

Prioritizing Attention in Fast Data: Principles and Promise

Peter Bailis, Edward Gan, Kexin Rong, Sahaana Suri
Stanford InfoLab

ABSTRACT

While data volumes continue to rise, the capacity of human attention remains limited. As a result, users need analytics engines that can assist in prioritizing attention in this *fast data* that is too large for manual inspection. We present a set of design principles for the design of fast data analytics engines that leverage the relative scarcity of human attention and overabundance of data: return fewer results, prioritize iterative analysis, and filter fast to compute less. We report on our early experiences employing these principles in the design and deployment of MacroBase, an open source analysis engine for prioritizing attention in fast data. By combining streaming operators for feature transformation, classification, and data summarization, MacroBase provides users with interpretable explanations of key behaviors, acting as a search engine for fast data.

1. INTRODUCTION

In an information-rich world, the wealth of information means a dearth of something else: a scarcity of whatever it is that information consumes. What information consumes is rather obvious: it consumes the attention of its recipients. [23]

Today, 45 years after Herbert Simon’s above observation, each of Facebook, Twitter, and LinkedIn ingest more than 12M events per second [4, 22, 26]. Simultaneously, a rise in automated data sources (i.e., the “Internet of Things”) is bringing even larger data volumes that are predicted to double every two years [1]. These volumes place a serious strain on analyst and analytic engine alike. As an example, several of today’s best-of-class application operators anecdotally reported using less than 6% of data they collect.

As a result, while the era of “big data” ushered an abundance of data, we believe that the impending era of *fast data* will be marked by an overabundance of data and a relative scarcity of resources to process and interpret it. That is, as data volumes continue to rise, human attention remains limited. The resulting challenge is to **prioritize attention**: information systems must—more than ever—assist in highlighting and contextualizing important behaviors, quickly and over large, diverse data sources. This problem is especially acute in tasks including data exploration, debugging of complex and predictive services, and high-volume monitoring. To illustrate

this difficulty, consider the following three questions, each of which corresponds to a real scenario reported by domain experts:

- A developer releases her mobile application on today’s Android ecosystem. She asks: is her application behaving reliably on all 24,000 distinct Android device types, and combinations of operating system and application releases?
- An industrial equipment manufacturer deploys tens of thousands of battery subsystems worldwide. Following a catastrophic failure of one battery, the manufacturer asks: are other batteries likely to also fail catastrophically?
- A geologist deploys an array of thousands of sensors, each monitoring the Earth’s activity at 100Hz, for decades. The geologist asks: are there significant seismic events captured via small, coupled interactions observed by the sensors?

At high volumes, analysts report that delivering timely answers to these questions via manual data analysis is infeasible. Moreover, in addition to overwhelming human attention, high data volumes can also overwhelm machine “attention,” or the ability to economically process this data due to computational overheads. As a result, analysts report that important behaviors are frequently overlooked, leading to inefficiency, outages, and unanswered questions.

To bridge this widening gap between limited attention and growing data volume, data-intensive systems must evolve. Just as the rise of big data was both enabled by and influenced a new generation of faster, cheaper, and often more capable analysis engines, the rise of fast data demands yet another re-design. Specifically, in prioritizing the use of scarce human and computational resources, fast data analysis engines must address three key challenges:

A.) Scarce resources for interpretation. In today’s high-volume deployments, data volumes far exceed humans’ capability to manually analyze events in real time. Users will be overwhelmed if a fast data system reports even a handful of raw data points per second. Exacerbating this problem, the combinatorial explosion of data attributes (e.g., application and firmware versions, hardware platforms) can mask important behaviors during manual inspection.

B.) Scarce resources for developing analyses. The statistical and predictive models that are critical in prioritizing attention also require many iterations of feature engineering, model selection, parameter tuning, performance optimization, and deployment. Moreover, accurately identifying important behaviors typically requires domain knowledge from experts. As a result, there is often a discrepancy between the skills of domain experts—who are unlikely to also be experts in machine learning, statistics, or databases—and the skills required to develop accurate, efficient analyses. Bridging this gap via large supporting teams of data scientists and engineers can be prohibitively expensive.

C.) Scarce resources for computation. Fast data exhibits extremely high volume, routinely in the millions of events per second as at the web services above. Processing these volumes is a challenge for any analytics engine, which will increasingly face difficult trade-offs between accuracy, completeness, and speed. These trade-offs are often unexplored in the statistics and machine learning literature, which frequently optimizes for accuracy instead of computational efficiency. Moreover, in many deployments, analytic models must evolve quickly as conditions change, and volumes will grow.

To investigate these challenges, we are developing MacroBase, a new analytics engine designed to prioritize attention in fast data.¹ MacroBase *i.)* produces outputs designed to prioritize attention while *ii.)* allowing iterative development of analyses and *iii.)* aggressively prioritizing computation over input data. That is, MacroBase ingests input from diverse, streaming data sources (e.g., relational databases, video feeds) and outputs high-level results designed for human consumption that contextualize and explain important behaviors (e.g., attributes of data that are disproportionately correlated with degraded or improved performance). For example, in an industrial deployment, MacroBase identified a buggy firmware release as it was deployed to a device fleet; in a mobile application deployment, MacroBase identified a previously unknown problematic interaction between the application and the battery in a relatively rare device model; in a third deployment, MacroBase identified poorly-configured containers within a datacenter. Instead of outputting all data points exhibiting these phenomena, MacroBase provided users with high-level explanations (e.g., “devices running firmware v42 are 73 times more likely to exhibit degraded performance”). To enable this functionality, MacroBase employs a customizable combination of high-performance streaming analytics operators for feature extraction, classification, and explanation [6]. Combined, MacroBase’s operators act as a search engine for fast data.

In this paper, we present design principles that enabled the above analyses and other early use cases, and discuss our initial experiences implementing, improving, and deploying the MacroBase prototype, which continues to evolve today. First, in §2, we outline a set of three design principles for prioritizing attention in fast data analytics: prioritize output presented to users (§2.1), prioritize iteration in developing analyses (§2.2), and prioritize computation on inputs (§2.3). The essence of this style can be captured by the phrase “scarcity mandates prioritization.” Second, in §3, we report on the early architecture, implementation, and use of the MacroBase engine. We describe how MacroBase embodies the above principles by producing interpretable, context-rich explanations and allowing users to iteratively improve efficiency and accuracy. In §4, we outline several research challenges inspired by our early experiences with the MacroBase prototype, which has performed analyses in at least six external industrial organizations and recently received its first external patches from engineers at two Internet services.

2. A FAST DATA MANIFESTO

Fast data analysis systems must prioritize the attention of end users, developers, and machines to highlight behaviors in data that matter. In this section, we introduce three design principles for prioritizing scarce attention. Figure 1 illustrates how *i.)* domain expert attention can be focused by delivering output that that summarizes, aggregates, and contextualizes fast data (§2.1), *ii.)* engineering resources can be intelligently allocated via interfaces that facilitate iterative development (§2.2), and, *iii.)* closest to data sources, limited computational resources can be allocated discriminatively to data that is most relevant to outputs (§2.3).

¹Available at <https://github.com/stanford-futuredata/macrobase/>

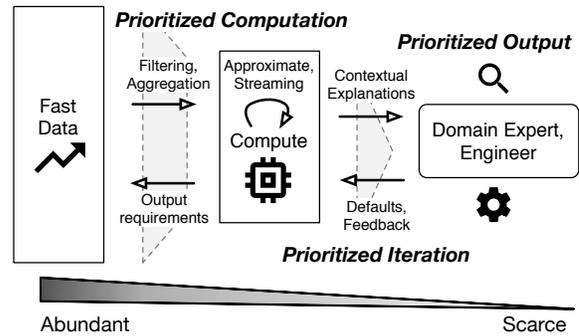


Figure 1: Fast Data System Design. A funnel of increasing scarcity from data to computational resources and humans guides the design of the system. Techniques for prioritizing computation and human attention exploit the resource imbalances at each level.

2.1 Principle: Prioritize Output

Given data streams that routinely contain hundreds of thousands or more events per second, fast data systems cannot afford to produce results that expose raw data; a handful of data points per second will overwhelm human operators. Even automated downstream consumers may be unable to handle raw data streams at scale.

Prioritize Output: *The design must deliver more information using less output.* Each result provided to the user costs attention. A few general results are better than many specific results.

To prioritize attention, fast data systems must be judicious in deciding which and what kind of results to return to end users. In contrast, systems that return a large number of raw records (e.g., relational database scans, file systems, message queues) will squander attention. To borrow from industrial designer Dieter Rams [17], fast data systems should output fewer, but better results. Fast data systems can prioritize attention by producing summaries and aggregates that contextualize and highlight key behaviors within and across records; for example, instead of reporting all 128K problematic records produced by device 1337, the system can simply return the device ID and a count of records. Fast data systems can further prioritize attention by presenting results in order of their importance and relevance to users. Systems can exploit natural hierarchies and ontologies to support aggregation and ranking, allowing users to drill down as needed from coarser to finer detail (e.g., from datacenter to rack, server, container, and process).

2.2 Principle: Prioritize Iteration

Iteration is key to developing high quality analyses. Modern advanced analytics workflows consist of many steps—including feature engineering, model selection, parameter tuning, and performance engineering—that are each inherently iterative and feedback-driven. This process is labor-intensive: today’s headline-grabbing advances in machine learning and predictive data products are delivered by large, well-financed teams of highly-skilled data scientists, statisticians, and DevOps engineers who perform these often tedious tasks. Due to these overheads, many domain experts are unable to translate their expertise into analyses. Fast data systems should both lower the barrier to analysis and accelerate this feedback-driven process by explicitly prioritizing iterative workflows.

Prioritize Iteration: *The design should allow iterative, feedback-driven development.* Give useful defaults, and make it easy to tune analysis pipelines and routines. It is slightly better to be flexible than perfectly precise.

Fast data systems must impose little burden on end users while providing means of easily tuning analyses to improve accuracy and scale. The first model is rarely the final model, but good defaults (or default packages) can provide rapid feedback. Subsequently, users should be empowered to quickly iterate and provide feedback on analyses while receiving timely results. Simultaneously, systems should provide power tools to more skilled end users that allow advanced configuration as needed.

Today’s imperative, ad-hoc “hairball” architectures hamper iterative development. The few systems that manage to solve one problem well are rarely amenable to repurposing in new tasks. Instead, fast data systems should instead be designed for modularity and incremental extensibility. Systems should allow end-users to make best use of their domain knowledge while absorbing as much of the burden of end-to-end model deployment as possible. They should augment the abilities of domain experts (and any supporting programmers and data scientists) by automating error-prone, tedious tasks while scaling best-of-class analyses to increasing volumes.

2.3 Principle: Prioritize Computation

At high volume, efficiently prioritizing limited computational resources is essential. Engines that provide advanced predictive analytics are usually focused on batch-oriented, less time-sensitive offline analyses, but online stream processing systems such as Storm and Spark Streaming leave the design and implementation of most complex analysis routines as an exercise for the end-user. Simultaneously, the machine learning and statistics community have historically optimized for prediction quality. As a result, the trade-offs between quality and speed in a given domain are often unexplored.

Fast data analysis engines should provide techniques for accurate, high-volume stream processing. To do so, they can leverage a key property of fast data: not all inputs contribute equally to the output.

Prioritize Computation: *The design must prioritize computation on inputs that most affect its output. The fastest way to compute is to avoid computation. Avoid computation on inputs that contribute less to output.*

Fast data systems should start from the output and work backwards to the input, doing as little work as needed on each piece of data, prioritizing computation over data that matters most. Gluing together black-box functions that are unaware of the final output will miss critical opportunities for fast compute; if a component is unaware of the final output, it is likely to waste time and resources. By quickly identifying inputs that most contribute to the output, we can aggressively prune the amount of computation required. This prioritizing fast compute presents many opportunities for sampling and pre-aggregation. Incremental algorithms that cache and reuse expensive computations are especially beneficial. To compute quickly, filter fast and compute less.

3. MACROBASE: A FAST DATA SYSTEM

To embody the principles in §2 and begin addressing the challenges of prioritizing attention, we have spent the last twelve months building a new fast data analysis engine called MacroBase. In this section, we report on our initial design decisions, experiences deploying the MacroBase prototype, and ongoing research within the system, which currently serves as an active vehicle for both research and a handful of production analyses. [6] and forthcoming publications provide additional details.

3.1 System Design

Recognizing the many challenges of prioritizing attention in fast data, when designing MacroBase, we decided it was better to begin

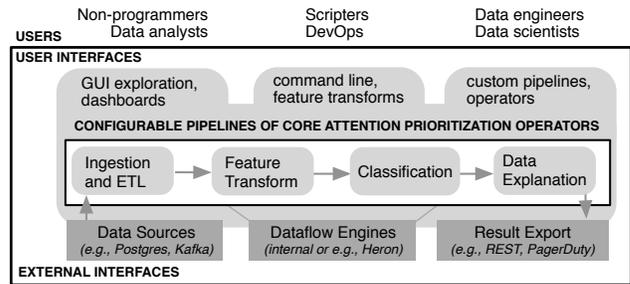


Figure 2: MacroBase System Architecture. MacroBase executes configurable dataflow pipelines of statistical analysis operators designed to prioritize attention in fast data streams.

by doing one thing well—and later do a few things well—than to do many things poorly at the outset. As a result, MacroBase started as a relatively lean open source prototype, targeting a single analytics workflow our collaborators struggled with: detecting and understanding systematic anomalies in their mobile telematics application (e.g., adverse interactions between users’ hardware devices and the application’s trip detector). Subsequently, via engagements in industries, online services, and automotive vehicles, we have begun to expand the set of functionality to include filtering, visualizing, and summarizing behaviors in both streaming and offline data.

Twelve months in, the system has a well-defined architecture and growing set of key operators, which we describe in this section. We believe it provides a useful template for designing search engine-like systems for diverse fast data scenarios.

3.1.1 System Architecture

Modularity. As an analytic engine for prioritizing attention in fast data, MacroBase provides an extensible set of interfaces and modules to aid users in quickly understanding important behaviors in high-volume data streams. It was apparent early on that different data sources and domains required customization to express features that matter most: for example, tabular, relational, time series, and image data could all leverage common outlier detection and explanation operators, but each domain required different pre-processing and feature extraction steps. As a result, one of MacroBase’s major design objectives is to facilitate easy extension and reconfiguration of the provided functionality without having to reason about low-level systems concerns (e.g., robustness, streaming execution).

Concretely, MacroBase provides a set of composable, streaming dataflow operators designed to prioritize attention. These operators perform tasks including feature extraction, supervised and unsupervised classification, explanation and summarization. Unlike a traditional relational database, with its clear set of logical functionality such as operators for selection, projection, and join operations, the core set of logical operators required to prioritize attention has yet to be designed and implemented. As a result, both interface design and efficient operator implementation are core research challenges; we describe progress on each in detail here.

Dataflow. Although MacroBase’s core computational primitives continue to evolve, we have found that expressing them via a dataflow computation model is useful. Many developers of recent specialized analytics engines expended considerable effort devising new techniques for scheduling, partitioning, and scaling their specialized analytics (most notably, in graph processing), only to see most performance gains recouped (or even surpassed) by traditional, dataflow-oriented engines [2, 11, 13]. In MacroBase, we sought to learn from this history, short-circuiting the process by developing specialized dataflow operators from (almost) the start; our early

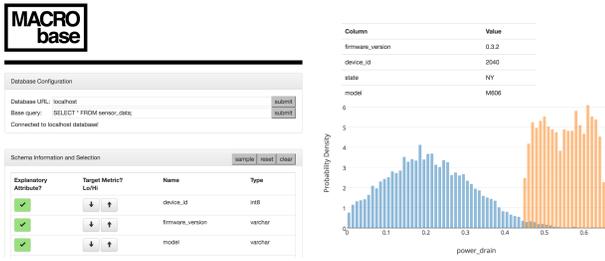


Figure 3: MacroBase’s Default Input and Output. By default, MacroBase users select key performance metrics and attributes in their input data; MacroBase subsequently reports combinations of attributes that are correlated with abnormal metric readings. In this example, MacroBase shows device ID 2030 is disproportionately correlated with high power drain readings; MacroBase provides a list of such explanations, ranked by their *risk ratio*.

non-dataflow, imperatively-specified prototypes were brittle to adapt in a rapidly-expanding set of early use cases.

The use of dataflow confers several benefits. MacroBase’s operators can be interleaved with traditional (e.g., relational) analytics operators, or re-used across task-specific dataflow graphs (i.e., pipelines). Decoupling dataflow specification from execution (i.e., operator placement and scheduling) also allows us to execute operator pipelines in a number of execution modes.

MacroBase’s core dataflow pipelines currently contain a sequence of data ingestion, feature extraction, classification, and explanation operators (Figure 2). As we describe below and in [6], within this pipeline, individual operators for each task—as well as the entire pipeline structure—is reconfigurable, and the system contains a growing library of alternative operators to draw from.

Execution Modes. MacroBase operates in two major modes. First, since all operators are expressed using a streaming dataflow interface, MacroBase can continuously process input data. Second, MacroBase can execute pipelines in a “one-pass” batch execution, typically reserved for historical and exploratory analysis. The dataflow architecture shown in Figure 2 makes it easy to re-use the same operators in each mode. This allowing users with varying skill levels to obtain results by clicking on simple graphical UIs (Figure 3), configuring pipelines in YAML markdown or Java, or encoding new statistical operators for use in custom pipelines.

Non-goals: low-level dataflow, data storage. In designing MacroBase, we sought to capitalize on two trends. First, streaming distributed dataflow engines (e.g., Storm, Spark Streaming, Heron) have become commoditized. It seemed ill-advised to reimplement this basic functionality (e.g., scheduling, distribution, fault tolerance). Instead, we focused on the task of implementing end-to-end streaming analyses: what computations should a streaming dataflow engine actually perform? Thus, MacroBase uses dataflow but our core efforts are not centered on the low-level details of dataflow execution. Instead, our focus is on accurate and efficient core computational operators that prioritize user attention. Second, large-scale storage systems are also increasingly commoditized (e.g., HDFS, Kafka). In response, we chose to delegate almost all aspects of data storage to external systems.

As a result, MacroBase’s design is focused on computation, bridging the modularity and extensibility of tuple-at-a-time dataflow engines and the functionality of domain-specific analysis tools (e.g., financial fraud and network intrusion detection systems).

3.1.2 Prioritizing Output

MacroBase’s design for prioritizing output was informed by challenges that even highly capable engineers reported in making use of

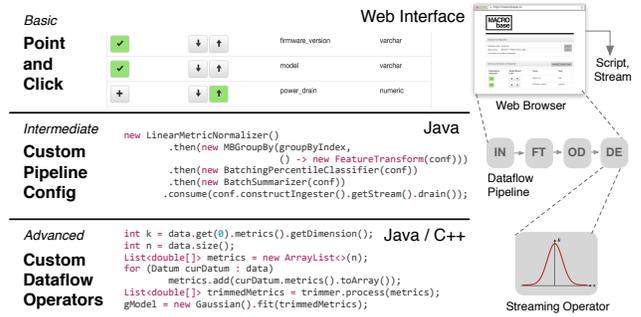


Figure 4: MacroBase Interfaces. MacroBase provides a range of interfaces for users of varying skill to perform analyses. A web-based point-and-click UI allows easy data exploration and export to command-line scripting and/or streaming. MacroBase also provides interfaces for users to configure custom pipelines of analytic operators (e.g., using feature transforms, or combining with relational operators including group-by) and for expert users to author their own streaming operators.

fast data volumes generated by their production applications. For example, similar to the difficulties we encountered in mobile application deployments, engineers at a major online service reported difficulty in determining if exceptions reported from end-user devices are correlated with hardware make and model, application version, and/or users’ physical locations. Today, the engineers can spend hours to days manually grouping the exceptions to identify significant commonalities. In response, we designed MacroBase’s outputs to automate this tedious process, enabling users to focus on a few trends and attributes that are disproportionately correlated with exceptions, rather than on thousands of error messages.

In MacroBase’s default analysis routine [6], users highlight fields of interest within each input data point (e.g., *power_drain*, *user_id*) as either a key performance *metric* (e.g., *power_drain* should not be too high) or as an explanatory *attribute* (e.g., *user_id*). MacroBase then outputs a set of explanations regarding correlated behaviors within the selected metrics: if a user highlights *power_drain* as a target metric and *device_type* as a target attribute, MacroBase might report that devices of type D104 are 12.4 times more likely to have abnormally high power drain than the overall population. MacroBase delivers a ranked list of these explanations according to their corresponding prevalence and degree of severity.

3.1.3 Prioritizing Iteration

Several user profiles have emerged during our initial MacroBase deployments, ranging from domain analysts with limited programming experience to seasoned infrastructure engineers capable of linking against the core MacroBase operators in their own dataflow engines. To accommodate such diverse user backgrounds, MacroBase provides a range of interfaces, from a point-and-click interface to a programmable pipeline interface to interfaces for authoring custom operators (Figure 4). These interfaces enable users of varying skill levels to quickly obtain initial results and further improve result quality by iteratively refining their analyses.

Easy: Point-and-Click. The simplest MacroBase interface is a web-based graphical user interface. The user inputs a database server and base table (e.g., `SELECT * FROM sensor_data;`) from which MacroBase ingests data. MacroBase then presents the user with the data schema, from which he can mark columns of interest as either a key performance *metric* or as an explanatory *attribute*. As above, MacroBase subsequently produces results in the form of explanations, or combinations of attributes that are disproportionately correlated with abnormal metrics. The user can see an overview

of the distributions corresponding to each explanation as well as the overall data and/or drill down into the raw records. Moreover, he can save the configured query and run it programmatically as a one-shot batch job or as a streaming query.

Intermediate: Custom Pipelines. While the GUI provides easy out of the box experience for many users, it is currently limited to a relatively simple analysis pipeline. Users may wish to customize their analyses: for example, instead of finding the highest latencies in a datacenter, a system operator wanted to find high latencies on a per-task basis. Accordingly, users can author custom pipelines, typically modifications of the default analysis. In the monitoring example above, MacroBase performed a per-task analysis by combining the default classification operator with a group-by operator.

This pipeline interface is especially useful in incorporating domain expertise and prior knowledge. For example, a dataset capturing electrical usage exhibited regular spikes, corresponding to refrigerator cooling; rather than classifying each spike as anomalous, we sought to identify instances when the duration of the spike lengthened, indicating increased activity. Compared to analyzing individual points as in the point-and-click GUI, this scenario required the system to operate on a different data type (time series) and perform an additional pre-processing step (a short-time Fourier transform) to extract the waveform. For many similar scenarios, simply adding an extra feature transformation to the pipeline—via temporal restructuring, domain-specific processing functions, or manually combining metrics—can enable a wider range of analyses without modifying the remainder of the pipeline. MacroBase allows users to combine existing operators and feature transformations via this intermediate pipeline interface (currently expressed in Java).

Advanced: Custom Operators. In addition to composing custom pipelines, users should also be allowed to iteratively tune models. Users may wish to expressing high-level rules based on prior information, domain expertise, or labeled data: for example, an automotive engineer may wish to be alerted if a particular class of vehicles has nitrogen-oxide emissions in excess of regulatory guidelines. The system should allow users to encode this knowledge. Moreover, users with expertise in statistics and machine learning may wish to perform their own low-level streaming model implementation and hyperparameter tuning.

For these tasks, MacroBase provides another set of interfaces for implementing new analytics operators, with well-defined static typing for common kinds of operators (feature transformation, classification, explanation) to enable interoperability [6]. As MacroBase committers, we use these advanced interfaces to develop and expose new functionality (e.g., §3.2), which can be used and composed in custom operator pipelines.

3.1.4 Prioritized Computation

To prioritize attention, MacroBase draws upon an array of existing techniques in statistics and machine learning. However, we have regularly found that the most accurate techniques for each task were far too slow for our needs. For example, initial prototypes for an explanation operator revealed that combining standard outlier detection with out-of-the-box itemset mining and Lasso-based techniques yielded valuable results but required painful delays on even moderately-sized (i.e., 100k points) datasets.

Existing machine learning and data mining algorithms are already highly optimized to scale with growing data volume in analytics. However, while implementing MacroBase we found that the imbalance between plentiful data and the small handful of desired human results provided multiple opportunities for improving existing algorithms. Applying classic systems techniques—including predicate

pushdown, incremental memorization, partial materialization, cardinality estimation, approximate query processing, and shared scans—has accelerated common operations in MacroBase. Per the principle of prioritizing computation, by focusing on the end results of the computation, we can avoid a great deal of unnecessary work. Two simple examples from [6] illustrate this potential:

- A naïve strategy for summarizing results of an outlier detector via the risk ratio is to compute correlations in the inlying and outlying points independently, then compare them. However, we can instead appeal to one of the oldest techniques in database query optimization: cardinality estimation. The number of outlying data points is usually much smaller than the number of inlying data points; therefore, if we first find correlations with the smaller set of outlying data points, we can aggressively prune computation when looking for correlations in the inlying data points.
- One of MacroBase’s explanation operators is based on a heavy hitters “top K” sketch. Typically, heavy hitters sketches assume worst-case arrival of items in a stream; again, we can appeal to the structure of the data stream to improve running time. Because many fast data sources repeatedly generate data points, each item in the stream is likely to appear more than once. Therefore, MacroBase uses a new heavy-hitters sketch that amortizes the overhead of maintenance across multiple points. This leads to speedups of over an order of magnitude over existing sketches, which use less memory but exhibit slower update speed.

Stacked end-to-end, the several order-of-magnitude speedups we have achieved via the above techniques enable analysis over larger data sets. For a modestly-sized dataset such as the US campaign contributions from the last four presidential elections, this can mean the difference between response times of seconds instead of hours.

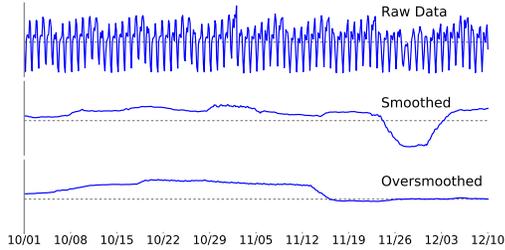
3.2 Ongoing and Future Work

MacroBase’s existing operators and pipelines have proven useful in several deployments. However, there are many promising and high-value opportunities for continued progress at several levels of the stack. Diverse data modalities (e.g., time series, video) merit different forms of output and corresponding computational operators. Efficient, accurate query specification remains challenging: new declarative interfaces and use of techniques including weak supervision and model ensembles can assist. Scale is a perennial concern, and a focus on end-to-end pipelines offers many opportunities for reuse of classic systems techniques including memoization, sketching, and approximate incremental computation. In this section, we illustrate this potential by reporting on ongoing improvements in prioritizing output, iteration, and computation.

3.2.1 Improving Output

Different data modalities are best understood using different forms of output. MacroBase’s default risk-ratio explanations work well for tabular numeric data but are less useful for time-series analyses where trends and waveforms are key to interpretability. For example, in modern application monitoring, time-series visualization via charting is increasingly important for tracking application behavior. However, raw data can be noisy and obscure significant trends in the overall system. For example, the top plot below illustrates an hourly moving average of transportation volume. The regular fluctuations obscure a much larger drop in volume during the week of Thanksgiving (11/27), which is only visible after the series is smoothed according to a weekly average (center). Identifying the best choice of window size that highlights this activity is non-

trivial; too large of a window obscures the trend entirely (bottom). In MacroBase, we are developing new techniques to automate this hyperparameter search for streaming time series data visualization.

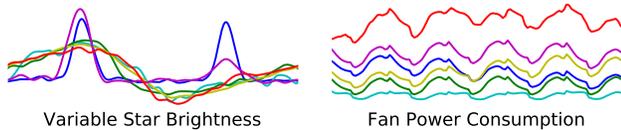


We believe that alternative, multi-modal techniques for displaying results will be complementary. For example, coupling the above temporal output with the risk ratio metric should enable the automated curation of an ideal set of dashboards for tracking important time-varying metrics that evolve as conditions change. Further developing prioritized output for high-dimensional, categorical, and video data is especially promising

3.2.2 Improving Iteration

MacroBase currently allows domain experts to quickly examine different sets of metrics and explanation attributes, and domain engineers to prototype new transforms and data pipelines. However, we are interested in further simplifying the process of using the system, especially for non-expert users.

For example, time series and multimodal input streams (e.g., video and sensors) are especially difficult to analyze when they are high dimensional, but similarity search and outlier classification remain key operations on these data streams. Dimensionality reduction provides a powerful tool for reducing data complexity [14], but many methods (e.g., PCA) are expensive, while others (e.g., Locality Sensitive Hashing) require time-consuming hyperparameter tuning, and the best tool for the job may vary from dataset to dataset. We have developed a new optimizer that evaluates a range of dimensionality reduction techniques and selects the best one that preserves a specified error bound. This is feasible, in part, due to the highly structured nature of many fast data streams (e.g., EEG data, power usage), such as those below:



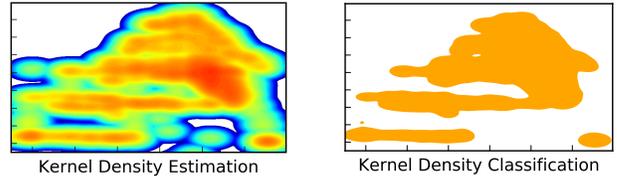
The result of this structure is two-fold. First, it allows dimensionality reductions far in excess of what pessimistic theoretical bounds otherwise suggest. Second, few data points are required to characterize most of the dataset. This permits aggressive sampling, allowing efficient fitting while preserving confidence on result quality. Automating much this process via an intelligent optimizer makes these techniques accessible to end users.

In the near term, we believe it is most beneficial to continue automating the many tasks that analysts report to be tedious. For example, we can leverage techniques like distant supervision [20] and active learning [9] to enable faster specification and validation of models. Optimizing for end-user behavior can further improve performance: for example, when developing a semi-automated method for selecting visualizations, we were able to prune the search space by eliminating visually indistinguishable alternatives. Broadly, we are excited by the potential of even incremental progress towards lowering the barrier to accessibility of (and cost of developing) complex, high-fidelity analytics.

3.2.3 Improving Scalability

Performance remains a core focus of our work. The key opportunity here is to marry systems-oriented performance optimization and the statistics and machine learning literature.

For many analyses, a small fraction of data is responsible for the majority of relevant results. Following the principle of prioritizing computation, once the engine has characterized “normal” behavior, as little computation as possible should be spent on non-anomalous data. For example, when using Kernel Density Estimation (KDE) [25]—a highly accurate but expensive statistical estimation technique—to model complex distributions and identify outlying data points, MacroBase avoids evaluating the actual densities: many analyses simply require identifying whether points lie in “sparse” and “dense” regions. The figures below illustrate the difference between this expensive, classic KDE (left) and binary KDE-based outlier classification (right) on a sensor dataset:



As most points are located in dense regions, we avoid expensive kernel evaluations by building a spatial index over the space and applying pruning rules to stop computing densities when they are known to be above or below the target threshold. This “predicate pushdown” allows empirical speedups of three to four orders of magnitude while still preserving KDE’s accuracy.

Determining which models are best-suited to operate at high volume (and retrain with low latency) and how to dynamically adapt model hyperparameters to changes in data streams remain open questions. The literature contains thousands of models with varying accuracy-speed trade-offs. For example, LSTM networks [15] are adept at automatically featurizing time-series data but are considerably slower than linear models or rules. Harnessing the predictive power of these models without compromising on scale is non-trivial. A key commonality among time-varying models is that few models—from convolutional networks to LSTMs—leverage temporal locality at test time. That is, most models simply evaluate each data point (e.g., video frame) without taking into account the fact that previous data points are similar (and are therefore computationally similar). In these regimes, memoizing and computing incremental approximation results has delivered exciting preliminary results.

4. EARLY EXPERIENCE

Several early use cases shaped MacroBase’s design. In this section, we describe several lessons learned from external users and analyses in a handful of domains.

Onboarding and Architectural Layering. MacroBase usage has largely mirrored its architectural layering: most users interact with the web UI, several use custom pipelines, and few write custom operators. That is, today, most users of the system begin with MacroBase’s web UI for preliminary data exploration. Again, the UI performs one analysis reliably with limited configuration, providing some functionality with limited effort. Subsequently, users typically reach out to our team or post GitHub issues to request additional functionality. For a majority of these requests, our team has responded by authoring custom pipelines, which now requires from ten minutes (for a simple group-by) to several hours (for a more complex explanation, or custom data import). For a handful of remaining requests, we (or our users) developed custom operators to provide

missing functionality. Looking forward, we hope to lower the bar to usability of complex functionality, especially around authoring custom pipelines, so they become mostly self-service like the web UI. A declarative or graphical representation appears promising

Domain Expert Usage. Bridging the gap between domain expertise and statistical machine learning skills has been a prominent theme in our user interactions. Several MacroBase users are world experts in their application domains, but lack the experience, time, or resources to take advantage of complex statistical analysis. For example, hardware engineers from a major industrial manufacturer wished to identify interactions between firmware releases, battery behavior, and equipment metrics. Though domain experts, they possessed limited experience dealing with large amounts of data. They required a tool capable of ingesting data from their data warehouse and exporting results to spreadsheets for further analysis with minimal configuration. These experts reported that MacroBase’s web UI highlighted a problematic firmware release as it was deployed to their device fleet. Here, MacroBase’s initial emphasis on doing one thing well—attribute correlation over unsupervised classification of abnormal behavior—was a good fit for expert needs. We hope to make subsequent core analyses as easy to use.

Systems Expert Usage. We have also found that MacroBase’s operators are useful for expert systems engineers who are less familiar with high-volume, streaming classification and explanation. For instance, every major online service has a data engineering team for collecting large numbers of metrics. However, analyzing this data at scale, with limited resources, remains a serious problem.

As an example, infrastructure engineers at a large Internet company wished to identify transient failures and slowdowns across a hierarchy of machines and processes. Having excellent knowledge of systems engineering, these engineers were able to build a monitoring tool that used hand-tuned rules and thresholds to identify abnormalities. However, their system relied heavily on past experience from application developers utilizing the data pipeline—the system had to know what it was looking for to identify it. MacroBase’s unsupervised classification and statistical explanation operators were a useful complement to their static rules and thresholds, identifying issues that were below the error threshold but deserved attention.

Beyond Errors. Prioritizing attention is not just about finding critical errors. As the above discussion highlights, many deployments already have a set of static thresholds to generate alerts and warnings. We believe opportunity also lies in detecting both degraded and improved behavior, especially in behaviors that are highly correlated with a possibly small subset of the overall population. For example, application developers at a mobile application startup wished to find abnormal, platform-specific app behaviors. Despite having thorough application logs, these issues were difficult to detect proactively at scale—the low overall error rate obscured the fact that small subsets of customers can experience high error rates. Instead, developers often relied on end-users to identify such problematic behavior. MacroBase’s use of the risk ratio to identify subgroups where anomalies are most likely to occur helped discover previously unknown application behaviors. In these cases, MacroBase’s output acted less as an alert and more as a report (i.e., not suitable for paging on-call staff at 3AM but likely worthy of a 10AM report). We are excited about the potential to prioritize attention in similar proactive scenarios.

Looking forward, we believe current practice in high-volume error explanation and diagnosis is especially ripe for improvement. Many application operators report that large data volumes (often collected from a diverse internal services) rule out many computationally-intensive and labor-intensive methods. However, from a statistical

perspective, this scale is an advantage: with many points, we can test more hypotheses (e.g., attribute risk-ratio calculations) than would otherwise be statistically valid [6]. Thus, utilizing metrics like the risk ratio that are intuitive and efficient to compute is promising.

5. RELATED WORK

MacroBase draws inspiration from conventional dataflow engines, statistical machine learning, and human-computer interaction.

Existing analysis techniques and systems. The literature contains a rich set of outlier detection and data explanation algorithms to draw upon in prioritizing attention [16, 18]. However, in applying many of these existing methods, we have found that the scale of fast data requires non-trivial adaptations, especially in the streaming setting. In particular, a prominent focus on accuracy—as opposed to trade-offs between speed and accuracy—leaves many relevant portions of the algorithmic design space unexplored. Several existing libraries for machine learning—including scikit-learn and MLlib [19]—provide useful collections of analysis operators (which we have ourselves utilized in developing early prototype analyses) but are generally not designed to produce interpretable output by default and seldom support streaming execution. As we have discussed, prioritizing output in high-volume streams offers new algorithmic opportunities that can improve scalability.

Perhaps most importantly, there is effectively no system that integrates core operators for prioritizing attention in a single framework (and certainly no streaming system that does so at scale). As a result, it is unclear how to compose operators and how to incorporate them into modular, scalable infrastructure. There are many domain-specific examples of success (e.g., in financial activity monitoring, network intrusion detection) [7], but there is little work on how the many potentially useful statistical and machine learning techniques behind each can be reused via extensible interfaces or combined and optimized in an end-to-end solution. It is as if we had thousands of papers on cost estimation, storage access methods, and concurrency control protocols and somehow expected the lessons and architectures developed in projects such as System R [5], Ingres [24], and Gamma [12] to magically materialize. The result is a wonderful opportunity for systems research.

State of practice. The number of scalable dataflow frameworks is likely at an all-time high [21]. However, the actual implementation of scalable streaming complex analysis operators is typically left as an exercise for the user. As a result, practitioners often need to author their own operators for prioritizing attention—a time-consuming, costly process. Most applications for prioritizing attention we have encountered at scale (primarily in monitoring) rely on simple, static rules that scale well but fail to account for all but the most severe behaviors. Compared with these dataflow engines, our concerns are higher in the stack: determining effective semantics, composition, and implementation of core operators that deliver richer, more accurate results than the static thresholds prevalent in industry. Combining this new class of computational operators for prioritizing attention with existing relational stream processing concepts (e.g., [3, 8]) is especially promising; optimized operators for streaming complex analytics should execute alongside operators for streaming relational algebra. [6] provides more detail on MacroBase’s current default set of operators; this paper describes the design process, principles, and recurring challenges that have arisen thus far.

Data Mining. While MacroBase’s goals are conceptually similar to those of data mining, we approach the problem from three historically separate traditions. First, from an algorithmic perspective, we are most interested in leveraging (and scaling up) recent advances in

Design Principle	Impact on System Design
Prioritize Output (§2.1): <i>Deliver more information using less output.</i>	Produce summaries and explanations instead of raw results for interpretability
	Rank results by relevance and severity to users
	Highlight unusual behaviors captured by unsupervised and semi-supervised learning
Prioritize Iteration (§2.2): <i>Allow iterative feedback-driven development.</i>	Provide reasonable defaults to jumpstart basic usage
	Expose a composable set of core operators for easy customization
	Make use of user feedback to tune all levels of the pipeline
Prioritize Computation (§2.3): <i>Focus on operations that most affect output.</i>	Aggressively prune computation via filtering, sampling, and branch-and-bound
	Trade off accuracy and completeness for performance where it has low impact
	Accelerate streaming operations via sketching and incremental data structures

Table 1: Impact of design principles on MacroBase’s system design choices described in this paper.

statistical machine learning while preserving their frequently rigorous quality guarantees. Second, from a systems perspective, we are interested in bringing more modular, functional system design to a traditionally less architecturally-inclined set of fields (i.e., machine learning and statistics) by building the next layer of functionality above commodity dataflow engines. Third, we view MacroBase as a concrete instantiation of a larger trend towards systems that not only train models but also enable end-to-end data product development and model deployment [10]; solely addressing model training ignores both the end-to-end needs of users and opportunities for optimization throughout the analysis pipeline. We believe the study of systems that enable modular, usable, efficient, and statistically-motivated analyses is especially promising and profitable given growing data volumes and analyst requirements.

6. CONCLUSIONS

The rise of fast data demands the development of data infrastructure to prioritize attention. In this paper, we outlined three design principles for prioritizing attention in fast data—prioritize output presented to users, prioritize iteration in developing analyses, and prioritize computation on inputs—that arose from and are embodied in a new analytics engine we are developing, called MacroBase (Table 1). In combining streaming feature extraction, classification, and explanation operators, MacroBase acts as a search engine for fast data. As we have reported, the initial reaction to the MacroBase prototype has been encouraging, with at least six independent successful deployments (to varying degrees of production-readiness) of the engine external to our lab (in addition to a number of internal trials) over a twelve-month period from project inception to present day. The principles, goals, and directions outlined in this short paper are a reflection of our experiences thus far.

There are many unanswered questions in prioritizing attention in fast data, a handful of which we have outlined. We intend to answer many over the next several years, and we hope others do the same. The opportunity for systems-oriented research to deliver reusable, composable, and efficient streaming complex analytics operators—beyond relational algebra—is substantial and is of increasing practical consequence. The early traction and the problems we have encountered have confirmed our belief in the importance of this problem domain. We cannot turn away our attention.

Acknowledgments

We thank the many members of the Stanford InfoLab, our collaborators at MIT and Waterloo, and the early adopters of the MacroBase prototype for providing feedback on and inspiration for this work. This research was supported in part by Toyota Research Institute, Intel, RWE AG, Visa, Keysight Technologies, Facebook, and VMWare and by the NSF Graduate Research Fellowship under

grants DGE-114747 and DGE-1656518. As MacroBase is open source and publicly available, there is no correspondence—either direct or implied—between the use cases described in this work and the above institutions that supported this research.

7. REFERENCES

- [1] Dell EMC Digital Universe Survey: The Digital Universe of Opportunities: Rich Data and the Increasing Value of the Internet of Things, 2014. <http://www.emc.com/leadership/digital-universe/>.
- [2] M. Abadi et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. In *OSDI*, 2016.
- [3] A. Arasu, S. Babu, and J. Widom. The CQL continuous query language: semantic foundations and query execution. *VLDBJ*, 15(2):121–142, 2006.
- [4] A. Asta. Observability at Twitter: technical overview, part I, 2016. <https://blog.twitter.com/2016/observability-at-twitter-technical-overview-part-i>.
- [5] M. M. Astrahan et al. System r: relational approach to database management. *TODS*, 1(2):97–137, 1976.
- [6] P. Bailis, E. Gan, S. Madden, D. Narayanan, K. Rong, and S. Suri. MacroBase: Prioritizing Attention in Fast Data. In *SIGMOD*, 2017.
- [7] V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):15, 2009.
- [8] S. Chandrasekaran et al. TelegraphCQ: Continuous dataflow processing for an uncertain world. In *CIDR*, 2003.
- [9] D. A. Cohn, Z. Ghahramani, and M. I. Jordan. Active learning with statistical models. *JAIR*, 1996.
- [10] D. Crankshaw et al. The missing piece in complex analytics: Low latency, scalable model management and serving with Velox. In *CIDR*, 2015.
- [11] A. Dave et al. GraphFrames: An integrated API for mixing graph and relational queries. In *GRADES*, 2016.
- [12] D. J. DeWitt et al. The Gamma database machine project. *TKDE*, 2(1):44–62, 1990.
- [13] X. Feng, A. Kumar, B. Recht, and C. Ré. Towards a unified architecture for in-RDBMS analytics. In *SIGMOD*, 2012.
- [14] I. K. Fodor. A survey of dimension reduction techniques, 2002. Technical Report UCRL-ID-148494, Lawrence Livermore National Laboratory.
- [15] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.
- [16] V. J. Hodge and J. Austin. A survey of outlier detection methodologies. *Artificial Intelligence Review*, 22(2):85–126, 2004.
- [17] S. Lovell, J. Ive, and K. Kemp. *Dieter Rams: As Little Design as Possible*. Phaidon Press, 2011.
- [18] A. Meliou, S. Roy, and D. Suciu. Causality and explanations in databases. In *VLDB*, 2014.
- [19] X. Meng et al. MLlib: Machine learning in Apache Spark. *JMLR*, 17(34), 2016.
- [20] M. Mintz et al. Distant supervision for relation extraction without labeled data. In *ACL*, 2009.
- [21] J. MSV. All the Apache streaming projects: An exploratory guide, 2016. <http://thenewstack.io/apache-streaming-projects-exploratory-guide/>.
- [22] T. Pelkonen et al. Gorilla: A fast, scalable, in-memory time series database. In *VLDB*, 2015.
- [23] H. A. Simon. Designing organizations for an information rich world. In *Computers, communications, and the public interest*, pages 37–72. 1971.
- [24] M. Stonebraker, G. Held, E. Wong, and P. Kreps. The design and implementation of ingres. *TODS*, 1(3):189–222, 1976.
- [25] M. Wand and M. Jones. *Kernel Smoothing*. Chapman & Hall/CRC Monographs on Statistics & Applied Probability. Taylor & Francis, 1994.
- [26] A. Woodie. Kafka tops 1 trillion messages per day at LinkedIn. *Datanami*, September 2015. <http://www.datanami.com/2015/09/02/>.