ASAP: Prioritizing Attention via Time Series Smoothing

YUTAO,WANG
student number:1023041136
{nanjing}@njupt.School of Cyberspace Security, Nanjing University of Posts and Telecommunications

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 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 , Ganglia , Graphite , OpenTSDB , Prometheus , and Facebook Gorilla , to cloud services including DataDog , New Relic , AWS CloudWatch , Google Stackdriver , and Microsoft Azure Monitor . 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 .

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 data: an hourly average of the number of NYC taxi passengers over 75 days in 2014 . Daily fluctuations of taxi volume dominate the

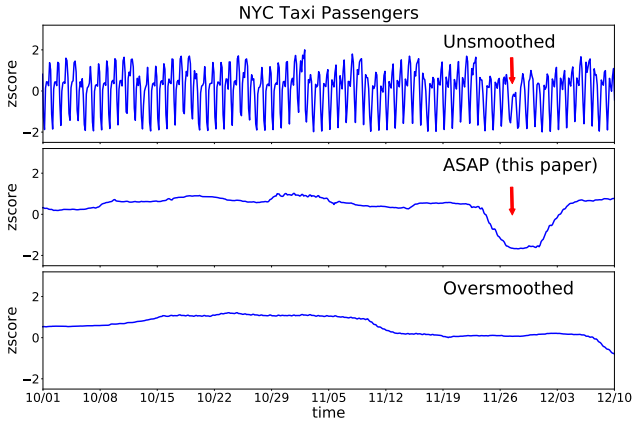

Figure 1: Normalized number of NYC taxi passengers over 10 weeks.[1] From top to bottom, the three plots show the hourly average (unsmoothed), the weekly average (smoothed) and the monthly average (oversmoothed) of the same time series. The arrows point to the week of Thanksgiving (11/27), when the number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

Proceedings of the VLDB Endowment, Vol. 10, No. 11

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

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* , 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* —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.,), 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 .

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 , 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 **??** presents ASAP's search strategy, optimizations, and streaming execution mode. Section **??** evaluates ASAP's visualization quality through two user studies, and ASAP's performance on a range of synthetic and real-world time series. Section **??** discusses related work, and Section **??** 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 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

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

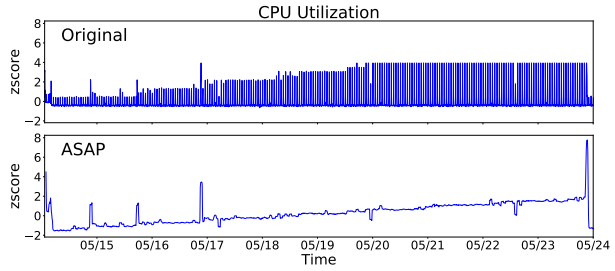

Figure 2: Server CPU usage across a cluster over ten days , visualized via a 5 minute average (raw) and an hourly average (via ASAP). The CPU usage spike around May 24th is obscured by frequent fluctuations in the raw time series.

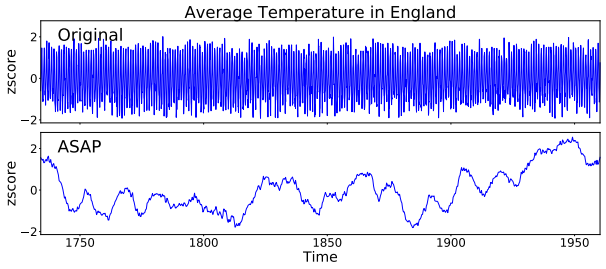

Figure 3: Temperature in England from 1723 to 1970 , visualized via a monthly average (raw) and 23-year average (via ASAP). Fluctuations in the raw time series obscure the overall trend.

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 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: 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 (, 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 , 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 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, 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 [**?**] 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* [**?**], a commonly-used measure in spatial statistics (especially geostatistics) that characterizes the spatial continuity (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

[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.

[4]This is not a theoretical example; in fact, the site from which we obtained this data 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.

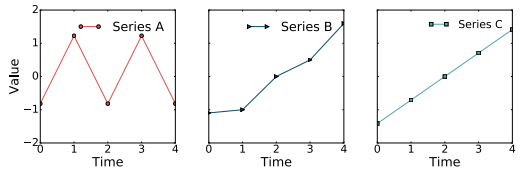

Figure 4: Three time series that appear visually distinct yet all have mean of zero and standard deviation of one. This example illustrates that standard summary statistics such as mean and standard deviation can fail to capture the visual "smoothness" of time series.

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.

4. REFERENCES

[1] K. Barnard, "Practical Color Constancy," PhD Thesis, School of Computing, Simon Fraser University, Burnaby, Canada, 1999.

[2] S. Bianco, G. Ciocca, C. Cusano, and R. Schettini, "Automatic Color Constancy Algorithm Selection and Combination," *Pattern Recognition*, vol. 43, no. 3, pp. 695–705, March 2010.

[3] Y. Boykov and V. Kolmogorov, "An Experimental Comparison of Min-Cut/Max-Flow Algorithms for Energy Minimization in Vision," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 9, pp. 1124–1137, September 2004.

[4] Y. Boykov, O. Veksler, and R. Zabih, "Efficient Approximate Energy Minimization via Graph Cuts," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, no. 12, pp. 1222–1239, November 2001.

[5] D. H. Brainard and W. T. Freeman, "Bayesian Color Constancy," *Journal of the Optical Society of America A*, vol. 14, no. 7, pp. 1393–1411, 1997.

[6] G. Buchsbaum, "A Spatial Processor Model for Color Perception," *Journal of the Franklin Institute*, vol. 310, no. 1, pp. 1–26, July 1980.

[7] M. Ebner, "Color Constancy Using Local Color Shifts," in *European Conference in Computer Vision*, pp. 276–287, Springer-Verlag, 2004.

[8] M. Ebner, *Color Constancy*, John Wiley & Sons, Chichester, England, 2007.

[9] G. D. Finlayson and S. Hordley, "Improving Gamut Mapping Color Constancy," *IEEE Transactions on Image Processing*, vol. 9, no. 10, pp. 1774–1783, October 2000.

[10] D. Forsyth, "A Novel Algorithm for Color Constancy," *International Journal of Computer Vision*, vol. 5, no. 1, pp. 5–36, 1990.

5. REFERENCES