VAStream: A Visual Analytics System for Fast Data Streams

Satya Katragadda[1], Raju Gottumukkala[1], Siva Venna[1], Nicholas Lipari[1], Shailendra Gaikwad[1], Murali Pusala[1], Jian Chen[2], Christoph W. Borst[1], Vijay Raghavan[1], Magdy Bayoumi[1]
[1]NSF Center for Visual and Decision Informatics, University of Louisiana at Lafayette, Lafayette, Louisiana
[2]Department of Geography, University of North Alabama, Florence, Alabama
satya@louisiana.edu,raju@louisiana.edu,sxv6878@louisiana.edu,nicholas.lipari@louisiana.edu,skg6619@louisiana.edu
muralipusala@louisiana.edu,jchen18@una.edu,cxb9999@louisiana.edu,raghavan@louisiana.edu,magdy@louisiana.edu

ABSTRACT

Processing high-volume, high-velocity data streams is an important big data problem in many sciences, engineering, and technology domains. There are many open-source distributed stream processing and cloud platforms that offer low-latency stream processing at scale, but the visualization and user-interaction components of these systems are limited to visualizing the outcome of stream processing results. Visual analysis represents a new form of analysis where the user has more control and interactive capabilities either to dynamically change the visualization, analytics or data management processes. VAStream provides an environment for big data stream processing along with interactive visualization capabilities. The system environment consists of hardware and software modules to optimize streaming data workflow (that includes data ingest, pre-processing, analytics, visualization, and collaboration components). The system environment is evaluated for two real-time streaming applications. The real-time event detection using social media streams uses text data arriving from sources such as Twitter to detect emerging events of interest. The real-time river sensor network analysis project uses unsupervised classification methods to classify sensor network streams arriving from the US river network to detect water quality problems. We discuss implementation details and provide performance comparison results of various individual stream processing operations for both stream processing applications.

KEYWORDS

Visual analytics, big data infrastructure, stream computing, machine learning, visualization

1 INTRODUCTION

Streaming data from sensors, video and social media streams is becoming increasingly common in many science, engineering and industry segments especially with the prevalence of both low-cost sensors and internet connectivity [1–3]. These high-velocity data streams should be processed in near-real time. These streams may also change with respect to the change in environment (i.e. volatility). There are several stream processing architectures and commercial cloud platforms such as Amazon Kinesis™, Google Data Flow™, and Microsoft Azure™ stream analytics that provide interfaces for users to both build and efficiently manage stream processing workflows. There are still three main challenges from in dealing with these data streams at the infrastructure (hardware and software) level: 1) How to reduce the latency while achieving high-throughput for end-to-end processing from data consumption to visualization (2) How to adapt to changing workloads or failures, and also (3) How to provide flexibility in the infrastructure to adapt to changing nature of data or user demands.

Visual analysis represents a form of analysis where the user has more control over the visualization, analytics or data management processes [4]. Achieving desirable performance for streaming applications in terms of accuracy, latency or efficiency requires optimization of many components including data engineering, machine learning techniques, and distributed processing components [5–8]. A typical end-to-end pipeline for processing streaming data includes pre-processing (to transform noisy data to clean data), analysis (that includes statistical or machine learning techniques or numerical simulations), storage, visualization, and user interaction components. Many of these processes may be performed in a distributed environment where the computational workload is shared among several nodes [9–11]. There are many programming frameworks such as Spark, Flink, Storm that facilitate in-memory processing of various stream processing tasks [1, 12, 13]. These existing systems offer low-latency stream processing at scale to extract relevant information or knowledge from the data streams that are presented to the user for visualization. Achieving end-to-end latency while maintaining throughput still remains a challenge. Also, many stream processing frameworks offer some adaptivity in terms of changing workloads or failures at the infrastructure level or application level. When dealing with streaming workloads multi-dimensional and multi-relational (e.g: graph streams) data, how to provide fault-tolerance and adaptivity remains a major research challenge. Finally, with data streams there is volatility (i.e. change in the nature of data). This volatility could be due to change in data sources or change in data (e.g. change in distribution such as concept drift) that requires changing models or adjusting the parameters of the model. Providing more control to data scientists in terms of the ability to change the data sources, processing techniques, adjusting analytics or machine learning models and visualizing techniques will help address some of the challenges. For example, sampling data can have a huge impact on both the model performance (in terms of accuracy), and computational performance (i.e. latency, throughput), and resource cost. Similarly choosing a model that needs to be updated frequently (i.e. a regression versus deep-learning model) or choosing features have similar effect on both model and computational performance.

In this paper, we present the design, system architecture, and performance results of VAStream: a visual analytics sandbox environment for processing high-volume, high-velocity data streams. The system is designed to support multiple big data projects as part of the NSF Center for Visual and Decision Informatics [14]. We

DOI: 10.1145/3332186.3332256

Table 1: Hardware Configuration for VAStream

Node Type	Nodes	RAM	SSD (PCIe)	HDD	GPU
Visualization	1	128 GB	3.2 TB	2 TB	P6000
Analytics	1	448 GB	3.2 TB	2 TB	2x P100
Streaming	6	256 GB	3.2 TB	2 TB	P100
Management	3	128 GB	N/A	10.8 TB	N/A
Storage	1	32 GB	N/A	192 TB	N/A
Total	**12**	**2272 GB**	**25.6 TB**	**218 TB**	**8xP100 1xP6000**

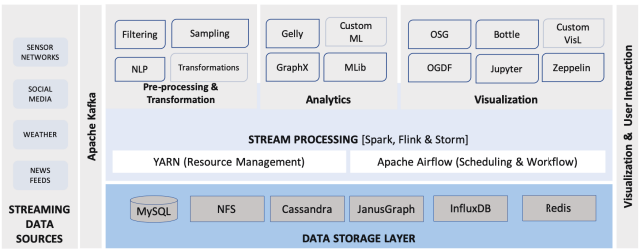


Figure 1: Software Stack on VAStream

present the overall hardware architecture, software stack, and cluster management infrastructure to support stream processing and visualization workflows. We discuss two different projects within the context of this framework - namely event detection from social media streams and classification of US river network streams. We also discuss the different analysis and visualization tasks from the user's side inspired by the Manifold framework [15] and present the performance measurements for these projects.

2 VASTREAM SYSTEM ARCHITECTURE

The goal of the VAStream system is to support (1) subscription to multiple live data streams, (2) distributed in-memory stream processing and analysis, (3) analytics and machine learning on data streams, (4) visualization and user interaction across multiple devices (i.e. browsers, Virtual Reality (VR) and multi-touch devices), (5) cluster management tools to support monitoring, resource management and interactive capabilities for users to change visualization, analysis techniques (querying, pattern matching or unsupervised machine learning) or data pre-processing modules. The overall system architecture consists of both hardware and software to support in-memory stream processing, analytics, visualization and interaction with multiple devices.

2.1 Hardware

The instrument's hardware is customized to support high-performance stream processing, analytics operations, fast machine learning and visualization of both static and continuous data streams. The major design goals of the system are driven by the performance demands of various ongoing big data related projects that fall in the visual analytics category, and also the need for some customizability on the system to run multiple analytics workloads while keeping a standardized architecture.

The cluster consists of 11 dual socket Intel Broadwell based nodes. These nodes are categorized as streaming, analytics, visualization, and infrastructure nodes based on some customization of memory or GPU's. The streaming nodes are designed to perform basic data cleaning and feature extraction on streaming data in near-real time. The analytics node is equipped to handle large in-memory data and trained models. The visualization node handles the real-time rendering of visualizations based on the outputs generated from the analytics node. Finally, the infrastructure nodes are dedicated to handle the basic operation of the cluster and are responsible for the resource management of the cluster.

All the nodes use Dell PE r730/xd motherboard and have 2x14-core 2.4 GHz processor with DDR4-2440 RAM with 32Kb L2 cache and 35MB (1.9 MB per core). The systems have NVMe SSD as their primary storage to support low-latency data management operations. There are 8 P100 GPU's in total to support fast machine learning for multiple users. Each of the GPU contains 3584 cores and delivers over 9.3 TFLOPS of FP16 processing power. 2 of the GPU's are placed in the analytics node to support fast graph traversal and deep learning. The analytics node has 448GB RAM to support low-latency data access. The visualization node is equipped with a P6000 GPU for fast rendering and visual layout generation. There is also NFS storage to store large data sets. The NFS is primarily used to store data for offline analysis rather than stream-processing. The nodes are connected with a 10Gbps Ethernet connection. The system is connected to the university's science DMZ and has 40G connectivity. Table 1 provides a the hardware of VAStream.

2.2 Software

Figure 1 shows the different components of VAStream software stack. We used the standard Cloudera stack to further customize the system [16]. Cloudera provides many tools and packages to manage a big data cluster. Cloudera Manager provides basic functionality to configure the cluster, monitor the resources, jobs and health of the system, track and manage resources, and diagnose any issues with the cluster [17]. We customized our cluster configuration to be suitable for stream processing. First, the HDFS file system is not used for stream processing as this adds additional latency. Second, several open source data management tools were installed in the data storage layer to serve each of the stream processing, analysis and visualization applications. Third, we developed in-house libraries to support stream processing, analytics, visualization and user interaction across multiple device.

The streaming data is originally served by multiple *streaming data sources* through RSS or JSON feeds which are aggregated by Kafka before being sent to VAStream. The VAStream system could consume multiple data streams into the data ingestion layer. The type of data that is currently consumed include weather, river network, live video, social media streams. But this could potentially consume multiple types of data streams relevant to infrastructures, traffic, and disasters. Kafka provides basic light weight stream processing such as filtering and aggregation on these streams. For example, one may want to create a new tuple by joining two different streams such as rainfall data and water stage sensor data from USGS.

The system supports multiple *stream processing frameworks*, namely Spark, Flink and Storm [1, 12, 13]. There are two categories of streaming, namely native streaming that does continuous processing of streams as they arrive and processing data in micro-batchs. Native streaming is typically applied for event-based processing, and microbatch-based processing involves processing few records every few seconds. Each streaming framework has its own advantages and offer trade-offs in terms of performance and features [2]. Spark is micro-batch based streaming, whereas Storm and Flink are native stream-processing frameworks. The primary advantage of Spark is wide-spread adoption and integration with several machine learning packages and data management tools. Cloudera has the ability to manage multiple streaming jobs at the same time. Our use cases are limited to Spark streaming in this paper. The streaming applications use various pre-processing & transformation libraries for data transformations prior to analysis. For example, transforming Twitter streams to features required for machine learning models (applied for sentiment analysis or event detection) requires human-written text to be translated to machine readable form. Similarly, streaming data, whether it is time-series data or video data, requires libraries for filtering, sampling, and transformations. The Analytics packages on VAStream has various machine learning libraries. MLib for instance has several machine learning algorithms such as classification, regression, decision trees, clustering, etc. and is the most popular machine learning package for Spark. Many of our ongoing projects are graph-based. GraphX and Gelly provide graph processing on streaming graphs for Spark and Flink respectively. There are also several visualization libraries for graph-layout, VR-based visualization and multi-touch interaction. In addition to the existing machine learning libraries, VAStream has custom-built general purpose pre-processing, analysis, and visualization applications that can be used by the user.

The *Data storage layer* provides storage for low latency streaming and visualization pipelines. The in-memory data-storage is handled through a combination of Redis, Cassandra, and HBase [18–20]. In addition to these databases, VAStream also handles specialized databases like JanusGraph and InfluxDB to handle graph data and time-series data [21, 22]. These databases enable us to store the data at different stages of the stream processing pipeline (raw data, intermediary data, and outcomes from analysis). They support queries on both historical and streaming data.

The system also has various visualization libraries to support graph layout and graph sampling and summarization. These include OpenSceneGraph (3D graphics API), OGDF (graph drawing) and Python Bottle (a micro web-framework). Also, the *visualization and user interaction* component is handled both through existing tools (i.e. Hue and Zepplin), and custom built user interaction modules.

All the software and frameworks mentioned in this section can be integrated into the user's workflow using Apache Airflow. Airflow is a platform to programmatically author, schedule, and monitor workflows. The user can execute an existing workflow or design their own workflow based on his/her requirement. Figure 2 displays a workflow designed using airflow to detect events from social media streams. This capability enables the user to interact with the data and modify various modules in the data analysis pipeline. However, the direct acyclic graph (DAG) based work-flow scheduling does not strictly work with streaming data, so the job is scheduled to run at regular intervals as a batch job.

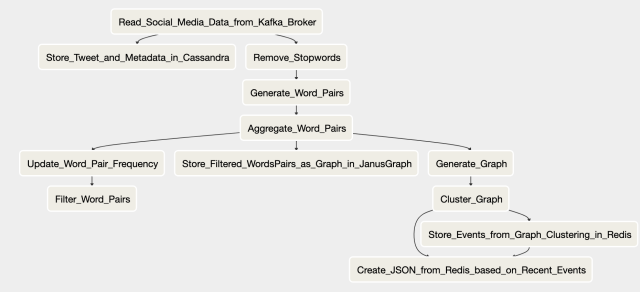


Figure 2: Workflow designed in Apache Airflow to detect events from social media data

Cloudera manager provides the monitoring of the server resources and deployment of various packages. Job scheduling and resource management is handled through YARN. A user's job is scheduled to run in the VAStream environment if there are a minimum amount of resources available. If the required resources are not available the job is added to the scheduler and is executed when the resources are available.

3 VISUAL ANALYTICS ON DATA STREAMS

VAStream system supports continuous and low-latency end-to-end processing of real-time streams where the outcomes of stream processing are pushed to a visualization system. There are three main drivers for the system to adopt visual analytics approach for data streams. One, the ongoing research projects at the NSF Center for Visual and Decision Informatics that have various visual analytics components. Second, most of the users of the system are advanced users i.e. students and researchers working in data science and visualization – so the system serves as an experimental platform. The third reason, which is the most important driver is the nature of streaming data – which is not only high-volume and high-velocity but highly volatility. This means the uncertainty of use and availability of new data sources, the inherent nature of many sensor streams that change distribution or properties and decisions on data storage and management are primarily user driven. The Visual analytics system should provide the user flexibility to interact, explore, and incorporate the user's knowledge into the analysis though an interface. Figure 3 provides an overview of a typical visual analytics workflow on data streams using various components in the VAStream.

The input data is gathered from real-time data streams through Apache Kafka. This input data is then ingested into the processing pipeline. Once, ingested raw data is cleaned and meaningful features are extracted from the cleaned data. These features are then passed along to the analytics module to extract knowledge based on supervised or unsupervised algorithms. This knowledge is usually an outcome, like detected events, anomalies etc. These outcomes are stored in an in-memory database to serve the visualizations to the user.

A copy of data is stored in a database for post-hoc analysis. This database is usually disk based to handle large amount of data.

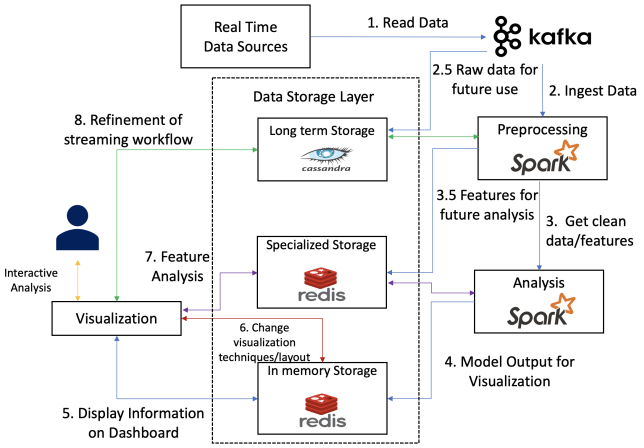


Figure 3: A representation of a typical visual analytics workflow

The intermediary data like features that are generated by the pre-processing module are also stored in a temporary in-memory database. All the data in this database is moved to longer term disk-based storage after some time, which means that the data is no longer required for further analysis.

In Figure 3, the arrows in blue (1 - 5) represent the traditional streaming workflow. Red arrows (6) represent the inspection or exploration phase of the visual analytics process. The arrows in purple (7) represent the exploration or reasoning phase. Finally, the green arrows (8) represent the refinement portion of the visual analytics process. All the latter processes are initiated by the user and are executed as batch jobs. The idea of inspection, explanation, and refinement phases is based on the manifold framework by Zhang et al. [15].

In the next section, we explain two different applications that are built using the workflow template shown in Figure 3.

4 USE CASES

We discuss the implementation details and performance measurements of two streaming application on VAStream. The social media based event detection application uses unstructured text arriving from Twitter to detect emerging events. The sensor stream classification application uses sensor streams arriving from US river network to analyze water quality. We discuss the users design tasks, techniques and tools used for pre-processing, analysis and visualization along with performance. We generate representative data streams to analyze the performance of both the applications with respect to their total execution time for various tasks.

4.1 Event Detection from Social Media Streams

Event detection from social media streams refers to extracting actual real-world events from social media conversations. Event detection methods automatically find emerging topics of interest based on these conversations without requiring keyword searches. This is very useful for situations such as public safety incidents or mass emergencies where events are rather unpredictable. One approach to model event detection is using a word co-occurrence graph. In a word co-occurrence graph, nodes represent words and a link between two words appearing in a single Tweet [23]. Real-time event detection on real-time social media streams require construction of a co-occurrence graph for every micro-batch of data. A typical micro-batch of one-minute data from Twitter Firehose contains 300K tweets. The resulting graph has around 4.5 million nodes and 24 million edges on average. One of the challenges with event detection methods is the volatility of data both in terms of changing content and noise. With the right user intervention, this noise can be further minimized by adjusting the parameters to filter important word pairs. Providing the right user interaction tools can allow the user to evaluate bias vs. the natural distribution of data.

Table 2 summarizes the different user interaction tasks for the event detection module based on the Manifold framework [15]. The investigation phase is for the end user of the event detection application that is interested in the output of the event detection model. The investigation phase includes (a) map layout that shows various active events, their location and the keywords associated with the event, (b) a force-directed network layout of word co-occurrence graph for an individual event, and (c) a Sankey diagram that displays the evolution of an event over time. Figure 4 provides an overview of visualizations during the investigation phase.

In the explanation phase, the user is trying to understand the influence of various features that are contributing to the output of models. For example, the user may notice that the events are not corresponding to actual ground-truth, which could be due to noise. Here the user views a simple bar chart to understand the frequency distribution of various word pairs. In this specific example, the bar chart helps understand that the parameter thresholds need to be changed. Figure 5 provides a sample visualization of word-pair frequency distributions for event detection from social media streams.

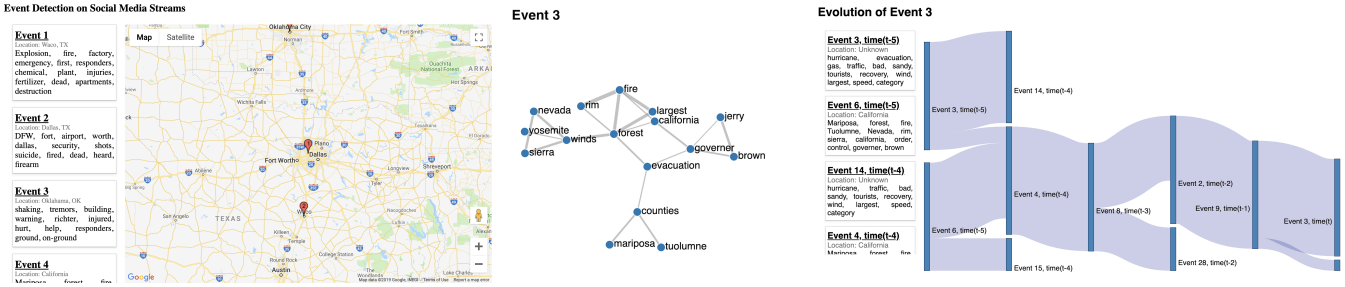
Finally, in the refinement stage, the user adjusts the streaming workflow that includes adjusting the data source, features, or changing the model. This phase may involve adjusting the parameter thresholds, changing the sampling rate, or tapping to a more reliable data source. Figure 6 shows visualization on a multi-touch display and mobile devices. More information about multi-touch visualization of this data is provided in [24].

4.1.1 Results. The experiments are conducted using a simulated twitter steam that generated about 300k tweets per minute. The streams are collected in one-minute intervals. The execution time is measured from the time when the program starts to retrieve the input data from the database or Kafka broker to the time the output is rendered on the browser. Figure 7 shows the execution time for each of the retrieval, pre-processing, analysis, and visualization modules and the total time to accomplish the tasks presented in Table 2. The latency for each module is computed by executing each independent as a separate task. The input data for pre-processing task is read from Kafka, and the output (if any) is written back to Kafka. The latency of each task is computed between the time when the input data is read from the Kafka broker to the time when the data is written back to the Kafka broker.

Task 1 represents the original event detection workflow. The original event detection takes 56.1 seconds to process 1-minute worth of data. Pre-processing and analysis modules are the most

Table 2: Details about goals, design tasks, pre-processing, analysis, visualization, and data sources for event detection on social media streams

Goal	low level design task	Source	Pre-processing	Analysis	Storage	Visualization
Inspection	T1. Display emerging events	Kafka	stop word removal word pair generation word pair filtering	graph generation graph clustering text filtering	Cassandra	map layout
	T1.1 select/filter events based on location/keywords	Redis				map layout
	T2. Display event evolution	Redis	graph generation	graph similarity graph evolution		Sankey diagram
	T2.1 Filter by time period	Redis	graph generation	graph similarity		force layout
Explanation	T3. Display frequency distribution of word pairs	Redis		binning		bar chart
	T4. Display change in frequency of active nodes	Redis		binning		bar chart
Refinement	T5. Update parameters to improve event detection accuracy	Cassandra	stop word removal word pair generation word pair filtering	graph generation graph clustering		map layout



(a) Map interface of events (b) Force layout of an event (c) Sankey diagram over previous 5 time periods

Figure 4: Visualizations for investigation phase in event detection from social media streams

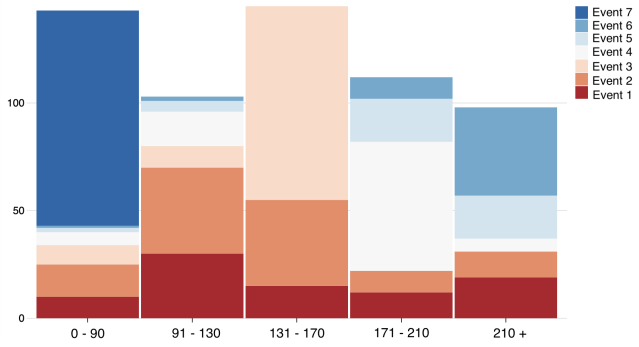


Figure 5: Word pair frequency distribution during the explanation phase of event detection from social media streams

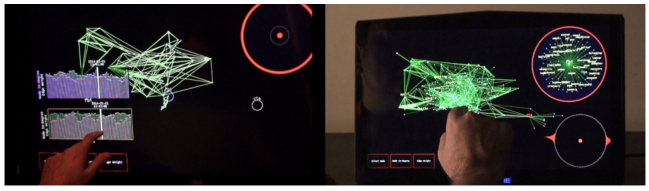


Figure 6: Various Interactions and Visualizations on the Multi touch interaction from the detected events

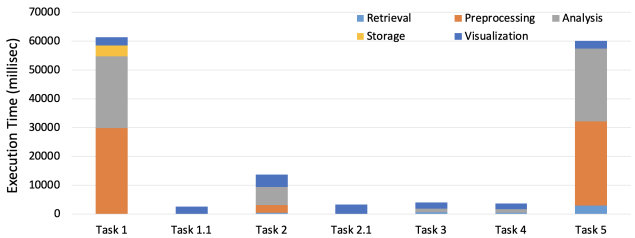


Figure 7: Break down of time taken to each module for each goal for event detection on social media streams

expensive operations with 29.74 seconds for pre-processing and 24.95 seconds for analysis. The pre-processing module involves stop word removal, word pair generation, and word pair filtering. All the word pairs need to be aggregated to filter them based on divergence score which is resource intensive and scales with the increase in the size of the data. The analysis phase involves graph generation and graph clustering. Graph generation is an expensive process compared to graph clustering with 78% of the time taken by graph generation and 22% of the time for graph clustering. The visualization of the events detected during each time-period takes

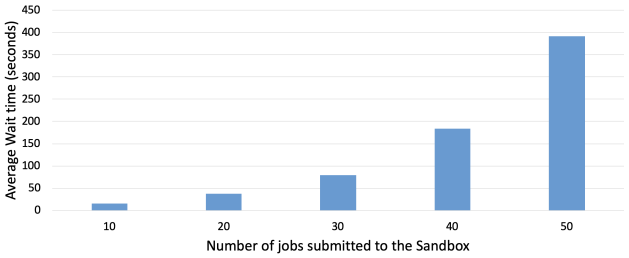


Figure 8: Average wait time while executing multiple jobs on VAStream

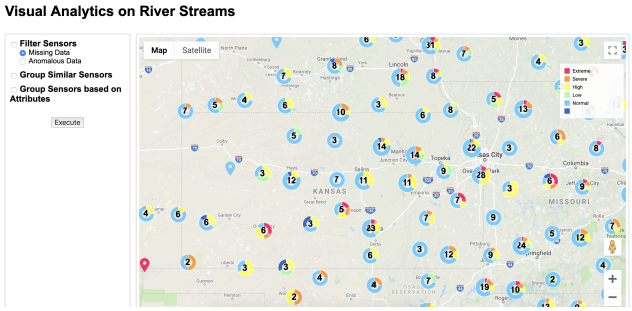


Figure 9: Dashboard for classification of river network streams

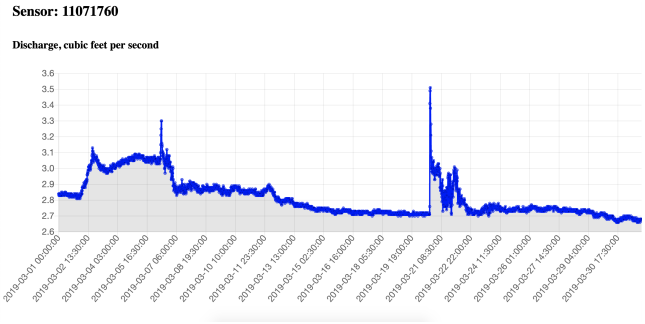


Figure 10: A time series display of attribute of a single sensor

about 2.8 seconds on average. The average number of events generated during each time-period is 27. Each event has 12 nodes and 29 edges on average. Task 5 follows a similar workflow to Task 1 and takes 56.8 seconds to execute. The time taken to retrieve tweet data from Cassandra is 3.9 seconds. The data is stored in a NVMe SSD which has lower retrieval time compared to traditional HDD.

Task 2 takes 0.38 seconds to retrieve all the events from the previous 5 time periods from Redis datastore. The graphs are then constructed during the pre-processing phase which takes 2.7 seconds on average. These graphs are then analyzed to identify similar graphs from consecutive time-periods. The total number of graph comparisons are about 2917 on average. An evolution timeline is generated based on birth, death, merge, and split operations on these events. This evolution timeline is rendered as Sankey diagram. The analysis and visualization tasks take 6.2 and 3.1 seconds respectively. In total, the time to compute the evolution of an event takes 14.3 seconds. The user can also the most similar event from the previous time period, which is usually rendered using a force layout. The time to render both the events is 3.2 seconds.

The total time taken to achieve tasks 3 and 4 are 4.2 and 3.9 seconds respectively. Task 3 identifies the word-pair data from each event and retrieves the frequency of the word pairs from Redis data store. The data is then binned and displayed to the user. The binning the word pairs takes 1.3 seconds. Similarly binning the nodes for Task 5 takes about 1.2 seconds.

We also analyzed the wait time of multiple visual analytics workflows to evaluate how simultaneous execution of multiple stream processing workflows affect total execution time. Various combination of tasks are associated with investigation, explanation, and refining phases are used to simulate the workload. Figure 8 shows the average wait time for multiple goals on VAStream. The wait time is the time difference between the job submission and job executing times. VAStream can execute 7 visual analytics jobs simultaneously with minimum impact to end-to-end latency. The average waiting time increases to 15.3 seconds for 10 visual analytics jobs are submitted. The average wait time is 37.39 seconds for 20 visual analytics jobs submitted to VAStream. The average wait time increases to 79.55, 183.47, and 391.32 seconds as the number of jobs increases to 30, 40, and 50 respectively. The CPU utilization of the cluster ranges from 75% to 90% for all the workloads. The memory utilization ranges from 112 GB to 187GB.

4.2 Classification of River Network Streams

Classification of river network data streams enable users to perform interactive querying and analysis of real-time stream-flow status and water quality issues. USGS National Water Information system consists of 10,615 river gauge stations nationwide for inland and coastal river networks with 15-60 minute update frequency [25]. With the current and historical water data for the nation, many applications such as flood monitoring & forecasting, water resources management, and pollution control & monitoring can be done most of time at local or regional scale. For example, eutrophication, excessive richness of nutrients and minerals in a water body inducing excessive growth of algae, is a leading cause of impairment of freshwater [26]. We can classify the USGS river network sensory data streams to see where is at risk of eutrophication problem. One major challenge working with real-time sensor data is the concept drift which causes the old model no longer valid. In this application, users can determine which features are most critical in determining the status of a sensor, update the model as needed and view the status of the sensor.

We generated a representative sample data for 10 attributes (water temperature, air pressure, water depth, water discharge, flow velocity, PH, salinity, oxygen, nitrogen, and phosphorus) based on real-data collected from. We generated data at a sampling rate of 15 minutes from 10,615 sensors. The status of each sensor is determined based on supervised or unsupervised model with inputs of historical records for the same senor and real-time observations from neighboring sensors upstream and downstream.

Table 3 provides a typical interaction of a user with the visual analytics application to classify river network streams. The tasks are

Table 3: Details about goals, design tasks, pre-processing, analysis, visualization, and data sources for classification of river network streams

Goal	low level design task	Source	Pre-processing	Analysis	Storage	Visualization
Inspection	T1. Display current status of sensors T1.1 Classify using logistic regression T1.2 Classify using k-means	Kafka	fix missing data anomaly detection data normalization	logistic regression k-means clustering	InfluxDB Redis	map layout
	T2. Display historical status of sensor	Redis				time series
Explanation	T3. Display distribution of sensor readings	InfluxDB		binning		bar chart
	T4. Identify correlation between different features	InfluxDB		binning		bar chart
Refinement	T5. Update logistic regression model	InfluxDB	fix missing data anomaly detection data normalization	logistic regression		map layout
	T6. Apply k-means using selected features	InfluxDB	fix missing data anomaly detection data normalization	k-means clustering		map layout

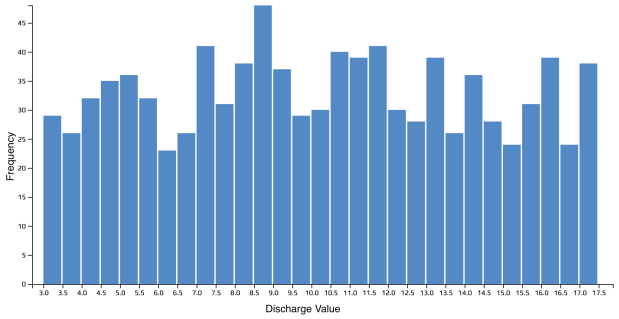


Figure 11: Binning of a specific features

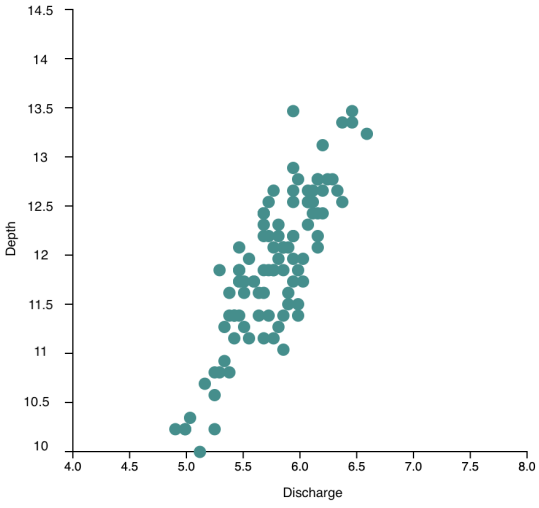


Figure 12: Correlation analysis between depth and discharge of a sensor

categorized into investigation, exploration, and refinement phases. In the investigation phase, the user explores the status of the river sensor network on a map-based interface. The interface is presented

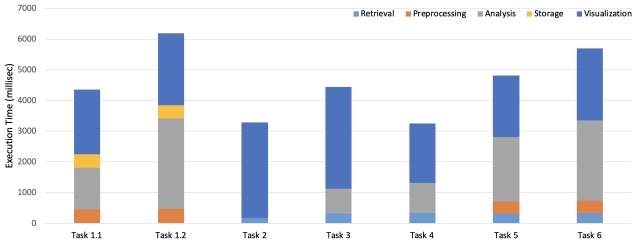


Figure 13: Break down of time taken to each module for each task for classification of river networks

in Figure 9. The user can use either the logistic regression to calculate the status of the sensor or use k-means clustering to identify similar sensors. The user can also look at the historical status of the sensor to get an additional context of the current value. A typical historical sensor status interface is shown in Figure 10. This allows the user to come up with a hypothesis if the current status of the sensors is accurate or not.

In the explanation step, the user can look at the attributes associated with the sensors and can look at the distribution of attributes (Figure 11) and correlation between different attributes (Figure 12) to determine is a feature is important for grouping or classification. In the refinement step, the user can request the model be rebuilt based on a subset or superset of features to determine if the change in model improves the classification or grouping of sensors.

4.2.1 Results. Classification of river networks application is evaluated over 30 time-periods, and the results are averaged over the 30 time-periods. The time taken is computed based on the total time taken from when the program starts to retrieve the input data from the database or Kafka broker to the completion of the visualization. The latency of each task is computed between the time when the input data is read from the Kafka broker to the time when the data is written back to the Kafka broker and are presented in Figure 13.

The total time required to read the data from the Kafka broker, clean the data, apply logistic regression on each sensor, and finally visualize the sensor status takes 2.2 seconds. The majority of the time is consumed by applying logistic regression on each sensor, which takes about 0.92 seconds. This time also includes the time taken to include updating the logistic regression if the user labels mis-classified data, if not all the data is used to train the data. The

time taken to clean the data is about .46 seconds, which includes the time taken to identify missing data, identify anomalies, and perform data normalization. In total, task 1.1 takes about 4.4 seconds. Task 1.2 takes 6.2 seconds with 3.1 seconds to perform k-means clustering on the data. The time taken to view historical values of the sensors takes 3.7 seconds with time taken to visualize the time series data at 3.1 seconds.

The time taken for tasks 3 and 4 are about 3.4 seconds and 1.9 seconds. The binning operation on one feature takes only 0.7 seconds on average. On the other hand, the time to compute correlation between two features take 0.9 seconds using spark data frames. Time taken to train a new logistic regression model for task 5 is about 1.9 seconds. The time taken to view a k-means clustering on a new set of data takes 2.4 seconds.

5 CONCLUSION AND FUTURE WORK

This paper provides design, system architecture, and performance analysis of VAStream - a visual analytics sandbox environment for processing high-volume, high-velocity data streams. The system is designed to support multiple big data projects as part of the NSF Center for Visual and Decision Informatics [15]. We present the overall hardware architecture, software stack, and cluster management infrastructure to support stream processing and visualization. The data management layer provides storage medium (in-memory, disk, etc.) and databases (text, relational data, graphs, time-series) to facilitate low-latency stream processing and visual analytics. We also use Airflow - a workflow management tool that enables the user to build, schedule, and execute custom workflows on data in real-time enhances the visual analytics process. We presented both the implementation details and performance measurements of two streaming applications on VAStream.

We encountered many opportunities to improve the performance of visual analysis on data streams. Our existing visualizations for both the applications limited to browser given the size of the dataset. However, big data visualization is computationally expensive and rendering these datasets require distributed techniques for rendering. Our existing work only covered the latency in terms of execution time. But efficient resource management and throughput analysis of heterogeneous streaming workflows still needs to be investigated. There are several streaming data sources, machine learning techniques, algorithms, and data management tools that could be integrated into VAStreams. Finally, improvements can be done with respect to managing the real-time stream processing workflows for visual analytics.

ACKNOWLEDGMENTS

This work is supported by the National Science Foundation under grant numbers CNS-1429526 and CNS-1650551.

REFERENCES

[1] Matei Zaharia, Reynold S Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, Xiangrui Meng, Josh Rosen, Shivaram Venkataraman, Michael J Franklin, et al. Apache spark: a unified engine for big data processing. *Communications of the ACM*, 59(11):56–65, 2016.

[2] Sanket Chintapalli, Derek Dagit, Bobby Evans, Reza Farivar, Thomas Graves, Mark Holderbaugh, Zhuo Liu, Kyle Nusbaum, Kishorkumar Patil, Boyang Jerry Peng, et al. Benchmarking streaming computation engines: Storm, flink and spark streaming. In *2016 IEEE international parallel and distributed processing symposium workshops (IPDPSW)*, pages 1789–1792. IEEE, 2016.

[3] Xiangrui Meng, Joseph Bradley, Burak Yavuz, Evan Sparks, Shivaram Venkataraman, Davies Liu, Jeremy Freeman, DB Tsai, Manish Amde, Sean Owen, et al. Mllib: Machine learning in apache spark. *The Journal of Machine Learning Research*, 17 (1):1235–1241, 2016.

[4] Shixia Liu, Jialun Yin, Xiting Wang, Weiwei Cui, Kelei Cao, and Jian Pei. Online visual analytics of text streams. *IEEE transactions on visualization and computer graphics*, 22(11):2451–2466, 2016.

[5] Jonas Traub, Nikolaas Steenbergen, Philipp Grulich, Tilmann Rabl, and Volker Markl. I2: Interactive real-time visualization for streaming data. In *EDBT*, pages 526–529, 2017.

[6] Lokmanyathilak Govindan Sankar Selvan and Teng-Sheng Moh. A framework for fast-feedback opinion mining on twitter data streams. In *2015 International Conference on Collaboration Technologies and Systems (CTS)*, pages 314–318. IEEE, 2015.

[7] Jianlong Zhou, Zelin Li, Zongjian Zhang, Bin Liang, and Fang Chen. Visual analytics of relations of multi-attributes in big infrastructure data. In *2016 Big Data Visual Analytics (BDVA)*, pages 1–2. IEEE, 2016.

[8] Dominik Sacha, Hansi Senaratne, Bum Chul Kwon, Geoffrey Ellis, and Daniel A Keim. The role of uncertainty, awareness, and trust in visual analytics. *IEEE transactions on visualization and computer graphics*, 22(1):240–249, 2016.

[9] SR Venna, RN Gottumukkala, and VV Raghavan. Visual analytic decision-making environments for large-scale time-evolving graphs. In *Handbook of Statistics*, volume 35, pages 81–115. Elsevier, 2016.

[10] Satya Katragadda, Raju Gottumukkala, Murali Pusala, Vijay Raghavan, and Jessica Wojtkiewicz. Distributed real time link prediction on graph streams. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 2912–2917. IEEE, 2018.

[11] A MadhaviLatha and G Vijaya Kumar. Streaming data analysis using apache cassandra and zeppelin. *IJISET-International Journal of Innovative Science, Engineering & Technology*, 3(10), 2016.

[12] Paris Carbone, Asterios Katsifodimos, Stephan Ewen, Volker Markl, Seif Haridi, and Kostas Tzoumas. Apache flink: Stream and batch processing in a single engine. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 36(4), 2015.

[13] Muhammad Hussain Iqbal and Tariq Rahim Soomro. Big data analysis: Apache storm perspective. *International journal of computer trends and technology*, 19(1): 9–14, 2015.

[14] NSF Center for Visual and Decision Informatics. http://www.nsfcvdi.org. Accessed: 2019-03-31.

[15] Jiawei Zhang, Yang Wang, Piero Molino, Lezhi Li, and David S Ebert. Manifold: A model-agnostic framework for interpretation and diagnosis of machine learning models. *IEEE transactions on visualization and computer graphics*, 25(1):364–373, 2019.

[16] A Dell Deployment Guide. Dell| apache hadoop solution dell| cloudera solution deployment guide v1. 6.

[17] Cloudera Manager 5 Overview. https://www.cloudera.com/documentation/enterprise/5-12-x/topics/cm_intro_primer.html. Accessed: 2019-05-22.

[18] Josiah L Carlson. *Redis in action*. Manning Publications Co., 2013.

[19] Avinash Lakshman and Prashant Malik. Cassandra: a decentralized structured storage system. *ACM SIGOPS Operating Systems Review*, 44(2):35–40, 2010.

[20] Lars George. *HBase: the definitive guide: random access to your planet-size data.* " O'Reilly Media, Inc.", 2011.

[21] JanusGraph. https://janusgraph.org. Accessed: 2019-05-22.

[22] Todd Persen and Robert Winslow. Benchmarking influxdb vs. cassandra for time-series data, metrics & management. *InfluxData Tech. Pap*, 2016.

[23] Satya Katragadda, Shahid Virani, Ryan Benton, and Vijay Raghavan. Detection of event onset using twitter. In *2016 International Joint Conference on Neural Networks (IJCNN)*, pages 1539–1546. IEEE, 2016.

[24] Nicholas G Lipari, Christoph W Borst, and Mehmet Engin Tozal. Visual analytics using graph sampling and summarization on multitouch displays. In *International Symposium on Visual Computing*, pages 462–471. Springer, 2016.

[25] USGS Water Services. https://waterservices.usgs.gov. Accessed: 2019-03-31.

[26] Michael F Chislock, Enrique Doster, Rachel A Zitomer, and AE Wilson. Eutrophication: causes, consequences, and controls in aquatic ecosystems. *Nature Education Knowledge*, 4(4):10, 2013.