APACHE FLINK

Conceptual Architecture

Authors

Muhammad Omar	omar624@my.yorku.ca
Ratul Momen	ratulm@my.yorku.ca
Ruth Bezabeh	ruttkas@my.yorku.ca
Nishiket Singh	nishiket@my.yorku.ca
Corneille Bobda	nelius1k@my.yorku.ca
Bartolomeo Gisone	bgisone@my.yorku.ca

Abstract

Apache Flink, an open-source, distributed data processing framework, has emerged as a versatile and powerful tool in the domain of data stream processing and batch data processing. This report provides a comprehensive overview of Apache Flink’s conceptual architecture, exploring its key features and use cases

The report begins by introducing the fundamental components of Apache Flink's runtime architecture, such as the Job Manager, the Task manager, the Resource Manager and the Dispatcher. The report also mentions Flink’s architectural styles and design patterns that were observed when conducting our research. Further, the evolvability of Flink is discussed; notably how Flink proceeds with its development and the different responsibilities assigned

Furthermore, the report delves into the diverse range of use cases that Apache Flink caters to, including real-time data analytics and event-driven applications. These use cases display how Apache Flink is used by existing companies to enhance their systems. They emphasise Flink's flexibility and scalability, making it suitable for applications across various industries.

Additionally, the report illustrates diagrams that showcase Apache Flink’s structure with all its features. Apache Flink's external interfaces are also highlighted, the report addresses the kind of information that is transmitted to or from Apache Flink’s runtime environment.

The report concludes by summarising the key findings of our research. It also touches upon the future prospects and evolving trends in the field of stream processing. Finally, the lessons learned by the researchers are discussed.

In a data-driven world where real-time insights and batch processing are crucial for business success, Apache Flink stands as a robust, open-source solution capable of meeting the evolving demands of modern data processing applications. This report offers a valuable insight into the capabilities and potential of Apache Flink, making it a valuable resource for organisations and data engineers seeking to harness the power of stream processing and batch data processing for their data-intensive workloads.

Introduction and Overview

This report discusses the Apache Flink framework and its conceptual architecture. This report aims to offer a clear and thorough understanding of Apache Flink, its core components, architecture, dataflow, concurrency, and higher level concepts such as evolvability, goals, use cases, and testability.

Apache Flink, developed by the Apache Software Foundation, is an open source, distributed stream processing engine designed to process and analyse continuous stream or batched data in real time, running continuously, with minimal downtime, and processing data as it is ingested.

Distributed Stream Processing Engines are designed to process and analyse continuous streams of data in real time. These engines enable organisations to gain insight, make decisions, and respond to events as the data is generated, rather than in batch processing, after it is ingested.

Ingested data are partitioned into smaller segments, which are then distributed to a cluster of machines or processors. Each node is responsible for processing a subset of the data, performing various operations, including: filtering, transformation, aggregation, and event processing. Once the operations are complete, the processed data can be sent to various destinations, such as databases, dashboards, or downstream systems.

Key features of Apache Flink include:

1. **Dynamic Stream and Batch Processing:** Apache Flink supports stream and batch processing. It is capable of seamlessly switching between these two modes, making it effective in a wide range of data processing requirements
2. **Event Time Processing:** Flink allows the handling of out of order data and late arriving events, which is essential for accurate and reliable data analysis
3. **Fault Tolerance:** Flink comes paired with fault tolerance mechanisms to ensure integrity of data. It uses distributed snapshots and state checkpoints to recover the state of the machine in the event of a failure.
4. **High Throughput:** Flinks is designed for high throughput, enabling it to process larger volumes of data with relatively low response times.
5. **Scalability:** Flink can be scaled horizontally, meaning that by adding more resources/processors allows for a higher volume of data. This makes it suitable for big data applications.

Report Organization	
Introduction to Apache Flink	This section will provide a brief introduction to Apache Flink, how it works, its functionality, and key features.
Architecture	This section will discuss the overall structure of Apache Flink, describing each major component and the interactions among them.
Concurrency	The section will cover the concurrency present in the Apache Flink system. It will describe how the concurrency works, and the benefit that it brings.
Dataflow	This section describes the global control flow of data during execution time
Architectural Styles	This section will introduce the architectural styles used in Apache Flinks architecture
Design Patterns	This section will introduce any design patterns present in the Apache Flink architecture, describing where and how it is used.
Evolvability	This section will discuss and cover the evolution and evolvability of Apache Flink, with an emphasis on the major players in its evolution.
Implications for Division of Responsibility	This section will discuss the implications and assumptions made towards participating developers. This section will list various types of developers and contributors, and briefly explain each of their responsibilities and implications.
Derivation Process and Alternatives	This section will cover the derivation process of how the findings came to be discovered. Furthermore, it will discuss any alternatives that were pitched, and why they were not deemed fit.
External Interfaces	This section will provide an overview of the information that is transmitted from and to the system. This section will focus on the channels through which information is passed through, as well as the content of the information.
Use Cases	This section will provide a couple use cases which illustrate the use of the Apache Flink system. This section will provide overviews of the specific use cases and describe how and precisely where Apache Flink is used.
Data Dictionary	This section is a glossary that defines all key terms used in this report.

Naming Conventions	This section will list any naming conventions or abbreviations used in this report. It will define the convention or abbreviation, and indicate where in the report it is used.
Conclusions	This section will summarise all key findings and any proposals for the future.
Lessons Learned	This section will document any regrets or noteworthy lessons.
References	This section will list all references to any sources used in the development of this report and research for the conceptual architecture.

Architecture

Component Breakdown

JobManager

A fundamental part of the Apache Flink system. It is responsible for coordinating and managing Flink jobs. The JobManager plays a role in many important functions such as:

1. Job Submission and Execution: JobManager accepts and manages job submissions, receiving jobs from the client and generating a job execution plan.
2. Job Monitoring: JobManager provides job monitoring ability. It collects, reports, and displays job related information, such as progress, execution time, and resource utilisation.
3. Failure Recovery: Using a high availability setup using multiple JobManagers, Flink prevents single point of failure. A HA setup will make use of multiple JobManagers, with one being the leader and the other JobManagers stay dormant. In the event that the lead JobManager fails, one of the dormant JobManagers will take lead and continue execution.
4. Checkpointing: fault tolerance is enforced with checkpointing. The JobManager coordinates the process of creating checkpoints, allowing Flink to recover from failures.

TaskManager

TaskManagers execute the tasks in the Flink job. It is also responsible for the exchange of data streams.

1. Task Execution: Responsible for executing the individual tasks of a Flink job.
2. Parallelism: TaskManagers can manage multiple tasks concurrently and distribute the workload across multiple processors.
3. Data Shuffling: During a task, when required, the TaskManager can redistribute data among parallel tasks.
4. Data Buffering: TaskManagers can buffer data as it is ingested, meaning that it can maintain a portion of the data before it is passed to another task.

There can be multiple TaskManagers in a Flink application. Each TaskManager runs on a separate machine/processor, working together with other TaskManagers to complete tasks in parallel.

Resource Manager

The ResourceManager plays a key role in managing and allocating resources for Flink. It supervises the distribution of resources ensuring efficient execution of Flink jobs. The ResourceManager is a major player in large scale stream and batch processing workload.

1. Resource Allocation: ResourceManager allocates resources such as CPU, memory, and task slots. Task slots are the computational units that are used by TaskManagers. The Resource manager ensures that resources are evenly and appropriately allocated for the efficient execution and completion of multiple Flink jobs.
2. Dynamic Scaling: ResourceManager is capable of distributing resources dynamically. This means that as resource requirements change in a task, the resource manager can shift resources on the go.
3. Scheduling: ResourceManager assists in job scheduling by ensuring the Flink job is being assigned to the appropriate task manager with appropriate resources.
4. Yarn Integration: ResourceManager can be configured to work with YARN ecosystem

Dispatcher

The Dispatcher assists in the submission of Flink jobs. Upon a submission, the Dispatcher creates a new JobMaster for that job. It also provides information about job executions.

JobMaster
The JobMaster is responsible for managing the execution of a JobGraph. Each Flink job has its own job master. Some of its key tasks include:

1. Scheduling: JobMaster is responsible for generating an execution sequence for a Flink job. It decides the order of the Jobs operations and data exchange.
2. Task Management: JobMaster tracks the status, progress, and dependencies of the tasks in a Flink job.
3. Dynamic Scaling: JobMaster can dynamically add or remove TaskManagers as it deems fit through analysing workload or available resources.

Concurrency

- Parallelism allows concurrency of data processing and takes advantage of the available cluster resources. You can configure the degree of parallelism of operators individually to control how many parallel instances of the operator will be running. Tasks are distributed among multiple task managers and each task manager can run multiple parallel operator instances within available slots. The number of slots on each Task Manager determines the maximum parallelism for the entire Flink cluster.
- Flink handles data partitioning and distribution and is responsible for input data making it way to parallel instances of operators which allows operators to process different subsets of data concurrently.
- Flink supports stateful processing allowing operators to maintain state across multiple events. This state is partitioned and distributed among Task Managers which enables operators to work on different parts of the state concurrently.
- Checkpointing mechanism allows for concurrent snapshots of operator states to be taken, which is useful in the event of failures, where individual operator states can be

recovered concurrently. These checkpoints are taken periodically and independently for each operator.
- It is crucial in event time processing to handle out-of-order events efficiently. Flink processes events with different timestamps concurrently based on watermarking and windowing i.e embedding a unique identifier within the data.
- The execution plan is optimised by chaining together operators that can be executed together without shuffling data between them, hence reducing data movement and improving concurrency.
- Flink also supports asynchronous I/O operations within operators to perform non-blocking, concurrent operations like external service calls, multithreading, event loops, message passing in actors etc.
- Apache Flink is designed for high levels of concurrency, allowing you to process data efficiently and in parallel across distributed Task Managers. This is a fundamental aspect of Flink's performance and scalability, enabling it to handle large-scale data processing workloads with low-latency and high throughput. The degree of concurrency can be configured based on your specific requirements and resources available in your Flink cluster.

Dataflow

- Unbounded data streams are data streams that have no end, they're continuously processed and usually require the events to be processed in a specific order in order for result completeness to be reasoned
- Bounded data streams are data streams that have a defined start and end, and are usually ingested completely before computations begin or end. They also do not need to be ordered because bounded data streams can be sorted after fully ingested.
- Stateful Stream processing. Some operations remember information across multiple events, hence states are used. As well, states are used to make apache Flink fault tolerant.
- JobManager: Upon receiving a job submission, JobManager is responsible for initialising the job and ensuring that required resources are available to execute said job. JobManager coordinates the execution of the job by assigning these jobs to Task Manager within the cluster. It determines how the data moves across the clusters and controls the significant part of the data flow. It is also responsible for the coordination of the checkpointing process and handles recovery in the case of a failure. It ensures that the state of the failed task is restored from the latest checkpoint.
- Taskmanager executes the tasks of a dataflow, and buffers and exchanges the data streams. The Taskmanager can contain multiple task slots which are individual components of each processing task
- StateBackend assists with the savepoint system that allows for continuous updates to whatever database is attached to the program. This allows for states to be saved to begin with which helps for stateful data processing, especially with event driven applications.
- Checkpoint coordinator takes in data streams and utilises the savepoint system to continuously store the processed information into whatever database is linked to it.

Architectural Styles

Client-server: Flink clients request jobs and Flink provides stream processing services

Pipe - filter: Within the TaskManager there are multiple operations that perform transformation on an ordered stream of data and pass downstream.

Design patterns

Facade design pattern: The Flink client only sees the JobManager and the JobManager abstracts away the other systems, reducing complexity.

State design pattern: The JobManager transforms the job graph submitted by the client to an ExecutionGraph. This ExecutionGraph has a job status associated with it. During normal execution it goes from created, scheduled, deploying, initialising to finished. In case of failure by the system it can enter failing, restarting and failed, while during user cancellation of a job the state will be cancelling then cancelled.

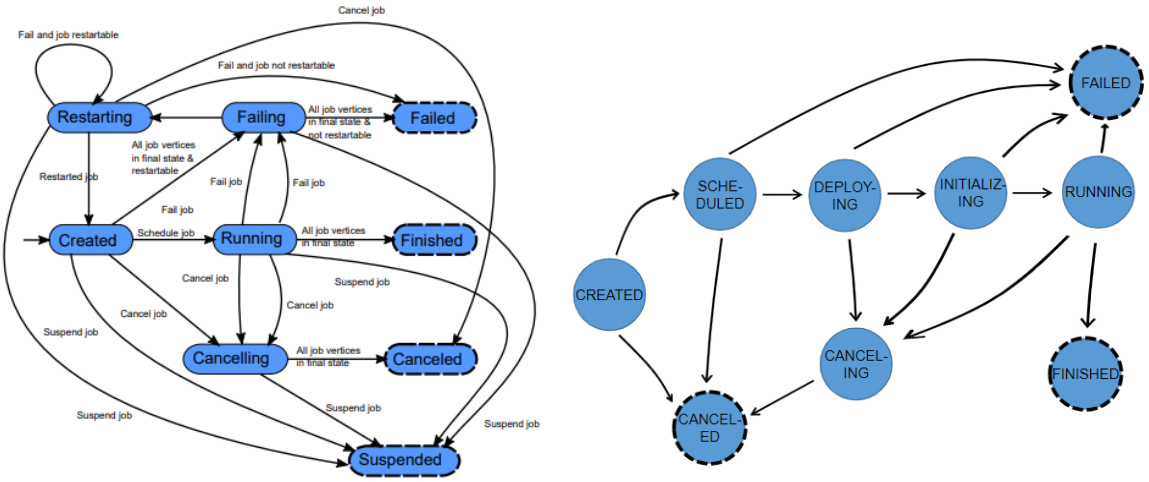

Fig 1.1 The states and transformations of an ExecutionGraph

Master-slave design pattern: The JobManager invokes the task managers and multiple independent task managers run in parallel to provide the same service.

Evolvability

Community Involvement: Since Apache Flink is open source, the software is developed, updated, and maintained by a diverse community of programmers and testers. Apache Flink has a Technical Steering Committee (TSC) that oversees the project's evolution and direction. TSC helps prioritise development and makes important decisions of Flinks updates.

RoadMap: Flink follows a release schedule that includes both major and minor releases. The roadmap is typically defined by the community and outlines the planned features and improvement for future releases.

Feedback: Feedback from users and real world use cases play a role in developing Flink. This feedback helps identify performance bugs, missing features, and issues users are

experiencing. User feedback is assisted with JIRA and mailing lists. JIRA enables users to track and report bugs, and mailing lists enable users to request features and engage in discussions about the platform. These softwares create fast, efficient, and organised feedback for Apache Flink.

Integration: Flink has and continues to evolve by integrating itself with other technologies. Apart from running as a standalone application, Flink integrates with common resource managers such as Hadoop YARN, Apache Mesos, and Kubernetes.

Optimization: Developers continue to work on optimising the performance and scalability for the software. This optimization includes resource utilisation, latency, and enhancing throughput. Developers also continue to evolve security features to protect data and applications, keeping up to speed with emerging technologies in the data processing and streaming sector to remain relevant and usable in the world of rapidly evolving technology.

Implications for Division of Responsibility

Developer	Responsibilities	Implications
Application Developer	Responsible for designing the data processing logic, define transformations, manage state, and specify windowing operations	Need advanced understanding of Flink API and data processing concepts.
System Administration	Responsible for deploying, managing, and maintaining the Flink cluster. Allocate resources, monitor health, and ensure high availability. Configure Flink for optimal resource utilisation, handle scaling, and ensure failure prevention	Expert in cluster management and coordination tools.
Framework Developer	Contribute to the framework itself. Work on enhancing core system, performance improvements, features, and fixing bugs	Need a deep understanding of Flink codebase, distributed systems, and stream processing. Work on evolving the project, addressing community needs, and ensuring stability.
Data engineers	Work on data connectors and integration with external data sources. Ensure data can be ingested into and extracted from Flinks applications seamlessly.	Advanced knowledge in data integration tech and knowledge of Flinks connector APIs.
QA Engineers	Testing Flink applications and the framework itself. Develop and perform tests, validate correctness and reliability of Flinks features	Strong understanding of Flink’s goals, features, and behaviour. Contribute to stability and reliability of Flink applications
Security	Identify and mitigate security vulnerabilities. Ensure data privacy, access control, and authentication	Expert in data security, encryption, and authentication.

Derivation Process and Alternatives

The conceptual architecture was derived by studying the supplied documentation from the Apache organisation and its team. The documentation supplied much information in terms of the components that were present within the system and the features offered. It went into much depth about the internal mechanisms and the names and parts within that allowed Apache Flink to offer its features. On top of that it further delved into the concepts and the unique properties that were required in order to make the entire system run. In terms of the actual alternatives offered, there seems to be quite a number of them, as data processing engines are readily available. Some of the alternatives seem to come from within the Apache organisation as well. Hadoop, Samza, Apache Spark and Apache Storm are all frameworks that offer data processing. Although most of the other frameworks seem to work just as well, they usually focus on either batch processing or stream, with the only exception being Apache Spark. The alternatives as well offer their own filesystems while Apache Flink does not.

Diagrams

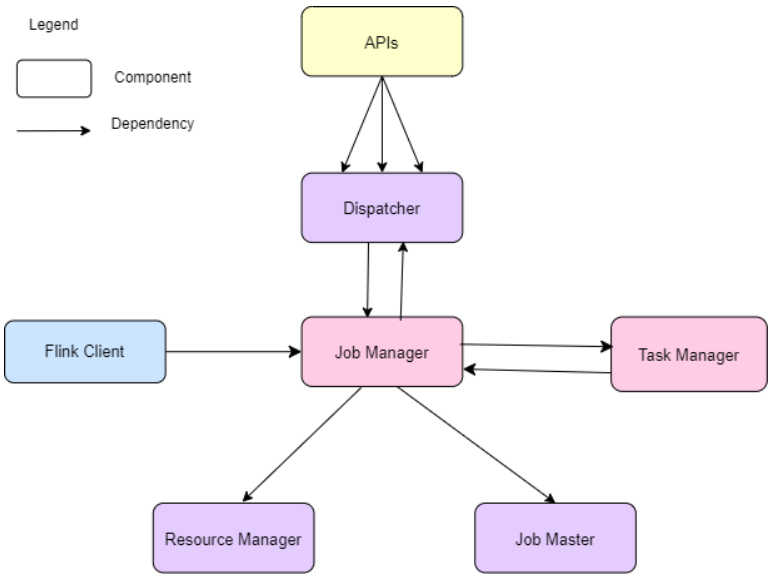

Fig 1.0 Dependency diagram of Flink's major components

External Interfaces

Flink can be configured to work with different interfaces.

Flink's User Interface: Used to monitor the status of the cluster and running job.
High Availability Service Provider: Used to recover from faults by having multiple JobManagers on standby. It can be implemented using Zookeeper or Kubernetes.
File Storage and Persistency: Used for checkpointing and recovery by connecting to external file storage systems.
Resource Provider: Used to deploy Flink in a distributed cluster using Kubernetes or YARN
Metrics Storage: Used to save internal and job specific metrics.
Application-level data sources and sinks: Used for colocating frequently used data with Flink.

Use Cases

UC1: Pinterest Real-Time Experiment Analysis

Pinterest runs thousands of experiments a day and uses Flink to filter and aggregate the data collected in real time. Real-time processing is especially useful because the outcomes of these experiments need to be acted upon as soon as possible. Experiments performing well and generating click-through are re-ramped beyond the initial 3 day experiment runtime and allocated to a larger group of users while those performing poorly are cancelled to prevent a decline in customer retention.

Pinterest uses two separate Flink programs, one for filtering and one for aggregation. Kafka Topics for events, which are any user action on the site, and experiment activations, which are records created when users are entered into experiments, are data sources for the filtering Flink program. Two filtering jobs are then performed before the data is outputted into ‘filtered events’ and ‘filtered experiment activations ' kafka topics to be queued into the aggregation Flink program where more operations are performed. The final processed data is sent to Pinterest's analytics engine to be analysed.

Stateful processing is used to keep a 3 day timer keyed state for each experiment as well as keeping count, while timely processing is used in a 15 minute tumbling window operation that uses processing time to send processed data every 15 minutes.

Filter events job - keeps only business critical events as well as strips any data fields not necessary for analytics. The progress is saved every five seconds for checkpointing.
Filter experiment activations job - Keeps activations that were triggered within the last three days or allocated to a larger group within the same time frame.

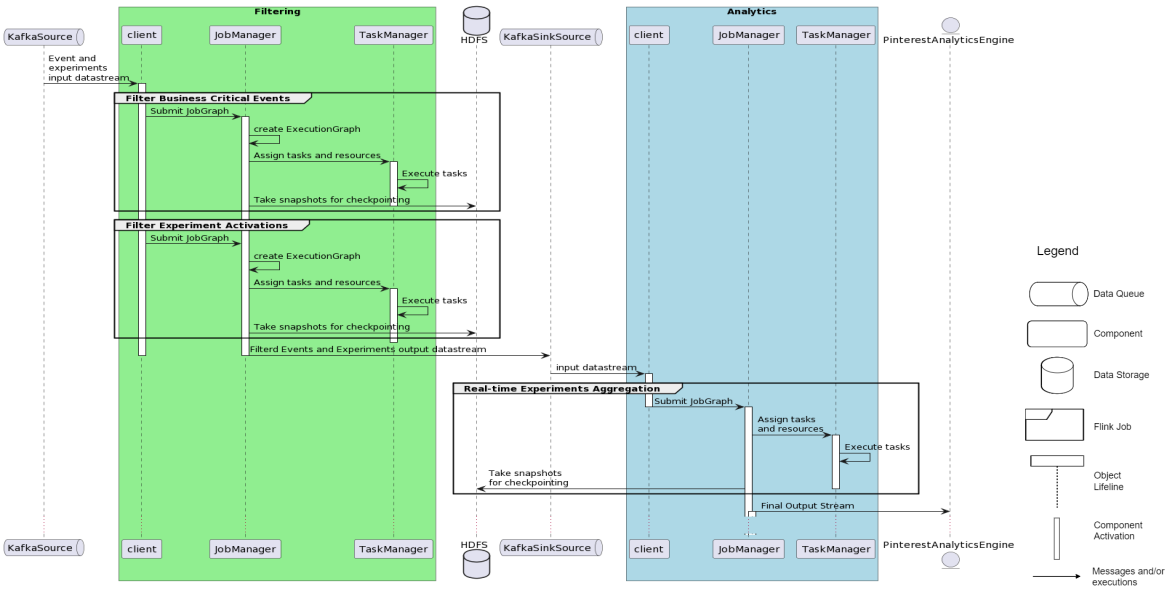

Fig 2.0 Sequence diagram showing the flow of process within and between pinterest’s two Flink applications

Operations by Real-time aggregation Job:

De-duplicate events - sends the first instance of a user event and discards subsequent events with the same userId, event_type and timestamp.
Find first trigger time - record only the first activation per user per experiment since a user could be entered into the same experiment multiple times.
Join activations with events and De-duplicate joined events - join user events and activations using user and activation IDs and discard duplicate joined events
Numerator Computer and Denominator computer - Keep count of events in the 3 day time frame.

Finally the processed data is sent to pinterests analytics engine using rest protocols every 15 minutes- using a 15 minute tumbling window that uses processing time.

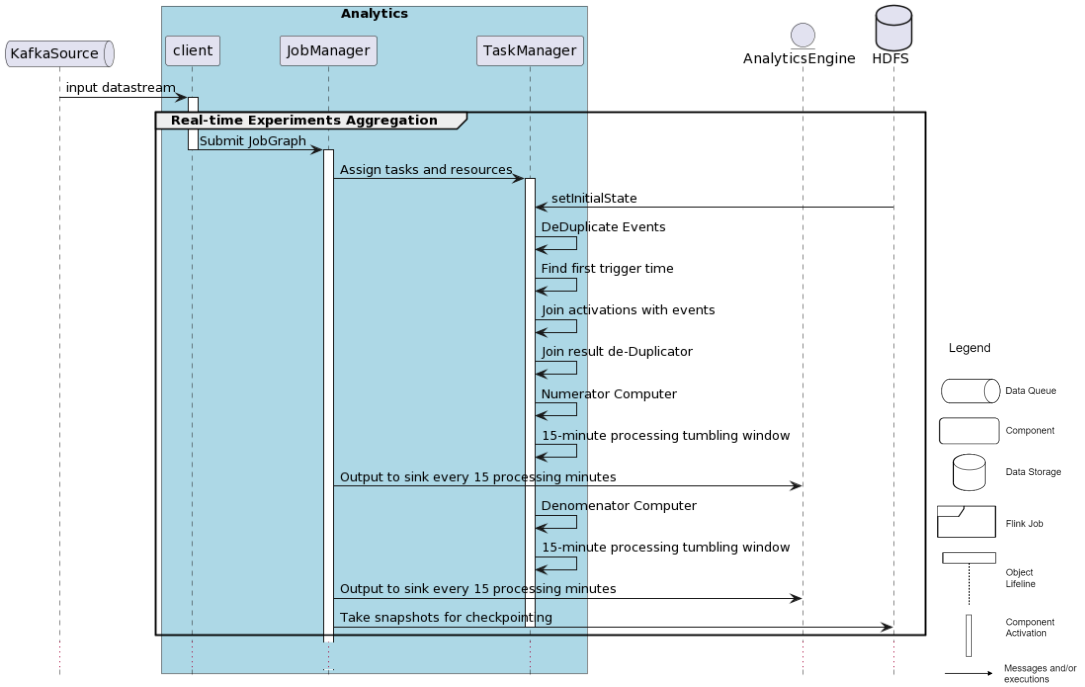

Fig 2.1 Sequence diagram showing the flow of process within pinterest's Flink aggregation application

UC1: Bouygues Telecom Real-Time Monitoring and Analysis

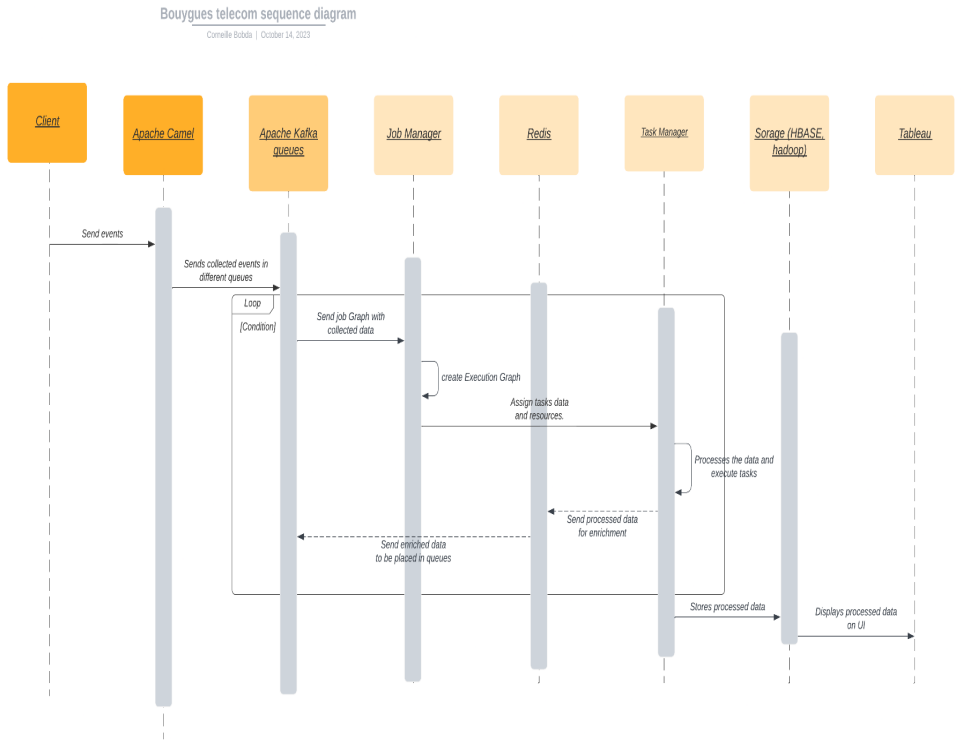

Fig 2.2 Sequence diagram showing process flow Bouygues Telecom Real-Time Monitoring and Analysis

Data Dictionary

- A **Job** is a running flink application
- The **Job Graph** is the data processing pipeline through which the event data is streaming
- An **operator** transforms event streams in the pipeline
- A **sink** is where the stream processing business logic goes
- A **snapshot** is a consistent collection of all task managers local states

Naming Convention

- **PascalCase:** PascalCase, which is used by removing spaces and capitalising the first letter of each word, is used in this report whenever Flink components are mentioned.
- **HA** (Used in the description of JobManager): High Availability

Conclusions

A summary of your key findings and proposals for future directions.

Lessons Learned

References

1. https://nightlies.apache.org/flink/flink-docs-release-1.17/
2. https://www.youtube.com/watch?v=DkNeyCW-eH0&t=37s&ab_channel=GOTOConferences
3. https://jbcodeforce.github.io/flink-studies/cep
4. *Real-time experiment analytics at Pinterest using Apache Flink* - https://medium.com/pinterest-engineering/real-time-experiment-analytics-at-pinterest-using-apache-flink-841c8df98dc2
5. *A brief history of time with Apache Flink* - https://www.youtube.com/watch?v=izYsMQWeUbE&t=896s
6. https://delinea.com/products/secret-server/features/distributed-engines#:~:text=A%20Distributed%20Engine%20is%20composed,for%20a%20particular%20network%20area.