VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"Jnana Sangama", Belagavi – 590 018.

A Technical Seminar Report

on

"WEB PERFORMANCE EVALUATION OF HIGH VOLUME STREAMING DATA VISUALIZATION"

Submitted in partial fulfillment for the award of

Bachelor of Engineering in Computer Science and Engineering

By

SUDEEP G N [1CK19CS081]

Under the Guidance of

Mr. NARAYANASWAMY H

Associate Professor, Dept. of CS&E.

C. BYREGOWDA INSTITUTE OF TECHNOLOGY

An ISO 9001:2015 Certified Institute

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Kolar – Srinivaspur Road,

Kolar – 563101

2022-2023

C BYREGOWDA INSTITUTE OF TECHNOLOGY

An ISO 9001:2015 Certified Institute

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Kolar-Srinivasapura road,

Kolar-563101

CERTIFICATE

This is to certify that the technical seminar work entitled "WEB PERFORMANCE EVALUATION OF HIGH VOLUME STREAMING DATA VISUALIZATION" is a bonafide work carried out by **SUDEEP G N (1CK19CS081)** in partial fulfillment for the award of Bachelor of Engineering in Computer Science & Engineering of the Visvesvaraya Technological University, Belagavi during the year 2022-23. It is certified that all corrections/suggestions indicated for the internal assessment have been incorporated in the report. The technical seminar report has been approved as it satisfies the academic requirements in respect of seminar work prescribed for the VIII Semester Bachelor of Engineering Degree.

Signature of the Guide	Signature of the Seminar Coordinator	Signature of the HOD
Mr. Narayanaswamy H	**Mrs. Manjula S S**	**Dr. S N Chandrashekara**
Assoc. Professor,	**Asst. Professor,**	**Prof. & HOD**
Dept. of CSE, CBIT	**Dept. of CSE, CBIT**	**Dept. of CSE, CBIT**

ABSTRACT

Many software and hardware applications generate an increasing volume of data and logs in real-time. Visual analytics is essential to support system monitoring and analysis of such data. For example, the world's largest radio telescope, the Square Kilometer Array (SKA), is expected to generate an estimated 160 TB a second of raw data captured from different sources. Transporting large amounts of data from distributed sources to a web browser for visualization is time-consuming due to data transport latencies. In addition, visualizing real-time data in the browser is challenging and limited by the data rates which a web browser can handle. We propose a novel low latency data streaming architecture, which uses a messaging system for real-time data transport to the web browser. Based on this architecture, we propose techniques and provide a tool for analyzing the performance of serialization protocols and the web-visualization rendering pipeline. We empirically evaluate the performance of our architecture using three visualizations use cases relevant to the SKA. Our system proved extremely useful in streaming high-volume data in real-time with low latency and greatly enhanced the web-visualization performance by enabling streaming an optimal number of data points to different visualizations.

DECLARATION

I, **SUDEEP G N** bearing USN **1CK19CS081** Student of 8th semester B.E., Computer Science and Engineering of VTU, declare that this seminar report entitled **"WEB PERFORMANCE EVALUATION OF HIGH VOLUME STREAMING DATA VISUALIZATION"**, embodies report of seminar work carried out under the guidance of **NARAYANASWAMY H** Assoc. Professor **Dept. Of CSE, CBIT** as partial fulfillment of the requirement of the award of the degree in Bachelor of Engineering, Computer Science and Engineering, affiliated to **VISVESVARAYA TECHNOLOGICAL UNIVERSITY, BELAGAVI** during academic year **2022-2023**. Further the content embodies in the project has not been submitted previously by anybody for the award of any other degree.

Place: Kolar Signature of Students

Date: **(SUDEEP G N)**

ACKNOWLEDGEMENT

The completion of any work is a showcase of constant dedication and co-operation of many people who lent their hands which went seen or unseen.

I am grateful to our Institution and Management, **C. BYREGOWDA INSTITUTE OF TECHNOLOGY** with its ideals and inspirations for having provided me with the facilities, which has made this seminar a success.

I would like to thank our beloved Principal **Dr. BASAVA KUMAR K G**, CBIT for his kind disposition for completing this undertaking successfully.

I express my heart full gratitude to **Dr. S N CHANDRASHEKARA**, Professor and HOD, Department of Computer science & Engineering, CBIT, for giving me guidance, valuable advice and support.

I express my heart full gratitude to **Prof. MANJULA S S**, Professor and seminar coordinator, Department of Computer science & Engineering, CBIT, for giving us guidance, valuable advice and support.

I extend my gratitude to our guide **Prof. NARAYANASWAMY H**, Assoc. Professor, Department of Computer Science & Engineering, CBIT for his valuable advice, support and constructive suggestions.

I also thank to all our professors and the entire department of Computer Science & Engineering for their co-operation and suggestions.

The report would be incomplete if I do not thank my parents and friends for their continuous encouragement and moral support.

SUDEEP G N - 1CK19CS081

CONTENTS

Abstract		**i**
Declaration		**ii**
Acknowledgement		**iii**
Table of Contents		**iv**
List of Figures		**v**

Chapter No.	**Chapter Name**	**Page No.**
1	**Introduction**	**1**
2	**Related Work**	**4**
	2.1 Radio Astronomy And Science Data Visualization	4
	2.2 Real-Time Data Visualization	5
	2.3 Web Visualization Performance Evaluation	6
3	**Real-Time Data at SKA**	**7**
	3.1 Real-Time Data Visualization Technology Overview & Challenges	8
4	**A Scalable Architecture For Streaming Data Visualization**	**10**
5	**Empirical Study Design**	**12**
	5.1 Methodology	12
	5.2 Use Cases	14
	5.3 Implementation And Experimental Setup	17
6	**Observation**	**18**
	6.1 Power Spectrum Plot	18
	6.2 Waterfall Plot Of Spectrogram	19
	6.3 Spectrograms Of Baselines And Polarizations	20
	Conclusion	**22**
	References	**23**

LIST OF FIGURES

FIG NO.	FIGURE NAME	PAGE NO.
3.1	The high-level data flow at the SKA.	07
4.1	A generic architecture for high-volume data streaming and visualization.	10
5.1	An end-to-end sequence diagram of dataflow with key performance indicators.	12
5.2	Example power spectrum plots computed from data from the Meerkat telescope.	14
5.3	An example waterfall plot of a spectrogram representing a baseline and a polarization.	15
5.4	A table of 77 small spectrograms, each spectrogram representing a baseline and a polarization.	16
6.1	power spectrum plot.	18
6.2	waterfall plot of spectrogram.	19
6.3	spectrograms of baselines and polarizations	20

Chapter 1

INTRODUCTION

Real-time data visualization is widely utilized to monitor the status and operation of hardware and software systems. Areas which benefit from visualizing large amounts of data in near real-time include, large scientific experiments network monitoring, mass spectroscopy, fraud detection, game analytics, radio communications, energy research, atmospheric science and others. As the scale of the monitored system increases the amount of data to be visualized also increases. Large data rates give rise to architectural and visualization bottlenecks which are not present for smaller visualization systems.

The Square Kilometer Array (SKA) telescope, will be a large hardware and software system in need of real-time monitoring. The SKA will be the largest radio telescope constructed to date and is expected to capture data at a rate of 160 TB/s , and will consist of roughly 100,000 antennas in Australia and South Africa. Owing to its scale and complexity the SKA requires a robust system to monitor its status and operation. To allow operators to monitor the health, and current status of the different parts of the telescope the SKA requires a real-time quality assessment visualization system. Visual monitoring systems are a feature of all current large radio telescopes, such as the Atacama Large Millimeter Array (ALMA), and the MeerKAT radio telescope. More generally real-time monitoring visualizations are used by large and data intensive scientific experiments, such as CERN, and LIGO. The SKA real-time visual monitoring system serves as an early concrete example of a system which must be able to handle the data rates future large-scale visual monitoring solutions, in a variety of fields, will experience.

Despite the ubiquity of real-time data visualization, little work has been done to address (a) the scalability of such systems and (b) how increased data rates impact the performance when displaying real-time visualizations in a web browser. The architectures used by existing real-time data visualization systems are pull-based,

Where the browser periodically polls (also called pulls) data from an intermediate source, e.g., a database and updates the display with new data. While this

approach is used widely, we observe two key limitations with this approach. Firstly, it does not scale favorably when compared to the SKA's latency requirements for high volume and high frequency data. Secondly, the performance of web browsers limits how much data can be processed, displayed, and updated within a given time frame. The browser may crash or the visualization lag if the rate at which is supplied to a web-based visualization exceeds what the browser is capable of.

To address these limitations, we start by conducting an empirical study to determine the latencies due to the different stages of an architecture which relies on polling. Based on this investigation, we propose an alternative, push-based, data streaming architecture. Motivated by the success of message brokers for message pushing applications, we find that they are able to support our latency requirements and stream the various types of data required for different visualizations in parallel. We thus adopt a message broker, designed for streaming data, to implement the push-based architecture for real-time data visualization applications.

Since the proposed push-based streaming architecture was able to supply data for real-time visualizations at rates greater than a browser may be able to cope with, the introduction of the push-based architecture necessitated measuring the browser performance when displaying various visualizations. These performance measurements allow us to optimize the streaming data rates for a browser, to minimize rendering lag when displaying real-time visualizations. The browser visualization performance measurement technique, necessitated by the increased throughput of the push-based architecture, is another novel contribution of this work. Three real world use cases are implemented to evaluate their web visualization performance as well as to evaluate the streaming data architecture.

The outcome of our work is as follows:

• We reviewed technologies used for radio astronomy data visualization and real-time data visualization in general. We discussed their limitations in high-frequency and high-volume real-time data visualization contexts, such as for the SKA.

• We proposed a scalable architecture for high-frequency and high-volume data streaming and visualization in the web browser. This architecture is transferable to different data intensive domains.

• We identified key performance indicators that contribute to latency in real-time web data visualization. We implemented three visualizations as use cases and analyzed their performances in terms of the identified key performance indicators.

• We developed a tool for supporting rapid development and evaluation of web visualizations and open-sourced the tool. Finally, we discuss the limitations of our study and propose recommendations for high-volume realtime data visualization in the web browser.

Chapter 2

RELATED WORK

2.1 Radio Astronomy And Science Data Visualization

In radio astronomy projects, such as the SKA, data visualization is commonly used to monitor instruments and visualize the data captured by the telescope. In the case of the Atacama Large Millimeter Array (ALMA) radio telescope data is saved in a database. The database facilitates on-demand retrieval as well as near real-time polling of the data for visualizations. ALMA's dashboards are implemented using a combination of technologies, including d3.js for the interactive data driven visualizations and SVG for rendering. MeerKAT, another radio telescope, uses a bespoke system for monitoring the antennas. MeerKAT's system uses a Redis Pub/Sub messaging system to transport data to the user interfaces. MeerKAT also uses Graphana, an off-the-shelf solution, for creating dashboards. A large science project outside radio astronomy, CERN uses the messaging system, e.g., Kafka, for transporting data from different sources to central databases, e.g., search engine (Elasticsearch), time series database (InfluxDB), Apache Hadoop and so on. They developed user interfaces and visualizations using off-the-shelf tools, e.g., Grapha, InfluxDB, Prometheus, Kibana, etc., for monitoring different systems. The LIGO Scientific Collaboration developed web-based real-time science data visualization but did not provide sufficient technical details of their visualization software and its architecture.

All the aforementioned works are motivated by the need for large-scale data visualization for real-time monitoring. However, these works lack technical contributions and offer limited insight into more general streaming data visualization system design and development. Our work addresses this gap with a more technical contribution to high-volume and lowlatency streaming data visualization system architecture and development.

2.2 Real-Time Data Visualization

Real-time and dynamic data visualization systems are of course also developed for a broader range of applications, including uses both within the wider sciences and industry. Examples include: network monitoring, analysis of vibrations during drilling operations, monitoring of sensors, manufacturing processes, control and hardware, urban air quality monitoring, etc. All these systems use databases to store the data received from the source and use the database polling technique for near real-time access and data visualization.

Visualization of real-time three-dimensional (3D) data, e.g., road environment data from light detection and ranging (LiDAR) is presented in, and 3D GIS data is presented in. We anticipate that the amount of data to be visualized in this project is large. However, there are limited technical details on how the data is streamed or transferred to the user interface in real-time.

Protopsaltis presented a survey of visualization methods, tools, and techniques for the IoT. They discuss different visualization tools, techniques, and challenges in developing visualizations for various IoT domains. However, there are no technical details on streaming data visualization systems.

Mass spectroscopy is another field where large amounts of data is commonly visualized using the browser. Examples include the seaMass software. This work involves streaming and then visualizing large amounts of scientific data, by sending a data set iteratively, based on the cognitive importance of the different features, thus quickly presenting the most important information to a user. However, this application differs from a monitoring application in that a selected data set is streamed to the browser in steps, as opposed to using streaming to visualize data which changes in real-time. In this work polling is used to stream the data to the client.

Messaging systems (e.g., Kafka, RabitMQ, Redis Pub/Sub, etc.) have been widely used for real-time data streaming and transport in a variety of application domains, e.g., IoT, messaging apps, radio astronomy data, and science data. Especially Kafka delivers high volume data streaming with low latency. However, the use of messaging systems for real-time streaming data visualization systems is rare in the literature.

2.3 Web Visualization Performance Evaluation

Hoetzlein analyzed graphics performance in rich internet applications using transparent 2D sprites. Their results show that the application performs better in the Google Chrome browser than in Internet Explorer and Firefox. They also implemented the application using WebGL-based GPU accelerated visualization, which outperformed the Flash and HTML5 Canvas implementations. Kee et al.suggested that interactive visualizations that require high performance should be implemented in WebGL and Canvas. The performance gain of visualizations implemented in WebGL is marginally higher than HTML5 Canvas implementation, whereas the development cost of a WebGL application is significantly greater.

Lee et al. surveyed and tested web-based data visualization tools and libraries, e.g., Google Charts, Flex, OFC, d3.js and, JfreeChart. They used 100,000 static data points for performance analysis.

None of these works address high volume, real-time and dynamic data visualization and its performance. In our work, we analyze the web visualization performance of large-scale and real-time streaming data.

Chapter 3

REAL-TIME DATA AT SKA

The SKA will consist of a large number of antennas which capture radio astronomical data, such captured data is called visibilities or visibility data. The first step of the data processing is that on-site ‘‘correlators’’ correlate the captured data. The correlated data is then sent to a distributed Science Data Processor (SDP) via a high-speed network. As shown in Fig. 3.1, the SDPs temporarily save the data in a very large Shared Memory. Different Applications access the data in real-time for, e.g., reconstruction of the sky image, astronomical discoveries, studying the astronomical signals, detecting anomalies, and so on. Metric Generators are distributed streaming data processing applications that generate different metrics in real-time from the streamed visibility data, e.g., power spectrum, the phase information of baselines, polarization, etc. The metric generators also generate metrics from data received from other applications, e.g., real-time radio frequency interference detectors, different sensors on the antennas, other instruments, and so on.

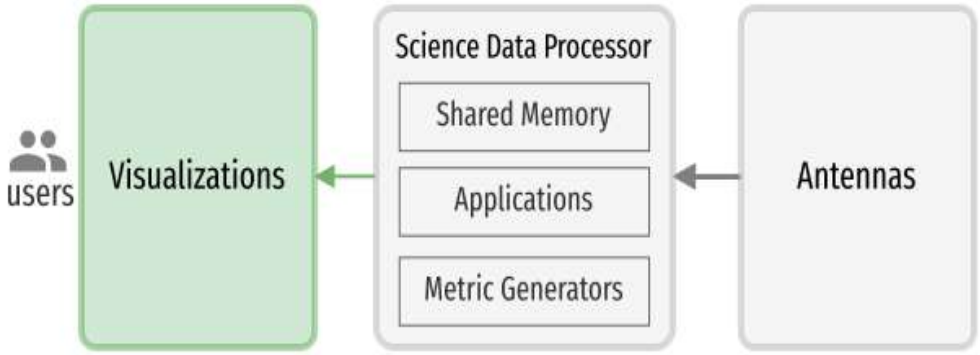

FIGURE 3.1. The high-level data flow at the SKA.

The data processing pipeline of the SKA is designed to implement expensive operations, e.g., extraction, cleaning, transformation, aggregation, etc., using low-level functions, e.g., metric generators, deployed in high-performance computing infrastructure. The metric generators also do additional pre-processing of data required to reduce consumers’ or web browsers’ need for further processing. While we will discuss the data structure of three metrics used for visualization use cases in this paper, the details about how the metric generators derive these metrics from the raw visibility data and the performance of the metric generators are not within the scope of this work. The essential functional role of our system is to transfer data

generated by the metric generators to the browser and visualize different metrics using different plots and dashboards as in Fig. 3.1. This is to convey the current state of the telescope to operators and radio astronomers.

3.1 Real-Time Data Visualization Technology Overview & Challenges

There are numerous off-the-shelf data visualization systems available for visualizing data in web browsers. In our first prototype, we adopted the polling data principle common in the literature. We used InfluxDB, a timeseries database to store the metrics generated by our metric generators and Grafana dashboard to continuously poll the data from the database and visualize. The high-level architecture of this prototype can be found in Fig. 3.2. The metric generators write new metrics to the timeseries database tables every second, hence the Grafana connector is configured to poll and retrieve the metrics from the database tables at every second. We used Grafana's built-in visualizations While testing and evaluating the prototype we observed two major problems (a) a significant latency in transporting the data to web browser, and (b) the browser rendering performance.

In order to identify which steps of the process contribute the most to the total latency we performed end-to-end profiling of the prototype.

- **Network latency –** The prototype is designed as a service-oriented architecture where the metric generators, databases, and web applications are developed and composed as separate services running in different containers. It requires some time to transfer the data from one service to another, e.g., the metric generators to the database and the database to the web browser.

- **Database read and write operation latency** – We noticed a significant latency in database operations. The database takes a significant amount of time to write the incoming metrics to an index and perform additional housekeeping operations, e.g., transaction, consistency or distributed transaction, linking records using primary and foreign keys, persistence, disk and in-memory data operations. We found that, on average, a synchronous InfluxDB read and/or write operation takes 100–150 milliseconds. Different

databases perform different housekeeping operations based on their characteristics. For example, a non-transactional database can save a significant amount of transactional (e.g., atomicity, consistency, isolation, and durability (ACID) properties of database) processing time and an in-memory database can achieve very low latency and high throughput as data stays in primary memory.

- **Polling latency** – Polling is a technique in which the browser regularly asks the database for new data. The web visualization functions make repeated requests to the database server for new metrics at a predefined interval (in our case, it is ∼250 milliseconds), These repeated requests waste resources. For example, each new incoming connection must be established, the HTTP headers must be passed, a query for new data must be performed, and a response (usually with no new data to offer) must be generated and delivered.
- **Web visualization latency** – In the web browser, ideally, each received data point should be processed and rendered into a visualization before the next data point arrives. If the metric generator generates and sends a metric to the browser at every Tfreq second interval, the payload must be visualized in less than Tfreq seconds. For the SKA, the value of Tfreq is 1 second.
- **Data processing latency** – Web browsers receive data as serialized payloads. The serialized payloads need to be deserialized, transformed and mapped to a data structure compatible with visualization functions which renders the data into plots.
- **Rendering time** – Rendering time depends on multiple factors, such as the amount of data to be processed, the processing functions, the visualization designs (plots) and their complexity, as well as the low-level rendering technologies used (e.g., SVG, Canvas).

Chapter 4

A SCALABLE ARCHITECTURE FOR STREAMING DATA VISUALIZATION

To address the inherent polling latency of the pull-based architecture, we adopted a messaging system and proposed an improved push-based architecture, seen in Fig. 4.1 The main objective of our proposed architecture is to stream data to a web browser using push-based communication protocols eliminating data transmission latency incurred due to periodic polling.

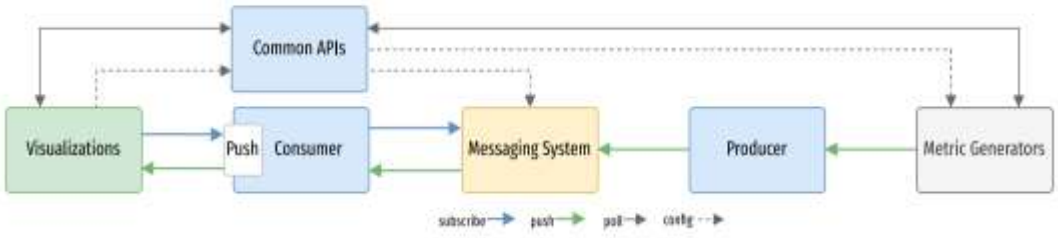

FIGURE 4.1 A generic architecture for high-volume data streaming and visualization.

There are four main components of the proposed architecture:

- **Messaging System** – The main component of our data streaming system is a messaging system. The messaging system sends messages between applications, processes, and servers, The broker or message broker, e.g., Apache Kafk is a publish-subscribe based messaging system. The broker handles all requests (e.g., produce, consume) and metadata from clients and keeps data replicated within the cluster. There can be one or more brokers in a cluster. Brokers use topics, which are categories or feed names to which records are stored and published. Our metric generators publish metrics on different topics. Applications, web visualizations subscribe to the messaging system and consume the metrics published into the relevant topics.
- **Producer** – Our metric generators publish different metrics to the message broker using the producer APIs. The producers are processes that push records into Kafka topics within the broker. The producers publish or write a stream of

events to one or more Kafka topics. The metrics or data are stored for a specified retention period, which can be configured using the producer APIs.

- **Consumer** – While producer applications write data to topics the consumer applications read from topics. The Consumer subscribes to one or more topics and reads records on these topics. A consumer reads messages from partitions, in an ordered fashion. For example, if messages m1, m2, m3, m4 are inserted into a topic in this order, the consumer will read them in the same order. Since every message has an offset, every time a consumer reads a message it stores the offset value in Kafka, denoting that it is the last message that the consumer read. Additionally, if at any point in time a consumer needs to go back in time and read older messages, it can do so by resetting the offset position.

  Web applications establish connections with consumers using WebSocket. The data read by the consumers are immediately transported to the web browser.

- **Common APIs** – These are a set of REST APIs implemented to perform various operations, e.g., configure different parameters of the message broker, sending commands to the metric generators (e.g., setting different averaging factors, selecting serialization protocols, configuring the streaming data processing engine, etc). We also implemented a set of APIs to pull data or metrics from the metric generators on-demand. The Kafka Confluent Control Center provides off-the-shelf APIs and a user interface for managing Kafka clusters.

Chapter 5

EMPIRICAL STUDY DESIGN

5.1 METHODOLOGY

In the first evaluation, we generate serialized payloads using different protocols on the server side and deserialize the payloads in the browser. This evaluation aims to measure the serialization and deserialization cost of the JSON and ProtoBuf protocols. That is, which serialization protocol generates smaller payloads and which is faster to deserialize.

For the second evaluation, we visualize the deserialized payloads in the web browser using appropriate plots implemented using JavaScript functions. This test aims to measure visualization latency, e.g., to determine how much time it takes to process and render payloads of different sizes into relevant plots.

FIGURE 5.1. An end-to-end sequence diagram of dataflow with key performance indicators

Our application has several components and we measure latencies of key performance indicators, as shown in sequence diagram Fig. 5.1

- **Serialized payload size (Sser) –** Metric generators serialize data to create payloads. There are two key performance indicators, e.g., serialization time and payload size involved at this stage. Analysis of serialization time is not within the scope of this paper, as serialization will happen in a high-performance computing environment as a part of the data pre-processing. However, we want the size of the serialized payload to be small, so it can be reliably handled by the message broker and quickly transferred through the network. The payload size of serialized data depends on the data format and serialization protocol used.

- **Transmission time (Ttrans)** – This measures the time spent sending metrics from metric generators to the browser. The components are deployed as distributed services, and the value of this parameter will depend on the proximity of the services, the network connection between the services, etc. In our test application, we deploy the services in multiple containers connected via the bridge network of a host machine.

- **Deserialization time (Tdser)** – The main latencies in the web browser are due to deserialization of payloads, data processing, and rendering cost. When the payloads are received in the web browser, they are deserialized using Javascript functions. The deserialized payload is a JavaScript object. The deserialization time can contribute significantly to the latency.

- **Rendering time (Trend)** – The deserialized JavaScript objects are visualized using corresponding visualization functions. Each visualization function executes a drawing function, which includes instructions on how the data will be plotted and rendered to the browser display. Internally, the browser executes Critical Rendering Path (CRP which involves JavaScript execution, rendering, and rasterization to create and display the visualization. With this metric we report the CRP time.

  The metric generators generate and send metrics to the browser every second (Tfreq), and each payload needs to be processed and rendered before the next payload arrives. We need to ensure that total latency in the browser, Trend + Tdser ≤ Tfreq. As if this combined latency is greater than Tfreq, one second in our case, the payload will accumulate, the browser heap size will increase, and the browser will become unresponsive.

5.2 USE CASES

To study the streaming architecture performance and the web visualization performance on the streamed data, we implement a simple use case, e.g., line plot, and two data-intensive use cases, e.g., spectrograms. These use cases were selected through discussions with SKA's domain scientists and radio astronomers.

1) **POWER SPECTRUM PLOT:** Multi-antenna radio telescopes rely on radio-interferometry. The mathematical details of interferometry are beyond the scope of this paper; however, the general idea is to combine the interference patterns generated by pairs of antennas (called baselines), based on their physical location, to obtain an image of the observation field after a series of mathematical operations.

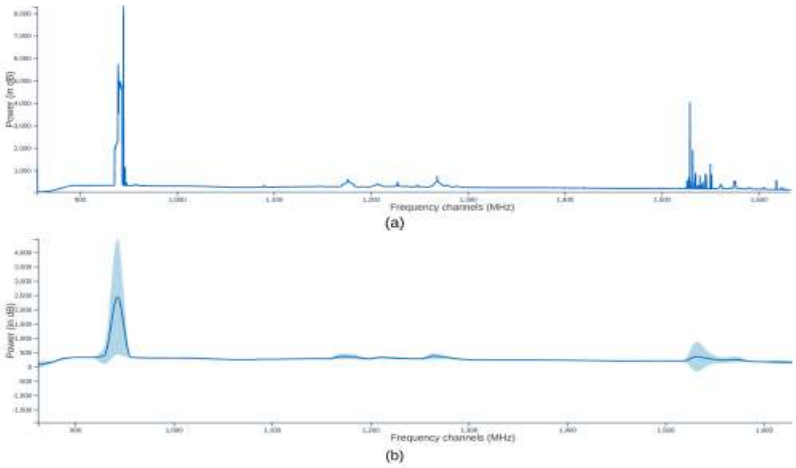

FIGURE 5.2. Example power spectrum plots computed from data from the Meerkat telescope.

Fig. 5.2 shows the distribution of the average autocorrelation values of all antennas across the frequency channels, i.e. the power spectral density (power spectrum) plot. Note that only the amplitude of the visibilities are used in this plot. Such a plot not only contains useful astronomical information but also helps with monitoring the functioning of the antennas. The SKA project, as the largest radio-telescope ever built, will operate on a maximum of 65,000 frequency channels.

2) WATERFALL PLOT OF SPECTROGRAM

The correlator is constantly producing cross and autocorrelation values for all the baselines. Therefore, it is of interest to the operator to see these values for different frequency channels over time for each baseline as a moving signal.

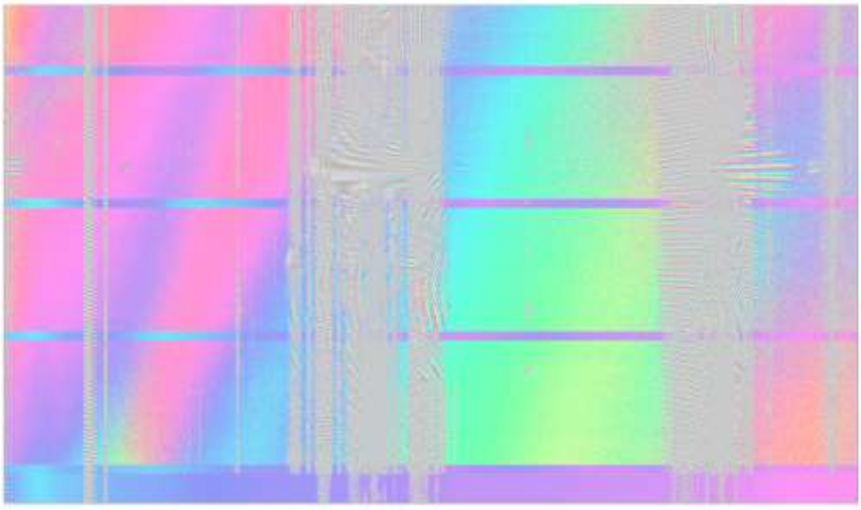

FIGURE 5.3. An example waterfall plot of a spectrogram representing a baseline and a polarization.

Fig. 5.3 shows an example snapshot of such a spectrogram visualizing the phase values of the visibilities as a function of time and frequency (a similar plot can also be generated for the amplitudes). This plot is for a given baseline and polarization (the received signal by each antenna and the visibilities contain four different polarizations).

A spectrogram has x and y dimensions of frequency channels or phase and time. Data flows down, in a waterfall plot manner, as in Fig. 10. This effect is achieved by shifting the block of previously stacked spectra down the screen by one pixel or one line, then a new line of data is added at the top, with the oldest line of spectral data disappearing off the bottom of the display. Fig. 5.3 shows the 4096 phase data points plotted as a function of time. The current time and newest data are always at the top of the plot. The data rate is set at 1 line/sec, and the waterfall plot is set to hold 600 lines (height of the display). Therefore, the oldest line, written 600 sec ago, is at the bottom.

3) SPECTROGRAMS OF BASELINES AND POLARIZATIONS

Both from the astronomical and observation management perspectives (e.g. monitoring the operation of the telescope), it is important to display the spectrogram plots for a collection of baselines. However, it should be noted that for an array of antennas such as the SKA, the number of baselines (all possible pairs of antennas) will be gigantic (e.g., 65,000 channels and 130,000 baselines). In fact, even for smaller radio telescopes the number of baselines (which grows quadratically with the number of antennas) might be too large to be meaningfully visualized. Therefore, the display should allow the user to select subgroups of baselines depending on the requirements of the operator. For instance, the operator might be only interested in autocorrelations to monitor the status of individual antennas, or from the imaging perspective a certain group of baselines (e.g. longer baselines or shorter ones) might be of interest.

FIGURE 5.4. A table of 77 small spectrograms, each spectrogram representing a baseline and a polarization.

The Spectrograms data model, which is a list containing multiple Spectrogram data. Fig. 5.4 shows an example snapshot of a series of spectrograms for different baselines and polarizations in a single display.

5.3 IMPLEMENTATION AND EXPERIMENTAL SETUP

We used the JavaScript library React to implement a web user interface and HTML SVG and Canvas graphics to implement the visualizations. As described earlier, we used push-based protocol, WebSocket, as an underlying protocol for communication between the web browser and consumer. The server side components, e.g., consumer, producer and metric generators are implemented in Python. The production version of the metric generators, data APIs, and the user interface are under development and are open sourced via GitLab. The tool developed for rapidly prototyping and testing different data models and visualizations is also open source and available via GitHub.

We considered two widely used serialization protocols, JSON and ProtoBuf. In order to compare their efficiency, we generate different serialized payloads using both JSON and ProtoBuf protocols and we record the payload sizes. We then deserialize the payloads in web browser using JavaScript functions and we record the deserialization times. In order to compare the rendering performance of a visualization function, we feed data to the visualization function every second. For each received data point the drawing function is executed and the changes are animated. We used the Chrome browser runtime performance analyzer to collect visualization critical rendering pipeline performance data.

The performance evaluation was conducted on Ubuntu 22.04 LTS powered by an Intel i9-9900K (8-Core/16-Thread, 16MB Cache, 4.7GHz across all cores), 32GB DDR4 XMP (2933MHz) memory, M.2 PCIe SSD storage, and Gigabit Ethernet network. A Chrome browser and our front-end and back-end services are deployed on this machine. For a production deployment, the back-end services should be deployed on a cluster of virtual machines with auto scaling capability.

Chapter 6

OBSERVATION

6.1 POWER SPECTRUM PLOT

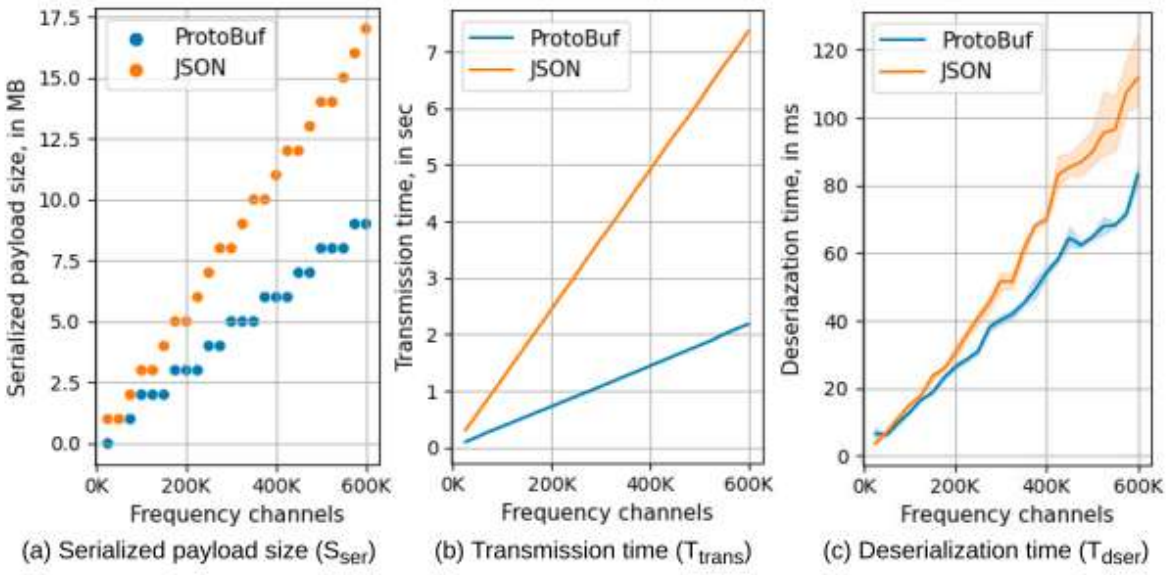

FIGURE 6.1. power spectrum plot.

Fig. 6.1(a) shows the difference in serialization cost between ProtoBuf or JSON encoding of the Spectrum data model. It can be seen that ProtoBuf generated payloads are significantly smaller than their JSON counterparts. As the number of channels increases, ProtoBuf's space efficiency improves further compared to JSON. For instance, when serializing 600,000 channels, the size of the JSON payload is approximately twice that of the equivalent ProtoBuf payload. This indicates that ProtoBuf is more efficient than JSON at encoding large amounts of data.

Fig. 6.1(b) illustrates the time required for the push-based architecture to transmit payloads from the metric generators to the browser. Two principal observations can be made from this data: (i) As expected, the transport time for smaller payloads is smaller than that for larger payloads, and the latency due to transmitting the data increases with the size of the payload. (ii) Additionally, JSON encoded payloads take longer to transport than ProtoBuf encoded payloads. This suggests that ProtoBuf is more efficient at transmitting data compared to JSON, particularly for larger payloads.

Fig. 6.1(c) demonstrates the deserialization time for ProtoBuf and JSON encoded payloads in the browser. It can be seen that the deserialization time for JSON payloads is higher than that for ProtoBuf payloads. This may be because the JSON payloads are larger than the corresponding ProtoBuf payloads, which may lead to longer deserialization times. This suggests that ProtoBuf is more efficient at deserializing data in the browser compared to JSON, particularly for larger payloads.

6.2 WATERFALL PLOT OF SPECTROGRAM

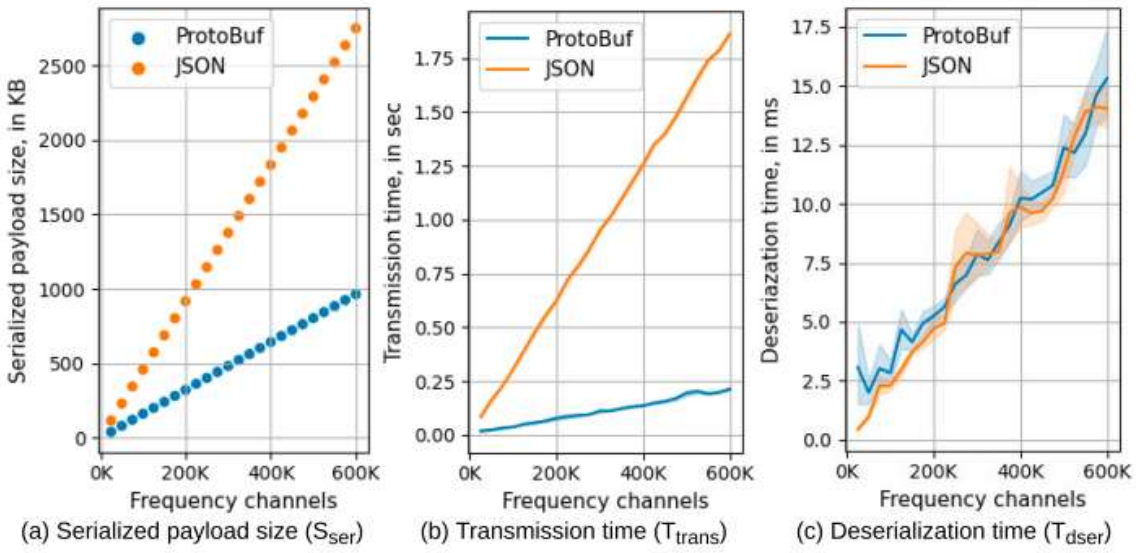

FIGURE 6.2. waterfall plot of spectrogram.

Fig. 6.2(a) illustrates the serialization cost when using ProtoBuf or JSON encoded payloads of the Spectrogram data model. The results show that the size of ProtoBuf payloads is marginally smaller than that of JSON payloads, only differing by 1~2 MB at 600,000 channels, as opposed to around 8 MB in the Spectrum data model. This suggests that ProtoBuf and JSON are relatively similar in terms of their serialization efficiency for the Spectrogram data model. It is worth noting that the size difference between the two encoding methods may vary depending on the specific characteristics of the data being serialized.

Fig. 6.2(b) demonstrates that the streaming cost for JSON encoded payloads is significantly higher than for ProtoBuf encoded payloads. The data shows that it takes nearly 2 seconds to transmit a JSON payload consisting of 600,000 channels, while it takes only 250 milliseconds to transmit a ProtoBuf payload containing the same data. This indicates that ProtoBuf is much more efficient at streaming Spectrogram data than JSON, particularly for

large payloads. This could be due to ProtoBuf's use of a binary encoding format, which is generally more efficient for transmission over a network compared to the text-based encoding of JSON.

Fig. 6.2(c) shows that the deserialization cost of the ProtoBuf and JSON encodings are almost the same. This suggests that there is little difference in the efficiency of the two encoding methods when it comes to deserialization. However, when compared to the results for the spectrum data model in Fig. 6.1(c), it is worth noting that the specific characteristics of the data being deserialized could affect the relative performance of ProtoBuf and JSON. Additionally, the performance of the deserialization process may be influenced by factors such as the hardware and software configurations of the system on which it is being performed.

6.3 SPECTROGRAMS OF BASELINES AND POLARIZATIONS

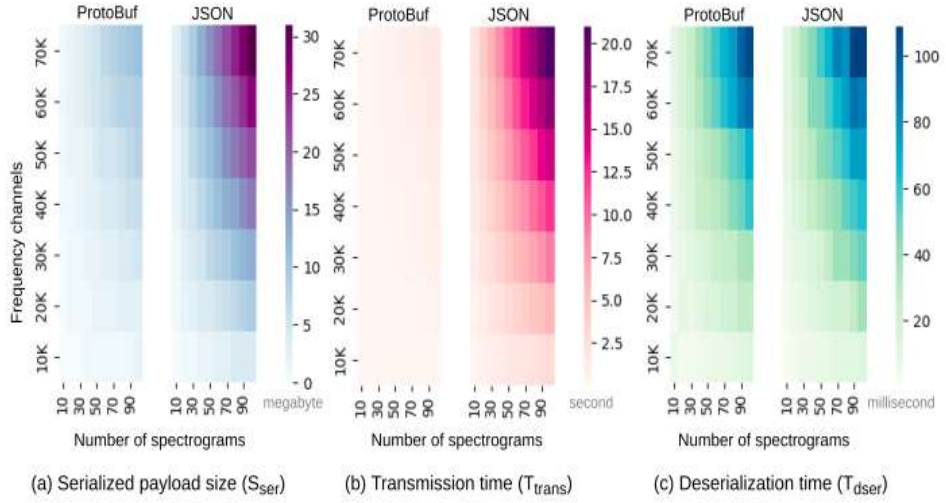

FIGURE 6.3 spectrograms of baselines and polarizations

Fig. 6.3(a) presents the serialization cost of ProtoBuf and JSON encoding of the Spectrograms data model. The results show that, as expected, the size of the ProtoBuf serialized data is marginally smaller than the size of the JSON serialized data. This indicates that ProtoBuf is slightly more efficient at serializing the Spectrograms data model compared to JSON. However, the difference in size between the two encoding

methods may be relatively small, depending on the specific characteristics of the data being serialized.

Fig. 6.3(b) illustrates that the streaming cost for JSON encoded payloads is significantly higher than for ProtoBuf encoded payloads. The data shows that it takes nearly 21 seconds to transmit a JSON payload consisting of 100 spectrograms, each containing 70,000 channels, while it takes only 1.6 seconds to transmit a ProtoBuf payload containing the same data. This demonstrates that ProtoBuf is much more efficient at streaming data than JSON, particularly for large payloads. This could be due to ProtoBuf's use of a binary encoding format, which is generally more efficient for transmission over a network compared to the text-based encoding of JSON.

Fig. 6.3(c) shows that the deserialization performance of the ProtoBuf and JSON encodings are similar. As expected, similar performance was also observed for the Spectrogram data model.

CONCLUSION

In this work, we presented an architecture for streaming high-volume data in the web browser in real-time and proposed methodologies for web visualization performance analysis. Because of the need to visualize various matrices to ensure day-to-day SKA telescope operations, support quality assurance of the telescope instruments, and rapid observation of large volumes of radio astronomy data captured by the telescope, we adopted data streaming architecture. Because of the limitations of the web browser in how much and how frequently it can process and render data, we introduced methods to measure this quantitatively. The streaming and visualization architecture can be generalized to stream various data types from different application domains. The web visualization performance, e.g., deserialization time and rendering time, may vary for data type and visualization functions or charts. Therefore, the performance of each data type and its visualization should be measured separately.

REFERENCES

[1] J. Henning and R. Smith, ‘‘A web-based system for creating, viewing, and editing precursor mass spectrometry ground truth data,’’ BMC Bioinf., vol. 21, no. 1, pp. 1–10, Dec. 2020.

[2] I. K. Choi, T. Jiang, S. R. Kankara, S. Wu, and X. Liu, ‘‘TopMSV: A webbased tool for top-down mass spectrometry data visualization,’’ J. Amer. Soc. Mass Spectrometry, vol. 32, no. 6, pp. 1312–1318, Jun. 2021.

[3] C. Maçãs, E. Polisciuc, and P. Machado, ‘‘ATOVis—A visualisation tool for the detection of financial fraud,’’ Inf. Visualizat., vol. 21, no. 4, pp. 371–392, Oct. 2022.

[4] J. S. Farnes, B. Mort, F. Dulwich, K. Adamek, A. Brown, J. Novotny, S. Salvini, and W. Armour, ‘‘Building the world’s largest radio telescope: The square Kilometre array science data processor,’’ in Proc. IEEE 14th Int. Conf. e-Sci., Oct. 2018, pp. 366–367.

[5] G. Hesse, C. Matthies, and M. Uflacker, ‘‘How fast can we insert? An empirical performance evaluation of apache kafka,’’ in Proc. IEEE 26th Int. Conf. Parallel Distrib. Syst. (ICPADS), Dec. 2020, pp. 641–648.

[6] A. Protopsaltis, P. Sarigiannidis, D. Margounakis, and A. Lytos, ‘‘Data visualization in Internet of Things: Tools, methodologies, and challenges,’’ in Proc. 15th Int. Conf. Availability, Rel. Secur., 2020, pp. 1–11.

[7] J. A. Miller, H. Zhu, and J. Zhang, ‘‘Guest editorial: Advances in web services research,’’ IEEE Trans. Services Comput., vol. 10, no. 1, pp. 5–8, Jan. 2017.

[8] S. Lee, J.-Y. Jo, and Y. Kim, ‘‘Performance testing of web-based data visualization,’’ in Proc. IEEE Int. Conf. Syst., Man, Cybern. (SMC), Oct. 2014, pp. 1648–1653.