Apache Flink Conceptual Architecture: Technical Report
Team: FlinkForce

Rafael Dolores[1] **Hashir Jamil**[1] **Alex Arnold**[1] **Zachary Ross**[1]
Nabaa Gaziy[1] **James Le**[1] **Walid AlDari**[1] **Maaz Siddiqui**[1]
[1]Department of Electrical Engineering & Computer Science, York University
{rafd47,hashirj,alex290,zachross,Nabaagz,jamesqml,walidald,msiddiqi}@my.yorku.ca

Abstract

This report examines the conceptual architecture of the open-source stream processing framework Apache Flink [1]. Flink is a streaming-first runtime that supports both batch processing and data streaming programs. It provides APIs in Java, Scala and Python, and supports very high throughput and low event latency at the same time. Apache Flink has multiple key stakeholders that play a significant role in ensuring the project remains successful. The high-level conceptual architecture of Apache Flink is examined, detailing its main components such as task managers, job managers, Flink clusters, and the Flink client. The important security measures that Apache Flink has in place to ensure data privacy, and comply with regulatory standards are also reviewed. Multiple use cases for Apache Flink are detailed such as real-time stream processing, batch processing, event-driven monitoring, and fault tolerance.

1 System Overview

1.1 High-Level Functionality

Apache Flink is a streaming-first runtime that supports both batch processing and data streaming programs. It provides APIs in Java and Scala, and supports very high throughput and low event latency at the same time. Flink also supports event time and out-of-order processing, flexible windowing, fault-tolerance with exactly-once processing guarantees, natural back-pressure in streaming programs, and libraries for graph processing, machine learning, and Complex Event Processing. Additionally, Flink features built-in support for iterative programs in the dataset API, custom memory management for efficient and robust switching between in-memory and out-of-core data processing algorithms. In summary, Apache Flink is a powerful and versatile framework for developing real-time and batch processing applications [2].

1.2 Key Stakeholders

Apache Flink has many key stakeholders. Many companies including Netflix, Uber, and Amazon use Apache Flink in their software systems [3–6]. The open source project is overseen by the Apache Software Foundation and has 1000+ contributors on its GitHub [2, 7]. Each of these stakeholders are invested in the success of Apache Flink and play a significant role in ensuring that the open source project continues to be successful. More details about the stakeholders' roles are discussed in sections 1.4 and 4.3.

1.3 Historical Evaluation

Apache Flink's journey began in 2010 when the research project "Stratosphere: Information Management on the Cloud," led by Volker Markl at the Technical University Berlin, laid its foundation. Flink initially emerged as a fork of Stratosphere's distributed execution engine and gained recognition as it joined the Apache Incubator in March 2014, subsequently being renamed Apache Flink. In March 2015, Apache Flink 0.9 marked a significant milestone by introducing the Flink Streaming API, enabling real-time data processing. The year 2016 witnessed the release of Flink 1.0, solidifying Apache Flink's stability and introducing an SQL-like querying API. Over the years, from 2016 to 2018 (versions 1.1 to 1.6), Apache Flink prioritized performance, fault tolerance, dynamic scaling, and event time processing. Between 2018 and 2021 (versions 1.7 to 1.14), Apache Flink evolved significantly, enhancing query optimization, offering Python support, streamlining deployment, integrating with the ecosystem, and ensuring exactly-once processing semantics. In the latest chapters of its development from 2021 to 2023 (versions 1.15 to 1.17), Apache Flink has continued to advance. Notably, it introduced a Delete and Update API in Flink SQL for batch processing, along with performance, stability, and usability improvements in batch execution. Version 1.17 introduced the "gateway mode" for SQL client interaction, allowing users to submit SQL queries to a SQL Gateway [8]. In the realm of stream processing, these updates also brought enhancements in streaming SQL semantics, checkpoint improvements, watermark alignment enhancements, and StateBackend upgrades. Apache Flink's journey underscores its ongoing commitment to evolving and meeting the needs of data processing in both batch and stream processing scenarios [9, 10].

1.4 Contributions

The Apache Flink community employs a diverse range of communication platforms to effectively engage with its users, tailoring the interaction to individual needs. GitHub serves as a hub for developers and users alike, enabling pull requests, code reviews, and development discussions. For issue tracking, JIRA is utilized, ensuring a systematic approach to problem resolution. Stack Overflow functions as a discussion board where users can seek answers to common queries or post new questions for community assistance. Furthermore, the community maintains an informative Wiki documentation, offering valuable resources for users to deepen their project understanding. Lastly, the Apache Flink community extends a helping hand to newcomers, providing mentorship and support to facilitate their integration into the project. This multi-faceted approach fosters a collaborative and supportive atmosphere, enhancing the overall experience for community members [11].

2 Research Methodology & Derivation Process

Before moving forward it is important to touch upon the process used to derive the conceptual architecture. Apache Flink is a very

large framework to build event-based distributed software solutions. The main challenge was not finding enough relevant information but to filter for the highest quality of information. The result was to use the various documentation and tutorials provided by the Apache Foundation as the primary basis for the process [1, 3, 8, 11–21]. Apache Flink's documentation is of very high quality and is the recommended starting point for anyone new to the framework. To augment this, the source code of Flink was surveyed by reding through key parts of the GitHub Repository [2]. This process was enough to derive the proposed conceptual architecture made using PlantUML.

Moving on, to build up the understanding further, various use cases in current large scale software systems wer studied. This allowed for better understanding of architectural patterns, data flow, use cases as well as creation of sequence diagrams [5, 6]. All sequence diagrams were made with PlantUML.

Finally, third party blogs and developer insights were studied to further understand the deeper concepts of Flink and its specific use cases [4, 9, 10, 22–26]. This was a key step for getting an outside perspective on the elements of the framework.

3 Conceptual Architecture of Apache Flink

3.1 Technology Stack

Apache Flink is not just a software system, it is itself a framework upon which to build applications. It is a very large, open-source system build with the following programming languages broken down by proportion:

- Java (85.9%)
- Scala (9.9%)
- Python (2.8%)
- Typescript (0.4%)
- Shell (0.5%)
- HiveQL (0.3%)
- Other (0.6%) [2]

The core of Apache Flink's computational features consist of Java & Scala. These modules make up the major features of Flink. They expose the main APIs that components of a Flink solution will need to interface with. Furthermore, the Python components make up PyFlink, an API facilitating the integration of Flink into Python applications [2].

Apache Flink also offers a web dashboard for real-time monitoring of the work being done by Flink integrations in applications. The dashboard is the minor Typescript component and it is built with the angular.js framework with a node.js build infrastructure [26].

3.2 Architectural Styles

Apache Flink is a large-scale distributed system that incorporates and hybridizes several architectural styles/patterns. Furthermore, when Flink is used as a framework to build a solution, it facilitates these large-scale applications to have Flink's architectural elements by proxy. In other words, Flink itself does not consist of these architectural elements by itself. Flink afterall, is a framework. The resulting application is what contains either one or more of these architectural elements. The major styles and the respective patterns are listed below:

3.2.1 Layered System of Services & APIs

Apache Flink's architecture is designed in layers, each serving specific functions in the data processing pipeline. The layered architecture of Flink helps in organizing and managing the complexities of distributed data processing. Flink follows loose layering, layers are not limited to only accessing the service offered by the adjacent layer below. The resulting Flink applications are loosely layered, offering versatility and flexibility in configuration. Here is an overview of the layered architecture of Apache Flink.

At the topmost layer exist Batch and Stream Processing APIs, Flink provides high-level APIs for both batch and stream processing. These APIs allow developers to define complex data transformations, aggregations, and computations using familiar programming constructs like maps, filters, windows, and joins. Flink supports both DataStream API for real-time stream processing and DataSet API for batch processing. Users interact with Flink primarily through these APIs to define their processing logic [14, 23].

Below this, Flink offers a Table API and supports SQL queries. These interfaces allow developers and data analysts to express their processing logic in a more declarative manner. Users can write SQL queries or use the Table API to perform operations on structured data, enabling seamless integration with existing tools and workflows [14, 23].

Beneath the higher-level APIs, Flink's core processing engines are based on DataStream and DataSet APIs. DataStream API processes unbounded streams of data in real-time, while DataSet API processes bounded datasets in batch mode. These APIs provide abstractions for handling the complexities of distributed data processing, including fault tolerance, state management, and data serialization [14, 23].

Next is the Runtime and Execution Engine. The runtime layer is responsible for executing the data processing tasks defined by the APIs. It includes various components such as JobManager, TaskManagers, and ResourceManager. JobManager coordinates job execution, scheduling tasks, and managing the overall job lifecycle. TaskManagers are responsible for executing tasks and managing the data exchange between tasks. ResourceManager oversees resource allocation, ensuring optimal utilization of cluster resources [14, 23].

At the bottom layer, Flink integrates with cluster management systems like Apache Mesos, Kubernetes, or YARN. These systems provide resource management capabilities, allowing Flink to efficiently utilize the available cluster resources. Flink's ResourceManager interacts with these cluster managers to request resources, allocate tasks, and handle failures [14, 23].

This layered architecture abstracts the complexities of distributed data processing, allowing developers to focus on defining their data transformations and computations without worrying about the intricacies of cluster management and fault tolerance. It provides a flexible and scalable framework for processing both batch and streaming data, making Apache Flink a powerful choice for various big data processing tasks. The architecture can be seen in Figure 1.

3.2.2 Distributed System

Flink has the capability to power highly scalable distributed systems for data stream processing. The Flink core is able to work with many cloud resource managers and even form custom clusters. This

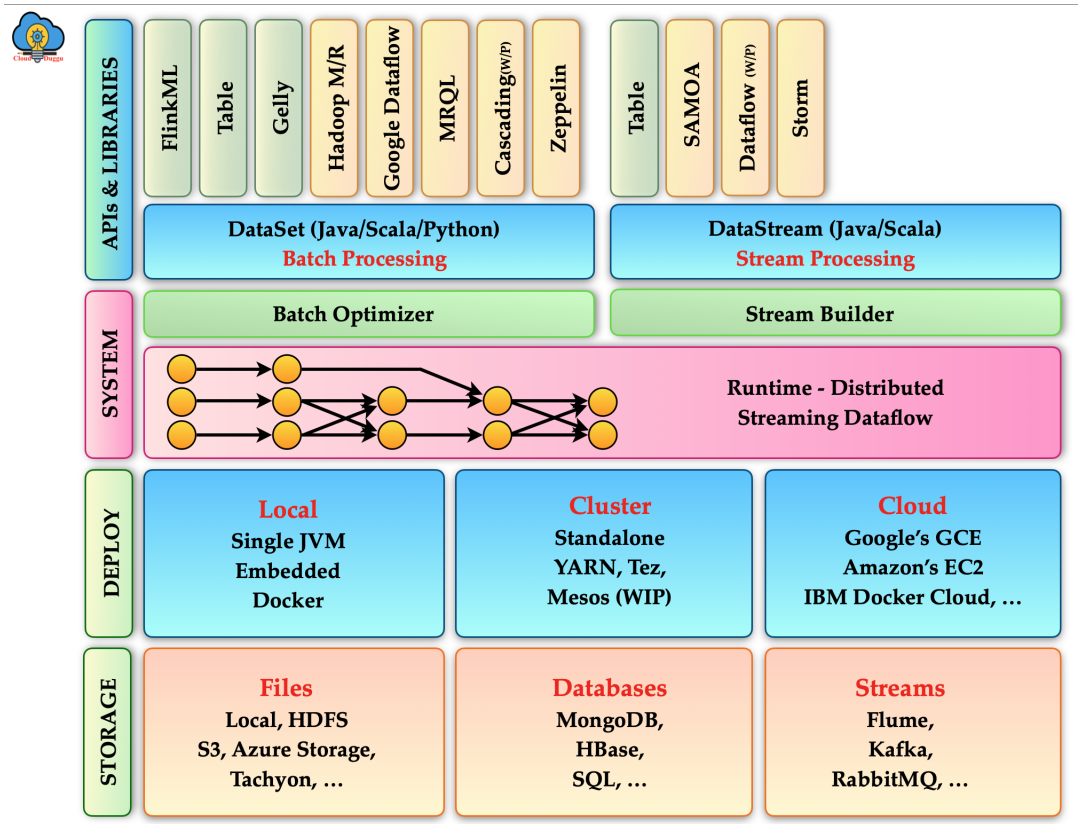

Fig. 1: The layers of services and APIs provided by Flink are shown sections titled System and APIs & Libraries. The bottom two layers show services adjacent to Flink [23].

allows Flink to scale up with several worker nodes managed via the Master-Slave pattern; the job manager nodes are the masters and the task manager nodes are the slaves [23]. This is shown in Figure 2. Furthermore, Flink has the ability to save backup snapshots of program states in distributed file systems that synchronize periodically with inidvidual nodes and their local file systems shown in Figure 3 [20].

3.2.3 Repository System for State Consistency & Durability

Flink uses local, in-memory/disk storage to maintain states for intermediate and final results of processed streams. This is shown in Figure 4. Flink applications must maintain a consistent state that is a computational result of processing streams/events. These states are stored locally in memory or specialized disk-based data structures. Moreover, the states are periodically saved to the distributed file system mentioned above [12].

3.2.4 Pipe & Filter for Batch and Stream Processing

Apache Flink enables real-time data processing through a Pipe and Filter model. Events produced are channeled through a Message Queue, a message broker like Kafka, and divided into partitions. These events are then fed into Flink's Data Source and processed with the Window Operator. The result is real-time insights based on event and window processing times [17]. This is shown in Figure 5.

Flink facilitates data processing by linking data sources to data sinks. Flink's support of bounded streams is executed by batch processing. Flink's support of unbounded streams is executed by continuous processing [12].

3.2.5 Client-Server System Communication

At a high level, Apache Flink's client-server architecture is designed to handle large-scale data processing and analytics in a distributed and fault-tolerant manner. Clients, which can be user applications or systems, submit jobs to the Flink cluster for processing. These jobs are typically data processing tasks defined using Flink's APIs or other compatible interfaces. The central component of the architecture is the JobManager, which acts as the master node. It receives job submissions from clients, coordinates the distributed execution, and monitors the overall job lifecycle.

On the other end, there are multiple TaskManagers, which serve as worker nodes in the Flink cluster. TaskManagers are responsible for executing the tasks specified in the submitted jobs. They manage the parallel execution of tasks, data exchange, and fault tolerance mechanisms. TaskManagers communicate with each other for data exchange and load balancing, ensuring efficient processing of tasks across the cluster.

The ResourceManager oversees resource allocation within the Flink cluster, ensuring that computational resources such as CPU and memory are efficiently distributed among TaskManagers based

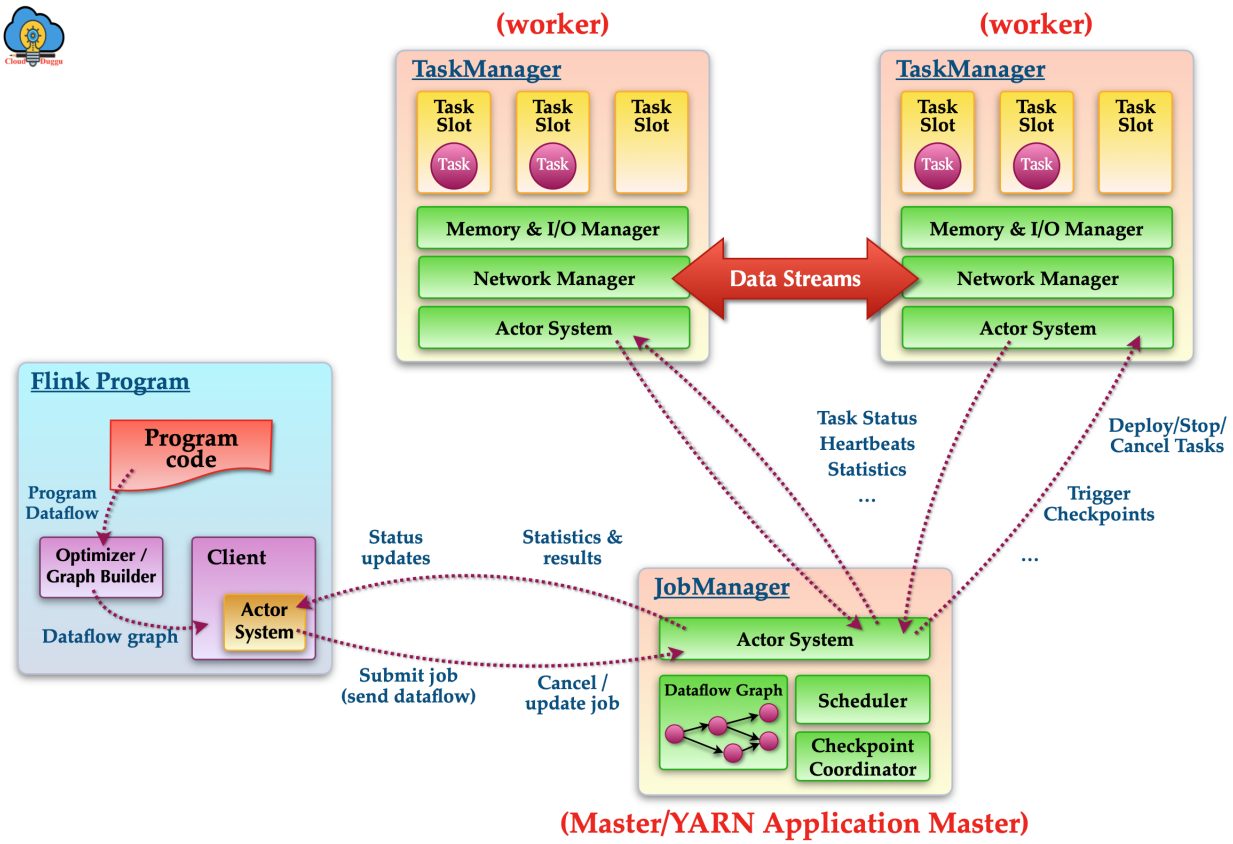

Fig. 2: The process tasks being shared across worker nodes being managed by the manager [23].

on the requirements of the running jobs. This dynamic allocation and deallocation of resources optimize the cluster's performance and responsiveness.

Additionally, Flink provides a RESTful API that allows clients to interact with the cluster programmatically. Clients can submit jobs, monitor their status, manage checkpoints, and access cluster metrics using HTTP requests. This RESTful interface enhances the cluster's usability and integration capabilities, enabling seamless interaction between Flink and other applications or monitoring tools. Overall, Flink's high-level client-server architecture ensures robust, scalable, and efficient processing of large-scale data workloads in both batch and streaming processing scenarios.

3.2.6 Implicit Invocation - Event-based Architecture

Flink applications respond to event data streams in a stateful manner. Systems react to events by performing computations, state updates and more. Flink has several semantic modes for various reference measures of application/event timestamps. Both batch analytics and stream analytics are performed in an event-driven manner [21]. This is shown in Figure 6.

3.3 Conceptual Architecture

Moving on, an exploration of the high-level architecture of Apache Flink, detailing its main components such as Task Managers, Job Managers, Flink Clusters, and the Flink Client. The architectural diagram below show the various components, their relationships and dependencies.

3.4 Flink Cluster

The Flink Cluster is the runtime environment where Flink applications execute. It comprises multiple components, including the Job Manager, Task Managers, and the necessary infrastructure to coordinate and execute Flink jobs. The cluster can operate in various cluster deployments, such as standalone, YARN, or Kubernetes, allowing it to integrate seamlessly with different cluster management and resource allocation systems. To accomplish this seamless integration Flink comes equipped with tools for native interaction with resource management systems.

3.5 Job Manager

Job Managers are central components in the Flink architecture, orchestrating the execution of Flink jobs. The Job Manager itself encompasses several internal components, each having specific responsibilities:

3.5.1 Dispatcher

REST API which acts as a gateway for submitting Flink jobs. It receives job submissions and starts a new JobManager instance for each submitted job, ensuring that each job's execution is isolated and independently managed.

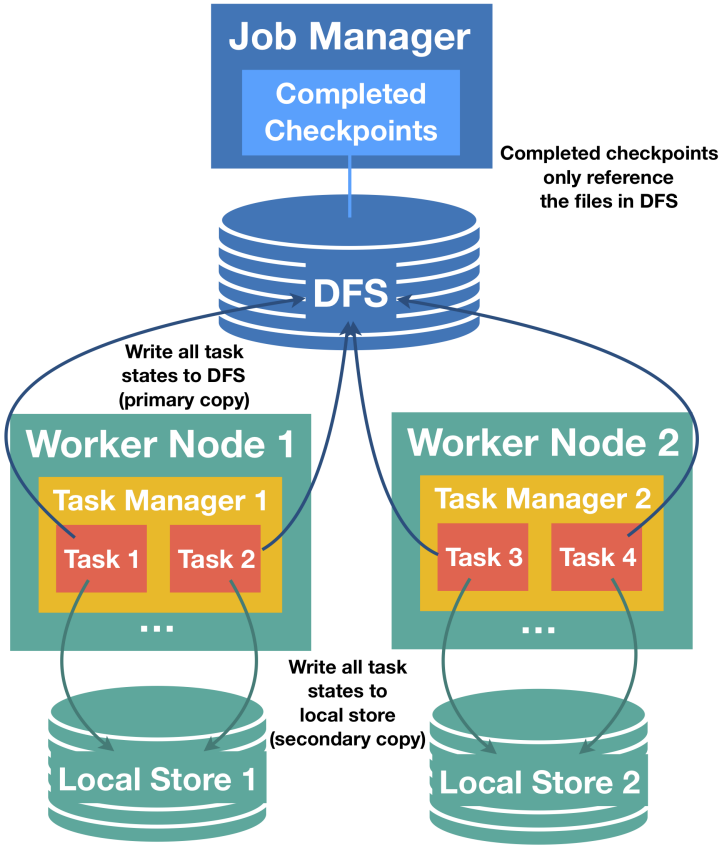

Fig. 3: Visualization of the snapshots being stored in the distributed file system and the local nodes. Arrows show synchronization of states [23].

3.5.2 Resource Manager

Manages the physical resources within the Flink cluster. It allocates resources to various tasks as required, ensuring that each task has the necessary resources for execution. The ResourceManager also handles the dynamic allocation and deallocation of resources, adapting to the workload and optimizing resource utilization.

3.5.3 JobMaster

Coordinates the execution of tasks for a specific job. It is responsible for scheduling tasks, managing their state, and recovering from task failures by restarting them. The JobMaster ensures that the job execution is reliable, consistent, and fault-tolerant.

3.6 Task Manager

Task Managers are the worker nodes in Flink's distributed architecture. They are responsible for executing the sub-tasks of a Flink job and processing the data streams. Each Task Manager has a set of slots where tasks can run concurrently. They manage the buffer, cache data, and execute the operations as per the defined job. Task Managers also handle the shuffling of data between different tasks, ensuring that data is properly exchanged and partitioned across different task managers for optimized processing.

3.7 State Backend

The State Backend in Flink plays a crucial role in the system's fault tolerance mechanism. It manages the state of the running jobs, ensuring that data is not lost in case of failures and that computations can be recovered and resumed. The State Backend is used for storing snapshots of the application's state at different points in time, a process known as checkpointing.

Checkpoints allow Flink to recover the state and resume computations in case of failures, ensuring that the system can continue processing data with consistency and reliability. The State Backend ensures that the state is stored and managed efficiently, allowing for fast recovery times, and minimizing the impact of failures on the application's performance and accuracy.

3.8 Clients & API

The Flink Client acts as the entry point for users to submit and manage their Flink jobs. It facilitates the interaction between the users and the Flink cluster, ensuring that the jobs are appropriately initialized, executed, and managed throughout their lifecycle. The client is equipped with various APIs, each tailored to meet different processing needs and use cases, enhancing the flexibility and usability of the Flink system.

Flink's rich set of APIs, such as the DataSet API, DataStream API, ProcessFunction API, and Table API & SQL, allow for a wide

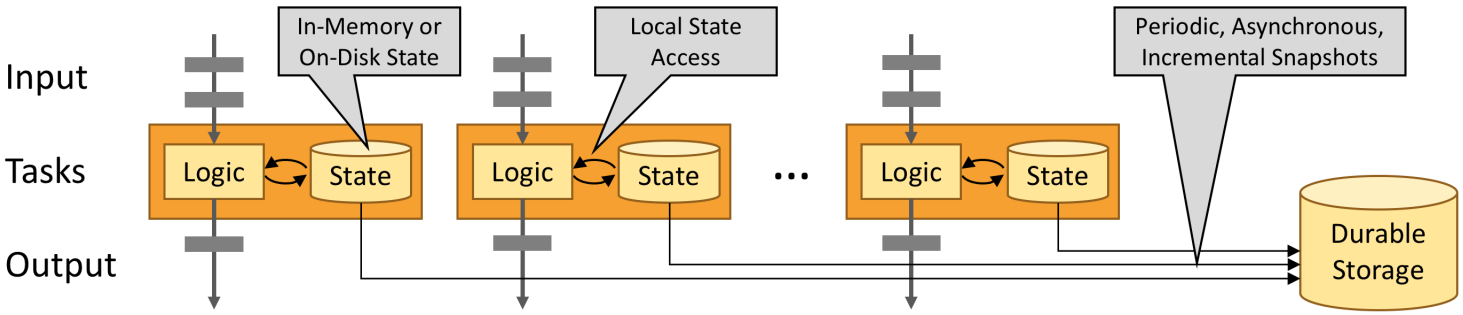

Fig. 4: States are maintained in memory and specialized disk-based data structures [12].

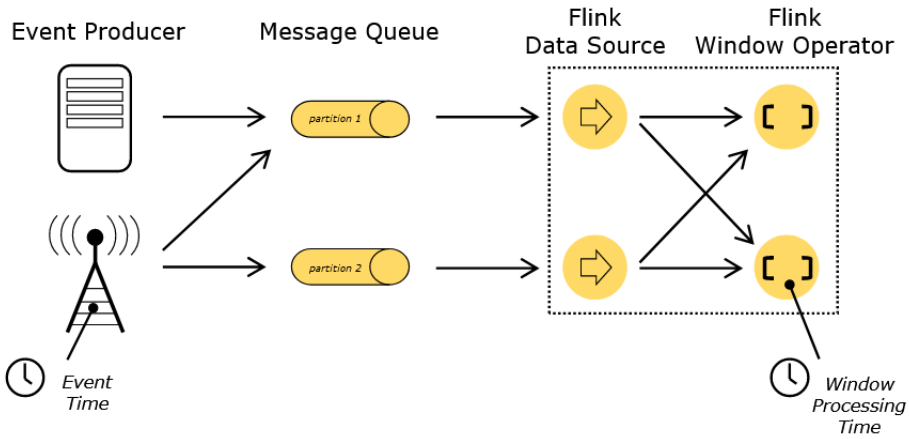

Fig. 5: Event producers and message quest interfaced with Flink to show Pipes & filter elements [17].

range of functionality . The DataSet API is primarily used for batch processing tasks, enabling operations on bounded datasets. In contrast, the DataStream API is designed for stream processing, allowing for operations on unbounded streams of data. The Table API & SQL provide a higher-level abstraction, offering users the ability to work with data using SQL queries and a table-like structure, suitable for both batch and stream processing. These APIs collectively empower developers to build robust, efficient, and versatile data processing applications using Apache Flink. Figure 8 shows the data flow between these components for a typical Flink Job.

4 Concurrency, Responsibility Analysis, and Quality Attributes

4.1 Concurrency Aspects

Concurrency in Apache Flink is a concept that is vital to its functioning. Given that the program deals with data pipelining, various data processing techniques are utilized and therefore mechanisms are put in place to aid in concurrent execution. We'll demonstrate concurrency by first listing an Apache mechanism or an architecture component, briefly describe its functioning, then explaining its concurrent purpose.

4.1.1 Operator

Operator is a tool/function that transforms an existing data stream into a new one. Operators vary in functionality, some are used to perform quick computations, while others are used to separate data [15]. Operators can also be configured to be set up in parallel to speed up computation by adjusting the parallelism feature in Apache Flink [19].

4.1.2 Checkpointing

Checkpointing is a fault-tolerance tool used to counter data inconsistencies when concurrently processing data streams. Snapshots are taken throughout data flow, so that when a failure occurs, the snapshot can be restored and data flows uninterrupted [18]. Thus it's useful in maintaining the correctness of concurrency.

4.1.3 Watermarks

Watermarks is a mechanism used to maintain event time in unbounded data processing. It works by flowing with the data stream and setting time stamps [17]. W(t), where t refers to a timestamp, if an event enters the stream with W(20), then any other event that enters afterwards is to have an event time greater than or equal to 20 [17].

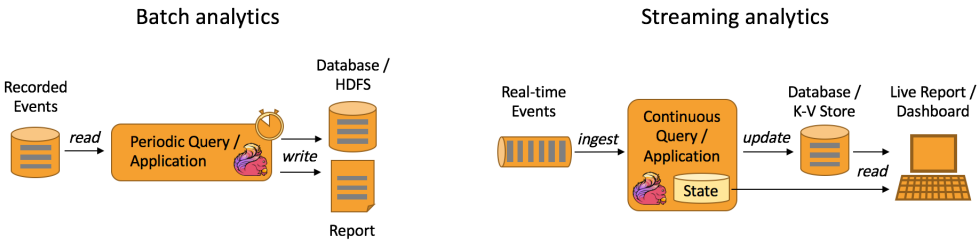

Fig. 6: Batch analytics and stream analytics modelled as event-driven tasks [21].

4.1.4 Job Manager

Job Manager is a runtime process that is focused on task assignment, failure handling, and checkpoint coordination. Task Manager is a run time process that is focused on executing tasks [16]. The Job Manager acts much like a CPU, assigning processes/tasks to each of its cores/Task managers. In turn the Core/Task manager assigns or runs the task in a specific thread/task slot [16].

Finally, task and operator chaining is a mechanism used to fuse operator tasks into one to aid in distributed execution and decrease overhead in thread to thread handovers [16].

4.2 Quality Attributes

In this section, we'll be discussing the 3 quality attributes of Apache Flink: Performance, Scalability, and Reliability.

4.2.1 Performance

Apache has incredible performance for select use cases such as: real-time data processing, complex event processing and Batch data processing [22]. Due to the massive support for various data pipelining features/functionalities [4] - Support for bounded and unbounded data streams, through state and context mechanisms [8]. - Support for event time and process time data streams, through Watermarks [8].

4.2.2 Scalability

Apache Flink has been built to work with common cluster managers such as Kubernetes, or also can be set up to operate as a standalone cluster [13]. Therefore it can be scaled to the needs of the user or the increasing demand of usage of software. Due to the nature of clusters and Apache, it's been programmed to stream data at any scale [8].

4.2.3 Reliability

Given the presence of mechanisms such as state snapshots, stream replay, and checkpointing. Apache Flink's data streaming functionality is fault tolerant ensuring smooth and correct flow of data [13]. Regardless of the presence of errors or failures. Furthermore, Apache Flink offers monitoring functionalities in order to track performance and state of the data pipelines [13].

4.3 Division of Responsibilities

Apache Flink, an open-source project driven by a global community of developers, operates with a unique structure. Rather than having a defined core group of primary developers, the project relies on a diverse community of contributors who actively maintain and enhance it. Among these contributors are the committers, individuals granted write access to the project's repositories, allowing them to modify code, documentation, and accept contributions [4]. This access is granted through a consensus decision by the active PMC (Project Management Committee) members, and committers may also declare emeritus status. The PMC serves as the official governing body, responsible for project oversight and management. Its duties encompass approving releases, managing shared resources like code repositories and mailing lists, representing the project, resolving license issues, nominating new PMC members and committers, and upholding the project's guidelines and ASF brand management policies [1]. In essence, Apache Flink's collaborative ecosystem ensures its growth, development, and adherence to Apache Software Foundation principles. Within the Apache Flink project, a dedicated position known as the "Release Manager" is integral to its operations, their role is to oversee the practical aspects of a release within the Flink project. They are tasked with guiding a Flink release from its inception, where community consensus is established, to its final distribution [4]. While Release Managers play a significant role in pushing out releases, it's important to note that they are not the ultimate authority responsible for ensuring compliance with release requirements. Instead, this responsibility falls on the broader PMC and, more specifically, the PMC chair, who serves as an officer of the Foundation. It's noteworthy that any committer within the Flink project has the opportunity to assume the role of a Release Manager, and a release process commences when the project community collectively agrees to initiate it.

4.4 Security Measures

The Apache Flink project prioritizes security measures, data privacy, and compliance to protect sensitive information and adhere to regulatory standards. The project complies to The Apache Software Foundation's licensing provisions, emphasizing a collaborative and legally compliant development environment. Flink has implemented robust security features, including authentication and authorization mechanisms, data encryption in transit and at rest, and role-based access control to ensure data security throughout its lifecycle [5]. Protocols such as Kerberos and TLS/SSL authentication are used to secure FlinFlink employs security protocols, including Kerberos and

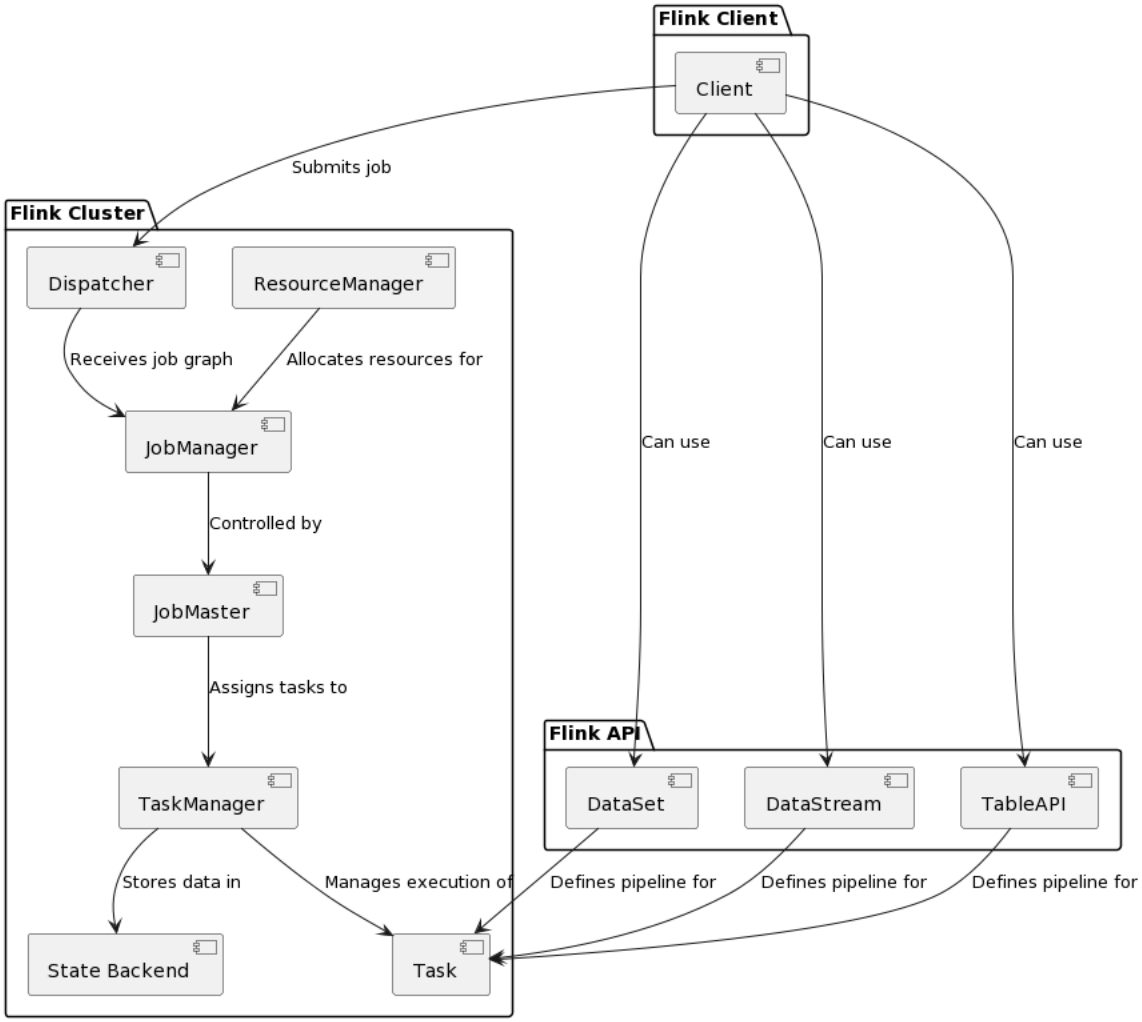

Fig. 7: Conceptual Architecture of Apache Flink, Showing Components & their Dependencies

TLS/SSL authentication, to enhance its protection measures [5]. The project actively monitors and addresses security vulnerabilities, often collaborating closely with the global open-source community to maintain a secure codebase. In terms of data privacy and compliance, Apache Flink enables users to establish data retention policies and comply with data protection regulations. This commitment to security and compliance establishes a secure and reliable environment for organizations relying on Flink for their data processing needs. Additionally, all Apache committers are required to maintain a signed Individual Contributor License Agreement (ICLA) with the Apache Software Foundation [4]. This legal framework ensures that contributions align with the Foundation's guidelines, fostering transparency and trust within the development process.

5 Visualization of Use Cases & Applications

5.1 Real-Time Stream Processing with IoT Sensors

IoT sensors, given their vast deployment in modern applications and their continuous data generation, are prime candidates for real-time stream processing. The sequence diagram encapsulates this by showcasing the real-time data processing and analysis flow of IoT sensor data using a Flink Cluster. As IoT sensors emit data, it is ingested into the Flink Cluster, specifically through TaskManager1, as coordinated by the JobManager. This ingested data is then passed to a Data Transformation Operator to verify its previous state from the State Backend. Following the transformation, the data is channeled to a Real-Time Analytics Operator for processing. This operator subsequently updates the current state of the sensor data in the State Backend and dispatches the real-time analytics results to an external Analytics Dashboard. Once the Analytics Dashboard acquires this data, it furnishes an acknowledgment back to the JobManager, thus bringing the data processing cycle to a close. Figure 12 shows this use case as a UML sequence diagram.

5.2 Batch Processing

This is the most fundamental use case. Initially, data is extracted from sensors and manufacturing logs. Within the Flink Cluster, this data undergoes a series of ETL (Extract, Transform, Load) operations—first fetched by the TaskManager, then transformed, and finally loaded into a Data Warehouse. Following the ETL processes,

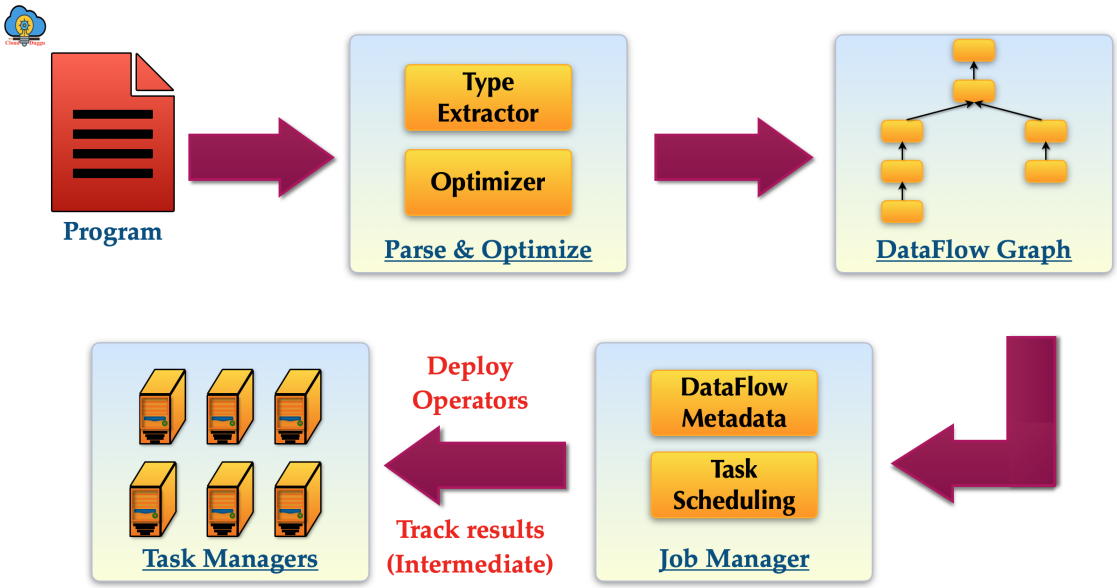

Fig. 8: The workflow of a typical Flink job going through the system components mentioned above [23]

the transformed data is analyzed to derive insights, particularly focusing on vaccine production parameters. The findings from this analysis are subsequently reflected on an analytics dashboard. At a high level, batch processing in Apache Flink involves processing large, finite sets of data all at once rather than continuously processing streaming data. This mode is particularly useful when dealing with historical data or executing heavy transformations where immediacy is not paramount. Flink's robust architecture allows for efficient parallel processing, resource management, and stateful computations, ensuring that large volumes of data can be processed swiftly and reliably. Figure 13 shows this use case as a UML sequence diagram.

5.3 User Recommendations using Stateful Computations

In the evolving landscape of e-commerce, understanding user behavior and promptly acting upon it is important. This sequence diagram delineates the intricate yet efficient process of real-time recommendation generation using a Flink Cluster on an e-commerce platform. When users interact with the platform by browsing or shopping, the Application captures their activities. These activities, in the form of user events, are submitted to the Flink Cluster's Dispatcher, which subsequently assigns the job to the JobManager. The Source Task is triggered to fetch these user events and is further processed by the UserEventOperator to retrieve the historical behavior of the user from the State Backend. To generate tailored recommendations, data might need to be shuffled between operators due to changes in parallelism or for stream grouping. This is represented by the Network Shuffle, which serializes, partitions, and forwards the user event data to the RecommendationOperator. After updating the user's behavior in the State Backend, the RecommendationOperator collaborates with the Recommendation Engine to generate personalized product suggestions. These suggestions are then relayed back to the e-commerce Application, which consequently enhances the user's browsing or shopping experience by presenting personalized content. Figure 14 shows this use case as a UML sequence diagram.

5.4 Fraud Detection via Event-Driven Monitoring

In the financial world, detecting fraud quickly is crucial to protect both businesses and their customers. The depicted sequence diagram shows how a Flink Cluster, paired with a financial system, processes transactions to catch suspicious activities. When the TransactionSystem initiates a transaction, it sends the details to the JobManager. The JobManager then prompts the TaskManager to handle the fraud detection task. To do this, the TaskManager fetches the transaction and passes it to the FraudDetection operator. This operator retrieves historical transaction data from the State Backend to make a well-informed assessment. If the current transaction seems out of place or suspicious, the FraudDetection flags it. To ensure prompt action, an alert is sent to the AlertService, which then confirms the receipt and handling of the potential threat. Figure 15 shows this use case as a UML sequence diagram.

5.5 Fault Tolerance

Fault tolerance is a cornerstone of Apache Flink, ensuring that data stream processing remains consistent even amidst potential system failures. The showcased sequence diagram commences with a user submitting a Flink job, which encapsulates tasks and prescribes data movement from a Source to a Sink. Post receipt, the JobManager entrusts this to the Job Master. TaskManager1 commences by fetching data in batches from the Source. As it processes this data, it periodically records system states as snapshots for recovery purposes. Instead of directly transmitting to the Sink, sometimes data is passed to other downstream TaskManagers, especially when operations like joins are involved or there's a change in the level of parallelism. As data traverses through the system, newer snapshots get registered. When a disruption is detected, the Job Master

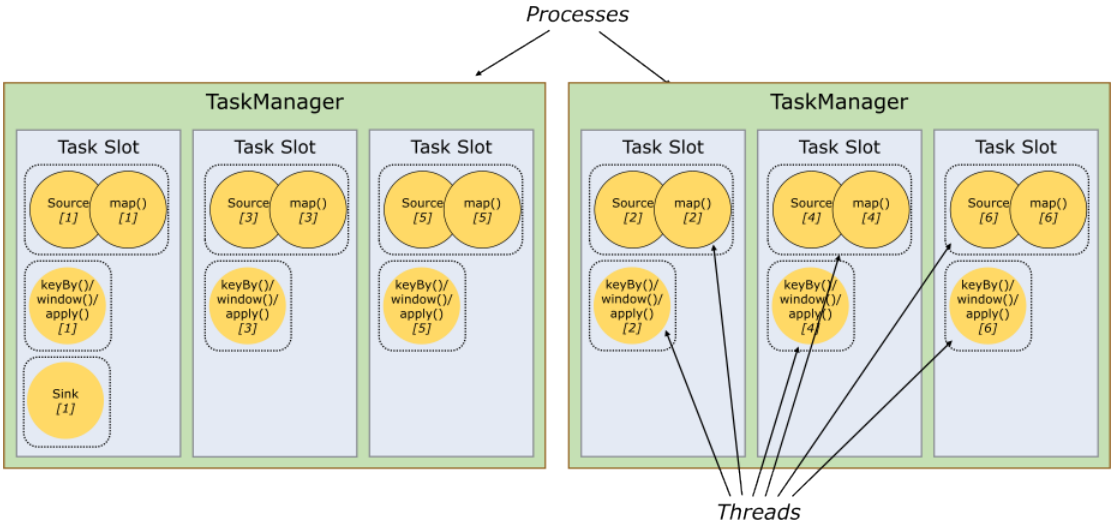

Fig. 9: Concurrency of Flink tasks spread over multiple parallel processes and further as threads [16]

promptly redistributes the task to another TaskManager. Using the most recent snapshot, this TaskManager restores its state and picks up the data processing baton, assuring both data integrity and system resilience. Figure 16 shows this use case as a UML sequence diagram.

6 Lessone Learned & Conclusion

To conclude, Apache Flink is a large-scale distributed system that encompasses many diverse architectural elements. Further, it can serve a variety of use-cases for industry level software solutions. These solutions encompass real-time processing of data, stateful, event-driven business logic, consistency in computation results, and fault tolerant processes.

Some of the key lessons learned are outlined as a last section. One valuable lesson is the importance of ensuring that every user/researcher of Apache Flink understands its fundamental, high-level structure. This involves gaining insight into components like the JobManager, TaskManager, and how data is processed within the system. A strong foundation in Flink's architecture is essential for effective collaboration and decision-making throughout any project involving Flink, whether it is one that implements the framework in a solution or reviews its usage for research purposes.

Another key lesson learned is feedback and iteration. In using a system as complex as Flink or researching its basics, one must be open to feedback from team members and stakeholders. Moreover, iterative approaches on the design/research are essential in being successful in any Flink project.

Finally, User-Friendly Documentation is a priority for large-scale distributed systems like Apache Flink. Creating user-friendly documentation is crucial for the software engineering community to understand and use open-source systems. Apache Flink stands out in the immense amount of high quality documentation. These documents are a holistic vault of educational content for Flink engineers and researchers. They encompass high quality textual descriptions, clear and concise diagrams and a wealth of soruce code examples to accelerate development/research understanding.

References

[1] "Apache Flink® — Stateful Computations over Data Streams." [Online]. Available: https://flink.apache.org/

[2] "Apache Flink," Oct. 2023, original-date: 2014-06-07T07:00:10Z. [Online]. Available: https://github.com/apache/flink

[3] "Powered By," section: what-is-flink. [Online]. Available: https://flink.apache.org/what-is-flink/powered-by/

[4] H. Temme, "3 Reasons Why You Need Apache Flink for Stream Processing," Jun. 2023. [Online]. Available: https://thenewstack.io/3-reasons-why-you-need-apache-flink-for-stream-processing/

[5] "Building Scalable Streaming Pipelines for Near Real-Time Features," Aug. 2021. [Online]. Available: https://www.uber.com/en-SE/blog/building-scalable-streaming-pipelines/

[6] "Apache Flink." [Online]. Available: https://aws.amazon.com/managed-service-apache-flink/

[7] "Welcome to The Apache Software Foundation!" [Online]. Available: https://www.apache.org/

[8] "Overview," section: docs. [Online]. Available: //nightlies.apache.org/flink/flink-docs-release-1.17/docs/dev/table/sql-gateway/overview/

[9] "Apache Flink® 1.3.0 and the Evolution of Stream Processing with Flink." [Online]. Available: https://www.ververica.com/blog/apache-flink-1-3-0-evolution-stream-processing

[10] "A Brief History of Flink: Tracing the Big Data Engine's Open-source Development | HackerNoon." [Online]. Available: https://hackernoon.com/a-brief-history-of-flink-tracing-the-big-data-engines-open-source-deve

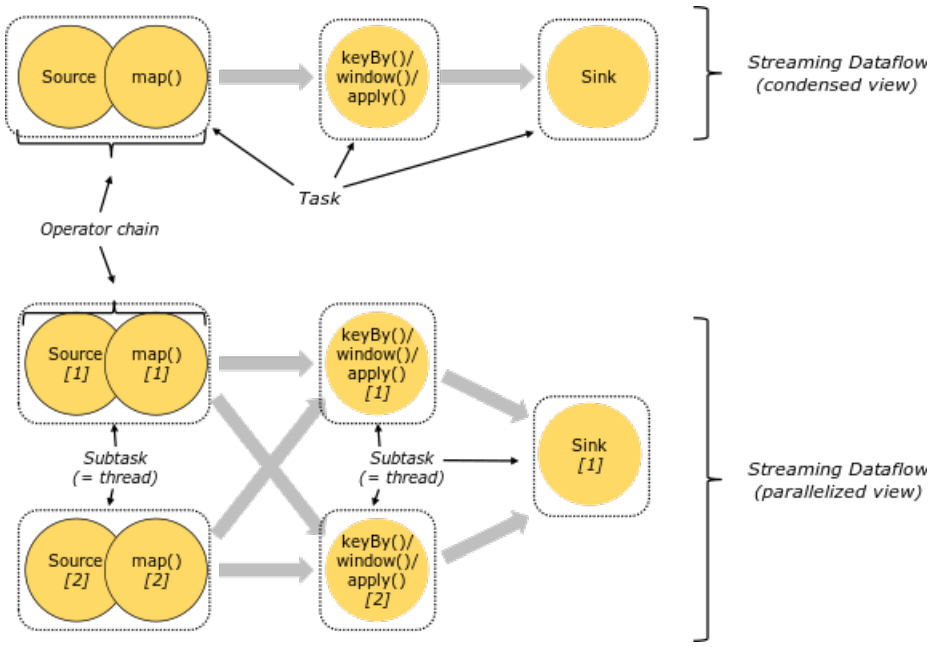

Fig. 10: Task and operator chaining [16].

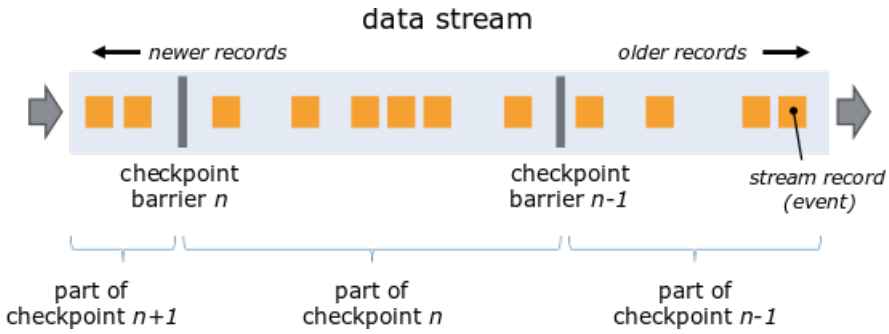

Fig. 11: Checkpointing process for fault tolerance [18]

[11] "Overview," section: how-to-contribute. [Online]. Available: https://flink.apache.org/how-to-contribute/overview/

[12] "Architecture," section: what-is-flink. [Online]. Available: https://flink.apache.org/what-is-flink/flink-architecture/

[13] "Flink Architecture," section: docs. [Online]. Available: //nightlies.apache.org/flink/flink-docs-release-1.17/docs/concepts/flink-architecture/

[14] "Apache Flink Documentation." [Online]. Available: //nightlies.apache.org/flink/flink-docs-release-1.17/

[15] "Operators." [Online]. Available: //nightlies.apache.org/flink/flink-docs-release-1.17/docs/dev/datastream/operators/

[16] "Flink Architecture," section: docs. [Online]. Available: //nightlies.apache.org/flink/flink-docs-master/docs/concepts/flink-architecture/

[17] "Timely Stream Processing," section: docs. [Online]. Available: //nightlies.apache.org/flink/flink-docs-release-1.17/docs/concepts/time/

[18] "Stateful Stream Processing," section: docs. [Online]. Available: //nightlies.apache.org/flink/flink-docs-release-1.17/docs/concepts/stateful-stream-processing/

[19] "Parallel Execution," section: docs. [Online]. Available: //nightlies.apache.org/flink/flink-docs-release-1.17/docs/dev/datastream/execution/parallel/

[20] "Tuning Checkpoints and Large State." [Online]. Available: //nightlies.apache.org/flink/flink-docs-release-1.13/docs/ops/state/large_state_tuning/

[21] "Use Cases," section: what-is-flink. [Online]. Available: https://flink.apache.org/what-is-flink/use-cases/

[22] Bitrock, "Apache Flink and Kafka Stream: A Comparative Analysis," Sep. 2023. [Online]. Available: https://medium.com/@BitrockIT/apache-flink-and-kafka-stream-a-comparative-analysis-f8cb5b946ec3

[23] "Apache Flink Architecture Tutorial." [Online]. Available: https://www.cloudduggu.com

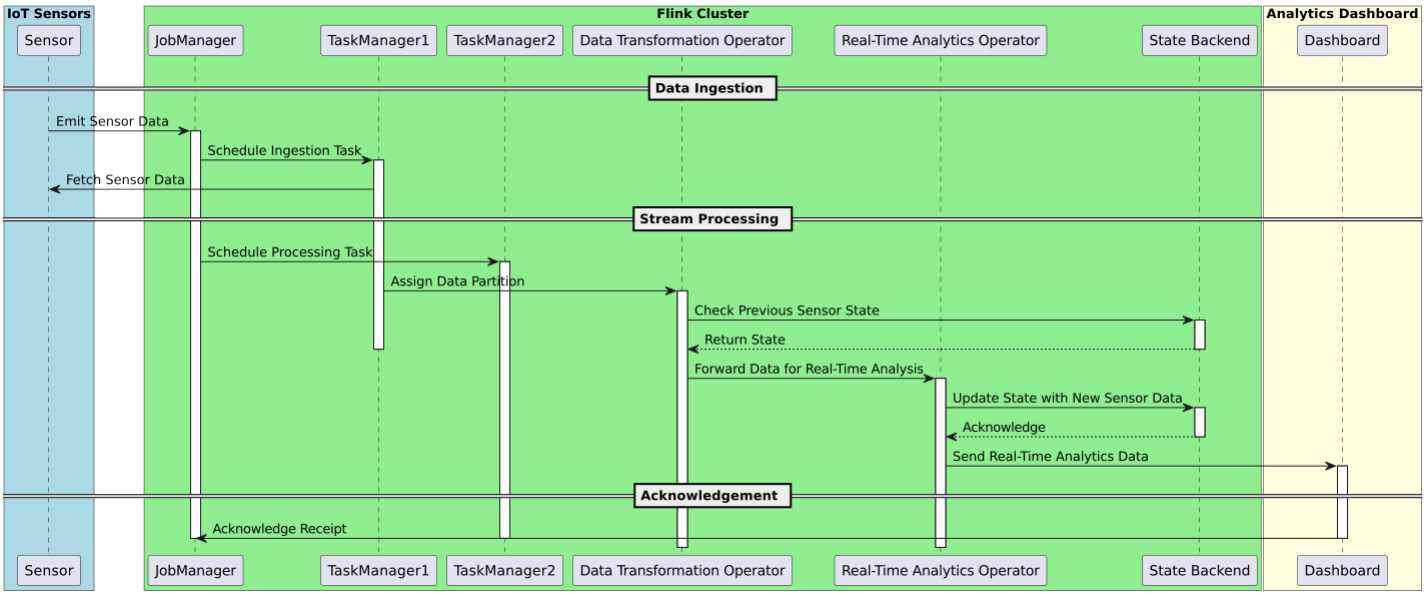

Fig. 12: Use Case 1, Real-Time Stream Processing with IoT Sensors

[24] "Securing Apache Flink | CDP Private Cloud." [Online]. Available: https://docs.cloudera.com/csa/1.11.0/security/topics/csa-flink-security-overview.html

[25] "Flink Bylaws - Apache Flink - Apache Software Foundation." [Online]. Available: https://cwiki.apache.org/confluence/display/FLINK/Flink+Bylaws

[26] "Apache Flink Web Dashboard." [Online]. Available: https://apache.googlesource.com/flink/+/release-1.0.2/flink-runtime-web/README.md

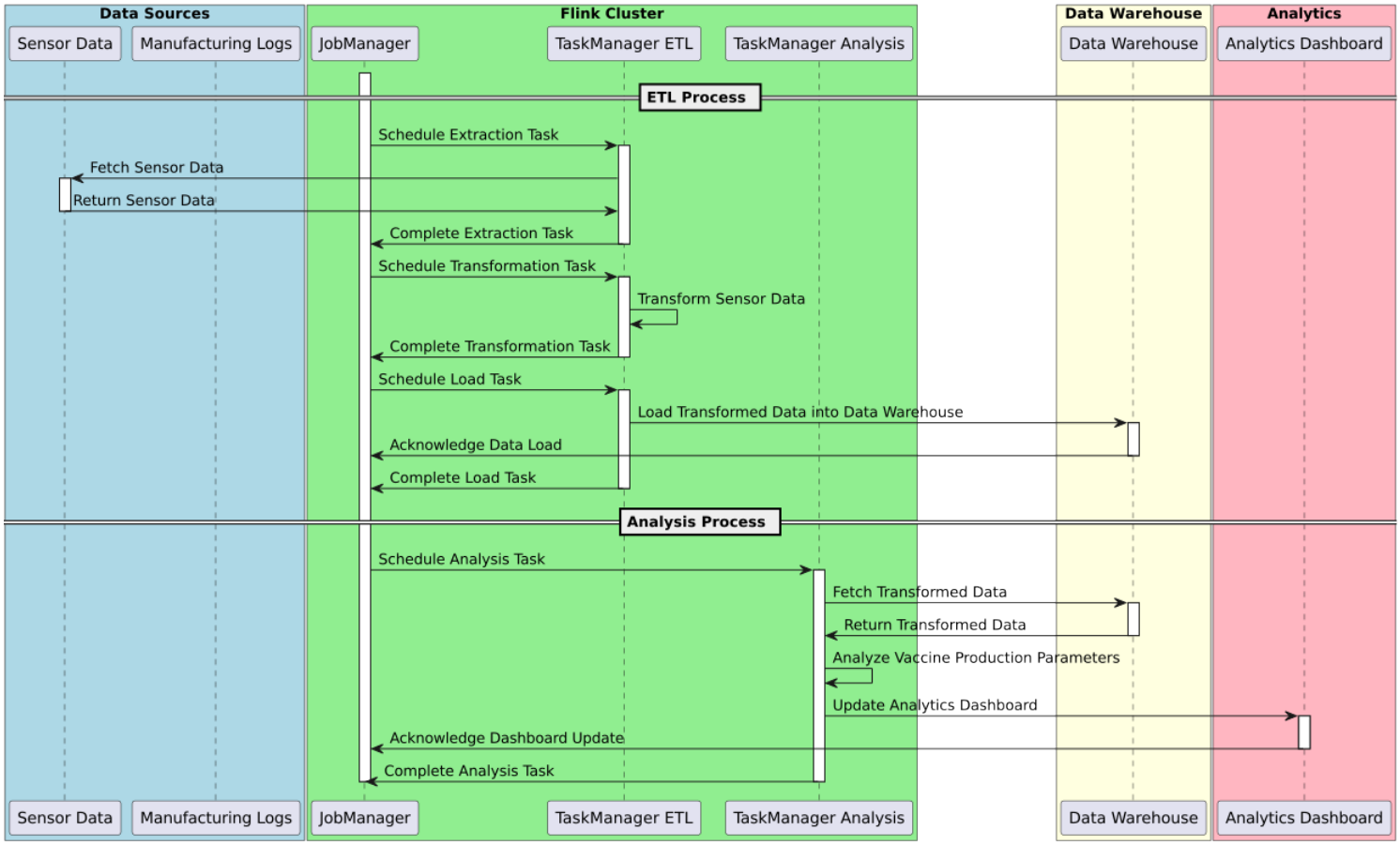

Fig. 13: Use Case 2, Batch Processing

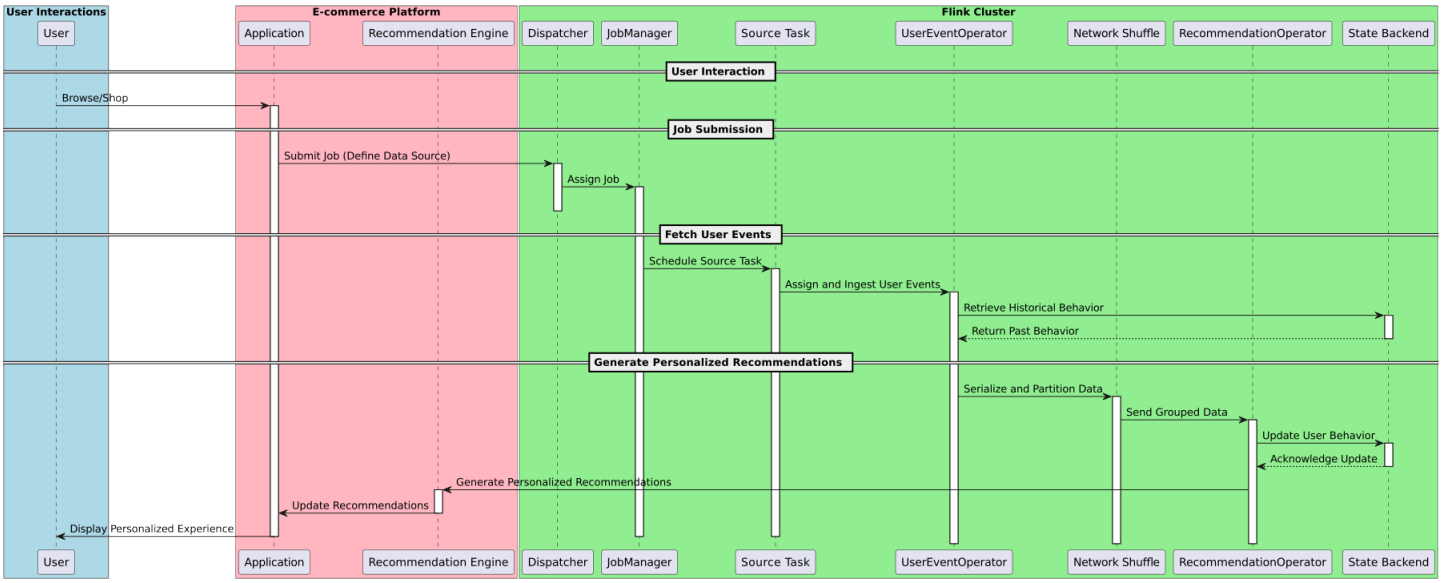

Fig. 14: Use Case 3, User Recommendations using Stateful Computations

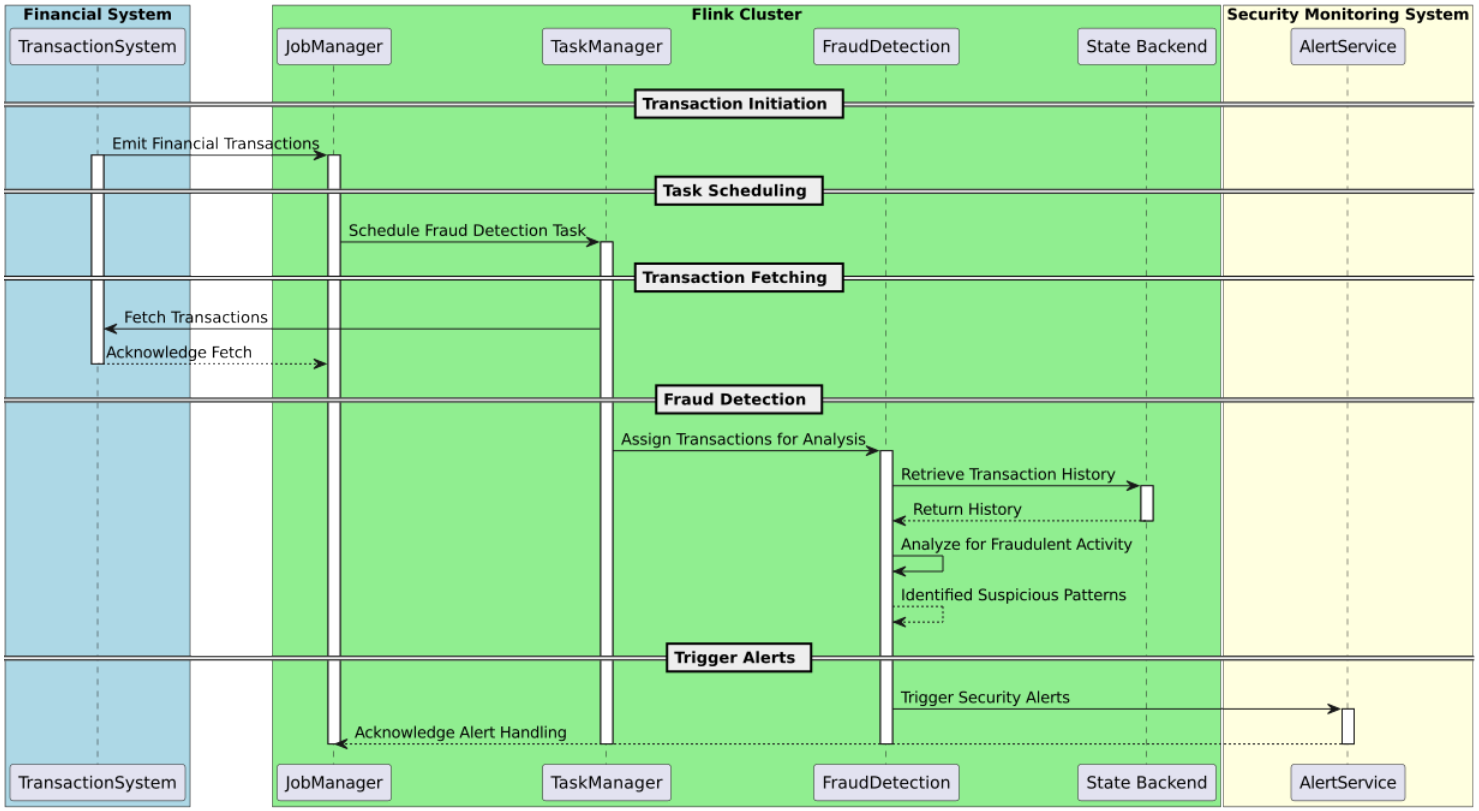

Fig. 15: Use Case 4, Fraud Detection via Event-Driven Monitoring

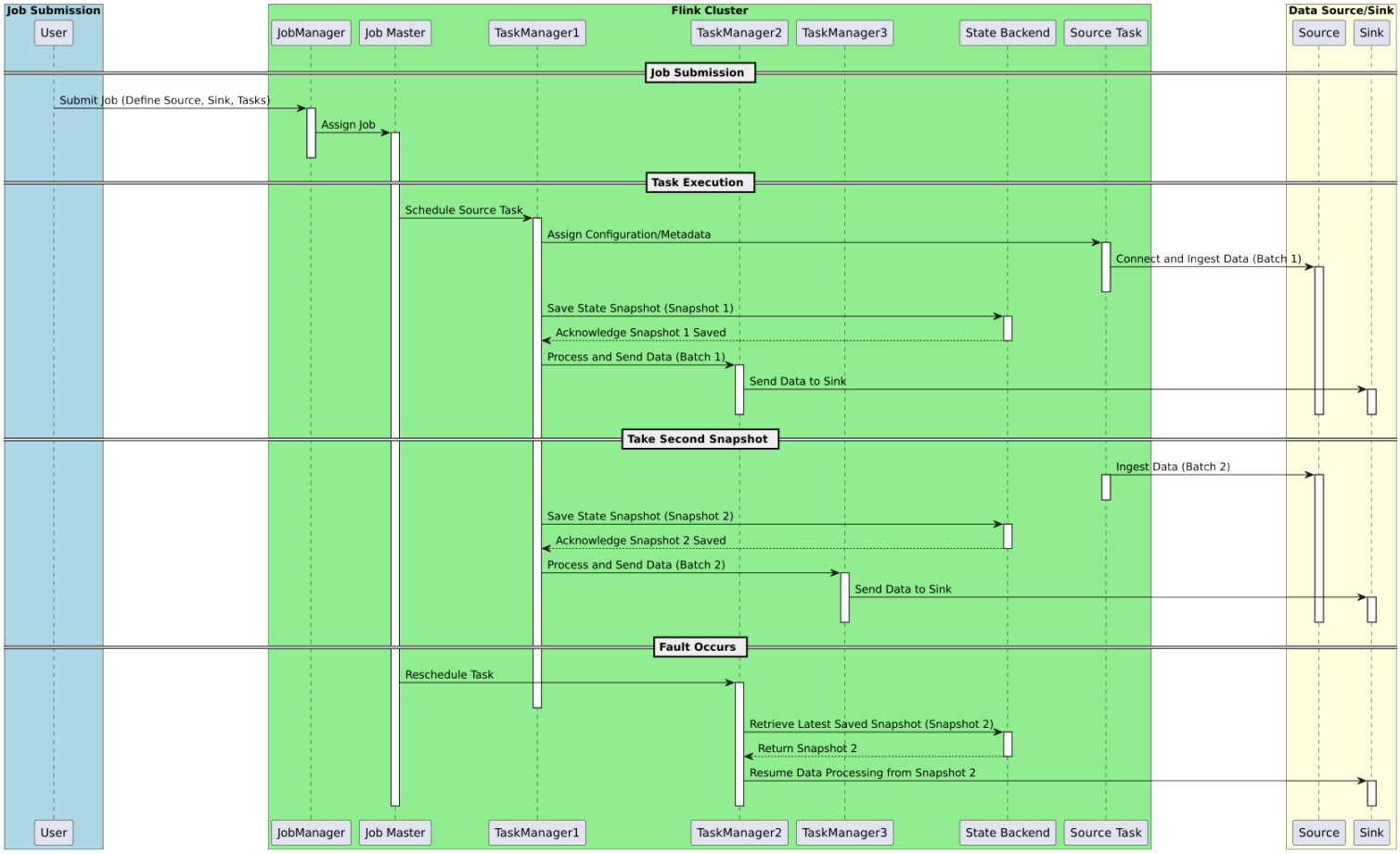

Fig. 16: Use Case 5, Fault Tolerance