MacroBase: Analytic Monitoring for the Internet of Things

Peter Bailis, Edward Gan, Samuel Madden†, Deepak Narayanan, Kexin Rong, Sahaana Suri
Stanford InfoLab and †MIT CSAIL

ABSTRACT

An increasing proportion of data today is generated by automated processes, sensors, and devices—collectively, the Internet of Things (IoT). IoT applications' rising data volume, demands for time-sensitive analysis, and heterogeneity exacerbate the challenge of identifying and highlighting important trends in IoT deployments. In response, we present MacroBase, a data analytics engine that performs statistically-informed *analytic monitoring* of IoT data streams by identifying deviations within streams and generating potential explanations for each. MacroBase is the first analytics engine to combine streaming outlier detection and streaming explanation operators, allowing cross-layer optimizations that deliver order-of-magnitude speedups over existing, primarily non-streaming alternatives. As a result, MacroBase can deliver accurate results at speeds of up to 2M events per second per query on a single core. MacroBase has delivered meaningful analytic monitoring results in production, including an IoT company monitoring hundreds of thousands of vehicles.

1. INTRODUCTION

An increasing proportion of data today is generated by automated processes, sensors, and systems in a variety of domains including manufacturing, mobile computing, and transportation—collectively, the Internet of Things (IoT). Inexpensive hardware, widespread communication networks, and decreased storage costs have led to tens of billions of dollars in recent commercial investment [57] and projected 40% year-over-year growth in overall data volume [49].

This increase in machine-generated data leads to a key data management challenge: while data volumes rise, the capacity of human attention remains fixed. It is increasingly intractable to understand IoT application behavior via manual data inspection. In a series of interviews, system operators and analysts in several IoT domains including datacenter operations, mobile applications, and industrial manufacturing highlighted a common desire for more sophisticated *analytic monitoring*, or automated detection of unusual or interesting behaviors in IoT data streams. However, this analytic monitoring of IoT streams is made particularly difficult given three key characteristics of IoT applications:

First, IoT applications generate immense data *volume*. For example, production infrastructure at LinkedIn and Twitter generates over 12M events per second [10, 80]. In addition to posing a clear scalability challenge, these high data rates are also a challenge for interpretability. We can identify interesting behaviors by applying one of the many techniques for online outlier detection in the literature, but the number of outlying points occurring in high-volume IoT data streams may be immense; it is infeasible for users to manually inspect thousands or more anomalies per second. Consequently, analytic monitoring for IoT requires the use of *data explanation* techniques that can group and categorize large numbers of outliers so users can diagnose their underlying causes.

Second, IoT applications often require *timely* analyses. Analysts frequently reported needs to identify and respond to errors (e.g., failing devices, resource oversubscription) in minutes or less. As a result, almost all IoT deployments we encountered in production used simple rule-based techniques that quickly fire in response to predefined threshold violations. While these rules identify catastrophic behavior, they often fail to detect systemic inefficiency and are difficult to adapt to scenarios where typical behavior may vary. In contrast, more sophisticated explanation techniques can overcome these shortcomings but are overwhelmingly designed for lower volume and offline operation (Section 2.2).

Third, IoT applications frequently exhibit *heterogeneous behavior*. IoT data streams may contain sources with varying operating characteristics. For example, the Android ecosystem contains over 24K distinct device types [65]: an Android application developer must reason about her application's behavior across each of these devices and their sensors, each version of the Android operating system, and each of her application releases. Manually identifying misbehaving or anomalous groups of software versions, devices, and users at scale and in a timely manner is highly challenging.

In response to these challenges, we present MacroBase, a new data analysis system that is specialized for high-performance analytic monitoring of IoT streams.[1] MacroBase's goal is to provide a flexible framework for executing analytic monitoring queries in a wide variety of current and future IoT domains while employing specialized analytic monitoring operators for improved efficiency and accuracy. Consequently, our focus in this paper is two-fold. First, we propose a general architecture for combining streaming outlier detection and streaming data explanation techniques and show how it can be used to enable analytic monitoring of a range of IoT applications. Second, we investigate the design of high-performance analytic monitoring operators that can take advantage of the combination of detection and explanation as well as the characteristics of IoT data streams in order to improve performance and result quality.

Architecturally, MacroBase is designed for extensibility. Given the variety of IoT applications as well as the diversity of outlier de-

[1] MacroBase is available as open source at `http://macrobase.io/`.

tection and data explanation techniques, MacroBase supports a range of IoT workloads and operators. By default, MacroBase executes analytic monitoring queries using a pipeline of domain-independent, unsupervised streaming detection and explanation operators that delivers results with limited end-user intervention. However, MacroBase users can tailor their queries by *i*.) adding domain-specific feature transformations (e.g., time-series operations such as autocorrelation and Fourier transformation) to their pipelines—without modification to the rest of the pipeline *ii*.) providing supervised detection rules (or labels) that can complement or replace unsupervised detectors and *iii*.) authoring custom streaming transformation, detection, and explanation operators, whose interoperability is enforced by MacroBase's type system.

EXAMPLE. *MacroBase's default operator pipeline may report that devices of type B264 running application version 2.26.3 are sixty times more likely to experience abnormally high power drain readings than the rest of the stream, indicating a potential problem with the interaction between B264 devices and version 2.26.3. If the user also wishes to find time-varying power spikes within the stream, she can employ a time-series feature transformation to identify time periods with abnormal time-varying patterns instead. She might later add a manual rule to capture all readings with power drain greater than* 100*W and a custom time-series explanation operator [56], all without modifying the remainder of the operator pipeline.*

As this example illustrates, compared to conventional dataflow-oriented analysis engines such as Heron [54] and Dryad [51], MacroBase provides users with a set of interfaces that are specialized for analytic monitoring. However, MacroBase's analytic monitoring operators can be combined with relational operators, and, just as relational operators can be executed via many dataflow implementations, MacroBase's operators preserve compatibility with existing streaming dataflow engines.

Using this flexible architecture, we subsequently develop a set of high-performance analytic monitoring operators that are specialized for handling the volume, time-sensitivity, and heterogeneity of IoT analyses as well as the combination of outlier detection and explanation. Specifically:

To provide responsive analysis over IoT streams in which a changing set of devices may exhibit outlying behavior, MacroBase's default operators are designed to track shifts in data. MacroBase leverages a novel sampler, called the *Adaptable Damped Reservoir* (ADR), which performs sampling over arbitrarily-sized, exponentially damped stream windows. MacroBase uses the ADR to incrementally train outlier detectors based on robust statistical estimation [45], which can reliably identify typical behavioral modes despite large numbers of extreme data points. MacroBase also adopts exponentially weighted sketching and streaming data structures [26, 77] in explanation, which further improve responsiveness and accuracy.

To provide interpretable explanations of often relatively rare behaviors in IoT streams, MacroBase adopts a metric from statistical epidemiology called the *relative risk ratio*, which describes the relative occurrence of key attributes (e.g., age, sex) among infected and healthy populations. MacroBase employs two IoT-specific optimizations in computing this statistic. First, MacroBase exploits the cardinality imbalance between detected outliers and inliers to accelerate explanation generation, an optimization enabled by the combination detection and explanation. Instead of inspecting outliers and inliers separately, MacroBase first examines outliers, then aggressively prunes its search over inliers. Second, because IoT streams contain repeated measurements from devices with similar attributes (e.g., firmware version), MacroBase utilizes this property to reduce update time in risk ratio computation via a new counting sketch, the *Amortized Maintenance Counter* (AMC). These optimizations improve performance while highlighting the small subset of stream attributes that matter most.

MacroBase's flexible architecture and optimized operators have proven useful in production. The system has already identified previously unknown behaviors in production, including a telematics company monitoring hundreds of thousands of vehicles. We report on these experiences and quantitatively evaluate MacroBase's performance and accuracy on both production telematics data as well as a range of publicly available real-world datasets. MacroBase's optimized operators exhibit order-of-magnitude performance improvements over less specialized operators at rates of up to 2M events per second per query while delivering accurate results in controlled studies using both synthetic and real-world data. As we show, this ability to quickly process large data volumes can also improve result quality by combating bias due to the multiple testing problem [72], thereby improving statistical significance of results. Finally, we demonstrate MacroBase's extensibility via case studies in mobile telematics, electricity metering, and video-based surveillance, and via integration with existing analysis frameworks.

Our contributions in this paper are as follows:

- MacroBase, an analytics engine and architecture for analytic monitoring of IoT data streams that is the first to combine streaming outlier detection and streaming data explanation.
- The Adaptable Damped Reservoir, the first exponentially damped reservoir sample to operate over arbitrary windows, which MacroBase leverages to improve detector responsiveness.
- An optimization for improving the efficiency of combined detection and explanation by exploiting cardinality imbalance between outliers and inliers in streams.
- The Amortized Maintenance Counter, a new heavy-hitters sketch that allows fast updates by amortizing sketch pruning across multiple observations of the same item.

The remainder of this paper proceeds as follows. Section 2 describes our target environment by presenting motivating use cases and discussing related work. Section 3 presents the MacroBase's interaction model and core pipeline architecture. Section 4 describes MacroBase's default streaming outlier detection operators and presents the ADR sampler. Section 5 describes MacroBase's default streaming explanation operator, including its cardinality-aware optimization and the AMC sketch. We experimentally evaluate MacroBase's accuracy and runtime, report on experiences in production, and demonstrate extensibility via case studies in Section 6. Section 7 concludes with a discussion of ongoing extensions.

2. TARGET ENVIRONMENT

MacroBase provides IoT application writers and system analysts an end-to-end analytic monitoring engine capable of highlighting outlying data within IoT streams while explaining the properties of the data that make it outlying. In this section, we describe our target analytic monitoring environment via three motivating use cases and a comparison with related work.

2.1 Analytic Monitoring Scenarios

As examples of the types of workloads we seek to support, we draw on three use cases from industry.

Mobile applications. Cambridge Mobile Telematics (CMT) is a five-year-old commercial IoT company whose mission is to make

roads safer by making drivers more aware of their driving habits. CMT provides drivers with a smartphone application and mobile sensor for their vehicles, and collects and analyzes data from many hundreds of thousands of vehicles at rates of tens of Hz. CMT uses this IoT data to provide users with feedback about their driving.

CMT's engineers report that analytic monitoring has proven especially challenging. CMT's operators, who include database and systems research veterans, report difficulty in answering several questions: is the CMT application behaving as expected? Are all users able to upload and view their trips? Are sensors operating at a sufficiently granular rate and in a power-efficient manner? The most severe problems in the CMT application are caught by quality assurance and customer service, but many behaviors are more pernicious. For example, Apple iOS 9.0 beta 1 introduced a buggy Bluetooth stack that prevented iOS devices from connecting to CMT's sensors. Few devices ran these versions, so the overall failure rate was low; as a result, CMT's data volume and heterogeneous install base obscured a potentially serious widespread issue in later releases of the application. Given low storage costs, CMT records all of the data required to perform analytic monitoring to detect such behaviors, yet CMT's engineers report they have lacked a solution for doing so in a timely and efficient manner.

In this paper, we report on our experiences deploying MacroBase at CMT, where the system has highlighted interesting behaviors such as those above, in production.

Datacenter operation. Datacenter and server operation represents one of the highest-volume sources for machine-generated data. In addition to the billion-plus events per minute volumes reported at Twitter and LinkedIn, engineers report a similar need to quickly identify misbehaving servers, applications, and virtual machines.

For example, Amazon AWS recently suffered a failure in its DynamoDB service, resulting in outages at sites including Netflix and Reddit. The Amazon engineers reported that "after we addressed the key issue...we were left with a low overall error rate, hovering between 0.15-0.25%. We knew there would be some cleanup to do after the event," and therefore the engineers deferred maintenance. However, the engineers "did not realize soon enough that this low overall error rate was giving some customers disproportionately high error rates" due to a misbehaving server partition [3].

This public postmortem is representative of many scenarios described by system operators. At a major social network, engineers report that the challenge of identifying transient slowdowns and failures across hosts and containers is exacerbated by the heterogeneity of workload tasks. Failure postmortems can take hours to days, and, due to the labor-intensive nature of manual analysis, engineers report an inability to efficiently and reliably identify slowdowns, leading to suspected inefficiency.

Unlike the CMT use case, we do not directly present results over production data from these scenarios. However, datacenter telemetry is an area of active research within the MacroBase project.

Industrial monitoring. IoT has spurred considerable interest in monitoring of industrial deployments. While many industrial systems already rely on legacy monitoring systems, several industrial application operators reported a desire for analytics and alerting that can adapt to new sensors and analyses. These industrial scenarios can have particularly important consequences. For example, an explosion and fire in July 2010 killed two workers at Horsehead Holding Corp.'s Monaca, PA, zinc manufacturing plant. Last year's postmortem review by the US Chemical Safety board revealed that "the high rate-of-change alarm warned that the [plant] was in imminent danger 10 minutes before it exploded, but there appears to have been no specific alarm to draw attention of the operator to the subtle but dangerous temperature changes that were taking place much (i.e. hours) earlier." The auditor noted that "it should be possible to design a more modern control system that could draw attention to trends that are potentially hazardous" [47].

In this paper, we evaluate analytic monitoring's potential to draw attention to behaviors within electrical utilities.

2.2 Related Work

Performing analytic monitoring in IoT applications as in the preceding scenarios requires high-volume, streaming analytics techniques. In this section, we discuss related techniques and systems from the literature.

Streaming and Specialized Analytics. MacroBase is a streaming data analysis system that is specialized for analytic monitoring. In its architecture, MacroBase builds upon a series of systems for streaming data and specialized, advanced analytics tasks. A range of systems from both academia [4, 20] and industry (e.g., Storm, StreamBase, IBM Oracle Streams) provide infrastructure for executing streaming queries. MacroBase adopts dataflow as its execution substrate, but its goal is to provide a set of high-level analytic monitoring operators; in MacroBase, dataflow is a means to an end rather than an end in itself. In designing a specialized engine, we were inspired by several past projects, including Gigascope (specialized for network monitoring) [27], WaveScope (specialized for signal processing) [35], MCDB (specialized for Monte Carlo-based operators) [52], and Bismarck (providing extensible aggregation for gradient-based optimization) [32]. In addition, a range of commercially-available analytics packages provide advanced analytics functionality—but, to the best of our knowledge, not the streaming explanation operations we seek here. MacroBase continues this tradition by providing a specialized set of operators for analytic monitoring of IoT, which in turn allows new optimizations within this design space.

Outlier detection. Outlier detection dates to at least the 19th century; the literature contains thousands of techniques from communities including statistics, machine learning, data mining, and information theory [7, 18, 43]. Outlier detection techniques have seen major success in several domains including network intrusion detection [31, 63], fraud detection (leveraging a variety of classifiers and techniques) [13, 66], and industrial automation and predictive maintenance [8, 58]. A considerable subset of these techniques operates over data streams [6, 17, 67, 76].

At stream volumes in the hundreds of thousands or more events per second, statistical outlier detection techniques will (by nature) produce a large stream of outlying data points. As a result, while outlier detection forms a core component of an IoT analytic monitoring solution, it must be coupled with streaming explanation (below). In the design of MacroBase, we treat the array of outlier detection techniques as inspiration for a modular architecture. In MacroBase's default pipeline, we leverage detectors based on robust statistics [45, 59], adapted to the streaming context. However, in this paper, we also demonstrate compatibility with detectors from Elki [74], Weka [37], RapidMiner [44], and OpenGamma [2].

Data explanation. Data explanation techniques assist in summarizing differences between datasets. The literature contains several recent explanation techniques leveraging decision-tree [22] and Apriori-like [38, 81] pruning, grid search [70, 83], data cubing [71], Bayesian statistics [79], visualization [16, 64], causal reasoning [82], and several others [30, 42, 50, 60]. While we are inspired by these results, none of these techniques executes over streaming data or at the scale we seek. Several exhibit runtime exponential in the number of attributes (which can number in the hundreds of thousands to

millions in the IoT data we examine) [71, 79] and, when reported, runtimes in the batch setting often vary from hundreds to approximately 10K points per second [70, 79, 81] (we also directly compare throughput with several techniques [22, 71, 79, 81] in Appendix D).

To address the demands of streaming operation and to scale to millions of events per second, MacroBase's explanation techniques draw on sketching and streaming data structures [21, 23, 24, 26, 28, 61, 73, 77] adapted to our use case of analytic monitoring for IoT. We view existing explanation techniques as a useful second step in analysis following the explanations generated by MacroBase, and we see promise in adapting these existing techniques to streaming execution at IoT volumes. Given our goal of providing a generic architecture for analytic monitoring, future improvements in streaming explanation should be complementary to our results here.

3. MacroBase ARCHITECTURE AND APIS

As an analytic monitoring engine for IoT, MacroBase is tasked with executing complex analytics over large stream volumes in a number of IoT application domains. As a result, MacroBase's architecture is designed to allow high-performance execution as well as the flexibility to operate in a range of domains using an array of detection and explanation operators. In this section, we describe MacroBase's query processing architecture approach to extensibility, and interaction modes.

Query pipelines. MacroBase performs analytic monitoring of IoT data by executing pipelines of specialized dataflow operators over input data streams. Each MacroBase *query* specifies a set of input data sources as well as a logical query plan, or *pipeline* of streaming operators, that describes the analytic monitoring computation.

MacroBase's pipeline design is guided by two principles. First, all operators operate over streams. Batch execution is supported by streaming over stored data. Second, MacroBase uses the compiler's type system to enforce a particular pipeline structure. Each operator must implement one of several type signatures (depicted in Table 1). In turn, the compiler enforces that all pipelines composed of these operators will adhere to a common structure that we describe below.

This *architecture via typing* strikes a balance between the elegance of more declarative but often less flexible interfaces, and the expressiveness of more imperative but often less composable interfaces. More precisely, this use of the type system facilitates three important kinds of interoperability. First, users can substitute streaming detection and explanation operators without concern for their interoperability. Early versions of the MacroBase prototype that lacked this pipeline modularity proved brittle to adapt. Second, users can write a range of domain-specific feature transformation operators (described in detail below) to perform advanced processing (e.g., time-series operations) to input streams without requiring expertise in outlier detection or explanation. Third, MacroBase's operators preserve compatibility with dataflow operators found in traditional stream processing engines. For example, an MacroBase pipeline can contain standard selection, project, join, windowing, aggregation, and group-by operators.

As dictated by MacroBase's type system, a MacroBase pipeline is structured as follows:

1.) Ingestion. MacroBase ingests data streams for analysis from a number of external data sources. For example, MacroBase's JDBC interface allows users to specify columns of interest from a base view defined according to a SQL query. MacroBase subsequently reads the result-set from the JDBC connector in order to construct data points to process, with one point per row in the view. MacroBase currently requires this initial operator perform any stream ordering.

Each data point contains a set of *metrics*, corresponding to IoT measurements (e.g., trip time, battery drain), and *attributes*, corresponding to metadata about the measurements (e.g., user ID and device ID). MacroBase uses metrics to detect abnormal or unusual events, and attributes to explain behaviors. In this paper, we consider real-valued metrics and categorical attributes.[2]

As an example, to detect the OS version problem at CMT, MacroBase could have used failed or short trips as metrics and device and OS type as attributes. To detect the outages at DynamoDB, MacroBase could have used failed requests as a metric and server or IP address as an attribute. To detect the Horsehead pressure losses, MacroBase could have used the pressure gauge readings as metrics and their locations as attributes along with an autocorrelation-enabled time-series pipeline (Section 6.4). Today, selecting attributes, metrics, and a pipeline is a user-initiated process; ongoing extensions (Section 7) seek to automate this.

2.) Feature Transformation. Following ingestion, MacroBase executes a series of optional domain-specific data transformations over the stream, allowing operations including time-series operations (e.g., windowing, seasonality removal, autocorrelation, frequency analysis), statistical operations (e.g., normalization, dimensionality reduction), and datatype-specific operations (e.g., hue extraction for images, optical flow for video). For example, in Section 6.4, execute a pipeline containing a grouped Fourier transform operator that aggregates the stream into hour-long windows, then outputs a stream containing the twenty lowest Fourier coefficients for each window as metrics and properties of the window time (hour of day, month) as attributes. Placing this feature transformation functionality at the start of the pipeline allows users to encode domain-specific analyses without modifying later stages. The base type of the stream is unchanged (*Point* $\rightarrow$ *Point*), allowing transform chaining. For specialized data types like video frames, operators can subclass *Point* to further increase the specificity of types (e.g., *VideoFramePoint*).

3.) Outlier Detection. Following ingestion, MacroBase performs outlier detection, labeling each *Point* as an inlier or an outlier based on its input metrics and producing a stream of labeled *Point* outputs (*Point* $\rightarrow$ (`bool`, *Point*)); we discuss binary classification here). This stage is responsible for both training and evaluating outlier detectors on the metrics in the incoming data stream. MacroBase supports a range of models, which we describe in Section 6. The simplest include rule-based models, which simply check specific metrics for particular values (e.g., whether the `Point` metric's L2-norm is greater than a fixed constant). In Section 4, we describe MacroBase's default unsupervised models, which perform density-based outlier classification. Users can also use operators that execute supervised and pre-trained models.

4.) Explanation. Rather than returning all outlying data points, given a stream of points classified as inliers and outliers, MacroBase performs summarization to generate *explanations* by finding attributes that are common among outliers and uncommon among stream inliers ((`bool`, *Point*)$\rightarrow$*Explanation*). As we describe in detail in Section 5, MacroBase's default pipeline returns explanations in the form of attribute-value combinations (e.g., device ID 5052) that are common among outlier points but uncommon among inlier points. For example, at CMT, MacroBase could highlight devices that were found in at least 0.1% of outlier trips and were at least 3 times more common among outliers than inliers. Explanation operators can subclass explanations to provide additional information, such as statistics about the explanation or representative sequences of points to contextualize time-series outliers.

[2] It is straightforward to handle continuous attributes via discretization (e.g., [82]). We provide two examples of discretization in Section 6.4.

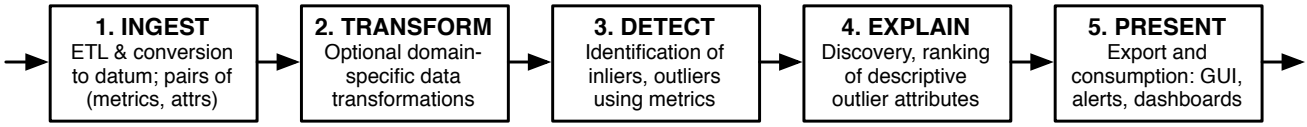


Figure 1: MacroBase's default analytic monitoring pipeline: MacroBase ingests IoT data as a series of points, which are scored and classified by an outlier detector operator, summarized by a streaming explanation operator, then ranked and presented to end users.

DATA TYPES	
Point := (`array<double>` metrics, `array<varchar>` attributes)	
Explanation := (`array<varchar>` attributes, *stats* statistics)	
OPERATOR INTERFACE	
Operator	*Type Signature*
Ingestor	external data source(s) → `stream<`*Point*`>`
Transformer	stream<*Point*> → `stream<`*Point*`>`
Detector	stream<*Point*> → `stream<(bool,` *Point*`)>`
Explainer	`stream<(bool,` *Point*`)>` → `stream<`*Explanation*`>`
Pipeline	Ingestor → `stream<`*Explanation*`>`

Table 1: MacroBase's core data and operator types. Each operator implements a strongly typed, stream-oriented dataflow interface specific to a given pipeline stage. A pipeline can utilize multiple operators of each type via transformations, such as group-by and one-to-many stream replication, as long as the pipeline ultimately returns a single stream of explanations.

Because MacroBase processes streaming data, explanation operators continuously summarize the stream. However, continuously emitting explanations may be wasteful if users only need explanations at the granularity of seconds, minutes, or longer. As a result, MacroBase's explanation operators are designed to emit explanations on demand, either in response to a user request, or in response to a periodic timer. In this way, explanation operators act as streaming view maintainers whose output can be queried at any time.

5.) Presentation. The number of output explanations may be large. As a result, most pipelines rank explanations before presentation according to statistics corresponding to each explanation. For example, by default, MacroBase sorts explanations by their degree of occurrence in the outliers and delivers this ranked list of explanations to a range of downstream consumers. MacroBase's default presentation mode is a static report rendered via a REST API or via a GUI. In the former, programmatic consumers (e.g., reporting tools such as PagerDuty) can automatically forward explanations to downstream reporting or operational systems. In the GUI, users can interactively inspect the outlier explanations and iteratively define their MacroBase queries. In practice, we have found that GUI-based exploration is an important first step in formulating standing MacroBase queries.

Extensibility. As we discussed in Section 1 and demonstrate in Section 6.4, MacroBase's pipeline architecture lends itself to three major means of extensibility. First, users can add new domain-specific feature transformations to the start of a pipeline without modifying the other operators. Second, users can input rules and/or labels to MacroBase to perform supervised classification. Third, users can write their own feature transformation, outlier detector, data explanation operators, and pipelines. This third option is the most labor-intensive, but is also the interface with which MacroBase's maintainers author new pipelines. These interfaces have also proven useful to non-experts: a master's student at Stanford and a master's student at MIT each implemented and tested a new outlier detector operator in less than a week of part-time work, and MacroBase's core maintainers currently require less than an afternoon of work to author and test a new pipeline.

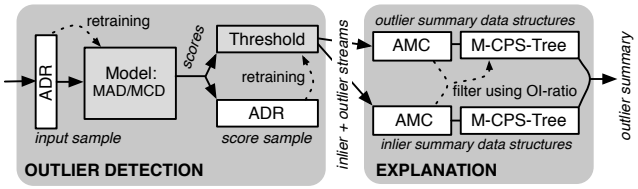


Figure 2: MDP: MacroBase's default streaming outlier detection (Section 4) and streaming explanation (Section 5) operators.

By providing a set of interfaces with which to extend pipelines (with varying expertise required), MacroBase places emphasis on "pay as you go" deployment. MacroBase's Default Operator Pipeline (MDP, which we illustrate in Figure 2 and describe in the following two sections) is optimized for efficient, accurate execution over a variety of data types without relying on labeled data or rules. It foregoes domain-specific feature extraction and instead operates directly over raw input metrics. However, as we illustrate in Section 6.4, this interface design also enable users to incorporate more sophisticated features such as domain-specific feature transformation, time-series analysis, and supervised models.

In this paper, we have chosen to present MacroBase's interfaces using an object-oriented interface, reflecting their current implementation. However, each of MacroBase's operator types is compatible with existing stream-to-relation semantics [9], theoretically allowing additional relational and stream-based processing between stages. Realizing this mapping and the potential for even higher-level declarative interfaces above MacroBase's analytic monitoring pipelines are promising areas for future work.

Operating modes. MacroBase supports three operating modes. First, MacroBase's graphical front-end allows users to interactively explore their data by configuring different inputs and selecting different combinations metrics and attributes. This is typically the first step in interacting with the engine. Second, MacroBase can execute one-shot queires that can be run programmatically in a single pass over data. Third, MacroBase can execute a streaming queries that can be run programmatically over potentially infinite stream of data. In streaming mode, MacroBase continuously ingests a stream of data points and supports exponentially decayed damped-window streaming to give precedence to more recent points (e.g., decreasing the importance of points at a rate of 50% every hour). MacroBase continuously re-renders query results, and if desired, triggers automated alerting for downstream consumers.

4. MDP OUTLIER DETECTION

MacroBase's outlier detection operators monitor the input for data points that exhibit deviant behavior. While MacroBase allows users to configure their own operators, in this section, we describe the design of MacroBase's default detection operators, which use robust estimation procedures to fit a distribution to data streams and identify the least likely points with the distribution using quantile estimation. To enable streaming execution, we introduce the ADR

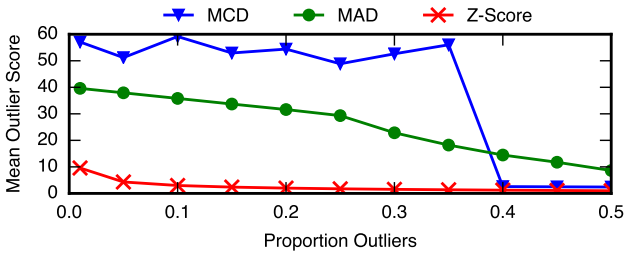


Figure 3: Discriminative power of estimators under contamination by outliers (high scores better). Robust methods (MCD, MAD) outperform the Z-score-based approach.

sampler, which MacroBase uses for model retraining and quantile estimation.

4.1 Use of Robust Statistics

A small number of anomalous behaviors may have a large effect on detector accuracy. As an example, consider the *Z-Score* of a point drawn from a univariate sample, which measures the number of standard deviations that a point lies away from the sample mean. This provides a normalized way to measure the "outlying"-ness of a point (e.g., a Z-Score of three indicates the point lies three standard deviations from the mean). However, the Z-Score is not robust to outliers: a single outlying value can skew the mean and standard deviation by an unbounded amount, limiting its usefulness.

To address this challenge, MacroBase's MDP pipeline leverages robust statistical estimation [45], a branch of statistics that pertains to finding statistical distributions for data that is mostly well-behaved but may contain a number of ill-behaved data points. If we can find a distribution that reliably fits most of the data, we can measure each point's distance from this distribution in order to find outliers [59].

A robust replacement for the Z-Score is to use the median and the Median Absolute Deviation (MAD) as a measure of distance in place of mean and standard deviation. The MAD measures the median of the absolute distance from each point in the sample to the sample median. Thus, each outlying data point has limited impact on the MAD score of all other points in the sample, since the median itself is resistant to outliers.

The Minimum Covariance Determinant (MCD) provides robust estimates for multivariate data. The MCD estimator finds the tightest group of points that best represents a sample, and summarizes the set of points according to its location and scatter (i.e., covariance) in metric space [46]. Given an estimate of location μ and scatter C, we can compute the distance between the distribution and a point x via the *Mahalanobis distance* $\sqrt{(x-\mu)^T C^{-1}(x-\mu)}$; intuitively, the Mahalanobis distance normalizes (or warps) the metric space via the scatter and then measures the distance to the center of the transformed space using the mean (see also Appendix A).

As Figure 3 demonstrates, MAD and MCD are empirically able to reliably identify points in outlier clusters despite increasing outlier contamination (experimental setup in Appendix A). Whereas MAD and MCD are resilient to contamination up to 50%, the Z-Score is unable to distinguish inliers and outliers under even modest contamination.

Classifying outliers. Given a query with a single, univariate metric, MDP uses a MAD-based detector, and, given a query with multiple metrics, MacroBase computes the MCD via an iterative approximation called FastMCD [69]. By adopting these unsupervised models, MDP can score points without requiring labels or rules from users. Subsequently, MDP uses a percentile-based cutoff over scores to identify the most extreme points in the sample. Points with scores above the percentile-based cutoff are classified as outliers, reflecting their distance from the body of the distribution.

As a notable caveat, MAD and MCD are parametric estimators, assigning scores based on an assumption that data is normally distributed. While extending these estimators to multi-modal behavior is straightforward [40] and MacroBase allows substitution of more sophisticated detectors (e.g., Appendix D), we do not consider them here. Instead, we have found that looking for far away points using these parametric estimators yields useful results: as we empirically demonstrate, many interesting behaviors in IoT applications manifest as extreme deviations from the overall population. Robustly locating the center of a population—while ignoring local, small-scale deviations in the body of the distribution—suffices to identify many important classes of outliers in the applications we study. The robust statistics literature discusses this phenomena in the case of the MCD [41] and echoes Aggarwal's observation that "even for arbitrary distributions, [extreme value analysis techniques] provide a good heuristic idea of the outlier scores of data points, even when they cannot be interpreted statistically" [7].

4.2 MDP Streaming Execution

Despite their utility, we are not aware of an existing algorithm for training MAD or MCD in a streaming context.[3] This is especially problematic because, as the distributions within data streams change over time, MDP estimators should be updated to reflect the change.

ADR: Adaptable Damped Reservoir. MDP's solution to the retraining problem is a novel adaptation of reservoir sampling over streams, which we call the Adaptable Damped Reservoir (ADR). The ADR maintains a sample of input data that is exponentially weighted towards more recent points; the key difference from traditional reservoir sampling is that the ADR operates over *arbitrary* window sizes, allowing greater flexibility than existing damped samplers. As Figure 2 illustrates, MDP maintains an ADR sample of the detector input to periodically recompute its estimator and a second ADR sample of the outlier scores to periodically recompute its classification threshold.

The classic reservoir sampling technique can be used to accumulate a uniform sample over a set of data using finite space and one pass [78]. The probability of insertion into the sample, or "reservoir," is inversely proportional to the number of points observed thus far. In the context of stream sampling, we can treat the stream as an infinitely long set of points and the contents of the reservoir as a uniform sample over the data observed so far.

In MacroBase, we wish to promptly reflect changes in the underlying data stream, and therefore we adapt a *weighted* sampling approach, in which the probability of data retention decays over time. The literature contains several existing algorithms for weighted reservoir sampling [5, 21, 28]. Most recently, Aggarwal described how to perform exponentially weighted sampling on a per-record basis: that is, the probability of insertion is an exponentially weighted function of the number of points observed so far [5]. While this is useful, as we demonstrate in Section 6, under workloads with variable arrival rates, we may wish to employ a decay policy that decays in *time*, not number of tuples; tuple-at-a-time decay may skew the reservoir towards periods of high stream volume.

To support more flexible reservoir behavior, MacroBase adapts an earlier variant of weighted reservoir sampling due to Chao [21, 28] to provide the first exponentially decayed reservoir sampler that works over arbitrary decay intervals. We call this variant the *Adaptable*

[3] Specifically, MAD requires computing the median of median distances, meaning streaming quantile estimation alone is insufficient. FastMCD is an inherently iterative algorithm that iteratively re-sorts data.

Algorithm 1 ADR: Adaptable Damped Reservoir

given: *max_size* $\in \mathbb{N}$; r: decay rate $\in (0,1)$
initialization: reservoir $R \leftarrow \{\}$; current weight $c_w \leftarrow 0$
function OBSERVE(x: point, w: weight)
  $c_w \leftarrow c_w + w$
  if $|R| <$ *max_size* **then**
    $R \leftarrow R \cup \{x\}$
  else with probability $\frac{1}{c_w}$
    remove random element from R and add x to R
function DECAY( )
  $c_w \leftarrow r \cdot c_w$

Damped Reservoir, or ADR (Algorithm 1). In contrast with existing approaches that decay on a per-tuple basis, the ADR separates the insertion process from the decay decision, allowing both time-based and tuple-based decay policies. Specifically, the ADR maintains a running count c_w of items inserted into the reservoir so far. Each time an item is inserted, c_w is incremented by one (or an arbitrary weight, if desired). With probability $\frac{1}{c_w}$, the item is placed into the reservoir and a random item is evicted from the reservoir. When the ADR is decayed (e.g., via a periodic timer or tuple count), its running count is multiplied by a decay factor (i.e., $c_w := (1-\alpha)c_w$).

MacroBase currently supports two decay policies: time-based decay, which decays the reservoir at a pre-specified rate measured according to real time, and batch-based decay, which decays the reservoir at a pre-specified rate measured by arbitrarily-sized batches of data points (Appendix A). The validity of this procedure follows from Chao's algorithm, which requires the user to manually manage weights and decay.

MacroBase's MDP uses the ADR to solve the model and percentile retraining and problems:

Maintaining inputs for training. Either on a tuple-based periodic basis or using a real-time timer, MDP periodically retrains models using an ADR that samples the input of the data stream. This streaming robust estimator maintenance and evaluation strategy is the first of which we are aware. We discuss the statistical validity of this procedure in Appendix D.

Maintaining percentile thresholds. While streaming quantile estimation is well studied, we were not able to find many computationally inexpensive options in an exponentially damped model with arbitrary window sizes. Thus, instead, MacroBase uses an ADR to sample the outlier scores produced by MAD and MCD. The ADR maintains an exponentially damped sample of the scores, which it uses to periodically compute the appropriate outlier score quantile.[4] A sample of size $O(\frac{1}{\varepsilon^2}\log(\frac{1}{\delta}))$ yields an ε-approximation of an arbitrary quantile with probability $1-\delta$ [14], so a ADR of size 20K provides an $\varepsilon = 1\%$, $\delta = 1\%$ approximation.

5. MDP OUTLIER EXPLANATION

Once MDP classifies the most outlying data points as outliers according to their metrics, MDP produces explanations that differentiate the inliers and outliers according to their attributes. In this section, we discuss how MacroBase explains outlying data using an severity metric from epidemiology, the *risk ratio*. We again begin with a discussion of an efficient batch-oriented operation employing outlier-aware optimizations, then discuss how MacroBase executes streaming explanation via the Amortized Maintenance Counter.

[4]This enables a simple mechanism for detecting quantile drift: if the proportion of outlier points significantly deviates from the target percentile (i.e., via application of a binomial proportion confidence interval), MDP should recompute the quantile.

Algorithm 2 MDP's Outlier-Aware Explanation Strategy

Given: minimum OI-ratio r and minimum support s
1: find attributes w/ support $\geq s$ in O and OI-ratio $\geq r$ in O, I
2: mine FP-tree over O using only attributes from (1)
3: filter (2) by removing patterns w/ OI-ratio $< r$ in I; return

5.1 Semantics: Support and OI-Ratio

MacroBase's outlier explanations describe attribute values that are common to the outliers but relatively uncommon to the inliers through two complementary metrics. First, to identify combinations of attribute values that are relatively uncommon in the inliers, MDP finds combinations with high *outlier-inlier (OI-ratio)*, or the ratio of its rate of occurrence (i.e., support) in the outliers to its rate of occurrence in the inliers. This metric corresponds to a standard diagnostic measure used in epidemiology called the *relative risk ratio*, which is used to determine potential causes for disease [62]. Second, to optionally eliminate explanations corresponding to rare but non-systemic combinations, MDP finds combinations with high *support*, or occurrence (by count) in the outliers. MDP accepts a minimum OI-ratio and level of outlier support as input parameters. As an example, *MDP* may find that 500 of 890 records flagged as outliers correspond to iPhone 6 devices (outlier support of 56.2%), but, if 80191 of 90922 records flagged as inliers also correspond to iPhone 6 devices (inlier support of 88.2%), we are likely uninterested in iPhone 6 as it has a low OI-ratio of 0.634.

5.2 Basic Explanation Strategy

To efficiently compute support and OI-ratio, we draw upon a connection to two classic problems in data mining: frequent itemset mining and emerging pattern mining. A naïve solution is to run an algorithm for frequent itemset mining twice, once on all inlier points and once on all outlier points, and then look for differences between the inlier and outlier sets. As we experimentally demonstrate in Section 6, this can be inefficient because we waste time searching over attributes in the inlier set that eventually get filtered due to insufficient support in the outlier set. In addition, the number of outliers is much smaller than the inliers, so processing the two sets independently ignores possibilities for additional pruning. To reduce this wasted effort, MacroBase takes advantage of both the cardinality imbalance between inliers and outliers as well as the joint explanation of each set.

Optimization: Item ratios are cheap. While computing OI-ratios for all attribute combinations is expensive, computing OI-ratios for single attributes is inexpensive: we can compute support counts over both inliers and outliers via a single pass over the attributes. Accordingly, MDP first computes OI-ratios for single attribute values, then computes attribute combinations using only attribute values with sufficient OI-ratios.

Optimization: Exploit cardinality imbalance. The cardinality of the outlier input stream is by definition much smaller than that of the input stream. Therefore, instead of mining the outlier supports and the inlier supports separately, MDP first finds outlier attribute value sets with minimum support and subsequently mines the inlier attributes, while only searching for attributes that were supported in the outliers.[5] This reduces the space of inlier attributes to explore.

End result. The result is a three-stage process (Algorithm 2). MDP first calculates the attribute values with minimum OI-ratio (support counting, followed by a filtering pass based on OI-ratio).

[5]In contrast with [55], this optimization for ratio computation is enabled by the fact that we wish to find supported itemsets to avoid overwhelming the user with explanations.

From the first stage's outlier attribute values, MDP then computes supported outlier attribute combinations. Finally, MDP computes the OI-ratio for each attribute combination based on their support in the inliers (support counting, followed by a filtering pass to exclude any attribute sets with insufficient OI-ratio).

Significance. We discuss confidence intervals on MDP explanations as well as quality improvements achievable by processing large data volumes in Appendix B.

Algorithms and Data Structures. In the one-pass setting, single attribute value counting is straightforward, requiring a single pass over the data; the streaming setting below is more interesting. We experimented with several itemset mining techniques and ultimately decided on prefix-tree-based approaches inspired by FPGrowth [39]. In brief, the FPGrowth algorithm maintains a frequency-descending prefix tree of attributes that can subsequently be mined by recursively generating a set of "conditional" trees. Corroborating recent benchmarks [33], the FPGrowth algorithm was fast and proved extensible in our streaming implementation below.

5.3 Streaming Explanation

As in MDP detection, streaming explanation mining proved more challenging. We again turn to the sketching and streaming literature in order to provide theoretically justified but practical summarization. We present the MDP implementation of single-attribute streaming explanation, then extend the approach to multi-attribute streaming explanation.

Implementation: Single Attribute Summarization. In a streaming setting, we wish to find individual attributes with sufficient support and OI-ratio. We would like to do so in a way that both respects bias in the stream and limits the overall amount of memory required to store the counts. Fortunately, the problem of maintaining a count of frequent items (i.e., *heavy hitters*, or attributes with top k occurrence) in data streams is well studied [24]. Given a heavy-hitters sketch over the inlier and outlier stream, we can compute an approximate support and OI-ratio for each attribute by comparing the contents of the sketches at any time.

Initially, we implemented the MDP item counter using the SpaceSaving algorithm [61], which has been demonstrated to provide excellent performance [25], with extensions to the exponentially decayed setting [26]. However, like many of the sketches in the literature, SpaceSaving was designed to strike a balance between sketch size and performance, with a strong emphasis on limited size. For example, in its heap-based variant, SpaceSaving maintains $\frac{1}{k}$-approximate counts for the top k item counts by maintaining a heap of the items. For a stream of size n, this requires $O(n\log(k))$ update time. (In the case of exponential decay, the linked-list-based variant can require $O(n^2)$ processing time.)

While logarithmic update time is modest for small sketches, given there are only two heavy-hitters sketches per MacroBase query, MDP can expend more memory on its sketches for better accuracy; for example, $1M$ items require less than ten megabytes of memory for double-encoded counts, which is small given modern server memory sizes. As a result, we developed a heavy-hitters sketch, called the *Amortized Maintenance Counter* (AMC, Algorithm 3), that occupies the opposite end of the design spectrum: the AMC uses a much greater amount of memory for a given accuracy level but is in turn faster to update and still limits total space utilization. The key insight behind the AMC is that, if we observe even a single item in the stream more than once, we can amortize the overhead of maintaining the sketch across multiple observations. In contrast, SpaceSaving maintains the sketch for every observation but in turn keeps the sketch size smaller.

Algorithm 3 AMC: Amortized Maintenance Counter

```
given: ε ∈ (0,1); r: decay rate ∈ (0,1)
initialization: (item → count) C ← {}; weight w_i ← 0
function OBSERVE(i: item, c: count)
    C[i] ← w_i + c if i ∉ C else C[i] + c
function MAINTAIN( )
    remove all but the 1/ε largest entries from C
    w_i ← the largest value just removed, or, if none removed, 0
function DECAY( )
    decay the value of all entries of C by r
    call MAINTAIN( )
```

AMC provides the same counting functionality as a traditional heavy-hitters sketch but exposes a second method, *maintain*, that is called to periodically prune the sketch size. AMC allows the sketch size to increase between calls to *maintain*, and, during maintenance, the sketch size is reduced to a desired stable size, which is specified as an input parameter. Therefore, the maximum size of the sketch is controlled by the period between calls to *maintain*: as in SpaceSaving, a stable size of $\frac{1}{\varepsilon}$ yields an $n\varepsilon$ approximation of the count of n points, but the size of the sketch may grow within a period. This separation of insertion and maintenance has two implications. First, it allows constant-time insertion, which we describe below. Second, it allows a range of maintenance policies, including a sized-based policy, which performs maintenance once the sketch reaches a prespecified upper bound, as well as a variable period policy, which operates over real-time and tuple-based windows (similar to ADR).

To implement this functionality, AMC maintains a set of approximate counts for all items that were among the most common in the previous period along with approximate counts for all other items that observed in the current period. During maintenance, AMC prunes all but the $\frac{1}{\varepsilon}$ items with highest counts and records the maximum count that is discarded (w_i). Upon insertion, AMC checks to see if the item is already stored. If so, the item's count is incremented. If not, AMC stores the item count plus w_i. If an item is not stored in the current window, the item must have had count less than or equal to w_i at the end of the previous period.

AMC has three major differences compared to SpaceSaving. First, AMC updates are constant time (hash table insertion) compared to $O(\log(\frac{1}{\varepsilon}))$ for SpaceSaving. Second, AMC has an additional maintenance step, which is amortized across all items seen in a window. Using a min-heap, with I items in the sketch, maintenance requires $O(I \cdot \log(\frac{1}{\varepsilon}))$ time. If we observe even one item more than once, this is faster than performing maintenance on every observation. Third, AMC has higher space overhead; in the limit, it must maintain all items it has seen between maintenance windows.

Implementation: Streaming Combinations. While AMC tracks single items, MDP also needs to track combinations of attributes. As such, we sought a tree-based technique that would admit exponentially damped arbitrary windows but eliminate the requirement that each attribute be stored in the tree, as in recent proposals such as the CPS-tree [77]. As a result, MDP adapts a combination of two data structures: AMC for the frequent attributes, and a novel adaptation of the CPS-Tree data structure to store frequent attributes.

Given the set of recently frequent items, MDP monitors the attribute stream for frequent attribute combinations by maintaining an approximately frequency-descending prefix tree of attribute values. MDP adopts the basic CPS-tree data structure [77], with several modifications, which we call the M-CPS-tree.

Like the CPS-tree, the M-CPS-tree maintains both the basic FP-tree data structures as well as a set of leaf nodes in the tree. However,

in an exponentially damped model, the CPS-tree stores at least one node for every item ever observed in the stream. This is infeasible at scale. As a compromise, the M-CPS-tree only stores items that were frequent in the previous window: at each window boundary, MacroBase updates the frequent item counts in the M-CPS-tree based on an AMC sketch. Any items that were frequent in the previous window but were not frequent in this window are removed from the tree. MacroBase then decays all frequency counts in the M-CPS-tree nodes and re-sorts the M-CPS-tree in frequency descending order (as in the CPS-tree, by traversing each path from leaf to root and re-inserting as needed). Subsequently, attribute insertion can continue as in the FP-tree, with the caveat that the item order is not maintained until query time.

In summary, MDP's streaming explanation operator consists of two primary parts: maintenance and querying. When a new data point arrives at the MacroBase summarization operator, MacroBase inserts each of the point's attributes into an AMC sketch. Subsequently, MacroBase inserts a subset of the point's attributes into a prefix tree that maintains an approximate, frequency descending order. When a window has elapsed, MacroBase decays the counts of the items as well as the counts in each node of the prefix tree. MacroBase removes any items that are no longer supported and rearranges the prefix tree in frequency-descending order. To produce explanations, MacroBase runs FPGrowth on the prefix tree.

6. EVALUATION

In this section, we evaluate the accuracy, efficiency, and flexibility of MacroBase and the MDP pipeline operators. Our goals are to demonstrate that:

- MacroBase's MDP pipeline is accurate: on controlled, synthetic data, under changes in stream behavior (via ADR), and over real-world workloads from the literature and in production (Section 6.1).
- MacroBase can process up to 2M points per second per query on a range of real-world datasets (Section 6.2).
- MacroBase's cardinality-aware explanation strategy produces meaningful speedups (average: 3.2× speedup; Section 6.3).
- MacroBase's use of AMC is up to 500× faster than existing sketches on production IoT data (Section 6.3).
- MacroBase's architecture is extensible, which we illustrate via three case studies (Section 6.4).

Experimental environment. We report results from deploying the MacroBase prototype on a server with four Intel Xeon E5-4657L 2.40GHz CPUs containing 12 cores per CPU and 1TB of RAM. To isolate the effects of pipeline processing, we exclude loading time from our results. By default, we issue MDP queries with a minimum support of 0.1% and minimum OI-ratio of 3, a target outlier percentile of 1%, ADR and AMC sizes of 10K, a decay rate of 0.01 every 100K points, and report the average of at least three runs per experiment. We vary these parameters in subsequent experiments in this section and the Appendix.

Implementation. We describe MacroBase's implementation, processing runtime, and approach to parallelism in Appendix C.

Large-scale datasets. To compare the efficiency of MacroBase and related techniques, we compiled a set of large-scale real-world datasets (Table 2) for evaluation (descriptions in Appendix D).

6.1 Result Quality

In this section, we focus on MacroBase's statistical result quality. We evaluate precision/recall on synthetic and real-world data, demonstrate adaptivity to changes in data streams, and report on experiences in production deployment.

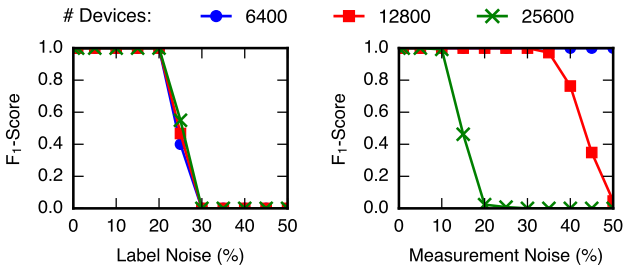


Figure 4: Precision-recall of explanations. Without noise, MDP exactly identifies misbehaving devices. MDP's use of OI-ratio improves resiliency to both label and measurement noise.

Synthetic dataset accuracy. We ran MDP over a synthetic dataset generated in line with those used to evaluate recent anomaly detection systems [70, 81]. The generated dataset contains 1M data points across a number of synthetic devices, each of which maps to a unique attribute ID and whose metrics are drawn from either an inlier distribution ($\mathcal{N}(10,10)$) or outlier distribution ($\mathcal{N}(70,10)$). We subsequently evaluated MacroBase's ability to automatically determine the device IDs corresponding to the outlying distribution. We report the F_1-score $\left(2 \cdot \frac{precision \cdot recall}{precision+recall}\right)$ for the set of device IDs identified as outliers as a proxy for reporting explanation quality.

Since MDP's statistical techniques are a natural match for this experimental setup, we also perturbed the base experiment to understand when MDP might underperform. We introduced two types of noise into the measurements to quantify their effects on MDP's performance. First, we introduced *label noise* by randomly assigning readings from the outlier distribution to inlying devices and vice-versa. Second, we introduced *measurement noise* by randomly assigning a proportion of both outlying and inlying points to a third, uniform distribution over the interval $[0, 80]$.

Figure 4 illustrates the results. In the the noiseless regions of Figure 4, MDP correctly identified 100% of the outlying devices. As the outlying devices are solely drawn from the outlier distribution, constructing outlier explanations via the OI-ratio enables MacroBase to perfectly recover the outlying device IDs. In contrast, techniques that rely solely on individual outlier classification deliver less accurate results on this workload [70, 81]. Under label noise, MacroBase robustly identified the outlying devices until approximately 25% noise, which corresponds a $3:1$ ratio of correct to incorrect labels. As our default OI-ratio is set to 3, exceeding this threshold causes rapid performance degradation. Under measurement noise, accuracy degrades linearly with the amount of noise. MDP is more robust to this type of noise when fewer devices are present. MDP accuracy suffers with a larger number of devices, as each device type is subject to more noisy readings.

In summary, for well-behaved data (e.g., $< 10-20\%$ noisy points), MDP is able to accurately identify correlated causes of outlying data. The noise threshold is improved by both MDP's use of robust methods as well as the use of OI-ratio to prune irrelevant summaries. Noise of this magnitude is likely rare in practice, and, if such noise exists, is likely indicative of another interesting behavior in the data.

Real world dataset accuracy. In addition to synthetic data, we also performed experiments to determine MacroBase's ability to accurately identify systemic abnormalities in real-world data. We evaluated MacroBase's ability to distinguish abnormally-behaving OLTP servers within a cluster, as defined according to data and man-

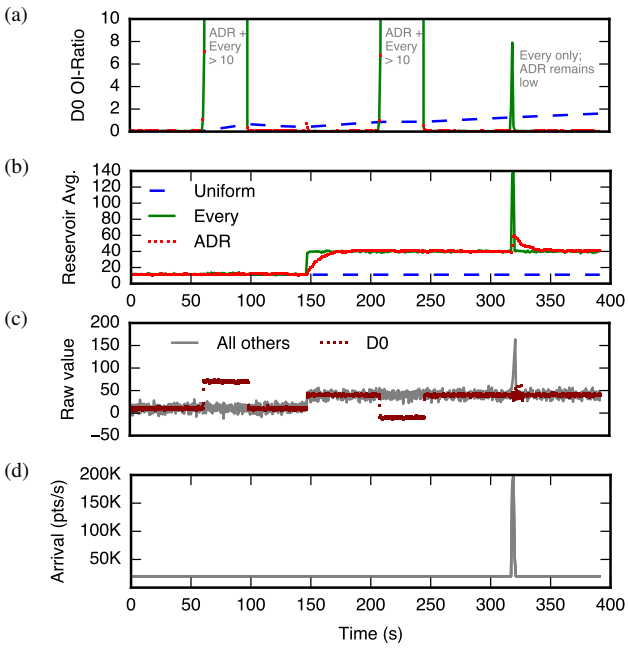


Figure 5: ADR provides greater adaptivity compared to tuple-at-a-time reservoir sampling and is more resilient to spikes in data volume (see text for details).

ual labels collected in a recent study [82] to diagnose performance issues within a single host. We performed a set of experiments, each corresponding to a distinct type of performance degradation within MySQL on a particular OLTP workload (TPC-C and TPC-E). For each experiment, we consider a cluster of eleven servers, where a single server behaves anomalously. Using over 200 operating systems and database performance counters, we ran MDP to identify the anomalous server.

We ran MDP with two sets of queries. In the former set, QS, MDP executed a query to find abnormal hosts using a single set of 15 metrics identified via feature selection techniques on a holdout of 2 clusters per experiment (i.e., one query for all anomalies). In the second set, QE, MDP executed a slow-hosts query using a set of metrics for each distinct anomaly type (e.g., network contention), again using a holdout of 2 clusters per experiment (i.e., one query per anomaly type).

As Table 4 (Appendix D) shows, under QS, MDP achieves top-1 accuracy of 86.1% on the holdout set across all forms of anomalies (top-3: 88.8%). For eight of nine anomalies, MDP's top-1 accuracy is even higher: 93.8%. However, for the ninth anomaly, which corresponds to a poorly written query, the metrics correlated with the anomalous behavior are substantially different. In contrast, QE, which targets each anomaly with a custom set of metrics, is able to identify behaviors reliably, leading to perfect top-3 accuracy.

Adaptivity. While the previous set of experiments operated over statically distributed data, we sought to understand the benefit of MDP's ability to adapt to changes in the input distribution via the exponential decay of ADR and AMC. We performed a controlled experiment over two types of time-varying behavior: changing underlying data distribution, and variable data arrival rate. We then compared the accuracy of MDP outlier detection across three sampling techniques: a uniform reservoir sample, a per-tuple exponentially decaying reservoir sample, and our proposed ADR.

Figure 5c displays the time-evolving stream representing 100 devices over which MDP operates. To begin, all devices produce readings drawn from a Gaussian $\mathcal{N}(10,10)$ distribution. After 50 seconds, a single device, $D0$, produces readings from $\mathcal{N}(70,10)$ before returning to the original distribution at 100 seconds. The second period (150s to 300s) is similar to the first, except we also introduce a shift in all devices' metrics: after 150 seconds, all devices produce readings from $\mathcal{N}(40,10)$, and, after 225 seconds, $D0$ produces readings from $\mathcal{N}(-10,10)$, returning to $\mathcal{N}(40,10)$ after 250 seconds. Finally from 300s to 400s, all devices experience a spike in data arrival rate. We introduce a four-second noise spike in the sensor readings at 320 seconds: the arrival rate rises by ten-fold, to over 200k points per second, with corresponding values drawn from a $\mathcal{N}(85,15)$ distribution (Figure 5d).

In the first time period, all three strategies detect $D0$ as an outlier, as reflected in its AMC OI-ratio in Figure 5a. After 100 seconds, when $D0$ returns to the inlier distribution, its OI-ratio drops. The reservoir averages remain unchanged in all strategies (Figure 5b). In the second time period, both adaptive reservoirs adjust to the new distribution by 170 seconds, while the uniform reservoir fails to adapt quickly (Figure 5b). As such, when $D0$ drops to $\mathcal{N}(-10,10)$ from time 225 through 250, only the two adaptive strategies track the change (Figure 5a). At time 300, the short noise spike appears in the sensor readings. The per-tuple reservoir is forced to absorb this noise, and the distribution in this reservoir spikes precipitously. As a result, $D0$, which remains at $\mathcal{N}(40,10)$ is falsely suspected as outlying. In contrast, the ADR average value rises slightly but never suspects $D0$ as an outlier. This illustrates the value of adaptivity to distribution changes and resilience to variable arrival rates.

Production results. MacroBase currently operates over a range of production data and has delivered valuable results in several domains. Here, we report on our experiences deploying MacroBase at CMT, where it identified several previously unknown behaviors. In one case, MacroBase highlighted a small number of users who experienced issues with their trip detection. In another case, MacroBase discovered a rare issue with the CMT application and a device-specific battery problem. Consultation and investigation with the CMT team confirmed these issues as previously unknown, and have since been addressed. This experience proved a useful demonstration of MacroBase's analytical monitoring functionality in a production environment and inspired several ongoing extensions (Section 7).

6.2 End-to-End Performance

In this section, we evaluate MacroBase's end-to-end performance on real-world datasets. For each dataset X, we execute two MacroBase queries: a simple query, with a single attribute and metric (denoted XS), and a complex query, with a larger set of attributes and, when available, multiple metrics (denoted XC). We then report throughput for two system configurations: one-shot execution that processes each stage in sequence and exponentially-weighted streaming execution (EWS) that processes points continuously. One-shot and EWS have different semantics, as reflected in the explanations they produce. While one-shot execution examines the entire dataset at once, exponentially weighted streaming prioritizes more recent points. Therefore, for datasets with few distinct attribute values (e.g., Accidents contains only nine types of weather conditions), the explanations will have high similarity. However, for datasets with many distinct attribute values (typically the complex queries, which have hundreds of thousands of possible combinations—e.g., Disburse has 138,338 different disbursement recipients), the explanations tend to differ. For this reason, we provide throughput results both with and without explanations, as well as the number of explanations generated by the simple (XS) and complex (XC) queries and their Jaccard similarity.

Dataset	Queries: Name	Metrics	Attrs	Points	Thru w/o Explain (pts/s): One-shot	EWS	Thru w/ Explain (pts/s): One-shot	EWS	# Explanations: One-shot	EWS	Jaccard Similarity
Liquor	LS	1	1	3.05M	1549.7K	967.6K	1053.3K	966.5K	28	33	0.74
	LC	2	4		385.9K	504.5K	270.3K	500.9K	500	334	0.35
Telecom	TS	1	1	10M	2317.9K	698.5K	360.7K	698.0K	469	1	0.00
	TC	5	2		208.2K	380.9K	178.3K	380.8K	675	1	0.00
Campaign	ES	1	1	10M	2579.0K	778.8K	1784.6K	778.6K	2	2	0.67
	EC	1	5		2426.9K	252.5K	618.5K	252.1K	22	19	0.17
Accidents	AS	1	1	430K	998.1K	786.0K	729.8K	784.3K	2	2	1.00
	AC	3	3		349.9K	417.8K	259.0K	413.4K	25	20	0.55
Disburse	FS	1	1	3.48M	1879.6K	1209.9K	1325.8K	1207.8K	41	38	0.84
	FC	1	6		1843.4K	346.7K	565.3K	344.9K	1710	153	0.05
CMTD	MS	1	1	10M	1958.6K	564.7K	354.7K	562.6K	46	53	0.63
	MC	7	6		182.6K	278.3K	147.9K	278.1K	255	98	0.29

Table 2: Datasets and query names, throughput, and explanations produced under one-shot and exponentially weighted streaming (EWS) execution. MacroBase's sustained throughput is up to three orders of magnitude higher than related systems.

Table 2 displays results across all queries. Throughput varied from 147K points per second (on *MC* with explanation) to over 2.5M points per second (on *TS* without explanation); the average throughput for one-shot execution was 1.39M points per second, and the average throughput for EWS was 599K points per second. The better-performing mode depended heavily on the particular data set and characteristics. In general, queries with multiple metrics were slower in one-shot than queries with single metrics (due to increased training time, as streaming trains over samples). Mining for each explanation incurred an approximately 22% overhead. In all cases, these queries far exceed the current arrival rate of data for each dataset. EWS typically returned fewer explanations (Telecom provides an extreme example) due to its temporal bias. In practice, users tune their decay on a per-application basis (e.g., at CMT, streaming queries may prioritize trips from the last hour to catch errors arising from the most recent deployment). These throughputs exceed those of related techniques we have encountered in the literature (by up to three orders of magnitude); we examine specific factors that contribute to this performance in the next section.

Runtime profiling. To further understand how each pipeline operator contributed to overall performance, we profiled MacroBase's one-shot execution (EWS is more challenging to accurately instrument due to its streaming execution). On *MC*, MacroBase spent approximately 52% of its execution training MCD, 21% scoring points, and 26% summarizing them. On *MS*, MacroBase spent approximately 54% of its execution training MAD, 16% scoring points, and 29% summarizing them. In contrast, on *FC*, which returned over 1000 explanations, MacroBase spent 31% of its execution training MAD, 4% scoring points, and 65% summarizing them. Thus, the overhead of each component is dependent on the query.

6.3 Microbenchmarks and Comparison

In this section, we explore two key aspects of MacroBase's design: cardinality-aware explanation and use of AMC sketches.

Cardinality-aware explanation. We evaluated the efficiency of MacroBase's OI-pruning compared to traditional FPGrowth. MacroBase leverages a unique pruning strategy that exploits the low cardinality of outliers, which delivers large speedups—on average, over 3× compared to unoptimized FPGrowth. Specifically, MacroBase's produces a summary of each datasets's inliers and outliers in 0.22–1.4 seconds. In contrast, running FPGrowth separately on inliers and outliers is, on average, 3.2× slower; compared to MacroBase's joint explanation according to support and OI-ratio, much of the time spent mining inliers (with insufficient OI-ratio) in FPGrowth is wasted. However, both MacroBase and FPGrowth must perform a linear pass over all of the inliers, which places a lower bound on the running time. The exact benefit of this explanation strategy depends on the OI-ratio (here, 3), which we vary in Appendix D.

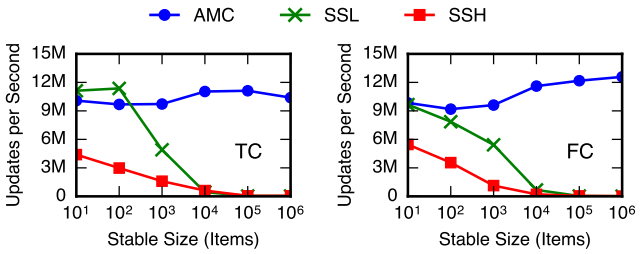


Figure 6: Streaming heavy hitters sketch comparison. AMC: Amortized Maintenance Counter with maintenance every 10K items; SSL: Space Saving List; SSH: Space Saving Hash. All share the same accuracy bound. Varying maintenance period produced similar results.

AMC Comparison. We also compare the performance of AMC with existing heavy-hitters sketches (Figure 6). AMC outperforms both implementations of SpaceSaving in all configurations by a margin of up to 500× except for SpaceSaving implemented using a list with fewer than 100 items. This is because the overhead of heap maintenance on every operation is expensive with even modestly-sized sketches, while the list traversal is costly for decayed, non-integer counts. In contrast, with an update period of 10K points, AMC sustains in excess of 10M updates per second. The primary cost of these performance improvements is additional space: for example, with a minimum sketch size of 10 items and update period of 10K points, AMC retains up to 10,010 items while each SpaceSaving sketch retains only 10. As a result, when memory sizes are especially constrained, SpaceSaving may be preferable, at a measurable cost to performance.

Additional results. In Appendix D, we provide additional results examining the distribution of outlier scores, the effect of varying support and OI-ratio, the effect of training over samples and operating over varying metric dimensions, the behavior of the M-CPS tree, preliminary scale-out behavior, comparing the runtime of MDP explanation to both existing batch explanation procedures, and MDP detection and explanation to operators from frameworks including Weka, Elki, and RapidMiner.

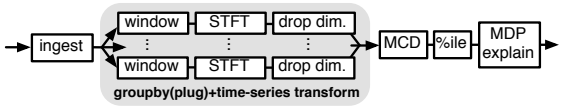
6.4 Case Studies and Extensibility

MacroBase is designed for extensibility, as we demonstrate via three case studies in three separate IoT domains. We report on the pipeline structures, performance, and interesting explanations from applying MacroBase over: hybrid supervised-unsupervised models, time-series data, and video surveillance data.

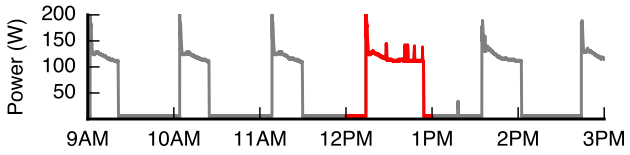
Hybrid Supervision. We demonstrate MacroBase's ability to easily synthesize supervised and unsupervised outlier detection models via a use case from CMT. Each trip in the CMTD dataset is accompanied by a supervised diagnostic score representing the trip quality. MDP's unsupervised operators can use this score as an input, but CMT also wishes to capture low-quality scores independent of their distribution in the population. Accordingly, we authored a new MacroBase pipeline that feeds CMT metrics to the MDP MCD operator but also feeds the diagnostic metric to a special rule-based operator that flags low quality scores as anomalies. The pipeline, which we depict below, performs a logical *or* over the two classification results:

With this hybrid supervision strategy, MacroBase identified additional behaviors within the CMTD dataset. Since the quality scores were generated external to MacroBase and the supervision rule in MacroBase was lightweight, runtime was unaffected. This kind of pipeline can easily be extended to more complex supervised models.

Time-series. Many interesting behaviors in IoT data are time-varying; MacroBase's architecture can detect these via feature transformation, which we demonstrate using a dataset of 16M points capturing a month of electricity usage from devices within a household [11]. We augment MDP by adding a sequence of feature transforms that *i.*) partition the stream by device ID, *ii.*) window the stream into hourly intervals, with attributes according to hour of day, day of week, and date, then *iii.*) apply a Discrete-Time Short-Term Fourier Transform (STFT) to each window, and truncate the transformed data to a fixed number of dimensions. As the diagram below shows, we feed the transformed stream into an unmodified MDP pipeline and search for outlying time periods and devices:

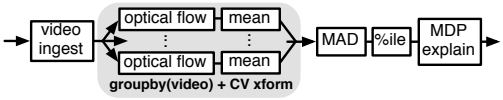


With this time-series pipeline, MacroBase detected several systemic periods of abnormal device behavior. For example, the following dataset of power usage by a household refrigerator spiked on an hourly basis (likely corresponding to compressor activity); instead of highlighting the hourly power spikes, MacroBase was able to detect that the refrigerator consistently behaved abnormally compared to other devices in the household and compared to other time periods between the hours of 12PM and 1PM—presumably, lunchtime—as highlighted in the excerpt below:

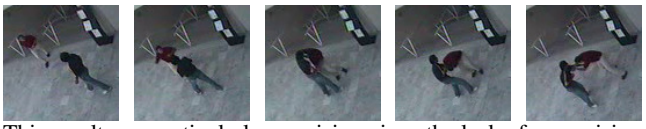


Without feature transformation, the entire MDP pipeline completed in 158ms. Feature transformation dominated the runtime, utilizing 516 seconds to transform the 16M points via unoptimized STFT.

Video Surveillance. We further highlight MacroBase's ability to easily operate over a wide array of data sources and domains by searching for interesting patterns in the CAVIAR video surveillance dataset [1]. Using OpenCV 3.1.0, we compute two types of time-series video feature transformations: first, we compute the average difference between pixels, and, second, we compute the average optical flow velocity. Each transformed frame is tagged with a time interval attribute, which we use to identify interesting video segments and, as depicted below, the remainder of the pipeline executes the standard MDP operators:



Using this pipeline, MacroBase detected periods of abnormal motion in the video dataset. Optical flow, which has been successfully applied in human action detection [29] produced particularly interesting results: for example, the MacroBase pipeline highlighted a three-second period in which two characters fought onscreen:



This result was particularly surprising given the lack of supervision in the pipeline. Like our STFT pipeline, feature transformation via optical flow dominated runtime (22s vs. 34ms for MDP); this is unsurprising given our CPU-based implementation of a relatively expensive algorithm but nevertheless illustrates MDP's ability to process video streams.

7. CONCLUSIONS AND FUTURE WORK

We have presented MacroBase, a new analytics engine designed to facilitate extensible, accurate, and fast analytic monitoring of IoT data streams. MacroBase provides a flexible architecture that combines streaming outlier detection and data explanation techniques to deliver interpretable summaries of unusual behavior in streams. MacroBase's default analytic monitoring operators, which include new sampling and sketching procedures, take advantage of this combination of detection and explanation and are specifically optimized for high-volume, time-sensitive, and heterogeneous IoT scenarios, resulting in improved performance and result quality. This emphasis on flexibility, accuracy, and speed has proven useful in several production deployments, where MacroBase has already identified previously unknown behaviors at high volume.

MacroBase is available as open source and is under active development. In addition, the system serves as the vehicle for a number of ongoing research efforts, including decentralized, edge-based analytic monitoring, temporally-aware explanation techniques, heterogeneous sensor data fusion, online non-parametric density estimation, and contextual outlier detection. New production use cases continue to stimulate the development of new analytic monitoring functionality to expand the set of supported domains and leverage the flexibility provided by MacroBase's pipeline architecture.

Acknowledgments

The authors would like to thank the many members of the Stanford InfoLab as well as the early production users of MacroBase for feedback and inspiration in the development of MacroBase; Ali Ghodsi, Joe Hellerstein, Mark Phillips, and Leif Walsh for useful feedback on this work; Xu Chu, Vivek Jain, Vayu Kishore, Ihab Ilyas, and Arsen Mamikonyan for ongoing contributions to the MacroBase

engine; and Pavlos S. Efraimidis for his lucid explanation and discussion of weighted reservoir sampling. This research is supported in part by Toyota Research, RWE AG, and Phillips Lighting.

8. REFERENCES

[1] Caviar test case scenarios. http://homepages.inf.ed.ac.uk/rbf/CAVIAR/.

[2] Opengamma, 2015. http://www.opengamma.com/.

[3] Summary of the Amazon DynamoDB service disruption and related impacts in the US-East region, 2015. https://aws.amazon.com/message/5467D2/.

[4] D. J. Abadi et al. The design of the borealis stream processing engine. In *CIDR*, 2005.

[5] C. C. Aggarwal. On biased reservoir sampling in the presence of stream evolution. In *VLDB*, 2006.

[6] C. C. Aggarwal. *Data streams: models and algorithms*, volume 31. Springer Science & Business Media, 2007.

[7] C. C. Aggarwal. *Outlier Analysis*. Springer, 2013.

[8] R. Ahmad and S. Kamaruddin. An overview of time-based and condition-based maintenance in industrial application. *Computers & Industrial Engineering*, 63(1):135–149, 2012.

[9] A. Arasu, S. Babu, and J. Widom. The cql continuous query language: semantic foundations and query execution. *The VLDB Journal*, 15(2):121–142, 2006.

[10] A. Asta. Observability at Twitter: technical overview, part i, 2016. https://blog.twitter.com/2016/observability-at-twitter-technical-overview-part-i.

[11] C. Beckel, W. Kleiminger, R. Cicchetti, T. Staake, and S. Santini. The ECO data set and the performance of non-intrusive load monitoring algorithms. In *BuildSys*. ACM, 2014.

[12] Y. Benjamini and D. Yekutieli. The control of the false discovery rate in multiple testing under dependency. *Annals of statistics*, pages 1165–1188, 2001.

[13] R. J. Bolton and D. J. Hand. Statistical fraud detection: A review. *Statistical science*, pages 235–249, 2002.

[14] C. Buragohain and S. Suri. Quantiles on streams. In *Encyclopedia of Database Systems*, pages 2235–2240. Springer, 2009.

[15] R. Butler, P. Davies, and M. Jhun. Asymptotics for the minimum covariance determinant estimator. *The Annals of Statistics*, pages 1385–1400, 1993.

[16] L. Cao, Q. Wang, and E. A. Rundensteiner. Interactive outlier exploration in big data streams. *Proceedings of the VLDB Endowment*, 7(13):1621–1624, 2014.

[17] L. Cao, D. Yang, Q. Wang, Y. Yu, J. Wang, and E. A. Rundensteiner. Scalable distance-based outlier detection over high-volume data streams. In *ICDE*, 2014.

[18] V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):15, 2009.

[19] B. Chandramouli, J. Goldstein, et al. Trill: A high-performance incremental query processor for diverse analytics. In *VLDB*, 2014.

[20] S. Chandrasekaran et al. TelegraphCQ: Continuous dataflow processing for an uncertain world. In *CIDR*, 2003.

[21] M. Chao. A general purpose unequal probability sampling plan. *Biometrika*, 69(3):653–656, 1982.

[22] M. Chen, A. X. Zheng, J. Lloyd, M. I. Jordan, and E. Brewer. Failure diagnosis using decision trees. In *ICAC*, 2004.

[23] J. Cheng et al. A survey on algorithms for mining frequent itemsets over data streams. *Knowledge and Information Systems*, 16(1):1–27, 2008.

[24] G. Cormode, M. Garofalakis, P. J. Haas, and C. Jermaine. Synopses for massive data: Samples, histograms, wavelets, sketches. *Foundations and Trends in Databases*, 4(1–3):1–294, 2012.

[25] G. Cormode and M. Hadjieleftheriou. Methods for finding frequent items in data streams. *The VLDB Journal*, 19(1):3–20, 2010.

[26] G. Cormode, F. Korn, and S. Tirthapura. Exponentially decayed aggregates on data streams. In *ICDE*. IEEE, 2008.

[27] C. Cranor, T. Johnson, O. Spataschek, and V. Shkapenyuk. Gigascope: a stream database for network applications. In *SIGMOD*, 2003.

[28] P. S. Efraimidis. Weighted random sampling over data streams. In *Algorithms, Probability, Networks, and Games*, pages 183–195. Springer, 2015.

[29] A. A. Efros, A. C. Berg, G. Mori, and J. Malik. Recognizing action at a distance. In *ICCV*, 2003.

[30] K. El Gebaly, P. Agrawal, L. Golab, F. Korn, and D. Srivastava. Interpretable and informative explanations of outcomes. In *VLDB*, 2014.

[31] T. Escamilla. *Intrusion detection: network security beyond the firewall*. John Wiley & Sons, Inc., 1998.

[32] X. Feng, A. Kumar, B. Recht, and C. Ré. Towards a unified architecture for in-RDBMS analytics. In *SIGMOD*, 2012.

[33] P. Fournier-Viger. SPMF: An Open-Source Data Mining Library – Performance, 2015. http://www.philippe-fournier-viger.com/spmf/.

[34] P. H. Garthwaite and I. Koch. Evaluating the contributions of individual variables to a quadratic form. *Australian & New Zealand Journal of Statistics*, 58(1):99–119, 2016.

[35] L. Girod, K. Jamieson, Y. Mei, R. Newton, S. Rost, A. Thiagarajan, H. Balakrishnan, and S. Madden. Wavescope: a signal-oriented data stream management system. In *ICDE*, 2006.

[36] M. Goldstein and S. Uchida. A comparative evaluation of unsupervised anomaly detection algorithms for multivariate data. *PLoS ONE*, 11(4):1–31, 04 2016.

[37] M. Hall et al. The WEKA data mining software: an update. *ACM SIGKDD explorations newsletter*, 11(1):10–18, 2009.

[38] J. Han et al. Frequent pattern mining: current status and future directions. *Data Mining and Knowledge Discovery*, 15(1):55–86, 2007.

[39] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *SIGMOD*, 2000.

[40] J. Hardin and D. M. Rocke. Outlier detection in the multiple cluster setting using the Minimum Covariance Determinant estimator. *Computational Statistics & Data Analysis*, 44(4):625–638, 2004.

[41] J. Hardin and D. M. Rocke. The distribution of robust distances. *Journal of Computational and Graphical Statistics*, 14(4):928–946, 2005.

[42] J. M. Hellerstein. Quantitative data cleaning for large databases. *United Nations Economic Commission for Europe (UNECE)*, 2008.

[43] V. J. Hodge and J. Austin. A survey of outlier detection methodologies. *Artificial Intelligence Review*, 22(2):85–126, 2004.

[44] M. Hofmann and R. Klinkenberg. *RapidMiner: Data mining use cases and business analytics applications*. CRC Press, 2013.

[45] P. J. Huber. *Robust statistics*. Springer, 2011.

[46] M. Hubert and M. Debruyne. Minimum covariance determinant. *Wiley interdisciplinary reviews: Computational statistics*, 2(1):36–43, 2010.

[47] W. H. Hunter. US Chemical Safety Board: analysis of Horsehead Corporation Monaca Refinery fatal explosion and fire, 2015. http://www.csb.gov/horsehead-holding-company-fatal-explosion-and-fire/.

[48] J.-H. Hwang, M. Balazinska, et al. High-availability algorithms for distributed stream processing. In *ICDE*, 2005.

[49] IDC. The digital universe of opportunities: Rich data and the increasing value of the internet of things, 2014. http://www.emc.com/leadership/digital-universe/.

[50] I. F. Ilyas and X. Chu. Trends in cleaning relational data: Consistency and deduplication. *Foundatations and Trends in Databases*, 2015.

[51] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly. Dryad: distributed data-parallel programs from sequential building blocks. In *EuroSys*, 2007.

[52] R. Jampani, F. Xu, M. Wu, L. L. Perez, C. Jermaine, and P. J. Haas. MCDB: a Monte Carlo approach to managing uncertain data. In *SIGMOD*, 2008.

[53] Y. Klonatos, C. Koch, T. Rompf, and H. Chafi. Building efficient query engines in a high-level language. In *VLDB*, 2014.

[54] S. Kulkarni, N. Bhagat, M. Fu, V. Kedigehalli, C. Kellogg, S. Mittal, J. M. Patel, K. Ramasamy, and S. Taneja. Twitter heron: Stream processing at scale. In *SIGMOD*, 2015.

[55] H. Li, J. Li, L. Wong, M. Feng, and Y.-P. Tan. Relative risk and odds ratio: A data mining perspective. In *PODC*, 2005.

[56] J. Lin, E. Keogh, and S. Lonardi. Visualizing and discovering non-trivial patterns in large time series databases. *Information visualization*, 4(2):61–82, 2005.

[57] J. Manyika. *McKinsey Global Institute: The internet of things: mapping the value beyond the hype*. 2015.

[58] R. Manzini, A. Regattieri, H. Pham, and E. Ferrari. *Maintenance for industrial systems*. Springer Science & Business Media, 2009.

[59] R. Maronna, D. Martin, and V. Yohai. *Robust statistics*. John Wiley & Sons, Chichester. ISBN, 2006.

[60] A. Meliou, S. Roy, and D. Suciu. Causality and explanations in databases. In *VLDB*, 2014.

[61] A. Metwally et al. Efficient computation of frequent and top-k elements in data streams. In *ICDT*. Springer, 2005.

[62] J. A. Morris and M. J. Gardner. Statistics in medicine: Calculating confidence intervals for relative risks (odds ratios) and standardised ratios and rates. *British medical journal (Clinical research ed.)*, 296(6632):1313, 1988.

[63] B. Mukherjee, L. T. Heberlein, and K. N. Levitt. Network intrusion detection. *Network, IEEE*, 8(3):26–41, 1994.

[64] V. Nair et al. Learning a hierarchical monitoring system for detecting and diagnosing service issues. In *KDD*, 2015.

[65] OpenSignal. Android fragmentation visualized (august 2015). http://opensignal.com/reports/2015/08/android-fragmentation/.

[66] C. Phua, V. Lee, K. Smith, and R. Gayler. A comprehensive survey of data mining-based fraud detection research. *arXiv preprint arXiv:1009.6119*, 2010.

[67] D. Pokrajac, A. Lazarevic, and L. J. Latecki. Incremental local outlier detection for data streams. In *CIDM*, 2007.

[68] B. Reiser. Confidence intervals for the Mahalanobis distance. *Communications in Statistics-Simulation and Computation*, 30(1):37–45, 2001.

[69] P. J. Rousseeuw and K. V. Driessen. A fast algorithm for the minimum covariance determinant estimator. *Technometrics*, 41(3):212–223, 1999.

[70] S. Roy, A. C. König, I. Dvorkin, and M. Kumar. Perfaugur: Robust diagnostics for performance anomalies in cloud services. In *ICDE*, 2015.

[71] S. Roy and D. Suciu. A formal approach to finding explanations for database queries. In *SIGMOD*, 2014.

[72] G. Rupert Jr et al. *Simultaneous statistical inference*. Springer Science & Business Media, 2012.
[73] F. I. Rusu. *Sketches for aggregate estimations over data streams*. PhD thesis, University of Florida, 2009.
[74] E. Schubert, A. Koos, T. Emrich, A. Züfle, K. A. Schmid, and A. Zimek. A framework for clustering uncertain data. In *VLDB*, 2015.
[75] W. Shi and B. Golam Kibria. On some confidence intervals for estimating the mean of a skewed population. *International Journal of Mathematical Education in Science and Technology*, 38(3):412–421, 2007.
[76] S. Subramaniam, T. Palpanas, D. Papadopoulos, V. Kalogeraki, and D. Gunopulos. Online outlier detection in sensor data using non-parametric models. In *VLDB*, 2006.
[77] S. K. Tanbeer et al. Sliding window-based frequent pattern mining over data streams. *Information sciences*, 179(22):3843–3865, 2009.
[78] J. S. Vitter. Random sampling with a reservoir. *ACM Transactions on Mathematical Software (TOMS)*, 11(1):37–57, 1985.
[79] X. Wang, X. L. Dong, and A. Meliou. Data x-ray: A diagnostic tool for data errors. In *SIGMOD*, 2015.
[80] A. Woodie. Kafka tops 1 trillion messages per day at LinkedIn. *Datanami*, September 2015. `http://www.datanami.com/2015/09/02/kafka-tops-1-trillion-messages-per-day-at-linkedin/`.
[81] E. Wu and S. Madden. Scorpion: Explaining away outliers in aggregate queries. In *VLDB*, 2013.
[82] D. Y. Yoon, N. Niu, and B. Mozafari. DBSherlock: A performance diagnostic tool for transactional databases. In *SIGMOD*, 2016.
[83] Z. Zheng, Y. Li, and Z. Lan. Anomaly localization in large-scale clusters. In *ICCC*, 2007.

APPENDIX

A. DETECTION

MCD. Computing the exact MCD requires examining all subsets of points to find the subset whose covariance matrix exhibits the minimum determinant. This is computationally intractable for even modestly-sized datasets. Instead, for the actual computation of the MCD covariance and mean, MacroBase adopts an iterative approximation called FastMCD [69]. In FastMCD, an initial subset of points S_0 is chosen from the input set of points P. FastMCD computes the covariance C_0 and mean μ_0 of S_0, then performs a "C-step" by finding the set S_1 of points in P that have the $|S_1|$ closest Mahalanobis distances (to C_0 and μ_0). FastMCD subsequently repeats C-steps (i.e., computes the covariance C_1 and mean μ_1 of S_1, selects a new subset S_2 of points in P, and repeats) until the change in the determinant of the sample covariance converges (i.e., $det(S_{i-1}) - det(S_i) < \varepsilon$, for small ε). To determine which dimensions are most anomalous in MCD, MacroBase uses the corr-max transformation [34].

Handling variable ADR arrival rates. We consider two policies for collecting samples using an ADR over real-time periods with variable tuple arrival rates. The first policy is to compute a uniform sample per decay period, with decay across periods. This can be achieved by maintaining an ADR for the stream contents from all prior periods and a regular, uniform reservoir sample for the current period. At the end of the period, the period sample can be inserted into the ADR. The second policy is to compute a uniform sample over time, with decay according to time. In this setting, we can choose a sampling period for the stream (e.g., 1 second), and, for each period, insert the average of all points within the sampling period into the ADR.

Contamination plot details. In Figure 3, we configured a synthetic dataset of 10M points with a combination of two distributions: a uniform *inlier* distribution D_i, with radius 50 centered at the origin, and a uniform *outlier* distribution D_o, with radius 50 centered at (1000, 1000). We varied the proportion of points in D_i and D_o to evaluate the effect of outlier contamination on the Z-Score, MAD, and MCD (using univariate points for Z-Score and MAD).

B. EXPLANATION

Confidence. To provide confidence intervals on its output explanations, MDP leverages existing results from the epidemiology literature, applied to the MDP data structures. For a given attribute combination appearing a_o times in the outliers and a_i times in the inliers, with an OI-ratio of o, n_o total outlier points, and n_i total inlier points, we can compute a $1-p\%$ confidence interval as:

$$\exp\left(\ln(o) \pm z_p \sqrt{\frac{1}{a_o} - \frac{1}{n_o} + \frac{1}{a_i} - \frac{1}{n_i}}\right)$$

where z_p is the z-score corresponding to the $1-\frac{p}{2}$ percentile [62]. For example, an attribute combination with OI-ratio of 5 appearing in 1% of the outlier tuples, with 1% outliers in a dataset of 10M points has a 95th percentile confidence interval of $(4.69, 5.33)$ (99th: $(4.60, 5.43)$). Given an OI-ratio threshold is 3, MacroBase can return this explanation with confidence.

However, because MDP performs a repeated set of statistical tests to find attribute combinations with sufficient OI-ratio, MDP subject to the multiple testing problem: large numbers of statistical tests are statistically likely to contain false positives. To address this problem, MDP applies a correction intervals. For example, under the Bonferroni correction [72], if a user seeks a confidence of $1-p$ and MDP tests k attribute combinations, MDP should instead assess the confidence at $1-\frac{p}{k}$. We can compute k at explanation time by recording the number of support computations.

k is likely to be large as, in the limit, MDP may examine the power set of all attribute values in the outliers. However, in our IoT scenario, k is less problematic. First, the pruning power of itemset mining eliminates many tests. Second, on production data, many of MacroBase's explanations have very high OI-ratio—often in the tens or hundreds. This is because many problematic IoT behaviors are highly systemic. Third, and perhaps most interestingly, MacroBase analyzes large streams. In the above example, even with $k = 10M$, the 95th percentile confidence interval is still $(4.14, 6.04)$. Compared to, say, medical studies with study sizes in the range of hundreds to thousands, the large volume of IoT data mitigates many of the problems associated with multiple testing. For example, given only 1000 points, $k = 10M$ yields a 95th percentile confidence interval of $(0, 776M)$, which is effectively meaningless. (This trend also applies to alternative corrective methods such as the Benjamini-Hochberg Procedure [12].) Thus, while the volumes of IoT data streams pose significant computational challenges, they can actually benefit the statistical quality of analytic monitoring results.

C. IMPLEMENTATION

In this section, we describe the MacroBase prototype implementation and runtime. As of June 2016, MacroBase's core comprises approximately 9,105 lines of Java, over 7,000 of which are devoted to operator implementation, along with an additional 1,000 lines of Javascript and HTML for the front-end and 7,600 lines of Java for diagnostics and prototype pipelines.

We chose Java due to its high productivity, support for higher-order functions, and popularity in open source. However, there is considerable performance overhead associated with the Java virtual machine (JVM). Despite interest in bytecode generation from high-level languages such as Scala and .NET [19, 53], we are unaware of any generally-available, production-strength operator generation tools for the JVM. As a result, MacroBase leaves performance on the table in exchange for programmer productivity. To understand

	LS	TS	ES	AS	FS	MS
Throughput (points/sec)	7.86M	8.70M	9.35M	12.31M	7.05M	6.22M
Speedup over Java	7.46×	24.11×	5.24×	16.87×	5.32×	17.54×

Table 3: Speedups of hand-optimized C++ over Java MacroBase prototype for simple queries (queries described in Section 6).

the performance gap, we rewrote a simplified MDP pipeline in hand-optimized C++. As Table 3 shows, we measure an average throughput gap of 12.76× for simple queries. Future advances in JVM code generation should only improve this gap.

MacroBase executes operator pipelines via a single-core dataflow execution engine. MacroBase's streaming dataflow decouples producers and consumers: each operator writes (i.e,. pushes) to an output stream but consumes tuples as they are pushed to the operator by the runtime (i.e., implements a `consume(OrderedList<Point>)` interface). This facilitates a range scheduling policies: operator execution can proceed sequentially, or by passing batches of tuples between operators. MacroBase supports several styles of pipeline construction, including a fluent, chained operator API (i.e., `ingester .then(transformer) .then(classifier)`). By default, MacroBase amortizes calls to `consume` across several thousand points, reducing function call overhead. This API also allows stream multiplexing and is compatible with a variety of existing dataflow execution engines, including Storm, Heron, and Amazon Streams, which could also form the basis of an execution substrate for MacroBase operators in the future. We demonstrate interoperability with several existing data mining frameworks in Appendix D.

The MacroBase prototype does not currently implement fault tolerance, although classic techniques such as state-based checkpointing are applicable here [48], especially as MDP's operators contain modest state. The MacroBase prototype is also oriented towards single-core deployment. For parallelism, users currently run one query per core; for example, in the event of larger deployments, we run multiple MacroBase queries on different partitions of data (e.g., one query pipeline per application cluster in a datacenter). However, we report on preliminary multi-core scale-out results in Appendix D.

The MacroBase prototype and all code evaluated in this paper are available online under a permissive open source license.

D. EXPERIMENTAL RESULTS

Dataset descriptions. CMTD contains user drives at CMT, including anonymized metadata such as phone model, drive length, and battery drain; Telecom contains aggregate internet, SMS, and telephone activity for a Milanese telecom; Accidents contains statistics about United Kingdom road accidents between 2012 and 2014, including road conditions, accident severity, and number of fatalities; Campaign contains all US Presidential campaign expenditures in election years between 2008 and 2016, including contributor name, occupation, and amount; Disburse contains all US House and Senate candidate disbursements in election years from 2010 through 2016, including candidate name, amount, and recipient name; and Liquor contains sales at liquor stores across the state of Iowa.

Several of these datasets, such as Disburse, are not explicitly IoT datasets. However, all but CMT are *i.*) publicly accessible, allowing reproducibility, and *ii.*) representative of many challenges we have encountered in analyzing production data beyond CMT in scale and general trends. While none of these datasets contain ground-truth labels, we have verified several of the explanations from our queries over CMT and plan to make all non-CMT explanation outputs publicly available.

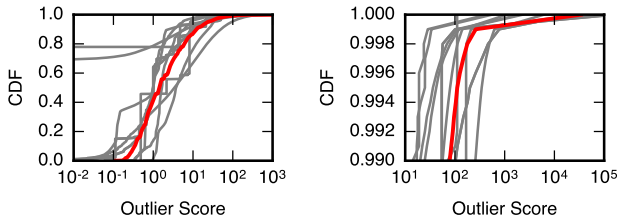


Figure 7: CDF of outlier scores for all datasets, with average in red; the datasets exhibit a long tail with extreme outlier scores at the 99th percentile and higher.

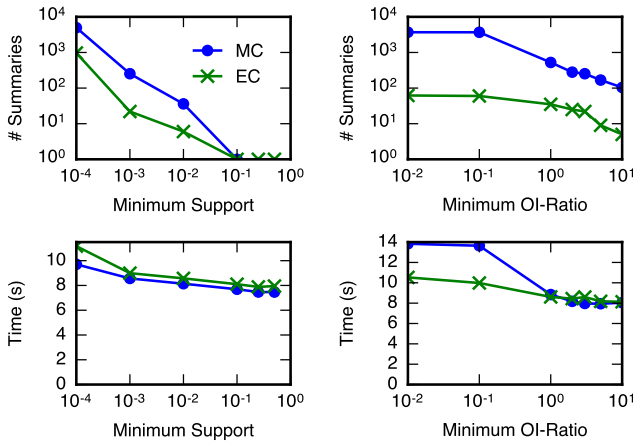


Figure 8: Number of summaries produced and summarization time under varying support (percentage) and OI-Ratio.

Score distribution. We plot the CDF of scores in each of our real-world dataset queries in Figure 7. While many points have high anomaly scores, the tail of the distribution (at the 99th percentile) is extreme: a very small proportion of points have outlier scores over over 150. Thus, by focusing on this small upper percentile, MacroBase highlights the most extreme behaviors.

Varying support and OI-Ratio. To understand the effect of support and OI-ratio on explanation, we varied each and measured the resulting runtime and the number of summaries produced on the EC and MC datasets, which we plot in Figure 8. Each dataset has few attributes with outlier support greater than 10%, but each had over 1700 with support greater than 0.001%. Past 0.01%, support had limited impact on runtime; most time in explanation is spent in simply iterating over the inliers rather than maintaining tree structures. This effect is further visible when varying the OI-Ratio, which has less than 40% impact on runtime yet leads to an order of magnitude change in number of summaries. We find that our default setting of support and OI-Ratio yields a reasonable trade-off between number of summaries and runtime.

Operating on samples. MDP periodically trains models using samples from the input distribution. The statistics literature offers confidence intervals on the MAD [75] and the Mahalanobis distance [68] (e.g., for a sample of size n, the confidence interval of MAD shrinks with $n^{1/2}$), while MCD converges at a rate of $n^{-1/2}$ [15]. To empirically evaluate their performance, we measured the accuracy and efficiency of training models on samples. In Figure 9, we plot the outlier classification F_1-Score versus sample size for the CMT queries. MAD precision and recall are unaffected by sampling, allowing a four order-of-magnitude speedup without loss in accuracy. In contrast, MCD accuracy is more sensitive due to variance in the sample selection: for example, training on a sample

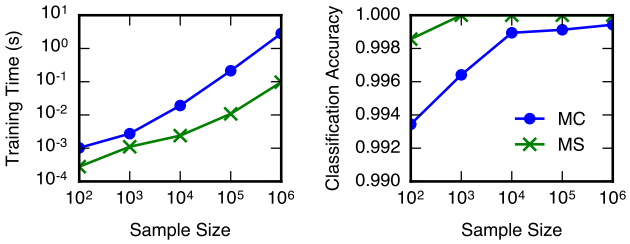


Figure 9: Behavior of MAD (MS) and MCD (MC) on samples.

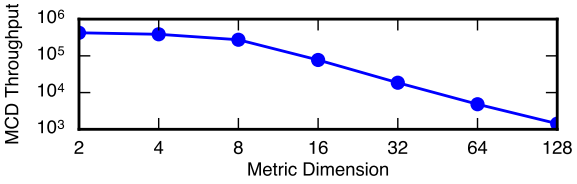


Figure 10: MCD throughput versus metric size.

of 10K points out of 10M yields a speedup of over five orders of magnitude but results in an F_1-Score of 0.48 (std dev: 0.32). A small number of MCD estimates have poor F-Scores. This variance is offset by the fact that models are retrained regularly under streaming execution.

Metric scalability. As Figure 10 demonstrates, MCD train and score throughput (here, over Gaussian data) is linearly affected by data dimensionality, encouraging the use of dimensionality reduction pre-processing techniques for complex data.

M-CPS and CPS behavior. We also investigated the behavior of the M-CPS-tree compared to the generic CPS-tree. The two data structures have different behaviors and semantics: the M-CPS-tree captures only itemsets that are frequent for at least two windows by leveraging an AMC sketch. In contrast, CPS-tree captures all frequent combinations of attributes but must insert each point's attributes into the tree (whether supported or not) and, in the limit, stores (and re-sorts) all items ever observed in the stream. As a result, across all queries except ES and EC, the CPS-tree was on average 130x slower than the M-CPS-tree (std dev: 213x); on ES and EC, the CPS-tree was over 1000x slower. The exact speedup was influenced by the number of distinct attribute values in the dataset: Accidents had few values, incurring 1.3x and 1.7x slowdowns, while Campaign had many incurring substantially greater slowdowns

Preliminary scale-out. As a preliminary assessment of MacroBase's potential for scale-out, we examined its behavior under a naïve, shared-nothing parallel execution strategy. We partitioned the data across a variable number of cores of a server containing four Intel Xeon E7-4830 2.13 GHz CPUs and processed each partition; upon completion, we return the union of each core's explanation. As Figure 11 shows, this strategy delivers excellent linear scalability. However, as each core effectively trains and summarizes a sample of the overall dataset, accuracy suffers due to both model drift (as in Figure 9) and lack of cross-partition cooperation in summarization. For example, with 32 partitions spanning 32 cores, FS achieves throughput nearing 29M points per second, with perfect recall, but only 12% accuracy. Improving accuracy while maintaining scalability is the subject of ongoing work.

Explanation runtime comparison. Following the large number of recent data explanation techniques (Section 2.2), we implemented several additional methods. The results of these methods are not comparable, and prior work has not evaluated these techniques with respect to one another in terms of semantics or performance. We do

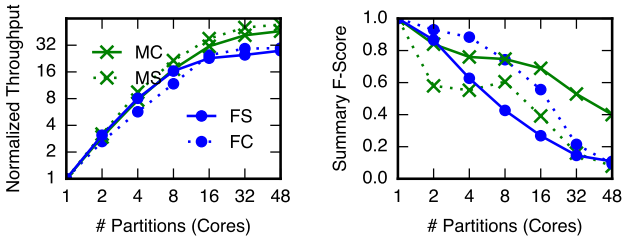


Figure 11: Behavior of naïve, shared-nothing scale-out.

	A1	A2	A3	A4	A5	A6	A7	A8	A9
TPC-C (QS: one MacroBase query per cluster): top-1: 88.8%, top-3: 88.8%									
Train top-1 correct (of 9)	9	9	9	9	9	8	9	9	8
Holdout top-1 correct (of 2)	2	2	2	2	2	2	2	2	0
TPC-C (QE: one MacroBase query per anomaly type): top-1: 83.3%, top-3: 100%									
Train top-1 correct (of 9)	9	9	9	9	9	8	9	9	7
Holdout top-1 correct (of 2)	2	2	2	2	2	1	2	2	0
TPC-E (QS: one MacroBase query per cluster): top-1: 83.3%, top-3: 88.8%									
Train top-1 correct (of 9)	9	9	9	9	9	8	9	9	0
Holdout top-1 correct (of 2)	2	2	2	2	2	1	2	2	0
TPC-E (QE: one MacroBase query per anomaly type): top-1: 94.4%, top-3: 100%									
Train top-1 correct (of 9)	9	9	9	9	9	8	9	9	6
Holdout top-1 correct (of 2)	2	2	2	2	2	1	2	2	2

Table 4: MDP accuracy on DBSherlock workload. A1: workload spike, A2: I/O stress, A3: DB backup, A4: table restore, A5: CPU stress, A6: flush log/table; A7: network congestion; A8: lock contention; A9: poorly written query. "Poor physical design" (from [82]) is excluded as the labeled anomalous regions did not exhibit significant correlations with any metrics.

not attempt a full comparison based on semantics but do perform a comparison based on running time, which we depict in Table 5. We compared to a data cubing strategy suggested by Roy and Suciu [71], which generates counts for all possible combinations (21x slower), Apriori itemset mining [38] (over 43x slower), and Data X-Ray [79]. Cubing works better for data with fewer attributes, while Data X-Ray is optimized for hierarchical data; we have verified with the authors of Data-XRay that, for MacroBase's effectively relational points, Data X-Ray will consider all combinations of categorical attributes unless stopping criteria are met. MacroBase's cardinality-aware explanation completes fastest for all queries.

Compatibility with existing frameworks. We implemented several additional MacroBase operators to validate interoperability with existing data mining packages. We were unable to find a single framework that implemented both unsupervised outlier detection and data summarization and had difficulty locating streaming implementations. Nevertheless, we implemented two MacroBase outlier detection operators using Weka 3.8.0's `KDTree` and Elki 0.7.0's `SmallMemoryKDTree`, an alternative FastMCD operator based on a recent RapidMiner extension (`CMGOSAnomalyDetection`) [36], an

Query	MB	FP	Cube	DT10	DT100	AP	XR
LC	1.01	4.64	DNF	7.21	77.00	DNF	DNF
TC	0.52	1.38	4.99	10.70	100.33	135.36	DNF
EC	0.95	2.82	16.63	16.19	145.75	50.08	DNF
AC	0.22	0.61	1.10	1.22	1.39	9.31	6.28
FC	1.40	3.96	71.82	15.11	126.31	76.54	DNF
MC	1.11	3.23	DNF	11.45	94.76	DNF	DNF

Table 5: Running time of explanation algorithms (s) for each complex query. MB: MacroBase, FP: FPGrowth, Cube: Data cubing; DT*X*: decision tree, maximum depth *X*; AP: A-Aprioi; XR: Data X-Ray. DNF: did not complete in 20 minutes.

alternative MAD operator from the OpenGamma 2.31.0, and an alternative FPGrowth-based summarizer based on SPMF version v.0.99i. As none of these packages allowed streaming operation (e.g., Weka allows adding points to a `KDTree` but does not allow removals, while Elki's `SmallMemoryKDTree` does not allow modification), we implemented batch versions. We do not perform accuracy comparisons here but note that the kNN performance was substantially slower ($>$100x) than MDP's operators (in line with recent findings [36]) and, while SPMF's operators were faster than our generic FPGrowth implementation, SPMF was still 2.8$\times$ slower than MacroBase due to MDP's cardinality-aware optimizations. The primary engineering overheads came from adapting to each framework's data formats; however, with a small number of utility classes, we were able to easily compose operators from different frameworks and also from MacroBase, without modification. Should these frameworks begin to prioritize streaming execution and/or explanation, this interoperability will prove especially fruitful in the future.