmentation of the normal and abnormal classes, but does not penalize mixed segments with values that appear in both classes (blue segments in the visualization). Take an extreme case, where all values appear in both classes (single mixed segment). This is the scenario with the worst separation power, but its segmentation entropy is 0, because it is treated as a single segment. This indicates that we need special treatment for mixed (blue) segments.

We assume the worst case distribution of normal and abnormal data points within the segment. This is the uniform distribution, which leads to most segmentation and highest entropy. For example, if a mixed segment c consists of 5 data points, 3 contributed from the normal class (N) and 2 contributed from the abnormal class (A), distributing them uniformly leads to 5 segments: (N,A,N,A,N). We denote this worst-case ordering of segment c as c^*. We assign a penalty term for each segment c, which is equal to the segmentation entropy of its worst-case ordering, c^*: $H_{Segmentation}(c^*)$. We thus define the regularized segmentation entropy:

$$H^+_{Segmentation} = H_{Segmentation} + \sum_{j=1}^{m} H_{Segmentation}(c^*_j) \tag{3}$$

The first term in this formula is the segmentation entropy of the feature, and the second term sums the regularization penalties of all mixed segments (m).

Accounting for feature size. Features may be of different sizes, as different event types may occur more frequently than others. The segmentation entropy is only comparable between two features f_1, f_2, if $|f_1.TS_A| = |f_2.TS_A|$ and $|f_1.TS_R| = |f_2.TS_R|$. However this does not hold for most features. To make these metrics comparable, we normalize segmentation entropy using class entropy and get the following definition for our entropy-based feature distance:

$$D(f) = \frac{H_{Class}(f)}{H^+_{Segmentation}(f)} \tag{4}$$

We use this distance function as a measure of single-feature reward. Features with perfect separation, such as the first two features of Figure 10, have reward equal to 1. Features with more complex segmentation have lower rewards. For the 4 features displayed in Figure 10, the rewards are 1, 1, 0.31, and 0.18, respectively.

5. CONSTRUCTING EXPLANATIONS

The entropy-base single-feature reward identifies the features that best distinguish the normal and abnormal periods. However, ranking the features based on this distance metric is not sufficient to generate explanations. We first have to address three additional challenges. First, it is not clear how to select a set of features from the ranked list. There is no specific constant k for selecting a set of top-k features, and moreover, such a set would likely not be meaningful as a top-k set is likely to contain highly-correlated features with redundant information. Second, there are cases where large distances are coincidental, and not associated with anomalies. Third, the rewards are computed for each feature individually, and due to submodularity, they are not additive. Determining how to combine features into an explanation requires eliminating redundancies due to feature correlations.

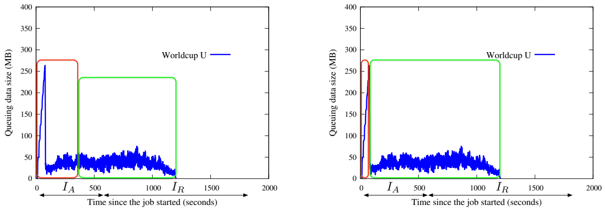
We proceed to describe the XStream approach to constructing explanations by addressing these challenges in three steps. Each step filters the feature set to eliminate features based on intuitive criteria, until we are left with a high-quality explanation.

5.1 Step 1: reward leap filtering

The entropy-based single-feature distance function produces a ranking of all features based on their individual rewards. Sharp changes in the reward between successive features in the ranking indicate a semantic change: Features that rank below a sharp drop in the reward are unlikely to contribute to an explanation. Therefore, features whose distance is low, relatively to other features, can be safely discarded.

5.2 Step 2: false positive filtering

It is possible for features to have high rewards due to reasons unrelated to the investigated anomaly. For example, a



(a) Temporal alignment (b) Point-based alignment

Figure 11: Two ways of alignment

feature that measures system uptime can have strong separating power between the annotated anomaly and reference regions (e.g., the anomaly is before the reference), but this is simply due to the nature of the particular feature, and it is not related to the anomaly. We call these features false positives. Our method for identifying and purging such false positives leverages other partitions (e.g., other Hadoop jobs in our running example). The intuition is that if a feature is a false positive, the feature will demonstrate similar behavior in other partitions without an indication of anomaly.

Identifying related partitions. We search the archived streams to identify similar partitions. Intuitively, such partitions should be results generated by the same query, monitoring the same Hadoop program, on the same dataset. XStream maintains a record of partitions in a partition table to facilitate fast retrieval. The partition table contains dimension attributes that record categorical information about the partition (e.g., $CEP-QueryID$, $HadoopJobName$, $Dataset$), and measure attributes that record partition statistics (e.g., monitoring duration, number of points). The system identifies related partitions, as those that match the dimension attributes.

Partition alignment. Once it discovers related partitions, XStream needs to map the annotated regions to each related partition. This alignment can be temporal-based or point-based. In temporal-based alignment, an annotation is mapped to a partition based on its temporal length. For example, in Figure 4, the abnormal period occupies 31% of temporal length; this annotation will align with the the first 31% of the temporal length in a related partition (Figure 11(a)). In point-based alignment an annotation is mapped to a partition based on the ratio of data points that it occupies in the monitoring graph. For example, the annotated high-memory usage partition of Figure 4 includes 113,070 points, with 2116 points falling in the abnormal annotation; this annotation will align with the first equal fraction of points in a related partition (Figure 11(b)). XStream selects the alignment for which the two partitions have the smallest relative difference. For example, if a related partition has 10% more points, but is 50% longer in time compared to the annotated partition, point-based alignment is preferred.

Interval labeling. Alignment maps the annotations to all related partitions. Now, these new annotations need to be labeled as normal or abnormal. XStream assigns labels through hierarchical clustering: a period that is placed in the same cluster as the annotated anomaly is labeled as abnormal. The clustering uses two distance functions: entropy-based, and normalized difference of frequencies. Periods whose cluster is far from the anomaly cluster are labeled as normal (reference). Finally, periods that cannot be assigned with certainty are discarded and not used later for validation.

Feature	Reward (annotated)	Reward (all)
Free memory size	1	0.77
Hadoop DataIO size	1	0.64
Num. of processes	1	0.64
Free swap size	1	1
Cached memory size	0.81	0.77
Buffer memory size	0.65	0.72

Figure 12: The six validated features after the removal of false positives.

In Figure 11(b), both intervals are assigned a "Reference" label. The left one is "Reference" because its frequency is significantly different from the annotated one (3.7 vs. 50.1); while the right one is "Reference" because both its frequency and value difference are quite small, meaning it is similar to the annotated "Reference" interval.

Feature validation. The process of partition discovering and automatic labeling generates a lot more labeled data that helps XStream filter out false positives, and improve the current set of features. Features that have high entropy reward on the annotated partition will be reevaluated on the large dataset. If the high reward is validated in the larger dataset as well, the feature is maintained; otherwise, it is discarded. In our running example, after the validation step, only 6 out of 670 features remain. Figure 12 shows the reward for each of these 6 features for the annotated partition and the augmented partition set.

5.3 Step 3: filtering by correlation clustering

After the validation step, we are usually left with a small set of features, which have high individual rewards, and the high rewards are likely related to the investigated anomaly. However, it is still possible that several of these features have information overlap. For example, two identical features, are good individually, but putting them together in an explanation does not increase the information content. We identify and remove correlated features using clustering.

We use pairwise correlation to identify similar features. We represent a feature as a node; two nodes are connected, if the pairwise correlation of the two features exceeds a threshold. We treat each connected component in this graph as a cluster, and select only one representative feature from each cluster. In our running example, the final six features are clustered into two clusters, one cluster with a single node, and another cluster with five nodes. Based on this result, the final explanation has two features.

5.4 Building final explanations

Once we make the final selection of features, the construction of an explanation is straightforward. For each selected feature, we can build a partial explanation in the format defined in Section 2.3. The feature name becomes the variable name. The value boundaries for the abnormal intervals become the constants. If a feature offers perfect separation during segmentation (Section 4), there is one boundary and only one predicate is built: e.g., the abnormal value range of feature f_1 is $(-\infty, 10]$, then the predicate is $f_1 \leq 10$. If a feature has more than one abnormal intervals, then multiple predicates are built to compose the explanation: e.g., the abnormal value ranges of feature f_2 are $(-\infty, 20]$,$[30, 50]$, and then the explanations are $f_2 \leq 20 \vee (f_2 \geq 30 \wedge f_2 \leq 50)$. Then we simply connect the partial explanations constructed from

No.	Anomaly	**Hadoop workload**
1	High memory	WC-frequent users
2	High memory	WC-sessions
3	Busy Disk	WC-frequent users
4	High High CPU	WC-frequent users
5	High High CPU	WC-sessions
6	Busy High CPU	Twitter trigram
7	High Busy Network	WC-sessions
8	High Busy Network	Twitter trigram

Figure 13: Workloads for evaluating the explanations returned by XStream.

different features using conjunction and write the final formula into the conjunctive normal form.

6. EVALUATION

We have implemented XStream on top of the SASE stream engine [3, 28]. In this section, we evaluate XStream on the conciseness, consistency, and prediction power of its returned explanations, and compare its performance with a range of alternative techniques in the literature. We further evaluate the efficiency of XStream when the explanation module is run concurrently with monitoring queries in an event stream system.

6.1 Experimental Setup

In our first use case, we monitored a Hadoop cluster of 30 nodes which was used intensively for experiments at the University of Massachusetts Amherst. To evaluate XStream for explaining anomalous observations, we used three Hadoop jobs: (A) Twitter Trigram: count trigrams in a twitter stream; (B) WC-Frequent users: find frequent users in a click stream; (C) WC-session: sessionization over a click stream. To enable the ground truth for evaluation, we manually created four types of anomalies by running additional programs to interfere with resource consumption: (1) High memory usage: the additional programs use up memory. (2) High CPU: the additional programs keep CPU busy. (3) Busy disk: the programs keep writing to disk. (4) Busy network: the programs keep transmitting data between nodes. By combining the anomaly types and Hadoop jobs, we create 8 workloads listed in Figure 13. The ground truth features are verified by a Hadoop expert.

Our second use case was supply chain management of an aerospace company. Due to confidentiality issues we were unable to get real data. Instead, we built a simulator to generate manufacturing data and anomalies such as faulty sensors and subpar material. Since both use cases generate similar results, we report results using the first use case in this section and refer the reader to Sec **??** for results of the second use case.

All of our experiments were run on a server with two Intel Xeon 2.67GHz, 6-core CPUs and 16GB memory. XStream is implemented in Java and runs on Java HotSpot 64-bit server VM 1.7 with the maximum heap size set to 8GB.

6.2 Effectiveness of Explanations by XStream

We compare XStream with a range of alternative techniques. We use **decision trees** to build explanations based on the latest version of weka, and **logistic regression** based on a popular R package. We consider two additional techniques, **majority voting** [17] and **data fusion** [21]. Both techniques make full use of every feature, and make prediction based on all features. Majority voting treats features equally and uses the label which counts the most as the prediction result. The fusion method fuses the prediction result from each feature based on their precision, recall and correlations. We compare these techniques on three measures: (1) consistency: selected features as compared against ground truth; (2) conciseness: the number of selected features; (3) prediction accuracy when the explanation is used as a prediction model on new test data.

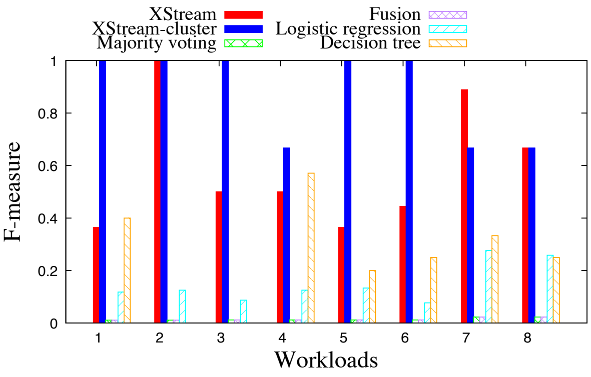


Figure 14: Consistency comparison

Consistency. First we compare the selected features of each algorithm with the ground truth features. The results are shown in Figure 14. X-axis represents different workloads (1 - 8), while Y-axis is the F-measure, namely, the harmonic mean of precision and recall regarding the inclusion of ground truth features in the returned explanations. XStream represents our results before applying clustering on selected features, while XStream-cluster represents results clustered by correlations (Section 5). We can see that XStream-cluster works better than XStream without clustering for most of workloads, and XStream-cluster provides much better quality than the alternative techniques. Majority voting and fusion do not select features, and hence their F-measures are low. Logistic regression and decision tree generate models with selected features, with sightly increased F-measures but still significantly below those of XStream-cluster.

Conciseness. Figure 15 shows the sizes of explanations from each solution. Here the Y-axis (in logarithmic scale) is the number of features selected by each solution, where the total number of available features is 345. "Ground truth" represents the number of features in ground truth, while "Ground truth cluster" represents the number of clusters after we apply clustering on the contained features. Again, majority voting and fusion do not select features, so the size is the same as the size of feature space. The models of logistic regression includes 20 - 30 features, which is roughly 10 times of the ground truth. Decision trees are more concise with less than 10 features selected. Overall, XStream outperforms other algorithms, and is quite close to the number of features in ground truth cluster.

Predication accuracy. In Figure 16 we compare the prediction accuracy of each method. The Y-axis represents F-measure for prediction over new test data. The F-measures of XStream, logistic regression and decision tree are quite stable, most of time above 0.95. Data fusion and majority voting fluctuate more. Overall, our method can provide consistent high-quality prediction power.

Effectiveness of the distance function. We finally

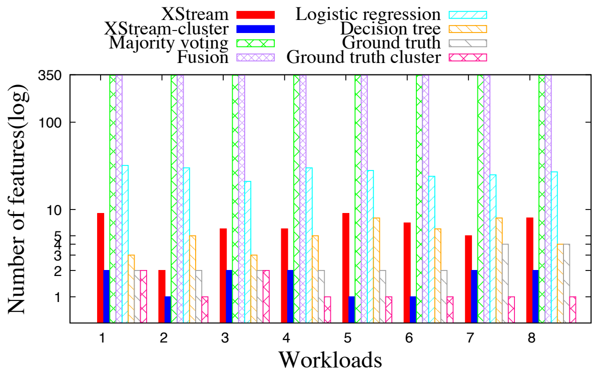


Figure 15: Conciseness comparison

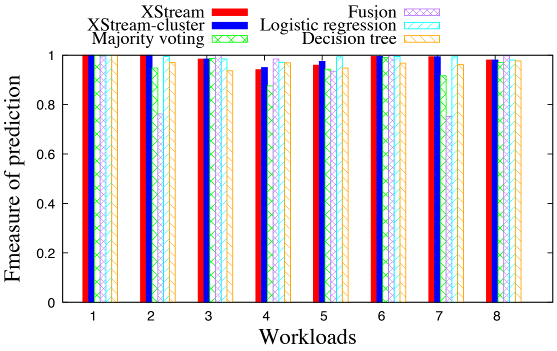


Figure 16: Prediction power comparison

demonstrate the effectiveness of our entropy-based distance function by comparing it with a set of existing distance functions [26] for time series: (1) Manhattan distance, (2) Euclidean distance, (3) DTW, (4) EDR, (5) ERP and (6) LCSS.

The results are shown in Figure 17. In each method, all available features are sorted by the distance function of choice in decreasing order. We measure the number of features retrieved from each sorted list in decreasing order in order to cover all the features in the ground truth, shown as the Y-axis. We see that our entropy distance is always the one using the minimum number of features to cover the ground truth. LCSS works well in the first two workloads, but it works poorly for workloads 3, 4, 5 and 6. This is because the ground truth features for the first two work loads have perfect separating power based on LCSS distance, while in other workloads they contain some signals. So LCSS is not as robust as our distance function. Other distance functions use large number of features all the time.

Summary. Our explanation algorithm outperforms other techniques in consistency and conciseness while achieving comparable, high predication accuracy. Specifically, XStream outperforms others in consistency by 3201% on average, up to 6037%. For conciseness, XStream reduces 90.5% of features on average, up to 99.5%. XStream is as good as other techniques on prediction quality: its F-measure on prediction is only slightly worse than logistic regression by 0.4%, while it is 3.3% higher than majority voting, 6.1% percent higher than fusion, and 1.9% higher than decision tree.

Our entropy distance function works better than existing distance functions on time series. It reduces the size of explanation by 94.6% on average, up to 97.2%, than other functions in order to cover the ground truth features.

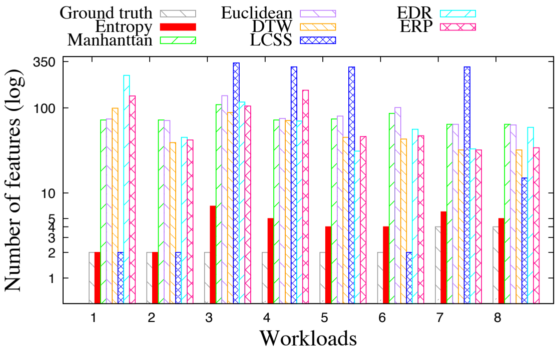


Figure 17: Distance function comparison

6.3 Efficiency of XStream

We further evaluate the efficiency of XStream. Our main result shows that our implementation is highly efficient: with 2000 concurrent monitoring queries, triggered explanation analysis returns explanations within a minute and affects the performance only slightly, delaying events processing by less than 1 second. Additional details are available at Sec C.

7. RELATED WORK

In the previous section, we compared our entropy distance with a set of state-of-the-art distance functions [26] and compared our techniques with prediction techniques including decision trees and logistic regression [2]. In this section we survey broadly related work.

CEP systems. There are a number of CEP systems in the research community [8, 19, 1, 23, 25]. These systems focus on passive monitoring using CEP queries by providing either more powerful query languages or better evaluation performance. Existing CEP techniques do not produce explanations for anomalous observations.

Explaining outliers in SQL query results. Scorpion [27] explains outliers in group-by aggregate queries. Users annotate outliers on the results of group-by queries, and then scorpion searches for predicates that remove these outliers while minimally affect the normal answers. It does not suit our problem because it works only for group-by aggregation queries and it searches through various subsets of the tuples that were used to compute the query answers. As shown for our example, Q1, the explanation of memory usage contention among different jobs cannot be generated from only those events that produced the monitoring results of Q1. Recent work [22] extends Scropion by supporting richer and insightful explanations by pre-computation and thus enables interactive explanation discovery. This work assumes a set of explanation templates given by the user and requires precomputation in a given database. Neither of the assumptions fits our problem setting.

Explaining outputs in iterative analytics. Recent work [7] focuses on tracking, maintaining, and querying lineage and "how" provenance in the context of arbitrary iterative data flows. It aims to create a set of recursively defined rules that determine which records in a data-parallel computation inputs, intermediate records, and outputs require explanation. It allows one to identify when (i.e., the points in the computation) and how a data collection changes, and provides explanations for only these few changes.

Data summarization. Given a table, recent work [9] constructs an explanation table and finds patterns affecting a binary value of each tuple. This problem is different from ours because it tries to summarize existing information and compute some statistical information, instead of discovering features which clearly separate normal and abnormal intervals in CEP monitoring.

Set-based distance function for time series. In addition to the lock-step and elastic distance functions we have compared with, Time series are also transformed into sets [20] for measurement. However, the design goal of the set-based function is to speed up the computation of existing elastic distance, so it is different from our entropy based distance function.

Anomaly detection. Anomaly detection or outlier detection [6, 14, 24] has been studied intensively in the data mining community. Researchers are still contributing actively on this area to achieve more efficient detection algorithms [15, 5]. However, the set of techniques in this area do not fit our problem. There are two main approaches. One is using a prediction model, which is learned on labeled or unlabeled data. Then incoming data is compared against with expected value by the model. If the difference is significant, the point or time series will be reported as outlier. The other approach is using distance functions, and outliers are those points or time series far from normal values. Both approaches report only outliers so that users know those outliers are different from normal values, but not the specific reasons (explanations) why they are different.

8. CONCLUSIONS

In this paper, we present XStream, a system that provides high-quality explanations for anomalous behaviors that users annotate on CEP-based monitoring results. Formulated as a submodular optimization problem, which is hard to solve, we provide a new approach that integrates a new entropy-based distance function and effective feature ranking and filtering methods. Evaluation results show that XStream outperforms existing techniques significantly in conciseness, by 3201% on average, and consistency, by 90.5% reduction of features needed, while achieving comparable high prediction power and retaining a highly efficient implementation of a data stream system.

To enable proactive monitoring in CEP systems, our future work will address temporal correlation in discovering explanations, automatic recognition and explanation of anomalous behaviors, and exploration of richer feature space to enable complex explanations.

9. REFERENCES

[1] D. J. Abadi, D. Carney, U. Çetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S. Zdonik. Aurora: a new model and architecture for data stream management. *The VLDB JournalâĂŤThe International Journal on Very Large Data Bases*, 12(2):120–139, 2003.

[2] C. C. Aggarwal. *Data Mining: The Textbook.* Springer Publishing Company, Incorporated, 2015.

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

[4] R. S. Barga, J. Goldstein, M. Ali, and M. Hong. Consistent streaming through time: A vision for event stream processing. *arXiv preprint cs/0612115*, 2006.

[5] L. Cao, J. Wang, and E. A. Rundensteiner. Sharing-aware outlier analytics over high-volume data streams. In *Proceedings of the 2016 International Conference on Management of Data*, pages 527–540. ACM, 2016.

[6] V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A survey. *ACM Computing Surveys (CSUR)*, 41(3):15, 2009.

[7] Z. Chothia, J. Liagouris, F. McSherry, and T. Roscoe. Explaining outputs in modern data analytics. *Proceedings of the VLDB Endowment*, 9(4), 2015.

[8] A. J. Demers, J. Gehrke, B. Panda, M. Riedewald, V. Sharma, and W. M. White. Cayuga: A general purpose event monitoring system. In *CIDR*, pages 412–422, 2007.

[9] K. El Gebaly, P. Agrawal, L. Golab, F. Korn, and D. Srivastava. Interpretable and informative explanations of outcomes. *Proceedings of the VLDB Endowment*, 8(1):61–72, 2014.

[10] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos. *Fast subsequence matching in time-series databases*, volume 23. ACM, 1994.

[11] U. Fayyad and K. B. Irani. Multi-interval discretization of continuous-valued attributes for classification learning. 1993.

[12] U. Feige, V. S. Mirrokni, and J. Vondrak. Maximizing non-monotone submodular functions. *SIAM Journal on Computing*, 40(4):1133–1153, 2011.

[13] Ganglia monitoring system. http://ganglia.sourceforge.net/.

[14] M. Gupta, J. Gao, C. Aggarwal, and J. Han. Outlier detection for temporal data. *Synthesis Lectures on Data Mining and Knowledge Discovery*, 5(1):1–129, 2014.

[15] H. Huang and S. P. Kasiviswanathan. Streaming anomaly detection using randomized matrix sketching. *Proceedings of the VLDB Endowment*, 9(3):192–203, 2015.

[16] M. Kuczma. *An introduction to the theory of functional equations and inequalities: Cauchy's equation and Jensen's inequality.* Springer Science & Business Media, 2009.

[17] L. Lam and S. Suen. Application of majority voting to pattern recognition: an analysis of its behavior and performance. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, 27(5):553–568, 1997.

[18] D. Luckham. *Event Processing for Business: Organizing the Real-Time Enterprise.* Wiley, 2011.

[19] Y. Mei and S. Madden. Zstream: a cost-based query processor for adaptively detecting composite events. In *SIGMOD Conference*, pages 193–206, 2009.

[20] J. Peng, H. Wang, J. Li, and H. Gao. Set-based similarity search for time series. In *Proceedings of the 2016 International Conference on Management of Data*, pages 2039–2052. ACM, 2016.

[21] R. Pochampally, A. Das Sarma, X. L. Dong, A. Meliou, and D. Srivastava. Fusing data with correlations. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 433–444. ACM, 2014.

[22] S. Roy, L. Orr, and D. Suciu. Explaining query answers with explanation-ready databases. *Proceedings of the VLDB Endowment*, 9(4):348–359, 2015.

[23] StreamSQL Team. StreamSQL: a data stream language extending SQL. http://blogs.streamsql.org/.

[24] L. Tran, L. Fan, and C. Shahabi. Distance based outlier detection for data streams. *Proceedings of the VLDB Endowment*, 9(4):1089–1100, 2015.

[25] D. Wang, E. A. Rundensteiner, and R. T. Ellison. Active complex event processing over event streams. *PVLDB*, 4(10):634–645, 2011.

[26] X. Wang, A. Mueen, H. Ding, G. Trajcevski, P. Scheuermann, and E. Keogh. Experimental comparison of representation methods and distance measures for time series data. *Data Mining and Knowledge Discovery*, 26(2):275–309, 2013.

[27] E. Wu and S. Madden. Scorpion: explaining away outliers in aggregate queries. *Proceedings of the VLDB Endowment*, 6(8):553–564, 2013.

[28] H. Zhang, Y. Diao, and N. Immerman. On complexity and optimization of expensive queries in complex event processing. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 217–228. ACM, 2014.

APPENDIX

A. OTHER WAYS OF FINDING MINIMUM EXPLANATIONS

A.1 Optimization with penalty

Let θ^i denote a selection vectors, where $\theta^i = (x_1, x_2, \ldots, x_p), \exists i, 1 \leq i \leq p, \forall j \neq i \wedge 1 \leq j \leq p, x_i = 1, x_j = 0$

$$\Theta^q = \begin{bmatrix} \theta^{m_1} \\ \theta^{m_2} \\ \vdots \\ \theta^{m_q} \end{bmatrix}$$

So Θ^q is a $q \times p$ matrix, and $F_j = \Theta^q \times V_{C_j}$, which is a vector with q coordinates.

$\mathbb{T} = \{\Theta^q = \begin{bmatrix} \theta^{m_1} \\ \theta^{m_2} \\ \vdots \\ \theta^{m_q} \end{bmatrix} |\theta^i = (x_1, x_2, \ldots, x_p), 1 \leq i, j \leq p, j \neq i, x_i = 1, x_j = 0, 1 \leq m_1 < m_2 < \ldots < m_q \leq p\}$

The objective function is used to find a subset of features as F from all the features of V_{C_j}, such that the distance between F_0 and F_1 is maximum while $|F|$ is minimal. We tried quite a few functions, and none of them works.

$$\underset{\Theta^q \in \mathbb{T}}{\text{argmax}} ||\Theta^q \cdot (V_{C_0} - V_{C_1})||_2^2 - \lambda ||\Theta^q||_1 \tag{5}$$

We choose **euclidean distance** for the object because it tends to select the most distinguished features. For one instance, there is no difference for the two distances. With two annotated instances, if one instance selects (a=0.8, b=0.5) while another instance selects(a=0.9,c=0.4), which set of feature should we consider? The Mahanttan distance would be the same. While the Euclidean distance prefers the latter. Or even a different case, (a=0.8, b=0.5) vs. (a=0.9,c=0.3).

While for the penalty term, we choose L_1 norm because it tends to return sparser results.

Function 5 is convex instead of concave according to Jesen's inequality[16]. Being convex means a local minimum value is a global minimum value, while being concave means a local maximum value is a global minimum value.

Based on Jesen's inequality, we can prove the convexity of Function 5 by proving Function 6 for any given a,

$$0 \leq a \leq 1$$

$$af(\Theta_1) + (1-a)f(\Theta_2) \geq f(a\Theta_1 + (1-a)\Theta_2) \tag{6}$$

Proof Sketch:

Left side:

$$af(\Theta_1) + (1-a)f(\Theta_2)$$

$$= a||\Theta_1 \cdot (V_{C_0} - V_{C_1})||_2^2 - a\lambda||\Theta_1||_1$$

$$+(1-a)(||\Theta_2 \cdot (V_{C_0} - V_{C_1})||_2^2 - \lambda||\Theta_2||_1)$$

$$= a||\Theta_1 \cdot (V_{C_0} - V_{C_1})||_2^2 + (1-a)(||\Theta_2 \cdot (V_{C_0} - V_{C_1})||_2^2)$$

$$-\lambda(||\Theta_1||_1 + (1-a)\lambda||\Theta_2||_1)$$

Right side:

$$f(a\Theta_1 + (1-a)\Theta_2)$$

$$= ||(a\Theta_1 + (1-a)\Theta_2) \cdot (V_{C_0} - V_{C_1})||_2^2 - \lambda||a\Theta_1 + (1-a)\Theta_2||_1$$

Let us remove the

$$-\lambda||a\Theta_1 + (1-a)\Theta_2||_1$$

on both sides.

Then we are trying to prove

$$||a\Theta_1 \cdot (V_{C_0} - V_{C_1})||_2^2 + ||(1-a)\Theta_2 \cdot (V_{C_0} - V_{C_1})||_2^2$$

$$\geq ||(a\Theta_1 + (1-a)\Theta_2) \cdot (V_{C_0} - V_{C_1})||_2^2$$

Then if we can prove for any dimension d of $V_{C_0} - V_{C_1}$ this inequation holds, this proof is done. Let V to denote $V_{C_0} - V_{C_1}$ for short.

$$a(\Theta_1 \cdot V)_d^2 + (1-a)(\Theta_2 \cdot V)_d^2$$

$$\geq ((a\Theta_1 + (1-a)\Theta_2) \cdot V)_d^2$$

$$a(\Theta_1 \cdot V)_d^2 + (1-a)(\Theta_2 \cdot V)_d^2$$

$$\geq a^2(\Theta_1 \cdot V)_d^2 + (1-a)^2(\Theta_2 \cdot V)_d^2 + 2a(1-a)(\Theta_1 \cdot V)_d(\Theta_2 \cdot V)$$

$$a(\Theta_1 \cdot V)_d^2 + (1-a)(\Theta_2 \cdot V)_d^2$$

$$-a^2(\Theta_1 \cdot V)_d^2 - (1-a)^2(\Theta_2 \cdot V)_d^2 - 2a(1-a)(\Theta_1 \cdot V)_d(\Theta_2 \cdot V) \geq 0$$

$$(a-a^2)(\Theta_1 \cdot V)_d^2 + (a-a^2)(\Theta_2 \cdot V)_d^2$$

$$-2(a-a^2)(\Theta_1 \cdot V)_d(\Theta_2 \cdot V) \geq 0$$

$$(a-a^2)((a\Theta_1 - (1-a)\Theta_2) \cdot V)_d^2 \geq 0$$

Given $0 \leq a \leq 1$, so $a - a^2 \geq 0$. And $((a\Theta_1 - (1-a)\Theta_2) \cdot V)_d^2$ is obviously larger than or euqal to 0. So the proof is done. □

Then we designed the concave version shown in Function 7 The problem of Function 7 is the maximum is reached when Θ^q is 0.

$$\underset{\Theta^q \in \mathbb{T}}{\text{argmax}}||\Theta^q \cdot (V_{C_0} - V_{C_1})||_2^2 - \lambda||\Theta^q||_2^2 (\lambda \geq 1) \qquad (7)$$

So we adjusted it as Function 8. It is required that ($\lambda_1 \geq 1, \lambda_1 > \lambda_2 > 0$). In practice, in the constraint set, $||\Theta^q||_2^2 = ||\Theta^q||_1$, so the meaning of Function 8 is: it only selects features with distance larger than $|\lambda_1 - \lambda_2|$. It can be proved. So the maximum value will be reached when all features with larger distance are selected while all features with smaller distance are filtered.

$$\underset{\Theta^q \in \mathbb{T}}{\text{argmax}}||\Theta^q \cdot (V_{C_0} - V_{C_1})||_2^2 - \lambda_1||\Theta^q||_2^2 + \lambda_2||\Theta^q||_1 \qquad (8)$$

In summary, those optimizations either cannot find optimal solution or the results are equal to uninteresting thresholds.

B. SYSTEM DESIGN

This section shows the system design of XStream. ~~We first discuss the requirements and design goal, then we present the system architecture, following are the detailed discussion on two modules for explanation.~~

B.1 Design ~~goal~~

~~The design goal includes two aspects: functionality and performance requirements. On~~ functionality, the explanation module should be highly integrated with existing CEP-based monitoring system, because it is triggered only when users ~~observer~~ anomalies from results of the monitoring system. For performance, it should meet two requirements: (1)after users trigger the explanation request, it should return answers as soon as possible, and the design goal is to be within 1 minute, which is a reasonable delay for such kind of tasks; (2)the explanation functionality should not hurt the monitoring performance seriously because the monitoring queries are always running and monitoring is the main purpose of the system, while explanation is ~~a supplementary function which are~~ supposed to run infrequently.

B.2 Architecture

With these design goals mentioned above, we design a CEP-based monitoring system with explanation functionality, and the new system is named XStream. The architecture of XStream is shown in Figure 18.

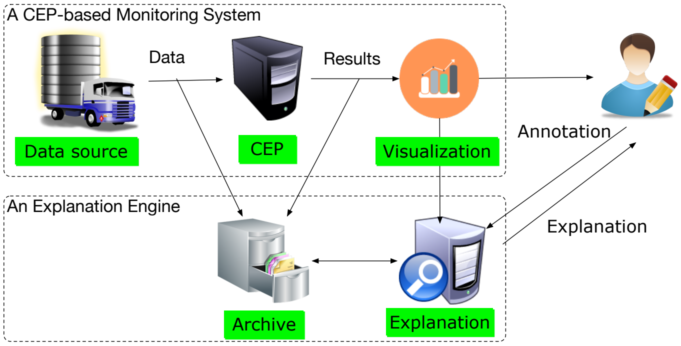
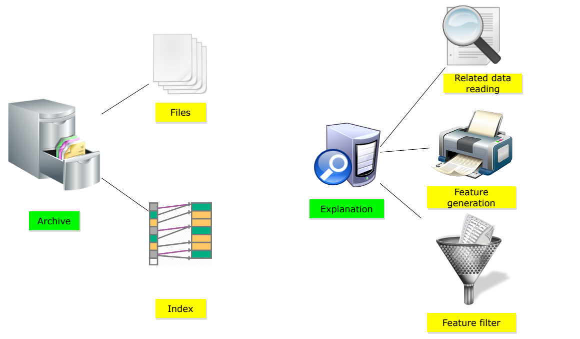


Figure 18: Architecture of XStream

Within the dashed rectangle~~, it~~ is a CEP-based monitoring system without explanation functionality. The monitoring system is composed of three components. ~~**Data source**~~ represents the input data stream, which works as a gateway to collect all types of events from outside the monitoring system. For example, in our Hadoop cluster monitoring use case, the data source module collects system metrics and Hadoop logs from all nodes across the whole cluster. The **CEP** module in the architecture represents a CEP engine, which is the workhorse of the monitoring system. Users submit queries to the CEP engine and the engine will match incoming events against all queries. The **visualization** module ~~visualize user specified results from~~ the queries.

We add two new modules ~~for explanation purpose, which are outside the dashed rectangle and discussed below.~~

~~**Archive module**~~ stores all events from ~~input stream~~ by timestamp, as shown in Figure 19(a). Events are written into disk by event types. Events of the same type will be written sequentially to disk. The sequential writing lowers the cost on I/O in both writing and reading. In order to avoid reading unnecessary events when ~~explanation~~ is triggered, events of the same type are chopped into smaller chunk files on disk. An index of the time range for each chunk is built. When events of ~~certain period~~ are requested, the archive module first looks up ~~via~~ the index, ~~then reads needed events~~. The choice of ~~chunk size needs to trade off~~ the I/O cost and index storage cost. A larger chunk size potentially increases the cost on reading unnecessary events, but it reduces the size of index: the extreme case is all events of the same type are written into one big file, and every time the explanation is triggered the module has to read from the beginning of the file, while the index has only one entry for each event type. A smaller chunk size is better at avoiding unnecessary events, but the index needs more space: the extreme case is each file only stores events for one timestamp, and every event read is needed but the index will hold a large number of entries for each event type.



(a) Archive Module (b) Explanation Module

Figure 19: Main modules of XStream

~~Explanation module~~ accepts ~~request~~ from users and analyzes related events to generate explanations, and its components are shown in Figure 19(b). Given user annotated abnormal intervals, ~~explanation engine~~ requests events in the specific time period according to ~~index~~ built by the archive module. Then all relevant events are converted to features in different granularities. After computing the entropy distance of every feature, and filtering noisy features, the module will return selected features as ~~explanation~~ to users.

The architecture is demonstrated to be effective: it returns explanations quickly and only slightly affects the performance of the monitoring engine. ~~See C~~ shows the detailed results on performance.

C. SYSTEM PERFORMANCE

Then we run experiments for performance. In the experiments, we first run 2000 monitoring queries for a few minutes, then trigger the explanation request every a few minutes. Each time we only trigger one explanation request, so there are no concurrent explanation ~~request.~~

In Figure 20, we show the number of delayed threads caused by the explanation function. Affected threads ~~means~~ the monitoring thread having a delay more than 0.01 second in processing incoming events~~, we~~ choose 0.01 second as the threshold because most events are processed within this range when no explanation analysis is triggered. In the figure we can see that, ~~most~~ use cases, only less than 25% of threads are affected. Only in Use case 3 , 26% of all threads are affected. In Use case 6, only 4.7% are affected. In Use case 8, no threads are affected. In summary, only a small portion of all monitoring queries are affected by the explanation function.

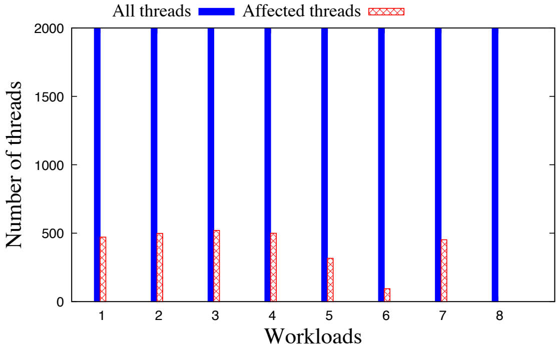


Figure 20: Total running threads vs. delayed threads

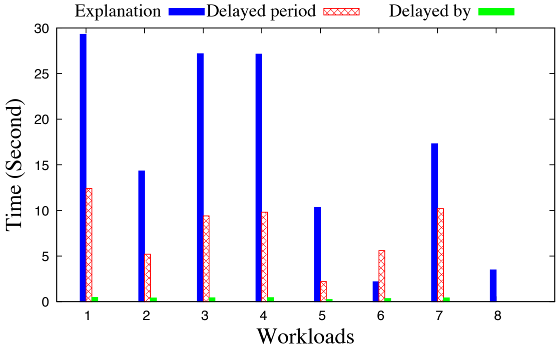


Figure 21: Explanation duration vs. affected duration vs. delayed distance

Figure 21 shows how seriously of those delayed threads are affected. The blue bars are the runtime of running explanation task independently, which means there are no monitoring queries running concurrently. The red bars show the duration of threads are affected. The delays do not last long, most of which are less than 10 seconds or so. As observed, these affected threads can catch up very quickly after the explanation triggered for some time. The green bars show how much time is delayed of each thread. It turns out that on average all the affected threads are delayed around 0.4 seconds. In Use case 5, the average delay time is 0.24 ~~seconds~~. For Use case 8, it is 0 second because no threads are affected. It means that, although some threads are affected, ~~but~~ they are still ~~going ahead~~, and only slightly behind the latest event. For monitoring purposes, such delay should be acceptable. In summary, after the explanation is triggered, a small portion of threads are affected slightly in a short period.

I don't really understand the last paragraph…

D. EVALUATION FOR SUPPLY CHAIN MANAGEMENT USE CASE

D.1 Detailed design of simulation

In order to study the problems in supply chain management, it is better to get some real traces from manufacturers. However, due to the confidentiality issues, we were unable to get real data. Thus a simulator is designed to generate manufacturing data to study this use case.

The two use cases shown above involves two categories of events: monitoring and materials. In practice, there must be other types of events, which are ignored here because we focus on studying the two types of use cases. So in simulation only those two categories of events will be generated.

In the monitoring category, we assume there are a number of different sensors reporting different measurements for the same place. **Rate:** each monitoring event series is reporting a specific measurement at a fixed rate, like one report per 10 seconds. **Value range:** there is a valid value range for each monitoring series, and any value outside the valid range is some abnormal value.

In the materials category, we assume there are a number of machines consuming materials at the same time. These machines are producing different parts for one product. **Rate:** while monitoring event series have fixed rate, the interval between material recording events are not fixed because manufacturing step might have different length of durations. **Value range:** we use numeric value to denote the quality of materials, and there is a valid bar for the quality value: any value equal to or larger than the bar is satisfying the standard, otherwise it is sub-par.These configurations are summarized in Table 1.

D.2 Anomalies

With the above settings, we can generate all events with different types occurred during the manufacturing period for a specific product. For normal products, all events are generated strictly following the event rates, and all values for monitoring category fall into the valid range and all material values are equal to or above the quality bar.

For anomalies, we simulated two different types of products corresponding to the use cases discussed above. A product with **missing monitoring** issue lacks some monitoring measurements during its manufacturing period. If a product has sub-par material issue, in the simulation of its manufacturing events, some values of its material recording events are under the quality bar.

In this usecase, CEP queries will be used to track the progress of each product. The historical query results will be archived. The analysis will be triggered when customers report quality problems. After a product is claimed to be problematic, the query result for this product will be compared against a few products which are guaranteed to be of good quality. Products have no claims during until expiration date are automatically labeled as normal. And the claimed product will be labeled as abnormal.

D.3 Evaluation

For the supply chain management use case, we created six anomalies: the first three use cases are about missing monitoring, and the last three use cases are about sub-par materials.

Explanation quality results are shown in Fig 22. Our techniques are providing much better explanations for every use case. Fig 23 compares the conciseness of results, and again our algorithm beats other techniques and the size is always the same as that of the ground truth. The prediction results are listed in Fig 24, and our techniques provide results as good as state-of-the-art techniques.

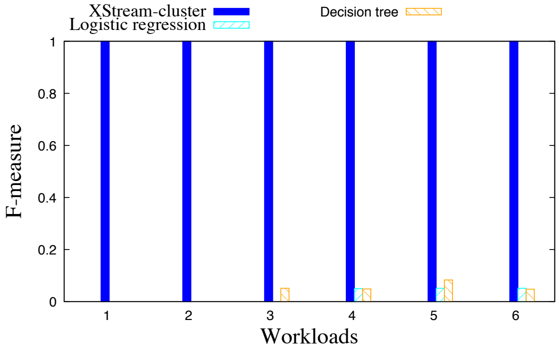


Figure 22: Supply chain management: consistency comparison

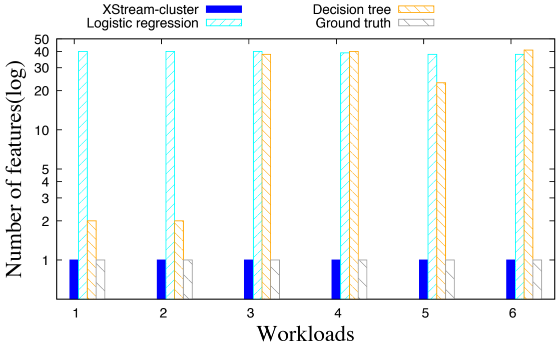


Figure 23: Supply chain management: conciseness comparison

Category	Event Rate	Num. of types	**Schema**	Valid values
Monitoring	Fixed rate	100	$\{(Monitoringname, timestamp, monitoringvalue)\}$	Valid range
Material	Non-Fixed rate	100	$\{(Machinename, timestamp, materialqualityvalue)\}$	Valid bar

Table 1: Simulation configurations.

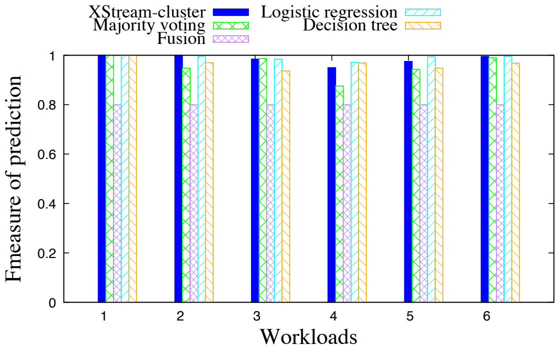


Figure 24: Supply chain management: prediction comparison