ASAP: Prioritizing Attention via Time Series Smoothing

Kexin Rong, Peter Bailis
Stanford InfoLab
{krong, pbailis}@cs.stanford.edu

ABSTRACT

Time series visualization of streaming telemetry (i.e., charting of key metrics such as server load over time) is increasingly prevalent in modern data platforms and applications. However, many existing systems simply plot the raw data streams as they arrive, often obscuring large-scale trends due to small-scale noise. We propose an alternative: to better prioritize end users' attention, smooth time series visualizations as much as possible to remove noise, while retaining large-scale structure to highlight significant deviations. We develop a new analytics operator called ASAP that automatically smooths streaming time series by adaptively optimizing the trade-off between noise reduction (i.e., variance) and trend retention (i.e., kurtosis). We introduce metrics to quantitatively assess the quality of smoothed plots and provide an efficient search strategy for optimizing these metrics that combines techniques from stream processing, user interface design, and signal processing via autocorrelation-based pruning, pixel-aware preaggregation, and on-demand refresh. We demonstrate that ASAP can improve users' accuracy in identifying long-term deviations in time series by up to 38.4% while reducing response times by up to 44.3%. Moreover, ASAP delivers these results several orders of magnitude faster than alternative search strategies.

1. INTRODUCTION

Data volumes continue to rise, fueled in large part by an increasing number of automated sources, including sensors, processes, and devices. For example, each of LinkedIn, Twitter, and Facebook reports that their production infrastructure generates over 12M events per second [16, 51, 68]. As a result, the past several years have seen an explosion in the development of platforms for managing, storing, and querying large-scale data streams of time-stamped data—i.e., time series—from on-premises databases including InfluxDB [6], Ganglia [3], Graphite [5], OpenTSDB [9], Prometheus [10], and Facebook Gorilla [51], to cloud services including DataDog [2], New Relic [8], AWS CloudWatch [1], Google Stackdriver [4], and Microsoft Azure Monitor [7]. These time series engines provide application authors, site operators, and "DevOps" engineers a means of performing monitoring, health checks, alerting, and analysis of unusual events such as failures [19, 32].

This work is licensed under the Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-nd/4.0/. For any use beyond those covered by this license, obtain permission by emailing info@vldb.org.
Proceedings of the VLDB Endowment, Vol. 10, No. 11
Copyright 2017 VLDB Endowment 2150-8097/17/07.

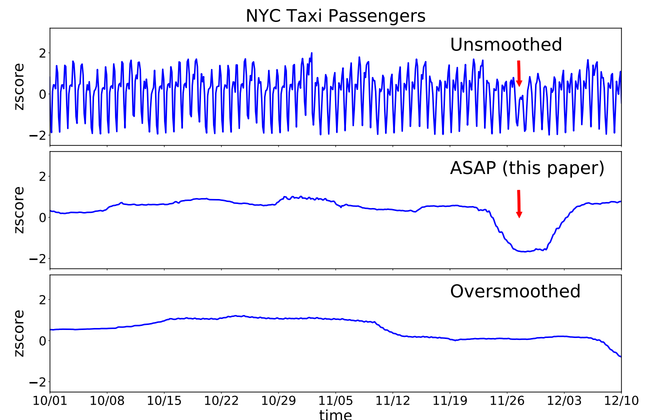

Figure 1: Normalized number of NYC taxi passengers over 10 weeks.[1] From top to bottom, the three plots show the hourly average (unsmoothed), the weekly average (smoothed) and the monthly average (oversmoothed) of the same time series. The arrows point to the week of Thanksgiving (11/27), when the number of passengers dips. This phenomenon is most prominent in the smoothed plot produced by ASAP, the subject of this paper.

While these engines have automated and optimized common tasks in the storage and processing of time series, effective visualization of time series remains a challenge. Specifically, in conversations with engineers using time series data and databases in cloud services, social networking, industrial manufacturing, electrical utilities, and mobile applications, we learned that many production time series visualizations (i.e., "dashboards") simply display raw data streams as they arrive. Engineers reported this display of raw data can be a poor match for production scenarios involving data exploration and debugging. That is, as data arrives in increasing volumes, even small-scale fluctuations in data values can obscure overall trends and behavior. For example, an electrical utility employs two staff to perform 24-hour monitoring of generators. It is critical that these staff quickly identify any systematic shifts of generator metrics in their monitoring dashboards, even those that are "sub-threshold" with respect to a critical alarm. Unfortunately, such sub-threshold events are easily obscured by short-term fluctuations in the visualization.

The resulting challenge in time series visualization at scale is presenting the *appropriate* plot that prioritizes users' attention towards significant deviations. To illustrate this challenge using public data, consider the time series depicted in Figure 1. The top plot shows raw

[1] Here and later in this paper, we depict z-scores [40] instead of raw values. This choice of visualization provides a means of normalizing the visual field across plots while still highlighting large-scale trends.

data: an hourly average of the number of NYC taxi passengers over 75 days in 2014 [41]. Daily fluctuations of taxi volume dominate the visual field, obscuring a significant long-term deviation: the number of taxi passengers experienced a sustained dip during the week of Thanksgiving. Ideally, we would smooth the local fluctuations to highlight this deviation in the visualization (Figure 1, middle). However, if we smooth too aggressively, the visualization may hide this trend entirely (Figure 1, bottom).

In this paper, we address the challenge of prioritizing attention in time series using a simple strategy: smooth time series visualizations as much as possible while preserving large-scale deviations. This raises two key questions. First, how can we quantitatively assess the quality of a given visualization in removing small-scale variations and highlighting significant deviations? Second, how can we use such a quantitative metric to produce high-quality visualizations quickly and at scale? We answer both questions through the design of a new time series visualization operator, called ASAP (Automatic Smoothing for Attention Prioritization), which quantifiably improves end-user accuracy and speed in identifying significant deviations in time series, and is optimized to execute at scale.[2]

To address the first question of quantitative metrics for prioritizing attention, we combine two statistics. First, we quantify the smoothness of a time series visualization via the *variance of first differences* [20], or the variation of difference between consecutive points in the series. By applying a moving average of increasing length, we can reduce this variance and smooth the plot. However, as illustrated by Figure 1, it is possible to oversmooth and obscure the trend entirely. Therefore, to prevent oversmoothing, we introduce a constraint based on preserving the *kurtosis* [25]—a measure of the "outlyingness" of a distribution—of the original time series, preserving its structure. Incidentally, this kurtosis measure can also determine when *not* to smooth (e.g., if a series has a few well-defined outlying regions). We demonstrate the utility of this combination of smoothness measure and constraint via two user studies: compared to displaying raw data, smoothing time series visualizations using these metrics improves users' accuracy in identifying anomalies by up to 38.4% and decreases response times by up to 44.3%.

Using these metrics, ASAP automatically chooses smoothing parameters on users' behalf, producing the smoothest visualization that retains large-scale deviations. Given a window of time to visualize (e.g., the past 30 minutes of a time series), ASAP selects and applies an appropriate smoothing parameter to the target series. Unlike existing smoothing techniques that are designed to produce visually indistinguishable representations of the original signal (e.g., [35, 57]), ASAP is designed to "distort" visualizations (e.g. by removing local fluctuations) to highlight key deviations (e.g. as in Figure 1) and prioritize end user attention [18].

There are three main challenges in enabling this efficient, automatic smoothing. First, our target workloads exhibit large data volumes—up to millions of events per second—so ASAP must produce legible visualizations despite high volume. Second, to support interactive use, ASAP must render quickly. As we demonstrate, an exhaustive search over smoothing parameters for 1M points requires over an hour, yet we target sub-second response times. Third, appropriate smoothing parameters may change over time: a high-quality parameter choice for one time period may oversmooth or undersmooth in another. Therefore, ASAP must adapt its smoothing parameters in response to changes in the streaming time series.

To address these challenges, ASAP combines techniques from stream processing, user interface design, and signal processing. First, to scale to large volumes, ASAP pushes constraints regarding the target end-user display into its design. ASAP exploits the fact that its results are designed to be displayed in a fixed number of pixels (e.g., maximum 1334 pixels at a time on the iPhone 7), and uses target resolution as a natural lower bound for the parameter search: there is rarely benefit in choosing parameters that would result in a resolution greater than the target display size. Accordingly, ASAP pre-aggregates data, thus reducing the search space. Second, to further improve rendering time, ASAP prunes the search space by searching for period-aligned time windows (i.e., time lag with high autocorrelation) for periodic data and performing binary search for aperiodic data; we demonstrate both analytically and empirically that this search strategy leads to smooth aggregated series. Third, to quickly respond to changes in fast-moving time series, ASAP avoids recomputing smoothing parameters from scratch upon the arrival of each new data point. Instead, ASAP reuses computation and re-renders visualizations on human-observable timescales.

In total, ASAP achieves its goals of efficient and automatic smoothing by treating visualization properties including end-user display constraints and limitations of human perception as critical design considerations. As we empirically demonstrate, this co-design yields useful results, quickly and without manual tuning. We have implemented ASAP as a time series explanation operator in the MacroBase fast data engine [17], and as a Javascript library. The resulting ASAP prototypes demonstrate order-of-magnitude runtime improvements over alternative search strategies while producing high-quality smoothed visualizations.

In summary, we make the following contributions in this work:

- ASAP, the first stream processing operator for automatically smoothing time series to reduce local variance (i.e., minimize roughness) while preserving large-scale deviations (i.e., preserving kurtosis) in visualization.
- Three optimizations for improving ASAP's execution speed that leverage *i*) target device resolution in pre-aggregation, *ii*) autocorrelation to exploit periodicity, *iii*) and partial materialization for streaming updates.
- A quantitative evaluation demonstrating ASAP's ability to improve user accuracy and response time and deliver order-of-magnitude performance improvements.

The remainder of this paper proceeds as follows. Section 2 describes ASAP's architecture and provides additional background regarding our target use cases. Section 3 formally introduces ASAP's problem definition and quantitative target metrics. Section 4 presents ASAP's search strategy, optimizations, and streaming execution mode. Section 5 evaluates ASAP's visualization quality through two user studies, and ASAP's performance on a range of synthetic and real-world time series. Section 6 discusses related work, and Section 7 concludes.

2. ARCHITECTURE AND USAGE

ASAP provides analysts and system operators an effective and efficient means of highlighting large-scale deviations in time series visualizations. In this section, we describe ASAP's usage and architecture, illustrated via two additional case studies.

Given an input time series (i.e., set of temporally ordered data points) and target interval for visualization (e.g., the last twelve hours of data), ASAP returns a transformed, smoothed time series (e.g., also of twelve hours, but with a smoothing function applied) for visualization. In the streaming setting, as new data points arrive, ASAP continuously smooths each fixed-size time interval, producing a sequence of smoothed time series. Thus, ASAP acts as a transformation over fixed-size sliding windows over a single time

[2]Demo and code available at http://futuredata.stanford.edu/asap/

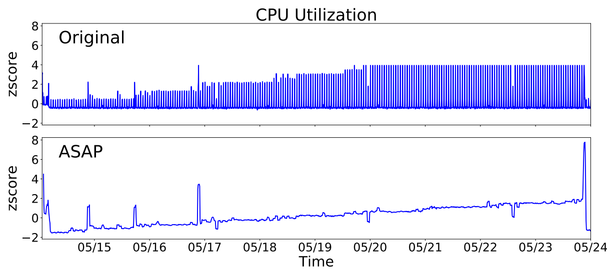

Figure 2: Server CPU usage across a cluster over ten days [41], visualized via a 5 minute average (raw) and an hourly average (via ASAP). The CPU usage spike around May 24th is obscured by frequent fluctuations in the raw time series.

series. When ASAP users change the range of time series to visualize (e.g., via zoom-in, zoom-out, scrolling), ASAP re-renders its output in accordance with the new range. For efficiency, ASAP also allows users to specify a target display resolution (in pixels) and a desired refresh rate (in seconds).

ASAP can run either client-side or server-side. For easy integration with web-based front-ends, ASAP can execute on the client; we provide a JavaScript library for doing so. However, for resource-constrained clients, or for servers with a large number of visualization consumers, ASAP can execute on the server, sending clients the smoothed stream; this is the execution mode that MacroBase [17] adopts, and MacroBase's ASAP implementation is portable to existing stream processing engines.

ASAP acts as a modular tool in time series visualization. It can ingest and process raw data from time series databases such as InfluxDB, as well as from visualization clients such as plotting libraries and frontends. For example, when building a monitoring dashboard, a DevOps engineer could employ ASAP and plot the smoothed results in his metrics console, or, alternatively, overlay the smoothed plot on top of the original time series. ASAP can also *post-process* outputs of time series analyses including motif discovery, anomaly detection, and clustering [38,39,44,70]: given a single time series as output from each of these analyses, ASAP can smooth the time series prior to visualization.

To further illustrate ASAP's potential uses in prioritizing attention in time series, we provide two additional case studies cases below, and additional examples of raw time series and their smoothed counterparts in our extended Technical Report ([54], Appendix C):

Application Monitoring. An on-call application operator is paged at 4AM due to an Amazon CloudWatch alarm on a sudden increase in CPU utilization on her Amazon Web Service cloud instances. After reading the alert message, she accesses her cluster telemetry plots that include CPU usage over the past ten days on her smartphone to obtain a basic understanding of the situation. However, the smartphone's display resolution is too small to effectively display all 4000 readings; as a result, the lines are closely stacked together in the plot, making CPU usage appear stable (Figure 2, top).[3] Unable to obtain useful insights from the plot, the operator must rise from bed and begin checking server logs manually to diagnose the issue. If she were to instead apply ASAP, the usage spike around May 24th would no longer be hidden by noise.

Historical Analyses. A researcher interested in climate change examines a data set of monthly temperature in England over 200

[3]This plot is inspired by an actual use case we encountered in production time series from a large cloud operator; high frequency fluctuations in the plot made it appear that a server was behaving abnormally, when in fact, its overall (smoothed) behavior was similar to others in the cluster.

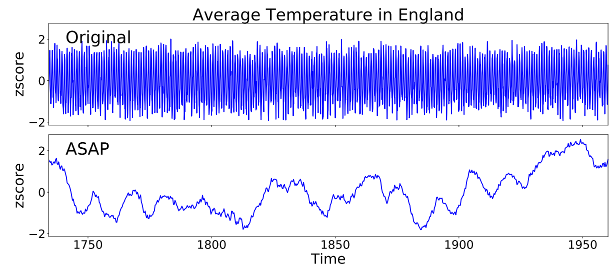

Figure 3: Temperature in England from 1723 to 1970 [34], visualized via a monthly average (raw) and 23-year average (via ASAP). Fluctuations in the raw time series obscure the overall trend.

years. When she initially plots the data to determine long-term trends, her plot spills over five lengths of her laptop screen.[4] Instead of having to scroll to compare temperature in the 1700s with the 1930s, she decides to plot the data herself to fit the entire time series onto one screen. Now, in the re-plotted data (Figure 3, top), seasonal fluctuations each year obscure the overall trend. Instead, if she were to instead use ASAP, she would see a clear trend of rising temperature in the 1900s (Figure 3, bottom).

3. PROBLEM DEFINITION

In this section, we introduce the two key metrics that ASAP uses to assess the quality of smoothed visualizations as well as its smoothing function. We subsequently cast ASAP's parameter search as an optimization problem.

3.1 Roughness Measure

As we have discussed, noise and/or frequent fluctuations can distract users from identifying large-scale trends in time series visualizations. Therefore, to prioritize user attention, we wish to *smooth as much as possible while preserving systematic deviations*. We first introduce a metric to quantify the degree of smoothing.

Standard summary statistics such as mean and standard deviation alone may not suffice to capture a time series's visual smoothness. For example, consider the three time series in Figure 4: a jagged line (series A), a slightly bent line (series B), and a straight line (series C). These time series appear different, yet all have a mean of zero and standard deviation of one. However, series C looks "smoother" than series A and series B because it has a constant slope. Put another way, the differences between consecutive points in series C have smaller variation than consecutive points in series A and B.

To formalize this intuition, we define the roughness (i.e., inverse "smoothness," to be minimized) of a time series as the standard deviation of the differences between consecutive points in the series. The smaller the variation of the differences, the smoother the time series. Formally, given time series $X = \{x_1, x_2, ..., x_N\}$, $x_i \in \mathbb{R}$, we adopt the concept of the *first difference* series [20] as:

$$\Delta X = \{\Delta x_1, \Delta x_2, ...\} \quad s.t. \quad \Delta x_i = x_{i+1} - x_i, i \in \{1, 2, ..., N-1\}$$

Subsequently, we can define the roughness of time series X as the standard deviation of the first difference series:

$$\text{roughness}(X) = \sigma(\Delta X)$$

This use of variance of differences is closely related to the concept of a *variogram* [22], a commonly-used measure in spatial statistics (especially geostatistics) that characterizes the spatial continuity

[4]This is not a theoretical example; in fact, the site from which we obtained this data [34] plots the time series in a six-page PDF. This presentation mode captures fine-grained structure but makes it difficult to determine long-term trends at a glance, as in Figure 3.

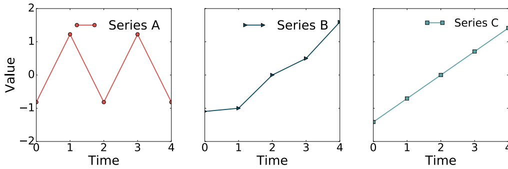

Figure 4: Three time series that appear visually distinct yet all have mean of zero and standard deviation of one. This example illustrates that standard summary statistics such as mean and standard deviation can fail to capture the visual "smoothness" of time series.

(or *surface roughness*) of a given dataset. By this definition, the roughness of the three time series in Figure 4 are 2.04, 0.4, and 0, respectively. Note that a time series will have roughness value of 0 if and only if the corresponding plot is a straight line (like series C). Specifically, a roughness value of 0 implies the differences between neighboring points are identical and therefore the plot corresponding to the series will have a constant slope, resulting in a straight line.

3.2 Preservation Measure

Per the above observation, if we simply minimize roughness, we will produce plots that approximate straight lines. In some cases, this is desirable; if the overall trend is a straight line, then removing noise may, in fact, result in a straight line. However, as our examples in Section 2 demonstrate, many meaningful trends are not accurately represented by straight lines. As a result, we need a measure of "trend preservation" that captures how well we are preserving large-scale deviations within the time series.

To quantify how well we are preserving large deviations in the original time series, we measure the distribution *kurtosis* [25]. Kurtosis captures "tailedness" of the probability distribution of a real-valued random variable, or how much mass is near the tails of the distribution. More formally, given a random variable X with mean μ and standard deviation σ, kurtosis is defined as the fourth standardized moment:

$$\text{Kurt}[X] = \frac{\mathbb{E}[(X-\mu)^4]}{\mathbb{E}[(X-\mu)^2]^2}$$

Higher kurtosis means that more of the variance is contributed by rare and extreme deviations, instead of more frequent and modestly sized deviations [66]. For reference, the kurtosis for univariate normal distribution is 3. Distributions with kurtosis less than 3, such as the uniform distribution, produce fewer and less extreme outliers compared to normal distributions. Distributions with kurtosis larger than 3, such as the Laplace distribution, have heavier tails compared to normal distributions. Figure 5 illustrates two time series sampled from the normal and Laplace distribution discussed above. Despite having the same mean and variance, kurtosis captures the two series' difference in tendency to produce outliers.

To prevent oversmoothing large-scale deviations in the original time series, we compare the kurtosis of the time series before and after applying the smoothing function. If the kurtosis of the original series is greater than or equal to the smoothed series, then the proportion of values that significantly deviate in the smoothed series is no smaller than the proportion in the original series. If smoothing is effective, then the smoothing will "concentrate" the values around regions of large deviation (i.e., significant shifts from the mean) and therefore highlight these deviations.

If the original time series only contains a few extreme outliers, the smoothing is likely to only average out the deviations, which we also account for in our parameter selection procedure. For example, consider a time series with all but one point in the range $[-1,1]$ and a single outlying point that has a value of 10. This outlier

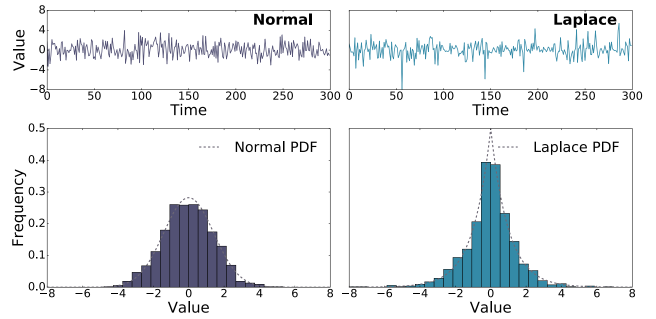

Figure 5: Time series and histograms sampled from a normal distribution (left) and a Laplace distribution (right). Despite having the same mean (0) and variance (2), the Laplace series includes a few large deviations, while the normal includes a large number of moderate deviations. The difference in tendency to produce outliers is captured by kurtosis: the normal distribution has a kurtosis of 3, while the Laplace distribution has a kurtosis of 6.

may be the most important piece of information that users would like to highlight in the time series, so applying a simple moving average only decreases the extent of this deviation (i.e., the kurtosis of the smoothed time series decreases). The kurtosis preservation constraint thus ensures we leave the original time series unsmoothed.

3.3 Smoothing Function

Given our roughness and preservation measure, we wish to smooth our time series as much as possible (i.e., minimizing roughness) while preserving large-scale deviations (i.e., preserving kurtosis). To perform the actual smoothing, we need a smoothing function.

In this paper, we focus on simple moving average (SMA) as the smoothing function. Three reasons motivate this choice. First, SMA is well studied in the stream processing literature, with several existing techniques for efficient execution and incremental maintenance [42]. We adopt these techniques, while using roughness and preservation metrics as a means of automatically tuning SMA parameters for visual effect. Second, SMA is also well studied in the signal processing community. Statistically, the moving average is optimal for recovering the underlying trend of the time series when the fluctuations about the trend are normally distributed [11], despite its light computational footprint and conceptual simplicity compared to alternatives. Third, we experimented with several alternatives including the MinMax aggregation, the Fourier transform [60], and the Savitzky-Golay filter [56]; SMA had fewer parameters to tune and proved more effective at smoothing per our target metrics. We include a visual comparison in [54].

Given input $w \in \mathbb{N}$, SMA averages every sequential set of w points in the original time series X to produce one point in the smoothed series Y. We can express SMA as:

$$\text{SMA}(X,w) = \{y_1, ... y_{N-w}\} \quad s.t. \quad y_i = \frac{1}{w}\sum_{j=0}^{w-1} x_{i+j}$$

When applying SMA over data streams with a sliding window, users can adjust its *window size* (number of points in each window) and *slide size* (distance between neighboring windows) parameter. In time series visualization, slide size determines the sampling frequency of the original time series and, therefore, the number of distinct, discrete data points in the smoothed plot. In this work, we focus on automatically selecting a window size for a given slide size. Instead of tuning slide size, we employ a policy that sets slide size according to the desired number of points (i.e., pixels) in the final visualization (i.e., $\frac{\text{\# original points}}{\text{\#desired points}}$). Increasing the slide size

beyond this threshold results in fewer data points than specified in the smoothed visualization, and decreasing the slide sizes results in a smoothed time series with more data points than available display resolution. Therefore, we found that varying the slide size did not dramatically improve visualization quality.

3.4 ASAP Problem Statement

Given our roughness and preservation measures and smoothing function, we present ASAP's problem statement as follows:

PROBLEM. *Given time series $X = \{x_1, x_2, ..., x_N\}$, let $Y = \{y_1, y_2, ..., y_{N-W}\}$ be the smoothed series of X obtained by applying a simple moving average with window size w (i.e., $y_i = \frac{1}{w}\sum_{j=0}^{w-1} x_{i+j}$). Find window size $\hat{w}$ where:*

$$\hat{w} = \arg\min_{w} \sigma(\Delta(Y)) \quad s.t. \quad \mathrm{Kurt}[Y] \geq \mathrm{Kurt}[X]$$

That is, we wish to reduce roughness in a given time series as much as possible by applying a sliding window average function to the data while preserving kurtosis.

4. ASAP

In this section, we describe ASAP's core search strategy and optimizations for solving the problem of smoothing parameter selection. We first focus on smoothing a single, fixed-length time series, beginning with a walkthrough of a strawman solution (Section 4.1). We then analyze the problem dynamics under a simple, IID distribution (Section 4.2) and, using the insights from this analysis, we develop a pruning optimization based on autocorrelation (Section 4.3). We then introduce a pixel-aware optimization that greatly reduces the input space via preaggregation (Section 4.4). Finally, we discuss the streaming setting (Section 4.5).

4.1 Strawman Solution

As a strawman solution, we could exhaustively search all possible window lengths and return the one that gives the smallest roughness measure while satisfying the kurtosis constraint. For each candidate window length, we need to smooth the series and evaluate the roughness and kurtosis. Each of these computations requires linear time ($O(N)$). However, there are also many candidates to evaluate: for a time series of size N, we may need to evaluate up to N possible window lengths, resulting in a total running time of $O(N^2)$. As we illustrate empirically in Section 5, in the regime where N is even modestly large, this computation can be prohibitively expensive.

We might consider improving the runtime of this exhaustive search by performing grid search via a sequence of larger step sizes, or by performing binary search. However, as we will demonstrate momentarily, the roughness metric is not guaranteed to be monotonic in window length and therefore, the above search strategies may deliver poor quality results. Instead, in the remainder of this section, we describe an alternative search strategy that is able to retain the quality of exhaustive search while achieving meaningful speedups by quickly pruning unpromising candidates and by optimizing for the desired pixel density.

4.2 Basic IID Analysis

To leverage properties of the roughness metric to speed ASAP's search, we first consider how window length affects the roughness and kurtosis of the smoothed series.

As a first step, consider a time series $X : \{x_1, x_2, ..., x_N\}$ consisting of samples drawn identically independently distributed (IID) from some distribution with mean μ and standard deviation σ. After applying a moving average of window length w, we obtain the smoothed series:

$$Y = \mathrm{SMA}(X, w), \quad y_i = \frac{1}{w}\sum_{j=0}^{w-1} x_{i+j}, i \in \{1, 2, ..., N-w\}$$

We denote the first difference series as $\Delta Y = \{\Delta y_1, \Delta y_2, ...\}$, where

$$\Delta y_i = y_{i+1} - y_i = \frac{1}{w}\sum_{j=0}^{w-1}(x_{i+j+1} - x_{i+j}) = \frac{1}{w}(x_{i+w} - x_i)$$

For convenience, we also denote the first $N-w$ points of X as $X_f = \{x_1, x_2, ..., x_{N-w}\}$ and the last $N-w$ points of X as $X_l = \{x_{w+1}, x_{w+2}, ..., x_N\}$. Then $\Delta Y = \frac{1}{w}(X_l - X_f)$, and roughness of the smoothed series Y can be written as:

$$\mathrm{roughness}(Y) = \sigma(\Delta Y) = \frac{1}{w}\sqrt{\mathrm{var}(X_f) + \mathrm{var}(X_l) - 2\mathrm{cov}(X_l, X_f)} \quad (1)$$

Since each x_i is drawn IID from the same distribution, we have $\mathrm{var}(X_f) = \mathrm{var}(X_l) = \sigma^2$ and $\mathrm{cov}(X_f, X_l) = 0$. Substituting in Equation 1 we obtain:

$$\mathrm{roughness}(Y) = \frac{\sqrt{2}\sigma}{w} \quad (2)$$

Therefore, for IID data, roughness linearly decreases with increased window size. Further, the kurtosis of random variable S, defined as the sum of independent random variables $R_1, R_2, ..., R_n$, is

$$\mathrm{Kurt}[S] - 3 = \frac{1}{(\sum_{j=1}^{n}\sigma_j^2)^2}\sum_{i=1}^{n}\sigma_i^4(\mathrm{Kurt}[R_i] - 3) \quad (3)$$

where σ_i is the standard deviation of random variable R_i. In our case, Y is the sum of w IID random variables X [49]. Thus, Equation 3 simplifies to

$$\mathrm{Kurt}[Y] - 3 = \frac{\mathrm{Kurt}[X] - 3}{w}. \quad (4)$$

Therefore, for IID series drawn from distributions with initial kurtosis less than 3, kurtosis monotonically increases with window length and for series drawn from distributions with initial kurtosis larger than 3, kurtosis monotonically decreases.

In summary, these results indicate that for IID data, we can simply search for the largest window length that satisfies kurtosis constraint via binary search. Specifically, given a range of candidate window lengths, ASAP applies SMA with window length that is in the middle of the range. If the resulting smoothed series violates the kurtosis constraint, ASAP searches the smaller half of the range; otherwise, ASAP searches the large half. This binary search routine is justified because the roughness of the smoothed series monotonically decreases with window length (Equation 2), and the kurtosis of the smoothed series monotonically decreases with window length or achieves its minimum at window length equals one (Equation 4).

However, many time series exhibit temporal correlations, which breaks the above IID assumption. This complicates the problem of window search, and we present a solution in the next subsection.

4.3 Optimization: Autocorrelation Pruning

We have just shown that, for IID data, binary search is accurate, yet many time series are not IID; instead, they are often periodic or exhibit other temporal correlations. For example, many servers and automated processes have regular workloads and exhibit periodic behavior across hourly, daily, or longer intervals.

To measure temporal correlations within a time series, we measure the time series *autocorrelation*, or the similarity of a signal with itself as a function of the time lag between two points [58]. Formally, given a process X whose mean μ and variance σ^2 are

time independent (i.e., is a *weakly stationary* process), denote X_t as the value produced by a given run of the process at time t. The lag τ autocorrelation function (ACF) on X is defined as

$$\text{ACF}(X,\tau) = \frac{\text{cov}(X_t, X_{t+\tau})}{\sigma^2} = \frac{\mathbb{E}[(X_t-\mu)(X_{t+\tau}-\mu)]}{\sigma^2}$$

The value of the autocorrelation function ranges from [-1, 1], with 1 indicating perfect correlation, 0 indicating the lack of correlation and -1 indicating anti-correlation.

4.3.1 Autocorrelation and Roughness

As suggested above, we can take advantage of the periodicity in the original time series to prune the search space. Specifically, given the original time series $X : \{x_1, x_2, ..., x_N\}$, and the smoothed series $Y : \{y_1, y_2, ..., y_{N-w}\}$ obtained by applying a moving average of window length w, we show that

$$\text{roughness}(Y) = \frac{\sqrt{2}\sigma}{w}\sqrt{1-\frac{N}{N-w}\text{ACF}(X,w)} \quad (5)$$

for a weakly stationary process X. We provide a full derivation of Equation 5 in [54], Appendix A.1; however, intuitively, this equation illustrates that window length and autocorrelation both affect roughness. For example, consider a time series recording the number of taxi trips taken over 30-minute intervals. Due to the regularity of commuting routines, this time series exhibits autocorrelation across week-long periods (e.g., a typical Monday is likely to be much more similar to another Monday than a typical Saturday). Furthermore, a rolling weekly average of the number of trips should, in expectation, have a smaller variance than rolling 6-day averages: for example, if people are more likely to take taxis during weekdays than during weekends, then the average from Monday to Saturday should be larger than the average from Tuesday to Sunday. Therefore, window lengths that align with periods of high autocorrelation make the resulting series smoother.

We experimentally validate this relationship on real world data ([54], Appendix A.1) and use this relationship to aggressively prune the space of windows to search (Section 4.3.3).

4.3.2 Autocorrelation and Kurtosis

In addition to roughness, we also investigate the impact of temporal correlations on the kurtosis constraint. We start with an example that illustrates how choosing window lengths with high temporal correlation (i.e., autocorrelation) leads to high kurtosis.

Consider a time series (sparkline below, left) consisting of a sine wave with 640 data points. Each complete sine wave is 32 data points long, and in the region from 320th to 336th data point, the peak of the sine wave is taller than usual. When applying a window that are multiples of the period, the smoothed series (sparkline below, right) is zero everywhere except around the region where the peak is higher. The smoothed series in the latter case has higher kurtosis because it only contains one large deviation from the mean. In contrast, applying a moving average with window length that is not a multiple of the period will not highlight this peak.

This example illustrates the case when applying moving average with window lengths aligning with the period of the time series can not only remove periodic behavior from the visualization (therefore highlighting deviations from period to period), but also the kurtosis of the smoothed series is also larger at the periodic window size. In ASAP, we find that, empirically, if a candidate window that is aligned with the time series period does not satisfy the kurtosis constraint, it is rare that a nearby candidate window that is off the period would satisfy the constraint instead; moreover, such a nearby aperiodic window would likely result in a rougher series.

Algorithm 1 Search for periodic data

```
Variables:
X: time series; candidates: array of candidate window lengths
acf[w]: autocorrelation for w; maxACF: maximum autocorrelation peak
opt: a set of states for the current best candidate in the search, including
     {roughness, wLB, window, largestFeasibleIdx}

function UPDATELB(wLB, w)                              ▷ Update lower bound
    return MAX(wLB, w·sqrt((1−maxACF)/(1−acf[w])))
function ISROUGHER(currentBestWindow, w)               ▷ Compare roughness
    return sqrt(1−acf[w])/w > sqrt(1−acf[currentBestWindow])/currentBestWindow
function SEARCHPERIODIC(X, candidates, opt)
    N = candidates.length
    for i ∈ {N, N-1, ..., 1} do                         ▷ Large to small
        w = candidates[i]
        if w < opt.wLB then                             ▷ Lower bound pruning
            break
        if ISROUGHER(opt.window, w) then                ▷ Roughness pruning
            continue
        Y = SMA(X, w)
        if ROUGHNESS(Y) < opt.roughness and
            KURT(Y) ≥ KURT(X) then                      ▷ Kurtosis constraint
            opt.window = w
            opt.roughness = ROUGHNESS(Y)
            opt.wLB = UPDATELB(opt.wLB, w)
            opt.largestFeasibleIdx = MAX(opt.largestFeasibleIdx, i)
    return opt
```

4.3.3 Pruning Strategies

Following the above observations, ASAP adopts the following two pruning strategies. The corresponding pseudocode for ASAP's search is listed in Algorithm 1.

Autocorrelation peaks. To quickly filter out suboptimal window lengths, ASAP searches for windows that correspond to periods of high autocorrelation. Specifically, ASAP only checks autocorrelation *peaks*, which are local maximums in the autocorrelation function and correspond to periods in the time series. For periodic datasets, these peaks are usually much higher than neighboring points, meaning that the corresponding roughness of the smoothed time series is much lower. This is justified by Equation 5—all else equal, roughness decreases with the increase of autocorrelation.

Naïvely computing autocorrelation via brute force requires $O(n^2)$ time; thus, a brute force this approach is unlikely to deliver speedups over the naïve exhaustive search for finding window length. However, we can improve the runtime of autocorrelation, to $O(n\log(n))$ time, using two Fast Fourier Transforms (FFT) [52]. In addition to providing asymptotic speedups, this approach also allows us to make use of optimized FFT routines designed for signal processing, in the form of mature software libraries and increasingly common hardware implementations (e.g., DSP accelerators).

Large to small. Since roughness decreases with window length (Equation 5, roughness is proportional to $\frac{1}{w}$), ASAP searches from larger to smaller window lengths. When two windows $w_1, w_2 (w_1 < w_2)$ have identical autocorrelation, the larger window will always have lower roughness under SMA. However, when the windows have different autocorrelations a_1, a_2, the smaller window w_1 will only provide lower roughness if $w_1 > w_2\sqrt{\frac{1-a_1}{1-a_2}}$. Moreover, since ASAP only considers autocorrelation peaks as candidate windows, a_1 is no larger than the largest autocorrelation peak in the time series, which we refer to as *maxACF*. Therefore, the smallest window w_1

Algorithm 2 Batch ASAP

```
function FINDWINDOW(X, opt)
    candidates = GETACFPEAKS(X)
    opt = SEARCHPERIODIC(X, candidates, opt)
    head = MAX(opt.wLB, candidates[opt.largestFeasibleIdx + 1])
    tail = MIN(maxWindow, candidates[opt.largestFeasibleIdx + 1])
    opt = BINARYSEARCH(X, head, tail, opt)
    return opt.window
```

that is able to produce smoother series than w_2 must satisfy

$$w_1 > w_2\sqrt{\frac{1-a_1}{1-a_2}} > w_2\sqrt{\frac{1-maxACF}{1-a_2}} \quad (6)$$

If ASAP finds a feasible window length for smoothing relatively early in the search, it uses Equation 6 to prune smaller windows that will not produce a smoother series (UPDATELB in Algorithm 1). Similarly, once ASAP has a feasible window, it can also prune window candidates whose roughness estimate (via Equation 5) is larger than the current best (ISROUGHER in Algorithm 1). In summary, the two pruning rules are complementary: the lower bound pruning reduces the search space from below, eliminating search candidates that are too small; the roughness estimate reduces the search space from above, further eliminating unpromising candidates above the lower bound.

Our pruning strategies exploit temporal correlations, which will be less effective for aperiodic data. However, per our analysis in Section 4.2, IID data is better-behaved under simple search. Therefore, ASAP falls back to binary search for aperiodic data. ASAP allows users to optionally specify a maximum window size to consider. Together, the search procedure is listed in Algorithm 2.

4.4 Optimization: Pixel-aware Preaggregation

In addition to leveraging statistical properties of the data, ASAP can also leverage perceptual properties of the target devices. That is, ASAP's smoothed time series are designed to be displayed on devices such as computer monitors, smartphones, and tablet screens for human consumption. Each of these target media has a limited resolution; as Table 1 illustrates, even high-end displays such as the 2016 Apple iMac 5K are limited in horizontal resolution to 5120 pixels, while displays such as the 2016 Apple Watch contain as few as 272 pixels. These pixel densities place restrictions on the amount of information that can be displayed in a plot.

ASAP is able to leverage these limited pixel densities to improve search time. Specifically, ASAP avoids searching for window lengths that would result in more points than pixels supported by the target device. For example, a datacenter server may report CPU utilization metrics every second (604,800 points per week). If an operator wants to view a plot of weekly CPU usage on her 2016 Retina MacBook Pro, she will only be able to see a maximum of 2304 distinct pixels as supported by the display resolution. If ASAP smooths using a window smaller than 262 seconds (i.e., $\frac{604,800}{2304}$), the resulting plot will contain more points than pixels on the operator's screen (i.e., to display all information in the original time series, the slide size must be no larger than window length). As a result, this *point-to-pixel* ratio places a lower bound on the window length that ASAP should search. In addition, the point-to-pixel ratio is also a useful proxy for the granularity of information content contained in a given pixel. While one could search for window lengths that correspond to sub-pixel boundaries, in practice, we have found that searching for windows that are integer multiples of the point-to-pixel ratio suffices to capture the majority of useful information in a plot. We provide an analysis in [54] (Appendix A.2), and empirically demonstrate these phenomena in Section 5.2.2.

Combined, these observations yield a powerful optimization for

Table 1: Popular devices and search space reduction achieved via pixel-aware preaggregation for a series of 1M points.

Device	Resolution	Reduction on 1M pts
38mm Apple Watch	272 x 340	3676x
Samsung Galaxy S7	1440 x 2560	694x
13" MacBook Pro	2304 x 1440	434x
Dell 34 Curved Monitor	3440 x 1440	291x
27" iMac Retina	5120 x 2880	195x

ASAP's search strategy. Given a target display resolution (or desired number of points for a plot), ASAP pushes this information into its search strategy by only searching windows that are integer multiples of point-to-pixel ratio. To implement this efficiently, ASAP preaggregates the data points according to groups of size corresponding to the point-to-pixel ratio, then proceeds to search over these preaggregated points. With this preaggregation, ASAP's performance is not dependent on the number of data points in the original time series but instead depends on the target resolution of the end device. As a result, in Section 5, we evaluate ASAP's performance over different target resolutions and demonstrate scalability to millions of incoming data points per second.

4.5 Streaming ASAP

ASAP is designed to process streams of time series and update plots as new data arrives. In this section, we describe how ASAP efficiently operates over data streams by combining techniques from traditional stream processing with constraints on human perception.

Basic Operations. As new data points arrive, ASAP must update its smoothing parameters to accommodate changes in the trends, such as periodicity. As in Section 4.4, in the streaming setting, we can preaggregate data as it arrives according to the point-to-pixel ratio. However, as data transits the duration of time ASAP is configured to smooth (e.g., the last 30 minutes of readings), ASAP must remove outdated points from the window. To manage this intermediate state, ASAP adapts techniques from streaming processing that sub-aggregate input streams for performance gain. That is, sliding window aggregates such as SMA can be computed more efficiently by sub-aggregating the incoming data into disjoint segments (i.e., *panes*) that are sizes of greatest common divisor of window and slide size [42]. We can perform similar pixel-aware preaggregations for data streams using panes.

ASAP maintains a linked list of all subaggregations in the window and, when prompted, re-executes the search routine from the previous section. Instead of recomputing the smoothing window from scratch, ASAP records the result of the previous rendering request and uses it as a "seed" for the new search. Specifically, since streams often exhibit similar behavior over time, the previous smoothing parameter could possibly apply to the current request. In this case, ASAP starts the new search with a known feasible window length, which enables the roughness estimation pruning procedure (ISROUGHER in Algorithm 1) to rule out candidates automatically.

Optimization: On-demand updates. A naïve strategy for updating ASAP's output is to update the plot upon arrival of each point. This is inefficient. For example, consider a data stream with a volume of one million points per second. Refreshing the plot for every data point requires updating the plot every 0.001 milliseconds. However, since humans can only perceive changes on the order of 60 events per second [33], this update rate is unnecessary. With pixel-aware preaggregation, we would refresh for each aggregated data point instead, the rate of which may still be higher than necessary. To visualize 10 minutes of data on a 27-inch iMac for example, pixel-aware preaggregation provides us aggregates data points that are 12ms apart (83Hz). As a result, we designed ASAP to only

Algorithm 3 Streaming ASAP

Variables: **X**: preaggregated time series; **interval**: refresh interval

```
function CHECKLASTWINDOW(X, opt)
    Y = SMA(X, opt.window)
    if KURT(Y) ≥ KURT(X) then
        update roughness and wLB for opt
    else
        re-initialize opt
    return opt
function UPDATEWINDOW(X, interval)
    while True do
        collect new data points until interval
        subaggregate new data points, and update X
        UPDATEACF(X)
        opt = CHECKLASTWINDOW(X, opt)
        FINDWINDOW(X, opt)
```

refresh at (configurable) timescales that are perceptible to humans. In our example above, a 1Hz update speed results in a $83\times$ reduction in the number of calls to the ASAP search routine; this reduction means we will either use less processing power and/or be able to process data at higher volumes. In Section 5.2.2, we empirically investigate the relationship between total runtime and refresh rate.

Putting it all together. Algorithm 3 shows the full streaming ASAP algorithm. ASAP aggregates the incoming data points according to the desired point-to-pixel ratio, and maintains a linked list of the aggregates. After collecting a refresh-interval-time worth of aggregates, ASAP updates data points in the current visualization, and recalculates the autocorrelation (UPDATEACF). ASAP then checks whether the window length from the last rendering request is still feasible (CHECKLASTWINDOW). If so, ASAP uses this previous window length to quickly improve the lower bound for the new search. Otherwise, ASAP starts the new search from scratch.

5. EXPERIMENTAL EVALUATION

In this section, we experimentally evaluate the quality and efficiency of ASAP's visualizations via two user studies and a series of performance benchmarks. Our goal is to demonstrate that:

- ASAP's visualizations improve both user accuracy and response time in identifying deviations (Section 5.1).
- ASAP identifies high quality windows quickly (Section 5.2.1).
- ASAP's optimizations—autocorrelation, pixel-aware aggregation and on-demand update—provide complementary speedups up to seven order-of-magnitude over baseline (Section 5.2.2).

5.1 User Studies

We first evaluate the empirical effectiveness of ASAP's visualizations via two user studies. We demonstrate that ASAP visualizations lead to faster and more accurate identifications of anomalies.

Visualization Techniques for Comparison. In each study, we compare ASAP's visualizations to a set of alternatives (cf. Section 6): *i*) the original data, *ii*) the M4 algorithm [35], *iii*) the Visvalingam-Whyatt algorithm [64], *iv*) piecewise aggregate approximation (PAA) [37] (PAA100 reduces the number of points to 100; PAA800 reduces to 800), and *v*) an "oversmoothed" plot generated by applying SMA with a window size of $\frac{1}{4}$ of the number of points. All plots are rendered using an 800 pixel resolution.

Datasets. We select five publicly-available time series described in Table 2 because each has known ground truth anomalies. We use this ground truth as a means of evaluating visualization quality by measuring users' ability to identify anomalous behaviors in the visualization and by assessing their preferences. Plots and text

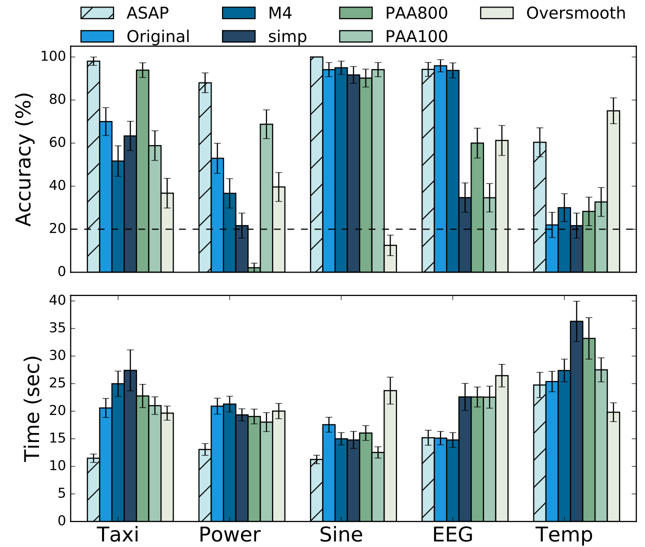

Figure 6: Accuracy in identifying anomalous regions and response times, with error bars indicating standard error of samples. On average, ASAP improves accuracy by 32.7% while reducing response time by 28.8% compared to other visualizations.

descriptions used in our user studies are available in Appendix B of the extended Technical Report [54].

5.1.1 *User Study I: Anomaly Identification*

To assess how different smoothing algorithms affect users' ability to identify anomalies in time series visualization, we ran a large-scale user study on Amazon Mechanical Turk, in accordance with Stanford University IRB guidelines.

In this first study, we presented users with textual descriptions of each dataset and anomaly, and asked them to select one out of the five equally-sized regions in a given time series visualization where the described anomaly occurred. Users performed anomaly identification using a single, randomly chosen visualization for each dataset, and, for each identification task, we recorded user's accuracy and response time. The user study involved 700 distinct Amazon Mechanical Turk workers, 406 of whom self-reported as intermediate or expert users of Excel, 324 of whom self-reported as intermediate or expert users of databases, and 288 of whom self-reported seeing time series at least once per month.

We report the accuracy and response time for the seven visualization techniques described above in Figure 6, where each bar in the plot represents an average of 50 users. When shown ASAP's visualizations, users were more likely to correctly identify the anomalous region, and to do so more quickly than alternatives. Specifically, users' accuracy of identifying the anomalous region increased by 21.3% when presented with ASAP's visualizations instead of the original time series, and users did so 23.9% more quickly. Compared to all other methods, users experience an average of 35.0% (max: 43.1%) increase in accuracy and 29.8% (max: 33.8%) decrease in response time with ASAP. ASAP led to most accurate results for all datasets except for the Temp dataset, in which the oversmooth strategy was able to better highlight (by 14.6%) a large increasing temperature trend over several decades, corresponding to the rise of global warming [65]. However, ASAP results in 38.4% more accurate identification than the raw data for this dataset. Overall, ASAP consistently produces high-quality plots, while the quality of alternative visualization methods varies widely across datasets. We provide additional results from a sensitivity analysis of the impact

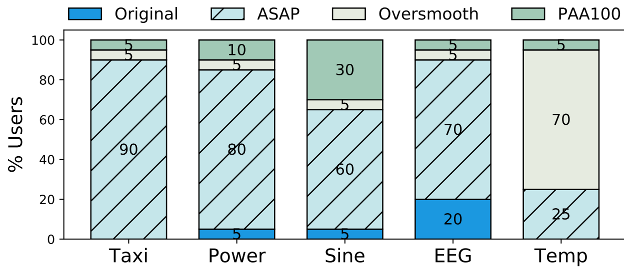

Figure 7: Visual preference study. Users prefer ASAP 65% of the time on average, and 59% more often than the original time series.

of roughness and kurtosis in [54] (Appendix B.2), where we show that ASAP also outperforms alternative configurations in average accuracy and response time.

5.1.2 *User Study II: Visual Preferences*

In addition to the above user study, which was based on a large crowdsourced sample, we performed a targeted user study with 20 graduate students in Computer Science. We retained the same datasets and descriptions of dataset and anomaly from the previous study, and asked users to select the *visualization* that best highlights the described anomaly in order to measure visualization preferences. In contrast with the previous study, due to smaller sample size, we presented a set of four visualizations–original, ASAP, PAA100, and oversmooth–anonymized and randomly permuted for each dataset.

Figure 7 presents results from this study. Across all five datasets, users preferred ASAP's visualizations as a means of visualizing anomalies in 65% of the trials (random: 25%). Specifically, for datasets Taxi (Figure B.4, Appendix), EEG (Figure B.5, Appendix), and Power (Figure B.7, Appendix), over 70% of users preferred ASAP's presentation of the time series. For these datasets, smoothing helps remove the high-frequency fluctuations in the original dataset and therefore better highlights the known anomalies. For dataset Sine (Figure B.6, Appendix), a simulated noisy sine wave with a small region where the period is halved, 60% users chose ASAP, followed by 30% choosing PAA100. In follow-up interviews, some users expressed uncertainty about this final plot: while the ASAP plot clearly highlights the anomaly, the PAA100 plot more closely resembles the description of the original signal. In the Temp dataset, 70% of users chose the oversmoothed plot, and 25% chose ASAP. For this dataset—which contains monthly temperature readings spanning over 250 years—aggressive smoothing better highlights the decade-long warming trend (Figure B.3, Appendix). In addition, no user preferred the original temperature plot, further confirming that smoothing is beneficial.

In summary, these results illustrate the utility of ASAP's target metrics in producing high-quality time series visualizations that highlight anomalous behavior.

5.2 Performance Analysis

While the above user studies illustrate the utility of ASAP's visualizations, it is critical that ASAP is able to render them quickly and over changing time series. To assess ASAP's end-to-end performance as well as the impact of each of its optimizations, we performed a series of performance benchmarks.

Implementation and Experimental Setup. We implemented an ASAP prototype as an explanation operator for processing output data streams in the MacroBase streaming analytics engine [17]. We report results from evaluating the prototype on a server with four Intel Xeon E5-4657L 2.4GHz CPUs containing 12 cores per CPU and 1TB of RAM (although we use considerably less RAM in processing). We exclude data loading time from our results but report all other computation time. We report results from the average of three or more trials per experiment. We use a set of 11 of datasets of varying sizes collected from a variety of application domains; Table 2 provides detailed descriptions of each dataset; we provide plots from each experiment in [54].

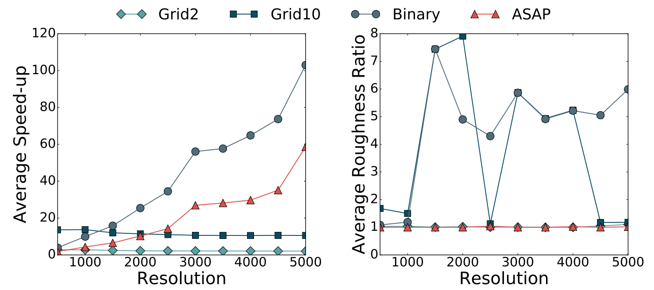

Figure 8: Throughput and quality of ASAP, grid search, and binary search over pre-aggregated time series according to varying target resolutions. Both plots report throughput and roughness compared to exhaustive search and report an average from the seven largest datasets in Table 2. ASAP exhibits similar speed-ups to binary search while retaining quality close to exhaustive search. ASAP's autocorrelation calculation incurs up to 50% overhead compared to binary search but its results are up to 7.5× smoother.

5.2.1 *End-to-End Performance*

To demonstrate ASAP's ability to find high-quality window sizes quickly, we evaluate ASAP's window quality and search time compared to alternative search strategies. We compare to exhaustive search, grid search of varying step size (2, 10), and binary search.

First, as Table 2 illustrates, with a target resolution of 1200 pixels, ASAP is able to find the same smoothing parameter as the exhaustive search for all datasets by checking an average of 8.64 candidates, instead of 113.64 candidates per dataset for the exhaustive search. For the Twitter_AAPL dataset, both exhaustive search and ASAP leave the visualization unsmoothed; this time series (Figure C.1, Appendix) is smooth except for a few unusual peaks, so further smoothing would have averaged out the peaks.

Second, we evaluate differences in wall-clock speed and achieved smoothness. All algorithms run on preaggregated data, so the throughput difference is only caused by the difference in search strategies; we further investigate the impact of pixel-aware preaggregation in Section 5.2.2. Figure 8 shows that ASAP is able to achieve up to 60× faster search time than exhaustive search over pre-aggregated series, with near-identical roughness ratio. ASAP's runtime performance scales comparably to binary search, although it lags by up to 50% due to its autocorrelation calculation. However, while ASAP produces high-quality smoothed visualizations, binary search is up to 7.5× rougher than ASAP. Grid search with step size two delivers similar-quality results as ASAP but fails to scale, while grid search with step size ten delivers the worst overall results. In summary, end-to-end, ASAP provides significant speedups over exhaustive search while retaining its quality of visualization. We provide additional runtime comparison with PAA and M4 in [54] (Appendix A.3).

5.2.2 *Impact of Optimizations*

In this section, we further evaluate the contribution of each of ASAP's optimizations—autocorrelation pruning, pixel-aware preaggregation, on-demand update—both individually and combined.

Table 2: Dataset descriptions and batch results from ASAP and exhaustive search over pre-aggregated data for target resolution 1200 pixels. ASAP finds the same choice of smoothing parameter as optimal, exhaustive search while searching an average of 13× fewer candidates.

Dataset	Description	# points	Duration	Exhaustive Search	ASAP
gas_sensor [45]	Recording of a chemical sensor exposed to a gas mixture	4,208,261	12 hours	window size: 26 # candidates: 115	window size: 26 # candidates: 7
EEG [39]	Excerpt of electrocardiogram	45,000	180 sec	window size: 22 # candidates: 119	window size: 22 # candidates: 21
Power [39]	Power consumption for a Dutch research facility in 1997	35,040	35040 sec	window size: 16 # candidates: 115	window size: 16 # candidates: 23
traffic_data [14]	Vehicle traffic observed between two points for 4 months	32,075	4 months	window size: 84 # candidates: 120	window size: 84 # candidates: 6
machine_temp [41]	Temperature of an internal component of an industrial machine	22,695	70 days	window size: 44 # candidates: 125	window size: 44 # candidates: 7
Twitter_AAPL [41]	A collection of Twitter mentions of Apple	15,902	2 months	window size: 1 # candidates: 120	window size: 1 # candidates: 7
ramp_traffic [45]	Car count on a freeway ramp in Los Angeles	8,640	1 month	window size: 96 # candidates: 117	window size: 96 # candidates: 5
sim_daily [41]	Simulated two week data with one abnormal day	4,033	2 weeks	window size: 72 # candidates: 100	window size: 72 # candidates: 5
Taxi [41]	Number of NYC taxi passengers in 30 min bucket	3,600	75 days	window size: 112 # candidates: 120	window size: 112 # candidates: 4
Temp [34]	Monthly temperature in England from 1723 to 1970	2,976	248 years	window size: 112 # candidates: 120	window size: 112 # candidates: 4
Sine [38]	Noisy sine wave with an anomaly that is half the usual period	800	800 sec	window size: 64 # candidates: 79	window size: 64 # candidates: 6

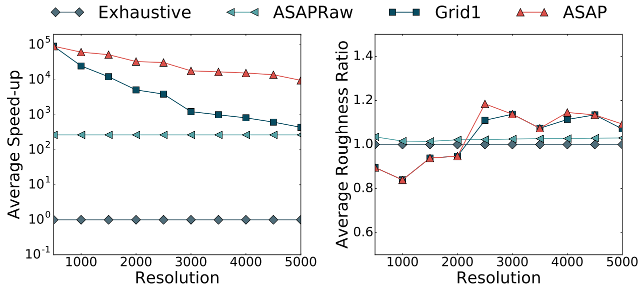

Figure 9: Throughput and quality of ASAP, exhaustive search on preaggregated time series over the baseline (exhaustive search over the original time series) under varying resolution. ASAP on aggregated time series is up to 4 orders of magnitude faster, while retaining roughness within 1.2× the baseline.

Pixel-aware preaggregation. We first perform a microbenchmark on the impact of pixel-aware preaggregation (Section 4.4) on both throughput and smoothness. Figure 9 shows the throughput and quality of ASAP and exhaustive search with and without pixel-aware preaggregation under varying target resolutions. With pixel-aware pre-aggregation, ASAP achieves roughness within 20% of exhaustive search over the raw series and sometimes outperforms exhaustive search because the initial pixel-aware preaggregation results in lower initial kurtosis. The preaggregation strategy enables a five and a 2.5 order-of-magnitude speedups over exhaustive search (Exhaustive) and ASAP on raw data (ASAPno-agg), respectively. In summary, pixel-aware preaggregation has a modest impact on result quality and massive impact on computational efficiency (i.e., sub-second versus hours to process 1M points). Should users desire exact result quality, they can still choose to disable pixel-aware preaggregation while retaining speedups from other optimizations. We provide additional analysis of pre-aggregation and additional experimental results in [54] (Appendix A.2).

On-demand update. To investigate the impact of the update interval in the streaming setting (Section 4.5), we vary ASAP's refresh rate and report throughput under each setting. The log-log plot (Figure 10) shows a linear relationship between the refresh interval and throughput. This is expected because updating the plot twice as often means that it would take twice as long to process the same number of points. For fast-moving streams, this strategy can save substantial computational resources.

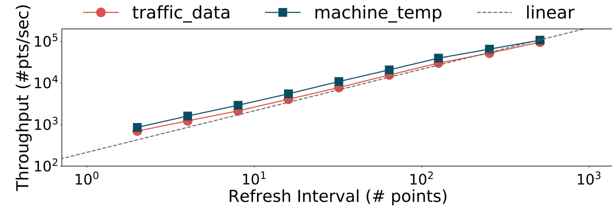

Figure 10: Throughput of streaming ASAP on two datasets, with varying refresh interval (measured in the number of points) for target resolution 2000 pixels in log-log scale. The plot captures a linear relationship between throughput and refresh interval as expected.

Factor Analysis. In addition to analyzing the impact of individual optimizations, we also investigate how ASAP's three main optimizations contribute to overall performance. Figure 11 (left) depicts a factor analysis, where we enable each optimization cumulatively in turn. Pixel-aware aggregation provides between two and four orders of magnitude improvement depending on the target resolution. Autocorrelation provides an additional two orders of magnitude. Finally, on demand update with a daily refresh interval (updating for every 288 points in the original series versus updating for each preaggregated point) provides another two order-of-magnitude speedups. These results demonstrate that ASAP's optimizations are additive and that end-to-end, optimized streaming ASAP is approximately seven orders of magnitude faster than the baseline.

To illustrate that no one ASAP optimization is responsible for all speedups, we perform a lesion study, where we remove each optimization from ASAP while keeping the others enabled (Figure 11, right). Removing on-demand update, pixel-aware aggregation, and autocorrelation-enabled pruning each decreases the throughput by approximately two to three orders of magnitude, in line with results from the previous experiment. Without pixel-aware preaggregation, ASAP makes no distinction between higher and lower resolution setting, so the throughput for both resolutions are near-identical. In

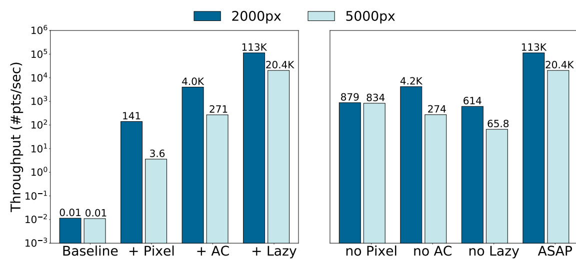

Figure 11: Factor analysis on machine_temp dataset under two display settings. Cumulatively enabling optimizations shows that each contributes positively to final throughput; combined, the three optimizations enable seven orders of magnitude speedup over the baseline. In addition, removing each optimization decreases the throughput by two to three orders of magnitude.

contrast, removing the other two optimizations degrades the performance for the higher resolution setting more. Thus, each of ASAP's optimizations is necessary to achieve maximum performance.

6. RELATED WORK

ASAP's design draws upon work from several domains including stream processing, data visualization, and signal processing.

Time Series Visualization. Data visualization management systems that automate and recommend visualizations to users have recently become a topic of active interest in the database and human-computer interaction community [69]. Recent systems including SeeDB [50], Voyager [67], and ZenVisage [59] focus on recommending visualizations for large-scale data sets, particularly for exploratory analysis of relational data [47]. In this paper, we focus on the visualization of deviations within time series.

Within the time series literature, which spans simplification and reduction [26, 29, 37, 53, 57, 64], information retrieval [31], and data mining [12, 28, 46, 55], visualization plays an important role in analyzing and understanding time series data [24]. There are a number of existing approaches to time series visualization [13]. Perhaps most closely related is M4 [35], which downsamples the original time series while preserving the shape—a perception-aware procedure [27]. Unlike M4 and many existing time series visualization techniques, which focus on producing visually indistinguishable representations of the original time series (often using fewer points) by optimizing metrics such as pixel accuracy [26, 29, 53, 57, 64], ASAP visually highlights large-scale deviations in the time series by smoothing short term fluctuations.

To illustrate this difference in goals, we compared ASAP, M4 and the Visvalingam-Whyatt line simplification algorithm [64] both on pixel accuracy (Appendix B.1, [54]) and on end user accuracy of identifying anomalies (Section 5.1): ASAP is far worse at preserving pixel accuracy (up to 93% worse, average: 91.8% worse) than M4 but improves accuracy by up to 51% (average: 26.7%) for end-user anomaly identification tasks. These trends were similar for piecewise aggregate approximation [37]—which, in contrast, was *not* originally designed for visualization. Despite differing objectives, we believe that pixel-preserving visualization techniques such as M4 are complementary to ASAP: a visualization dashboard could render the original time series using M4 and overlay with ASAP to highlight long-term deviations.

Signal Processing. Noise reduction is a classic and extremely well studied problem in signal processing. Common reduction techniques include the wavelet transform [23], convolution with smoothing filters [21, 56], and non-linear filters [63]. In this work, we study a specific type of linear smoothing filter—moving average—and the problem of its automatic parameter selection. Despite its simplicity, moving average is an effective time domain filter that is optimal at reducing random noise while retaining a sharp step response (i.e., rapid rise in the data) [11].

While there are many studies on parameter selection mechanisms for various smoothing functions [48], the objective of most of the above selection criteria is to minimize variants of reconstruction error to the original signal. In contrast, ASAP's quality metric is designed to visually highlight trends and large deviations, leading to a different optimization strategy. Biomedical researchers have explored ideas of selecting a moving average window size that highlights significantly deviating region of DNA sequences [61]. ASAP adopts a similar measure for quality—namely, preserving significant deviations in time series—but is empirically more efficient than the exhaustive approach described in the study.

Stream Processing. To enable efficient execution, ASAP is architected as a streaming operator and adapts stream processing techniques. As such, ASAP is compatible with and draws inspiration from the rich existing systems literature on architectures for combining signal processing and stream processing functionality [30, 36].

Specifically, aggregation over sliding windows has been widely recognized as a core operator over data streams. Sliding window semantics and efficient incremental maintenance techniques have been well-studied in the literature [15, 62]. ASAP adopts the sliding window aggregation model. However, instead of leaving users to select a window manually, in the parlance of machine learning, ASAP performs *hyperparameter tuning* [43] to automatically select a window that delivers smoothed plots that help improve end user's perception of long-term deviations in time series. We are unaware of any existing system—in production or in the literature—that performs this hyperparameter selection for smoothing time series plots. Thus, the primary challenge we address in this paper is efficiently and effectively performing this tuning via visualization-specific optimizations that leverage target display resolution, the periodicity of the signal, and on-demand updates informed by the limits of human perception.

7. CONCLUSIONS

In this paper, we introduced ASAP, a new data visualization operator that automatically smooths time series to reduce noise, prioritizing user attention towards systematic deviations in visualizations. We demonstrated that ASAP's target metrics—roughness and kurtosis—produce visualizations that enable users to quickly and accurately identify deviations in time series. We also introduced three optimizations—autocorrelation-based search, pixel-aware preaggregation and on-demand update—that provide order-of-magnitude speedups over alternatives without compromising quality. Looking forward, we are interested in further improving ASAP's scalability and in futher integrating ASAP with advanced analytics tasks including time series classification and alerting.

Acknowledgements

We thank the many members of the Stanford InfoLab, the Monitorama community, and Maneesh Agrawala for their valuable feedback on this work. This research was supported in part by affiliate members and other supporters of the Stanford DAWN project—Intel, Microsoft, Teradata, and VMware—as well as the Intel/NSF CPS Security grant #1505728, the Secure Internet of Things Project, the Stanford Data Science Initiative, and industrial gifts and support from Toyota Research Institute, Juniper Networks, Visa, Keysight, Hitachi, Facebook, Northrop Grumman, NetApp, and Google.

8. REFERENCES

[1] Amazon CloudWatch. https://aws.amazon.com/cloudwatch/.

[2] Datadog. https://www.datadoghq.com/.

[3] Ganglia Monitoring System. http://ganglia.info/.

[4] Google Stackdriver. https://cloud.google.com/stackdriver/.

[5] Graphite. https://graphiteapp.org/.

[6] InfluxDB. https://docs.influxdata.com/influxdb/.

[7] Microsoft Azure Monitor. https://docs.microsoft.com/azure/monitoring-and-diagnostics.

[8] New Relic. https://newrelic.com/.

[9] OpenTSDB. http://opentsdb.net/.

[10] Prometheus. https://prometheus.io/.

[11] CHAPTER 15 - moving average filters. In S. W. Smith, editor, *Digital Signal Processing*. 2003.

[12] R. Agrawal, K.-I. Lin, H. S. Sawhney, and K. Shim. Fast similarity search in the presence of noise, scaling, and translation in time-series databases. In *VLDB*, pages 490–501, 1995.

[13] W. Aigner, S. Miksch, H. Schumann, and C. Tominski. *Visualization of time-oriented data*. Springer, 2011.

[14] M. I. Ali et al. Citybench: A configurable benchmark to evaluate rsp engines using smart city datasets. In *ISWC*, pages 374–389, 2015.

[15] A. Arasu and J. Widom. Resource sharing in continuous sliding-window aggregates. In *VLDB*, pages 336–347, 2004.

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

[17] P. Bailis, E. Gan, et al. MacroBase: Prioritizing attention in fast data. In *SIGMOD*, pages 541–556, 2017.

[18] P. Bailis, E. Gan, K. Rong, and S. Suri. Prioritizing attention in fast data: Challenges and opportunities. In *CIDR*, 2017.

[19] B. Beyer, C. Jones, et al., editors. *Site Reliability Engineering: How Google Runs Production Systems*. O'Reilly, 2016.

[20] C. Chatfield. *The Analysis of Time Series: An Introduction, Sixth Edition*. 2016.

[21] J. Chen, J. Benesty, et al. New insights into the noise reduction wiener filter. *TASLP*, pages 1218–1234, 2006.

[22] N. Cressie. *Statistics for spatial data*. 1993.

[23] I. Daubechies. The wavelet transform, time-frequency localization and signal analysis. *IEEE Transactions on Information Theory*, 1990.

[24] M. de Oliveira and H. Levkowitz. From visual data exploration to visual data mining: A survey. *TVCG*, pages 378–394, 2003.

[25] L. T. DeCarlo. On the meaning and use of kurtosis. *Psychological methods*, 2(3):292, 1997.

[26] D. Douglas and T. Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica*, 1973.

[27] A. N. Eugene Wu. Towards perception-aware interactive data visualization systems. In *DSIA*, 2015.

[28] T.-c. Fu. A review on time series data mining. *Engineering Applications of Artificial Intelligence*, 24(1):164–181, 2011.

[29] T.-c. Fu, F.-l. Chung, R. Luk, and C.-m. Ng. Representing financial time series based on data point importance. *Engineering Applications of Artificial Intelligence*, pages 277 – 300, 2008.

[30] L. Girod, K. Jamieson, et al. Wavescope: a signal-oriented data stream management system. In *SenSys*, pages 421–422, 2006.

[31] H. Hochheiser and B. Shneiderman. Dynamic query tools for time series data sets: timebox widgets for interactive exploration. *Information Visualization*, pages 1–18, 2004.

[32] M. Httermann. *DevOps for developers*. Apress, 2012.

[33] V. Hulusić, G. Czanner, et al. Investigation of the beat rate effect on frame rate for animated content. In *SCCG*, pages 151–159, 2009.

[34] R. Hyndman. Time series data library. http://data.is/TSDLdemo.

[35] U. Jugel, Z. Jerzak, and other. M4: A visualization-oriented time series data aggregation. In *VLDB*, pages 797–808, 2014.

[36] Y. Katsis, Y. Freund, and Y. Papakonstantinou. Combining databases and signal processing in plato. In *CIDR*, 2015.

[37] E. Keogh et al. Dimensionality reduction for fast similarity search in large time series databases. *KAIS*, pages 263–286, 2001.

[38] E. Keogh et al. Finding surprising patterns in a time series database in linear time and space. In *KDD*, pages 550–556, 2002.

[39] E. Keogh, J. Lin, and A. Fu. HOT SAX: Efficiently finding the most unusual time series subsequence. In *ICDM*, pages 226–233, 2005.

[40] E. Kreyszig. *Advanced Engineering Mathematics*. Wiley, NY, fourth edition, 1979.

[41] A. Lavin and S. Ahmad. Evaluating real-time anomaly detection algorithms – the numenta anomaly benchmark. In *IEEE ICMLA*, pages 38–44, 2015.

[42] J. Li et al. No pane, no gain: Efficient evaluation of sliding-window aggregates over data streams. *SIGMOD Rec.*, pages 39–44, 2005.

[43] L. Li, K. Jamieson, et al. Hyperband: A novel bandit-based approach to hyperparameter optimization. *arXiv:1603.06560*, 2016.

[44] T. W. Liao. Clustering of time series data: a survey. *Pattern Recognition*, pages 1857–1874, 2005.

[45] M. Lichman. UCI machine learning repository, 2013. Accessed 19-Aug-2016.

[46] J. Lin, E. Keogh, et al. Visually mining and monitoring massive time series. In *KDD*, pages 460–469, 2004.

[47] J. Mackinlay, P. Hanrahan, and C. Stolte. Show me: Automatic presentation for visual analysis. *TVCG*, pages 1137–1144, 2007.

[48] J. S. Marron. Automatic smoothing parameter selection: A survey. *Empirical Economics*, 13(3):187–208, 1988.

[49] M. Nikulin. Excess coefficient. In M. Hazewinkel, editor, *Encyclopedia of Mathematics*. 2002.

[50] A. Parameswaran et al. SeeDB: Visualizing database queries efficiently. In *VLDB*, pages 325–328, 2013.

[51] T. Pelkonen et al. Gorilla: A fast, scalable, in-memory time series database. In *VLDB*, pages 1816–1827, 2015.

[52] W. H. Press, S. A. Teukolsky, et al. *Numerical Recipes in C (2nd Ed.): The Art of Scientific Computing*. 1992.

[53] K. Reumann and A. P. M. Witkam. Optimizing curve segmentation in computer graphics. *ICS*, 1974.

[54] K. Rong and P. Bailis. ASAP: Prioritizing attention via time series smoothing (extended version). *arXiv:1703.00983*, 2017.

[55] S. Salvador and P. Chan. Determining the number of clusters/segments in hierarchical clustering/segmentation algorithms. *Tools with Artificial Intelligence*, pages 576–584, 2004.

[56] A. Savitzky and M. J. E. Golay. Smoothing and differentiation of data by simplified least squares procedures. *Analytical Chemistry*, 1964.

[57] W. Shi and C. Cheung. Performance evaluation of line simplification algorithms for vector generalization. *The Cartographic Journal*, pages 27–44, 2006.

[58] R. H. Shumway and D. S. Stoffer. *Time Series Analysis and Its Applications*. Springer, 2005.

[59] T. Siddiqui, A. Kim, J. Lee, K. Karahalios, and A. Parameswaran. Effortless visual data exploration with zenvisage: An interactive and expressive visual analytics system. In *VLDB*, pages 457 – 468, 2017.

[60] J. O. Smith. *Spectral Audio Signal Processing*. 2011.

[61] F. Tajima. Determination of window size for analyzing dna sequences. *Journal of Molecular Evolution*, pages 470–473, 1991.

[62] K. Tangwongsan et al. General incremental sliding-window aggregation. In *VLDB*, pages 702–713, 2015.

[63] C. Tomasi and R. Manduchi. Bilateral filtering for gray and color images. In *ICCV*, pages 839–846, 1998.

[64] M. Visvalingam and J. D. Whyatt. Line generalisation by repeated elimination of points. *The Cartographic Journal*, pages 46–51, 1993.

[65] S. Weart. The carbon dioxide greenhouse effect. *The Discovery of Global Warming. American Institute of Physics.*

[66] P. H. Westfall. Kurtosis as Peakedness, 1905–2014. RIP. *The American Statistician*, pages 191–195, 2014.

[67] K. Wongsuphasawat et al. Voyager: Exploratory analysis via faceted browsing of visualization recommendations. *TVCG*, pages 649–658, 2016.

[68] 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/.

[69] E. Wu, L. Battle, and S. R. Madden. The case for data visualization management systems: vision paper. In *VLDB*, pages 903–906, 2014.

[70] Y. Zhu and D. Shasha. Efficient elastic burst detection in data streams. In *KDD*, pages 336–345, 2003.