

VIETNAM NATIONAL UNIVERSITY, HO CHI MINH CITY
HO CHI MINH UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING



Big Data (CO1007)

Assignment Report

Applying Streaming SQL in Real-Time Social Media Interaction Tracking

Mentor: Thoại Nam
Student: Trần Hà Tuấn Kiệt – 2011493
Nguyễn Đức Thụy – 2012158

Ho Chi Minh City, 12/2024



Table of Contents

1	Introduction about Social media Interaction	3
2	Problem Statement	5
2.1	Impact	5
2.2	Objectives	5
2.3	Approach	5
2.3.1	Data Collection	5
2.3.2	Concept	7
2.3.3	Tool	10
3	Implementation	11
3.1	Mastodon	11
3.2	Set Up Infrastructure	12
3.3	Arroyo Console guide	14
3.4	Develop Instances Count Measurement	16
4	Conclusion	20
5	Report links and details	22

List of Figures

1	Comparison between bounded dataset and unbounded dataset. Source: Arroyo .	4
2	Concept of Streaming SQL	7
3	Concept of Streaming SQL	8
4	Row-by-Row Processing	8
5	Sliding windows. Source: Arroyo	8
6	Tumbling windows. Source: Arroyo	8
7	Apply to Join Operation	9
8	Bounded Data in Time	9
9	Processing logic of Incremental Updates	9
10	Arroyo	10
11	Arroyo Dashboard. Source: Arroyo	10
12	Displaying deployment in Docker	13
13	Arroyo dashboard with our demo job running	14
14	Arroyo IDE	15
15	Details about a checkpoint of our demo job	16
16	Top 5 Mastodon instances within our sampling interval	19

List of Tables

1 Introduction about Social media Interaction

Social Media Interaction refers to the ways users engage and communicate with content, brands, and other users on social media platforms. It encompasses all forms of engagement, such as likes, comments, shares, mentions, direct messages, and reactions to posts. These interactions create a dynamic, two-way communication channel between individuals, communities, and businesses.

Social media interaction encompasses various elements that enable engagement and communication among users, brands, and communities. Engagement types like likes, comments, shares, retweets, and mentions, which allow users to express opinions and amplify content. Direct communication, such as private messages and replies, facilitates personal connections, while user-generated content (UGC) like reviews and testimonials highlights the creative involvement of users. Additionally, community engagement through group discussions and forums fosters a sense of belonging and collaboration.

The importance of social media interaction lies in its ability to provide real-time feedback, offering valuable insights into audience opinions, preferences, and sentiments. It enhances brand awareness by increasing visibility and trust, while also allowing businesses to analyze content performance and measure campaign success. Social media interaction builds strong relationships between brands and their audiences, encourages loyalty, and helps identify emerging trends and consumer behaviors. Ultimately, it serves as a powerful tool for driving meaningful connections and strategic growth in the digital landscape.

This concept underscores the importance of leveraging unbounded data streams to stay competitive in the dynamic world of digital engagement. The accompanying vibrant image of individuals in creative poses reflects the dynamic and lively nature of social media platforms, aligning with the essence of constant interaction and engagement.

Let's take a look at a comparison between bounded dataset and unbounded dataset in data processing.

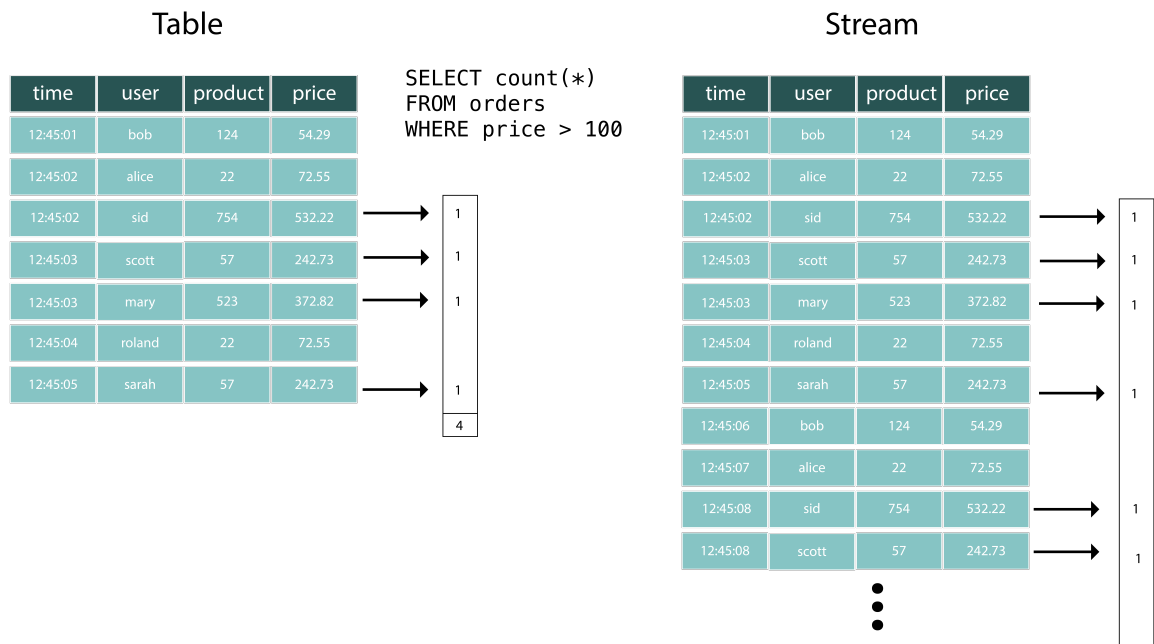


Figure 1: Comparison between bounded dataset and unbounded dataset. Source: Arroyo

A **table represents a bounded dataset**, meaning it contains a finite and complete set of data at any given time. This bounded nature allows for aggregate or summary calculations, such as counting records or calculating averages. For example, the query ‘**SELECT count(*) FROM orders WHERE price > 100**’ can aggregate the results in the table to produce a deterministic count because the data is static and complete.

On the other hand, **streams represent unbounded dataset**, where data continuously flows in overtime. Since streams are ongoing, queries on streams never actually end; they continuously process incoming data. For example, when applying the same query to a stream, each incoming event is processed individually, providing incremental updates to the results. This behavior is suitable for scenarios such as real-time analytics or monitoring, where data is continuously generated and processed.

In short, the **Figure 1** demonstrates this distinction effectively, showing how tables aggregate finite data into a static result, while streams generate dynamic, continuously updated outputs. This concept is essential in understanding how modern data systems like databases, stream processors, and big data frameworks handle and process information.

2 Problem Statement

2.1 Impact

The lack of real-time insights from unbounded social data significantly hampers the ability of businesses and organizations to respond promptly to trends and audience engagement. In the fast-paced digital landscape, trends evolve rapidly, and audience preferences shift dynamically. Without continuous access to real-time data streams, it becomes difficult to monitor and analyze social interactions effectively. This delay in gaining actionable insights prevents organizations from adapting strategies, capitalizing on emerging opportunities, or addressing potential challenges in a timely manner. Ultimately, the inability to respond to real-time engagement can lead to missed opportunities, reduced competitiveness, and weakened connections with audiences, making it crucial to leverage tools that process unbounded data streams for effective decision-making and sustained growth.

2.2 Objectives

The objective is to **implement a real-time social media interaction tracking system for platforms** like Mastodon and X (formerly Twitter). This system will process and analyze unbounded streams of social media data, enabling organizations to monitor user interactions, engagement trends, and audience behaviors as they happen. By providing actionable insights in real time, it empowers businesses to stay responsive to emerging trends, adapt strategies effectively, and maintain relevance in the fast-paced digital landscape. This highlights the importance of leveraging advanced data processing technologies to manage the dynamic nature of social media, ensuring informed and timely decision-making.

2.3 Approach

2.3.1 Data Collection

Mastodon

Mastodon is a decentralized social media platform structured as a **network of independent servers**, commonly referred to as instances. Each instance operates independently, serv-

ing unique communities and interests, yet they remain interconnected through a federated network. This decentralized approach contrasts with centralized platforms, fostering diversity and community-specific experiences.

- **Structure:** Mastodon's ecosystem consists of multiple independent servers, each catering to distinct communities while participating in a larger, interconnected federation.
- **Data access:** Mastodon provides real-time data streams through its Server-Sent Events (SSE) API, enabling access to ongoing activities like posts, mentions, hashtags, and interactions.
- **Data types:** The platform generates various types of data, including posts, mentions, hashtags, and user interactions, making it a rich source for social media analytics and engagement monitoring.

Mastodon presents unique challenges due to its decentralized structure, where data is distributed across numerous independent servers (instances). Aggregating and analyzing data from the entire network is inherently complex and resource-intensive, requiring sophisticated solutions to handle the fragmented ecosystem. Additionally, Mastodon's strong emphasis on user privacy, with strict controls and regulations, further complicates data collection and processing while ensuring compliance with user preferences and privacy standards.

X (formerly Twitter)

X, formerly known as Twitter, operates as a centralized social media platform with a unified database, making it distinct from decentralized systems like Mastodon. This centralized structure provides a single, cohesive source of data, which simplifies access and analysis for users, developers, and organizations.

- **Structure:** X is built on a centralized database architecture, enabling streamlined data management and easier access to platform-wide information.
- **Data access:** The platform offers real-time streams through the X API, allowing developers to track activities such as tweets, retweets, likes, mentions, and trending topics in real time.

- **Data types:** X generates various types of valuable data, including tweets, retweets, likes, mentions, and trends, making it a vital source for social media analytics and insights.

X faces notable challenges in its data access and usage policies. Strict **rate limits on API usage** restrict the volume of data that can be collected within a specific timeframe, creating barriers to conducting comprehensive and large-scale analyses. Additionally, **privacy regulations and API restrictions** further complicate data collection and processing, as developers and researchers must navigate limitations designed to protect user data while ensuring compliance with legal and ethical standards. These challenges necessitate innovative solutions to effectively leverage the platform's data while respecting its constraints.

2.3.2 Concept

Streaming SQL is an extension of traditional SQL, specifically designed to handle unbounded, real-time data streams. Unlike standard SQL, which processes finite datasets stored in static tables, Streaming SQL operates on continuous data streams that are constantly being updated, enabling real-time analysis and processing.



Figure 2: *Concept of Streaming SQL*


```
-- this will actually emit records because we've
placed a bound on time!
SELECT tumble(interval '1 hour') as hour, sum(price)
as sales
FROM orders
GROUP BY hour, store_id;
```

Figure 3: *Concept of Streaming SQL*

Dataflow Semantics

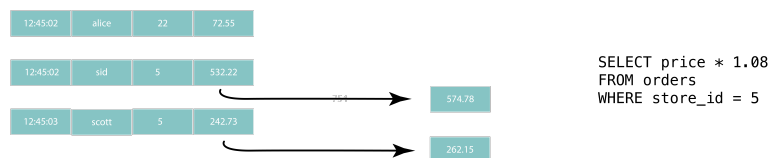


Figure 4: *Row-by-Row Processing*

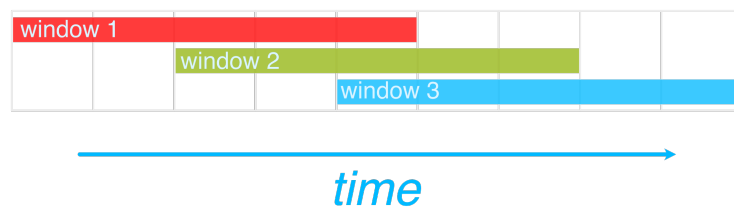


Figure 5: *Sliding windows. Source: Arroyo*

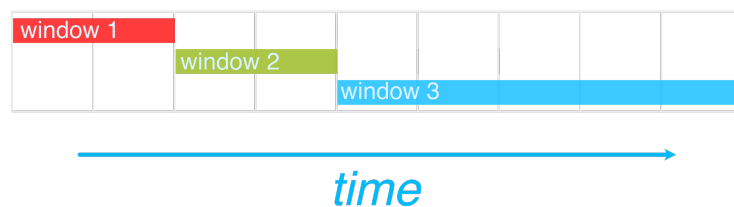


Figure 6: *Tumbling windows. Source: Arroyo*

```
CREATE VIEW orders_agg as (
  SELECT count(*) as orders, customer_id, tumble(interval '1 hour') as
  window
  FROM orders
  GROUP BY customer_id, window
);

--- same for pageviews_agg

SELECT O.window, O.customer_id, C.views, O.orders
FROM orders_agg as O
LEFT JOIN clicks_agg as C ON
  C.customer_id = O.customer_id AND
  C.window = O.window;
```

Figure 7: Apply to Join Operation

Update Semantics

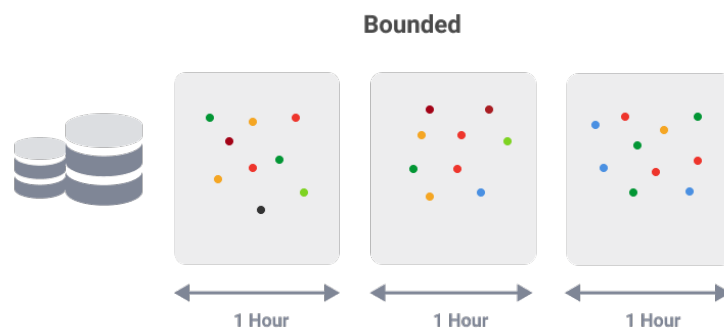


Figure 8: Bounded Data in Time

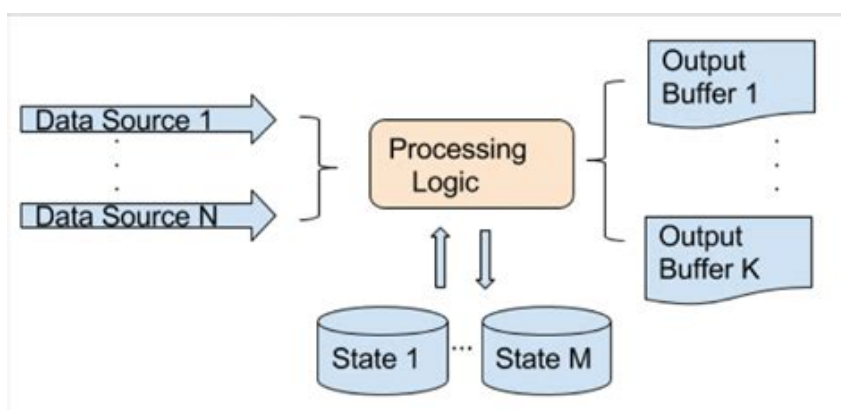


Figure 9: Processing logic of Incremental Updates

2.3.3 Tool



Figure 10: Arroyo

We will incorporate **Arroyo**, a distributed stream processing engine developed in Rust, as part of our technology stack to handle real-time data streams efficiently. Arroyo is designed to support SQL-based queries, making it accessible for processing large volumes of unbounded data streams with ease. Its high scalability ensures that it can handle significant workloads while maintaining **exactly-once semantics**, which is critical for ensuring data accuracy and consistency. This tool will enable us to implement robust, real-time data processing pipelines, making it an ideal choice for applications requiring low-latency and fault-tolerant stream analytics.

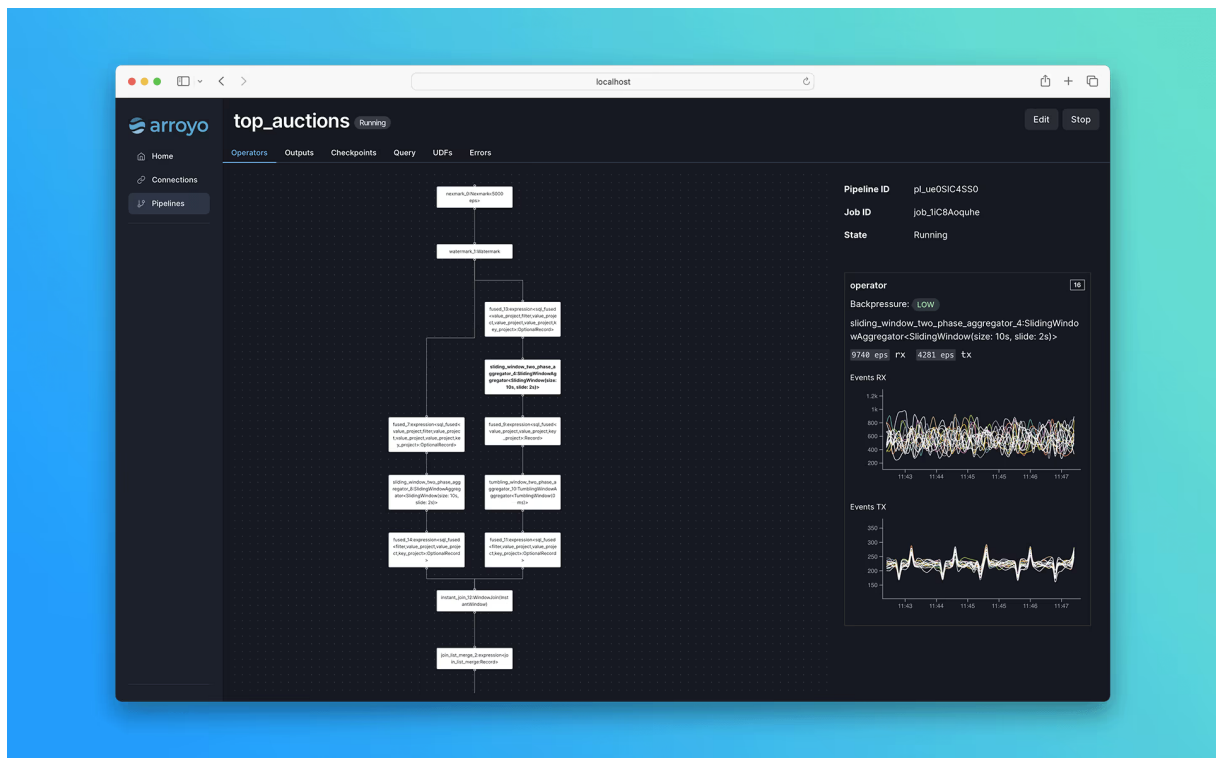


Figure 11: Arroyo Dashboard. Source: Arroyo

3 Implementation

In order to gain deeper insights into the operational dynamics and performance characteristics of decentralized social media platforms, our team has undertaken an innovative demonstration project leveraging the capabilities of Arroyo, a cutting-edge stream processing framework, to develop and implement a sophisticated event processing system specifically designed to monitor and analyze real-time activities across the Mastodon social network federation. This comprehensive monitoring solution serves multiple strategic objectives, including the evaluation of user engagement patterns, the assessment of network performance metrics, and the development of a more nuanced understanding of how decentralized social media architectures function in practice. Through this implementation, we aim to not only demonstrate the practical applications of stream processing technology in social media analytics but also to contribute valuable insights to the growing body of knowledge surrounding alternative social networking models that prioritize user privacy, data sovereignty, and community-driven governance structures.

3.1 Mastodon

Mastodon represents a new approach to social networking through decentralized architecture. Rather than operating as a single centralized platform controlled by one company, Mastodon functions as a network of independently operated servers, known as instances. Each instance runs on open-source software and communicates with other instances through a standardized protocol called ActivityPub [5, 8].

Mastodon has emerged as the most prominent platform within the Fediverse, offering functionality comparable to traditional social media services. Users can share short-form content called "toots" (similar to tweets) and amplify others' content through "boosts" (similar to retweets). The platform also incorporates community-focused features reminiscent of Reddit, with independent communities managed through separate instances, each with its own moderation policies.

The federated architecture of Mastodon serves as the foundation for our demonstration results, utilizing a sophisticated event-driven approach to track and analyze instance activity across the network. From a technical implementation perspective, the system operates by monitoring update events across all participating Mastodon instances through a specialized Server-

sent event protocol [1], which broadcasts these events to a designated endpoint whenever a user performs any update action within the network [3]. Our technical infrastructure systematically processes this event stream by first consuming the incoming events, then performing data extraction to isolate the instance domain information from the event payload, followed by implementing a rolling time-window aggregation mechanism that operates on 5-minute intervals to calculate the frequency of updates originating from each domain instance. The culmination of this processing pipeline results in the identification and selection of the five most active instances, determined by their respective update frequencies, which provides valuable insights into the distribution of user activity across the Mastodon federation network and helps us understand the patterns of engagement within the decentralized social media landscape.

3.2 Set Up Infrastructure

For our experimental infrastructure implementation, we made a strategic decision to deploy Arroyo on a Raspberry Pi model 3B+, a choice that aligns with our objectives to evaluate stream processing capabilities on accessible hardware platforms. Our server configuration encompasses a 1.4GHz 64-bit quad-core ARM Cortex-A53 processor complemented by 1 GB LPDDR2 SDRAM, with network connectivity provided through Gigabit Ethernet, establishing a robust foundation for our testing environment. This deliberate selection of commodity hardware serves multiple research purposes, primarily enabling us to conduct detailed performance monitoring of Arroyo while simultaneously investigating its operational characteristics on consumer-grade equipment. The decision holds particular relevance given the industry's current trajectory toward increased utilization of edge computing devices in stream processing applications. Our implementation offers an interesting comparative analysis opportunity against established solutions like Apache Flink, which operates within the Java Virtual Machine (JVM) ecosystem; in contrast, Arroyo's implementation in Rust suggests potentially lower resource overhead, making it an intriguing candidate for resource-constrained environments. This architectural choice not only supports our immediate research objectives but also provides valuable insights into the feasibility of deploying modern stream processing solutions on accessible hardware platforms, contributing to the broader discussion of edge computing capabilities in data processing applications.

For this project, we decided to use a Raspberry Pi model 3B+ to host the Arroyo, a choice

that aligns with our objectives to evaluate stream processing capabilities on accessible hardware platforms. Some machine specifications for our server includes: a 1.4GHz 64-bit quad-core ARM Cortex-A53 processor, a 1 GB LPDDR2 SDRAM, internet connection through Gigabit Ethernet [7]. This deliberate selection of commodity hardware serves multiple research purposes, primarily enabling us to conduct detailed performance monitoring of Arroyo while simultaneously investigating its operational characteristics on consumer-grade equipment. The decision holds particular relevance given the industry's current trajectory toward increased utilization of edge computing devices in stream processing applications. In contrast with Apache Flink, which was written in Java and execute on Java Virtual Machine (JVM), Arroyo's implementation in Rust suggests potentially lower resource overhead, making it an intriguing candidate for resource-constrained environments. This architectural choice not only supports our immediate research objectives but also provides valuable insights into the feasibility of deploying modern stream processing solutions on accessible hardware platforms, contributing to the broader discussion of edge computing capabilities in data processing applications.

```

ducth@raspi-vn-sg:~$ docker ps --size

```

CONTAINER ID	IMAGE	NAMES	COMMAND	CREATED	STATUS	PORTS
c472af1bc08e	ghcr.io/arroyosystems/arroyo:0.12.0	arroyo_stream-processor.1.xjet2aqa46eo027r6n8pky57d	"/app/arroyo cluster"	13 hours ago	Up 13 hours	5115/tcp
0dfca5697a18	postgres:17-alpine	arroyo_database.1.0lw96adpmlmrw3apo9qj02bya	"docker-entrypoint.s..."	13 hours ago	Up 13 hours	5432/tcp
						SIZE
						147kB (virtual 493MB)
						63B (virtual 269MB)

Figure 12: *Displaying deployment in Docker*

In alignment with contemporary best practices in containerization and microservices architecture, our deployment infrastructure leverages Docker Stack [4] technology to orchestrate and manage the operational components of our system, specifically focusing on two primary services: the Arroyo server implementation and its associated PostgreSQL database instance. This containerized approach not only ensures consistent deployment across different environments but also facilitates sophisticated resource management and monitoring capabilities. Our performance analysis, as illustrated in the accompanying system metrics visualization 12, demonstrates remarkably efficient resource utilization patterns, with both the Arroyo server and PostgreSQL database maintaining notably conservative memory footprints that aggregate to approximately 800 MB of total system memory consumption. This impressive resource efficiency validates our architectural decisions and suggests substantial potential for scaling our implementation across various deployment scenarios, from development environments to production

systems, while maintaining optimal performance characteristics and resource utilization patterns.

Our team source code for the `docker-compose.yaml` can be reviewed on our GitHub repository.

3.3 Arroyo Console guide

After successfully setting up and deploying Arroyo on our server infrastructure, we moved forward with designing and implementing the monitoring pipeline that would help us achieve our project objectives. Arroyo comes equipped with a user-friendly interface that makes the process of developing and deploying data processing jobs surprisingly straightforward and intuitive. To begin creating our monitoring solution, we simply navigated to the Arroyo Console, which serves as the central hub for managing all pipeline-related operations. The interface presents a clean and straightforward layout where users can find the **Create Pipeline** button prominently displayed, marking the starting point for building new data processing workflows. This streamlined approach to pipeline creation reflects Arroyo's overall design philosophy of making stream processing more accessible while maintaining the powerful capabilities needed for complex data analysis tasks. The simplicity of the interface proved to be a significant advantage for our team, allowing us to focus more on the actual implementation of our monitoring logic rather than getting caught up in complicated setup procedures or configuration details.

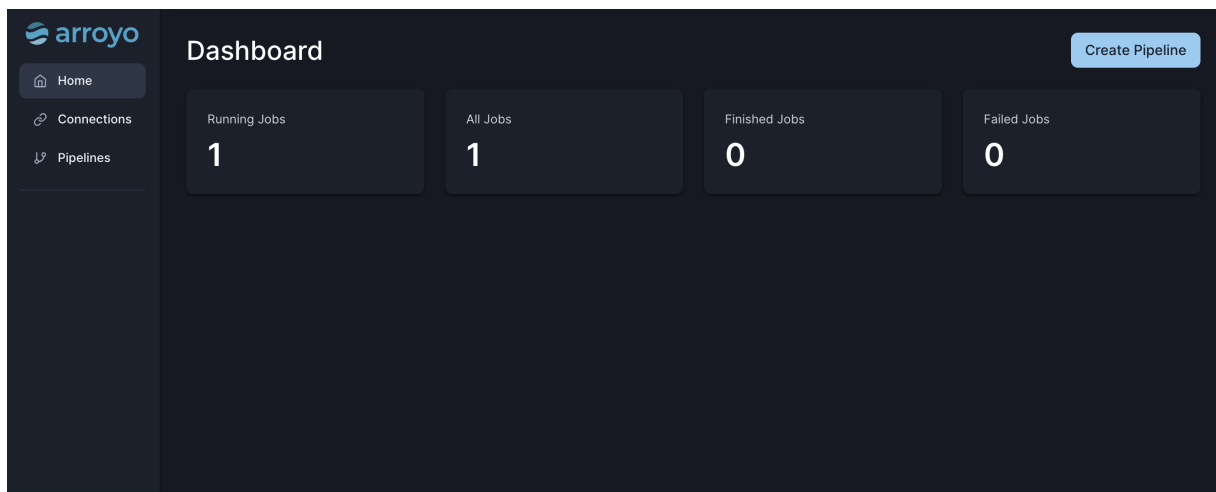


Figure 13: *Arroyo dashboard with our demo job running*

Arroyo offers a straightforward and user-friendly Interactive Development Environment

(IDE), designed to simplify the process of working with data. Within this IDE, users can effortlessly write SQL queries to read and manipulate data, enabling efficient calculations and transformations. The **Preview** feature, accessible via the menubar, allows users to quickly preview the output of their queries, providing immediate feedback and facilitating a more streamlined development cycle. This rapid feedback loop helps users fine-tune their queries before proceeding to deployment. Once the query meets the desired specifications, the **Launch** action can be used to deploy the code for actual execution, making the transition from development to production both seamless and efficient.

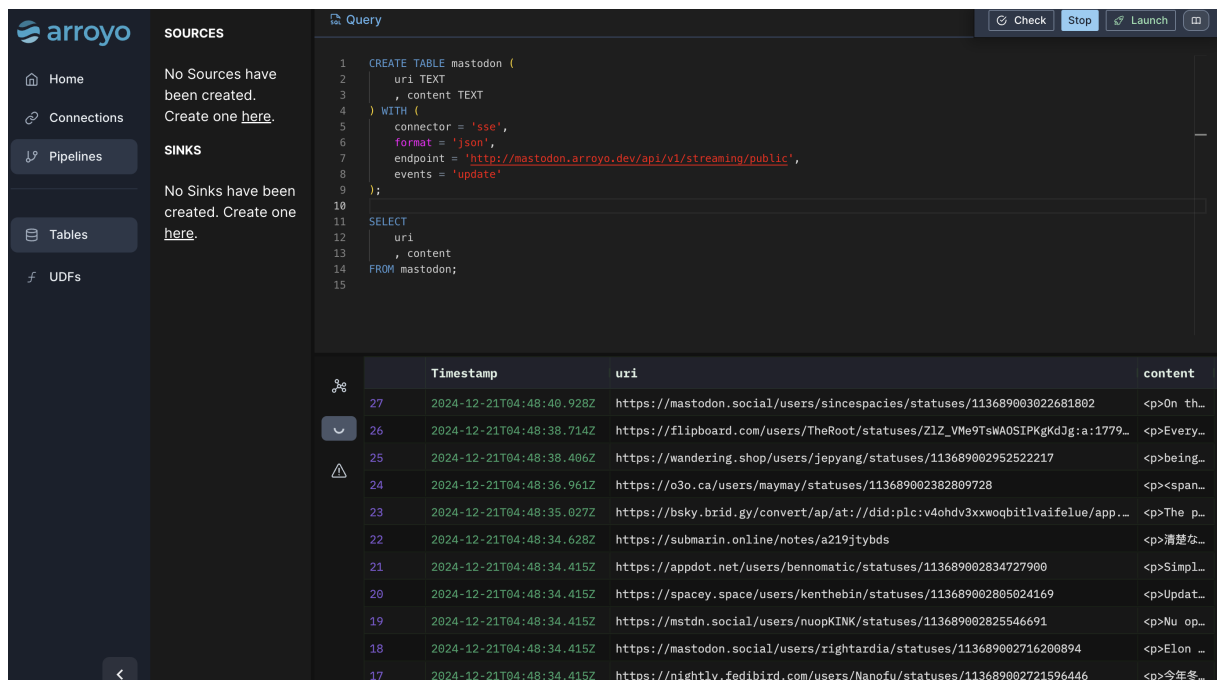


Figure 14: Arroyo IDE

Arroyo also provides a highly intuitive dashboard that offers comprehensive monitoring capabilities for running job details. This dashboard is equipped with a range of features designed to enhance usability and insight: it visually represents the query tree or query plan of the task, enabling a clear understanding of the job's structure; it allows users to tail the outputs of the job, providing real-time visibility into its operation; and it offers detailed technical insights into checkpoints, such as memory usage for each operator, the duration between operator lags, and other performance metrics. These features collectively support a deeper understanding of the job's behavior during deployment, making it easier for operators to quickly identify and address any abnormalities or inefficiencies that may arise during execution.

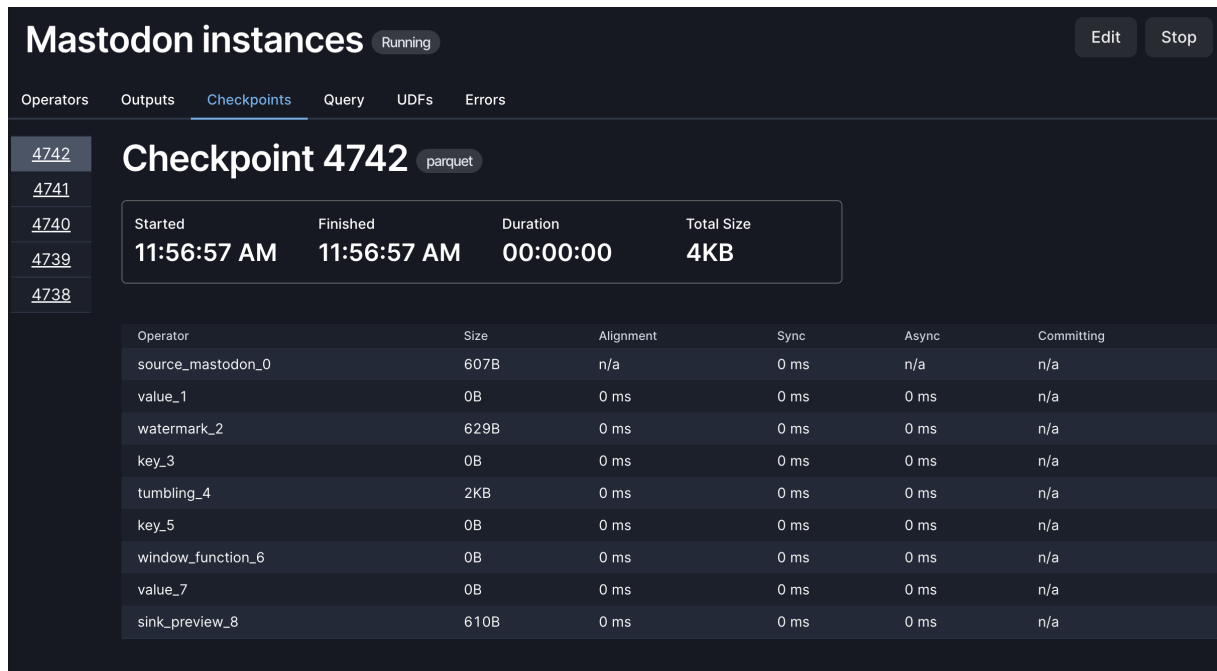


Figure 15: Details about a checkpoint of our demo job

3.4 Develop Instances Count Measurement

We now describe in depth the code to accomplish the aforementioned objective.

```

1
2 CREATE TABLE mastodon (
3     uri TEXT
4 ) WITH (
5     connector = 'sse',
6     format = 'json',
7     endpoint = 'http://mastodon.arroyo.dev/api/v1/streaming/public',
8     events = 'update'
9 );

```

First, this snippet of code will create a **Source** for incoming data into the system. Here, by using table as an abstraction, Arroyo simplifies the process of reading or writing to any other system. Any type of messaging fabrics: Apache Kafka, MQTT or SSE can be represented in a relational model. According to Mastodon documentation, incoming events are JSON-encoding, with a sample structure similar to below. As you can see, the schema of the object has a high complexity.

```

1 {

```

```
2  "id": "108914327388663283",
3  "created_at": "2022-08-30T23:05:46.839Z",
4  "language": null,
5  "uri": "https://letsalllovela.in/objects/e9cebb0c-7c75-414f-9608-20
6  b5628e52d7",
7  "content": "<span class=\"h-card\"><a class=\"u-url mention\" href=\"
8  https://pl.nulled.red/users/disarray\" rel=\"nofollow noopener
9  norereferrer\" target=\"_blank\">@<span>disarray</span></a></span> glad
10 i was able to help",
11 "account": {
12   "id": "464472",
13   "username": "freon",
14   "acct": "freon@letsalllovela.in",
15   "created_at": "2018-08-18T00:00:00.000Z",
16   "emojis": [],
17   "fields": [
18     { "name": "pronouns", "value": "emacs/xemacs (or he/they)", "
19     verified_at": null },
20     { "name": "age", "value": "23.66667", "verified_at": null }
21   ],
22   "media_attachments": [],
23   "mentions": [
24     { "id": "107946650784398271", "username": "disarray", "url": "https
25     ://pl.nulled.red/users/disarray", "acct": "disarray@pl.nulled.red" }
26   ],
27 }
```

However, to achieve our result, we will focus primarily in the field uri. Because of this, Arroyo helps us extract and parse this field into a separate column by setting the format to 'json'. That way, we can skip the step of actual parsing the JSON and get the property (using Arroyo json_get method).

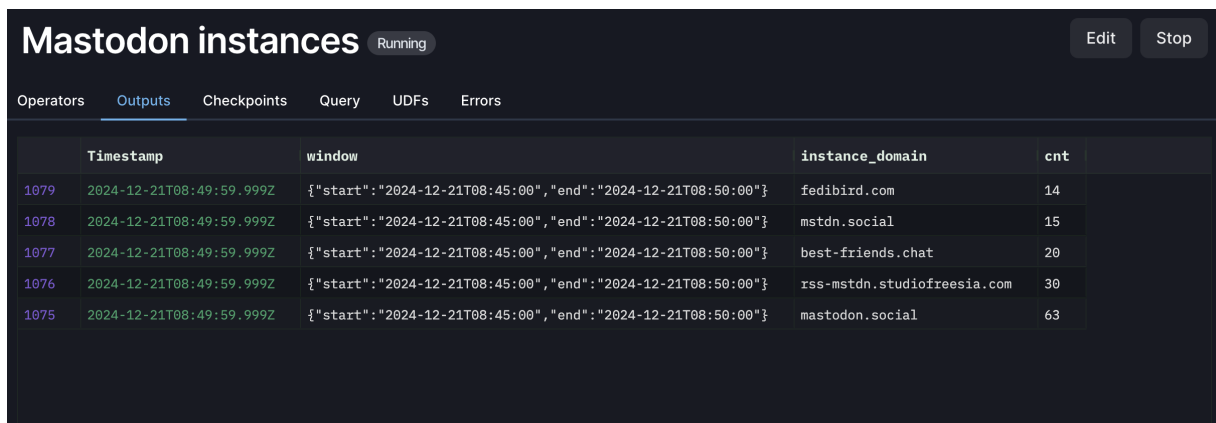
```
1 WITH extract_domain AS (
2   SELECT
3     split_part(
4       substr(uri, strpos(uri, '://') + 3),
5       '/',
```

```
6         1
7     ) AS instance_domain
8 FROM mastodon
9 )
```

Arroyo also supports the syntax of Common Table Expression (or CTE). This will support us with enhancing code readability, by replacing nested SELECT statement with table representing each step of the final transformation. For the first transformation, our team extracted the domain name section from the URI, which should reside between a scheme (such as `https://`) and a forward slash `/`. Arroyo SQL also supports many built-in string manipulation methods, like `strpos` to find a position of a substring within a text, `substr` to extract a substring between a beginning and ending position, and `split_part` to split a string into multiple substring based on a delimiter, and select a result of the splits.

```
1 , instance_domain_count AS (
2     SELECT
3         TUMBLE(interval '5 minutes') AS window
4         , instance_domain
5         , COUNT(*) AS cnt
6     FROM extract_domain
7     GROUP BY window, instance_domain
8 )
9 , instance_domain_sort AS (
10    SELECT
11        window
12        , instance_domain
13        , cnt
14        , ROW_NUMBER() OVER (PARTITION BY window ORDER BY cnt DESC) AS
15    order
16    FROM instance_domain_count
17 )
18 SELECT
19     window
20     , instance_domain
21     , cnt
22 FROM instance_domain_sort
WHERE order <= 5;
```

The next two steps is for our team to perform the complex aggregation on the incoming stream. To achieve our object, we performed aggregation on data based on fixed-size, non-overlapping windows of event incoming interval (which is also called **Tumbling Windows**). Then, we count for the number of updates for each group of the instances. Finally, ROW_NUMBER() is used to rank the instance based on their event counts, partition per interval window. The final SELECT then write out the result, but as we only interest in the top 5 mostly active instances, the WHERE clause does the filtering step.



	Timestamp	window	instance_domain	cnt
1079	2024-12-21T08:49:59.999Z	{"start": "2024-12-21T08:45:00", "end": "2024-12-21T08:50:00"}	fedibird.com	14
1078	2024-12-21T08:49:59.999Z	{"start": "2024-12-21T08:45:00", "end": "2024-12-21T08:50:00"}	mstdn.social	15
1077	2024-12-21T08:49:59.999Z	{"start": "2024-12-21T08:45:00", "end": "2024-12-21T08:50:00"}	best-friends.chat	20
1076	2024-12-21T08:49:59.999Z	{"start": "2024-12-21T08:45:00", "end": "2024-12-21T08:50:00"}	rss-mstdn.studiofreesia.com	30
1075	2024-12-21T08:49:59.999Z	{"start": "2024-12-21T08:45:00", "end": "2024-12-21T08:50:00"}	mastodon.social	63

Figure 16: Top 5 Mastodon instances within our sampling interval

The outcome is a table with a schema in figure 16. Here, it includes a window of processing interval, the domain name of the instance and its corresponding count. Most of the time, it will be the most active instance. It was not a surprise when considering that this instance has a high popularity, and to most people can be considered as the default server for Mastodon.

4 Conclusion

The deployment and operational insights gained from Arroyo highlight its exceptional suitability for managing real-time social media interaction tracking tasks. Its core behavior, centered on efficient stream processing and data analysis, underscores its capabilities as a modern tool optimized for both scalability and precision. Arroyo demonstrates how distributed stream processing systems, such as itself, can effectively bridge the gap between the volume and velocity of social media data streams and the real-time analytics demanded by contemporary platforms.

Arroyo's design principles are apparent in its resource efficiency and ease of use. Written in Rust, the framework offers significant performance benefits over traditional alternatives, as observed in its low latency and minimal resource footprint, even when deployed on commodity hardware like the Raspberry Pi. This outcome not only establishes Arroyo as a strong candidate for large-scale applications but also showcases its adaptability to constrained environments, enabling broader applications ranging from edge computing to traditional cloud environments.

The system's support for SQL-based streaming queries simplifies interaction and operational complexity. By abstracting the intricacies of continuous data ingestion and processing, Arroyo provides an accessible yet robust mechanism for monitoring and analyzing dynamic datasets, such as the decentralized activities within Mastodon's federated network. This feature enhances its usability for researchers, developers, and businesses aiming to derive actionable insights from real-time data sources.

Moreover, its integration with widely used tools and protocols, such as Docker for deployment and Server-Sent Events (SSE) APIs for real-time data fetching, ensures streamlined operational workflows. Arroyo's built-in IDE and dashboard further simplify development and monitoring, offering intuitive interfaces for query execution, job deployment, and resource tracking. These features collectively foster an ecosystem that minimizes barriers to entry while maximizing productivity.

The demonstrated real-time tracking pipeline offers a clear testament to Arroyo's strength in aggregating and analyzing unbounded data streams effectively. For instance, the ability to dynamically identify the most active Mastodon instances within specific time windows provides not only operational insights but also a foundation for exploring patterns within decentralized

networks. Such insights emphasize Arroyo's potential to contribute significantly to studies and applications focused on evolving paradigms in social media and real-time analytics.

In summary, Arroyo's behavior reflects its potential as a versatile tool, efficiently addressing the challenges associated with real-time data streams. Its implementation in this project affirms its practical utility and paves the way for its application in diverse domains requiring similar analytic capabilities.

5 Report links and details

We also attached here, source codes and a presentation video based on this report.

1. GitHub repo: <https://github.com/ducthuy-ng/bigdata-sem241>
2. Presentation video: https://www.youtube.com/watch?v=z8z_jYJGqDE

References

- [1] Server-sent events, 2024. URL <https://html.spec.whatwg.org/multipage/server-sent-events.html>.
- [2] Ra'ed M. Al-Khatib, Mohammed Al-Betar, Mohammed Awadallah, Khalid Nahar, Mohammed Abu Shquier, Ahmad Manasrah, and Ahmad Doumi. Mga-tsp: Modernized genetic algorithm for the traveling salesman problem. *International Journal of Reasoning-based Intelligent Systems*, 11:1, 01 2019. doi: 10.1504/IJRIS.2019.10019776.
- [3] Mastodon gGmbH. streaming api methods - mastodon documentation, 2022. URL <https://docs.joinmastodon.org/methods/streaming/#public>.
- [4] Docker Inc. Deploy a stack to a swarm, Dec 2020. URL <https://docs.docker.com/engine/swarm/stack-deploy/>.
- [5] Lucio La Cava, Sergio Greco, and Andrea Tagarelli. Understanding the growth of the fediverse through the lens of mastodon. *Applied Network Science*, 6(1), September 2021. ISSN 2364-8228. doi: 10.1007/s41109-021-00392-5. URL <http://dx.doi.org/10.1007/s41109-021-00392-5>.
- [6] Annu Lambora, Kunal Gupta, and Kriti Chopra. Genetic algorithm- a literature review. In *2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon)*, pages 380–384, 2019. doi: 10.1109/COMITCon.2019.8862255.
- [7] Raspberry Pi. Raspberry pi documentation - raspberry pi hardware. URL <https://www.raspberrypi.com/documentation/computers/raspberry-pi.html>.
- [8] W3C. Activitypub, Jan 2018. URL <https://www.w3.org/TR/activitypub/>.