BluePrint for Understanding Apache Flink's Concrete Architecture

Concrete Architecture

By: Analysis Paralysis

Authors:
Aisha Mahmood - aisha895@my.yorku.ca
Harsimran Saini - saini19@my.yorku.ca
Esha Vij - eshavij@my.yorku.ca
Harrish Elango - elangoha@my.yorku.ca
Adewusi Fashokun - korede@my.yorku.ca
Siddhanth Bakshi - sbakshi@my.yorku.ca

Abstract

This concrete architecture report provides a thorough overview of Apache Flink, specifically version 1.17.1. The subsystem that is focused on in this report is the Cluster Manager. This report provides an in-depth analysis of the Cluster Manager's architecture, including its top-level components and their connections. The architecture is broken down to show the underlying subsystems with the utilisation of tools such as Understand. Moreover, a containment file is also created to describe the process of extracting the architecture of the software system. This report examines the architectural styles used in Cluster Manager substem, highlighting the structural and behavioural patterns incorporated to allow the Cluster Manager to fulfil its responsibilities efficiently. Furthermore, the report explains noteworthy design patterns that are intrinsic to the subsystem's design, clarifying their significance and influence. The report also discusses the difference between the conceptual and concrete architecture of the overall architecture of Apache Flink, using a reflexion model. The report also outlines any noteworthy aspects of the architecture and its subsystems. Moreover, the report includes clear and concise diagrams to show the complex relationships between components, drawing on the insights gained from the architecture study to provide a visually appealing depiction of the Cluster Manager's structure. In addition, sequence diagrams are used to highlight significant features of the architecture and its subsystems. They clarify the flow and interactions inside the Cluster Manager and provide information about important operational sequences between its components. The report includes concurrency and use cases. The insights presented in this report are intended to facilitate a thorough grasp of the internal workings and design components of the Cluster Manager subsystem inside Apache Flink by offering a complete understanding of its architectural foundations.

Introduction and Overview

This paper primarily focuses on the Cluster Manager component within the Apache Flink software system. The Cluster Manager holds a pivotal role in the software's architecture, tasked with responsibilities such as scaling, deploying, and monitoring Apache Flink applications. A more profound examination of the Cluster Manager's architecture allows us to gain insights into the inner workings of Apache Flink.The Cluster Manager serves as the central coordinator for managing distributed tasks across a multitude of devices. It encompasses several integral components, including the JobManager, TaskManagers, and ResourceManager. The JobManager's core responsibility is to oversee and coordinate the execution of Flink's jobs, ensuring their efficient distribution across the available resources within the cluster.Conversely, TaskManagers are entrusted with executing tasks assigned by the JobManager. They handle the actual data processing workload and provide the essential computational resources, such as CPU and memory, for task execution. Cluster Managers are responsible for the distribution of tasks to multiple nodes, making critical decisions regarding which TaskManager should execute specific tasks based on the availability of resources.Furthermore, Cluster Managers actively monitor the health of nodes to preemptively reduce failures. In the unfortunate event of a failure, they initiate the recovery process, restarting tasks to maintain the system's robustness and reliability. A Lot of companies benefit from Cluster Managers fraud detection ability especially financial institutions that leverage Flink as it analyses data upon arrival to quickly check any fraudulent activities. Cluster Managers use Master-Slave Architecture, Observer Pattern and Resource Allocation Patterns as its design patterns and architectural styles.

Architecture

The Cluster Manager is a critical component of the overall architecture, responsible for managing and coordinating the execution of Flink applications (jobs) within a distributed cluster. It comprises various subsystems, with key components including the Job Manager, Job Master, Task Manager, Dispatcher, and Resource Manager. This integral unit plays a vital role in the entire Flink ecosystem. When a job is submitted, the Cluster Manager takes charge of overseeing the entire processing pipeline in Apache Flink. [6]

The primary functions associated with Cluster manager within Apache Flink are the following:

- **Job Submission and Scheduling**: Whenever a job is submitted to the Flink cluster, the Flink cluster uses the Job Manager for scheduling the jobs and attaches a Job Master to every job. It determines how to distribute the job's tasks across the available TaskManagers in the cluster
- **Resource Management:** The Flink cluster manages the cluster's available resources. The job manager maintains a synchronised connection with the resource manager to see the available resources and then assign the resource to the jobs accordingly, on the basis of availability. The Cluster Manager ensures that resources are allocated efficiently to different jobs and their individual tasks.
- **Fault Tolerance**: It monitors the TaskManager and JobManager and takes action for recovery, in case of any failure to ensure proper functioning of Flink in case of any hardware or software failure.
- **Scaling**: It can dynamically scale the cluster by adding or removing TaskManagers based on workload or resource requirements.
- **Monitoring and Logging**: The Cluster Manager often provides a web-based user interface for monitoring job progress, resource utilisation, and system metrics. It also handles logging and reporting for job execution.

To extract the dependencies, we obtained the source code for Apache Flink and ran it through a software tool called Understand, which allows us to navigate through the code and also has the ability to identify and show the dependencies between the different parts of the code. Although this program does have the ability to show the dependencies, it was shown from entity to entity which made it very large and hard to follow. To make visualising the dependencies easier, we first used the Understand software to extract the dependencies of Apache Flink onto a data file which we called Flink_UnderstandFileDependency.csv. After extracting the code dependencies into a .csv file, we had to convert it into another file format called ".raw.ta" which we created by running a Perl script named trransformUnderstand.pl that was given to us by the professor. This script created the .raw.ta file which we needed and has the same name as the .csv file.

The next step to extracting the dependencies for viewing was to create a containment file pertaining to the different subsystems in the code that we were interested in studying and how the different functions in the source code lined up with these subsystems that we decided were important. This started off as a tedious process as we were combing through the over 60000 lines of data manually, to decide if a dependency link was relevant to our subsystem of choice or not. As this was taking much longer than we liked, we decided that we were going to create a script to create the containment file. We used a Javascript script called generate-containment-file.js. This created the containment file we needed and grouped the functions into subsystems we created using JSON to create the structure of the different subsystems. With our .raw.ta file and our containment file created, we ran them both through a batch script given to us called createContainment.bat, which used both files to create two new files, called Flink_UnderstandFileDependency.ls.ta and Flink_UnderstandFileDependency.con.ta, the latter of which is what we used to run LSEdit on to visualise the dependencies of the source code.

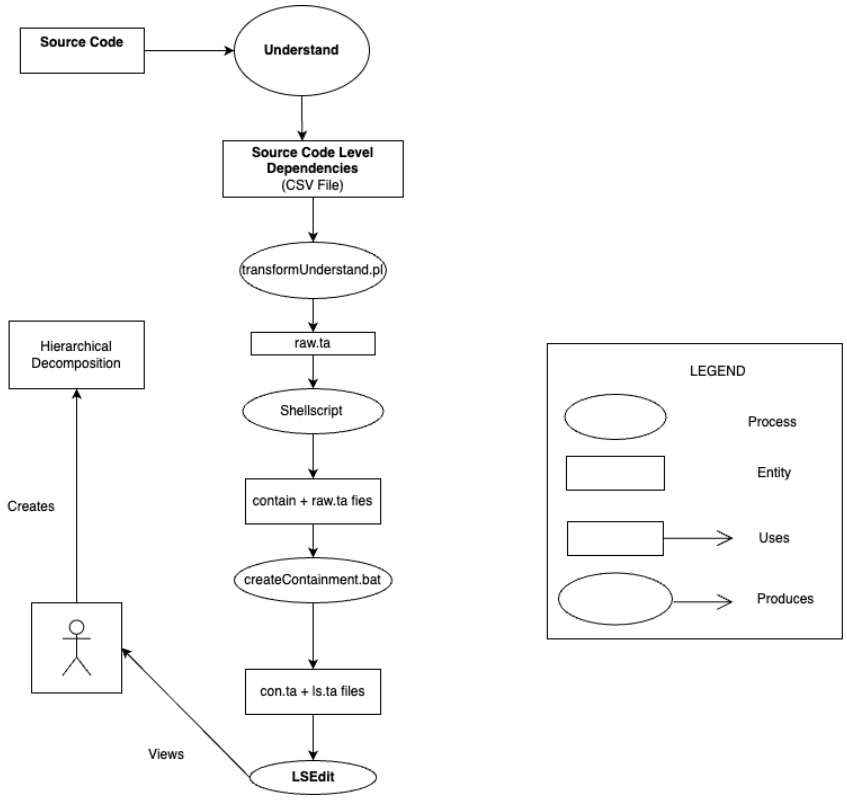

Figure 1- This is a flow chart that shows the process we took to extract the dependencies from Apache Flink.

Running LSEdit allowed us to visualise the extracted dependencies that we were interested in understanding and this is also where we learned that the Cluster Manager was not actually its own subsystem, but was a portion of the Runtime subsystem and was made up of many of the smaller subsystems found in “Runtime”.

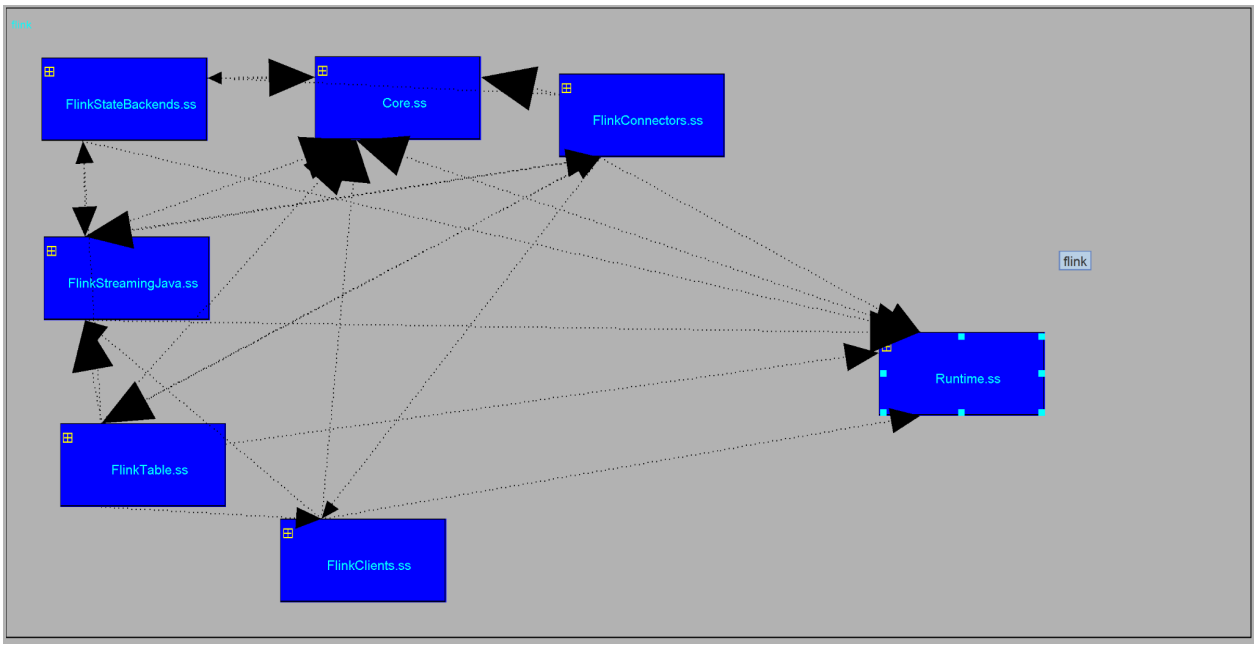

Figure 2- High level overview of the Apache Flink Architecture and all of the subsystems that make the software as a whole. All other subsystems in Flink appear to depend on the runtime subsystem.

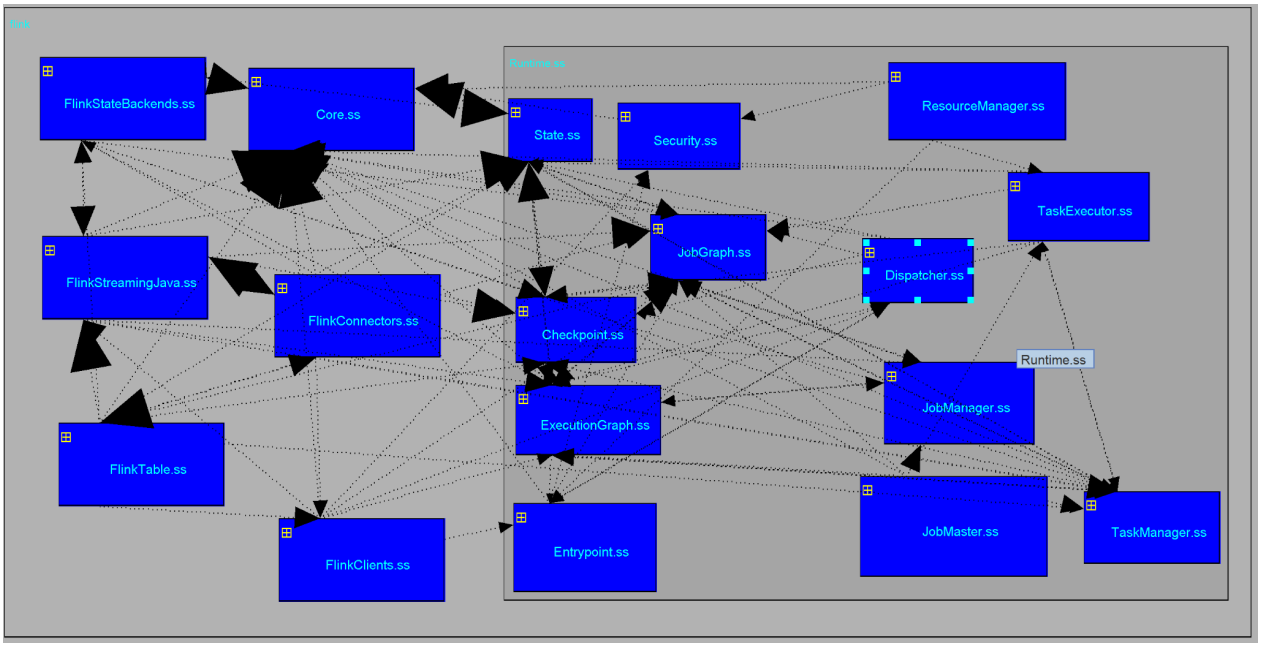

Figure 3- This is the high level overview of Apache Flink with the runtime subsystem opened up to see the different subsystems that make it up. The 6 subsystems on the left in the runtime subsystem are what actually make up the Cluster Manager.

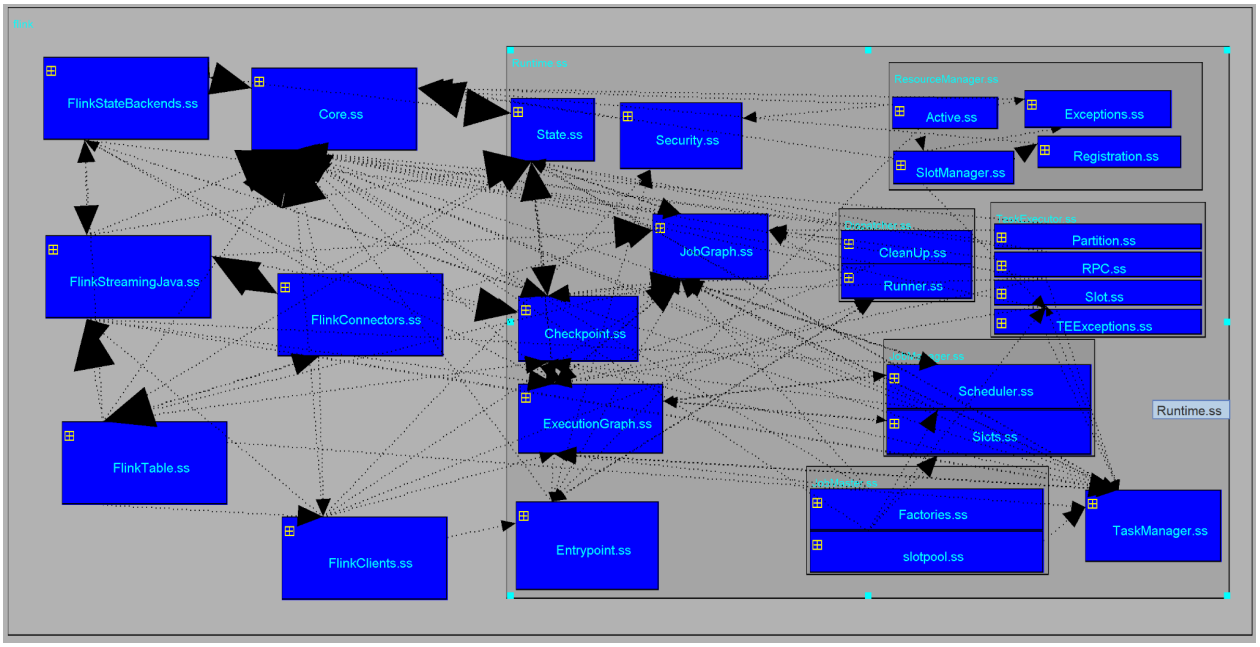

Figure 4- This is the next level down from the subsystem view above. This image has the subsystems that make up the Cluster Manager broken down into their constituent subsystems. This also allows you to see the dependencies between the different subsystems in the “Cluster Manager” and the other subsystems in Runtime.ss and the rest of the Flink program subsystems.

Architecture Styles

A number of different architectural styles are employed by Apache Flink to ensure that cluster management is handled in a well-organised and performant manner. Since this portion of the Flink system includes multiple subsystems like the JobManager, TaskManager, ResourceManager and JobMaster and the interactions between them, there are a number of noteworthy architectural styles to be seen. Some of these include:

Client-Server: Flink, through the use of its APIs, enables the submission of jobs to the Flink cluster. In particular, this functionality is implemented in the Dispatcher class. Flink's APIs also expose the status of the cluster (cluster health) as well as the state of both recently executed and currently running jobs. This makes it possible for external services to remain informed of the current state of the cluster, and report potential issues.

Master/Slave: Although the JobManager now consists of smaller components, like the ResourceManager and JobMaster, the JobManager has a master-slave relationship with the cluster's TaskManagers. This means that while there is only ever one active JobManager instance, this instance passes executable tasks to the available TaskManagers. To achieve this, it keeps track of the active TaskManagers and decides what tasks to pass to each of them. While this is the case, it is also important to note that in practice, there is usually more than one JobManager, but the other ones are kept on standby, in case there is an issue that causes the primary instance to fail. [5]

Pipe-and-filter: Arguably the most prominent architectural style employed in the implementation of the Flink cluster, pipe-and-filter is used in a number of subsystems. One of these is stateful stream processing, where messages are passed from the message producer, to the message queue, and finally to the Window Operator for processing.

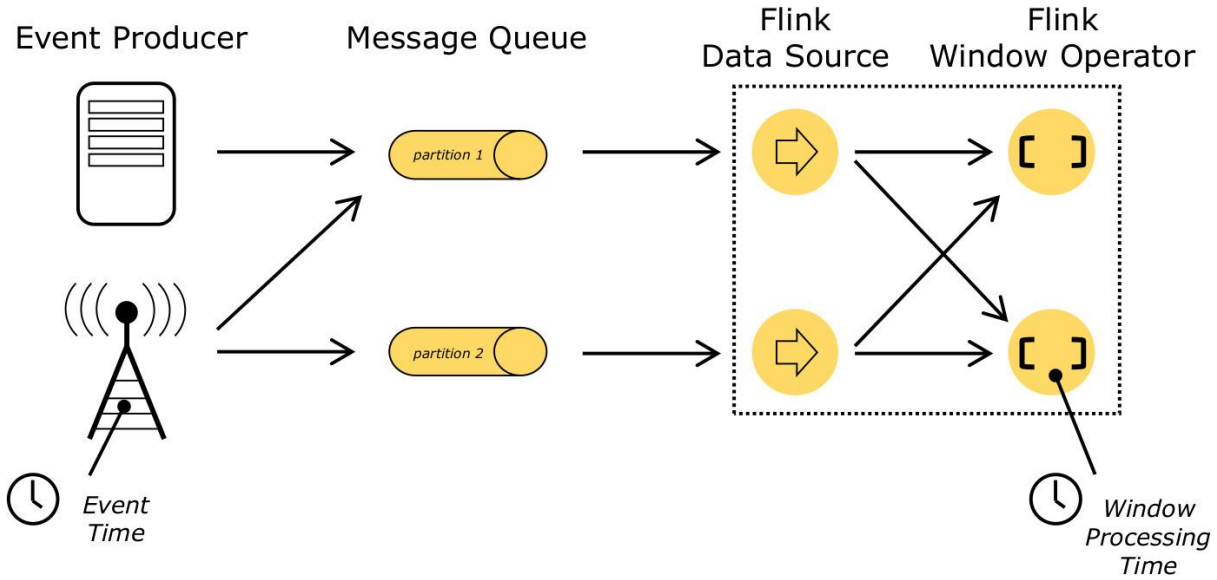

Figure 5: An overview of stream processing in Flink, using the pipe-and-filter architectural pattern.
Source: https://nightlies.apache.org/flink/flink-docs-release-1.17/docs/concepts/time

Design Patterns

Several design patterns are applied in Cluster Management:

1. Singleton Pattern:
This pattern ensures the Flink cluster session is managed with only a single instance of the cluster manager, maintaining a singular point of control.

2. Resource Management:
Resource Management adopts a centralised approach to allocate and release resources for Flink jobs in Apache Flink. It encompasses tasks such as configuring parallelism, optimising resource utilisation, and setting up TaskManagers.

3. Observer Pattern:
The Observer Pattern is employed to define mechanisms like JobManager, TaskManagers, and Flink Sessions. These components require notification of changes in observable objects during cluster events and may be responsible for monitoring or executing automated actions based on these events.

4. Master/Slave Pattern:
In this pattern, the JobManager serves as the master node, while the TaskManager acts as the slave node. This design allows for concurrent processing within Cluster Management. [4]

5. Builder Pattern:
The Builder Pattern is utilised in cluster management to create different representations of objects or job configurations using a set of function arguments. This simplifies the definition of data processing workflows and job parameters for the user. [3]

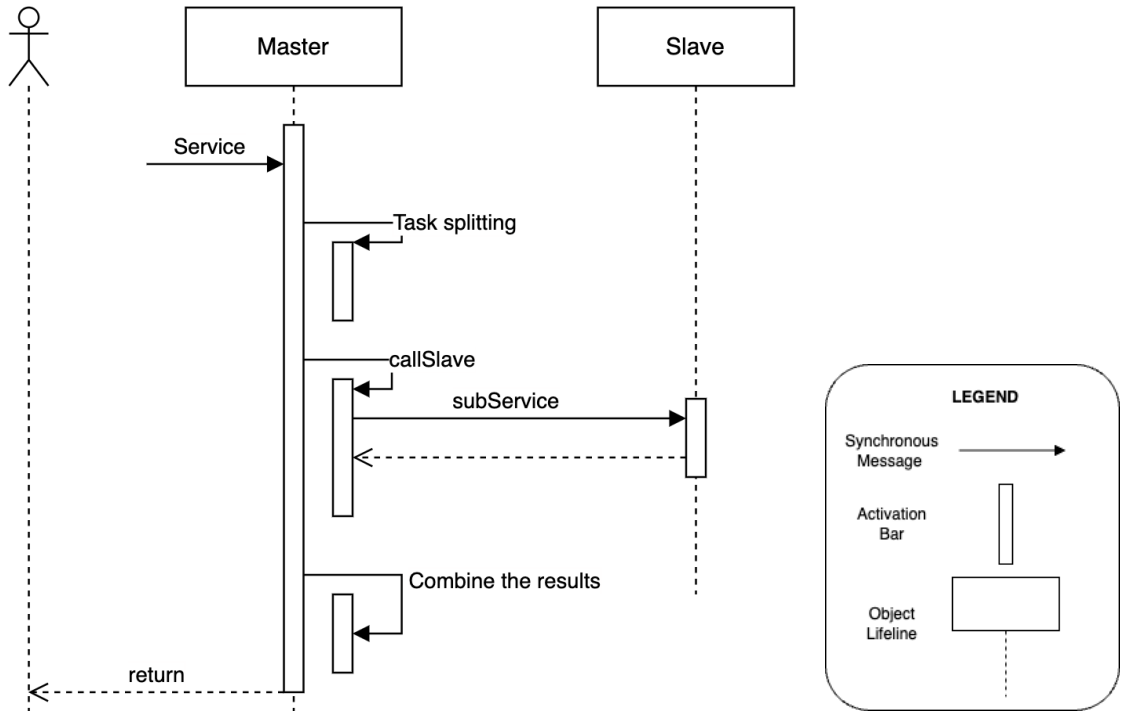

Figure 6- This sequence diagram illustrates the Master-Slave Design Pattern.

Reflexion Model

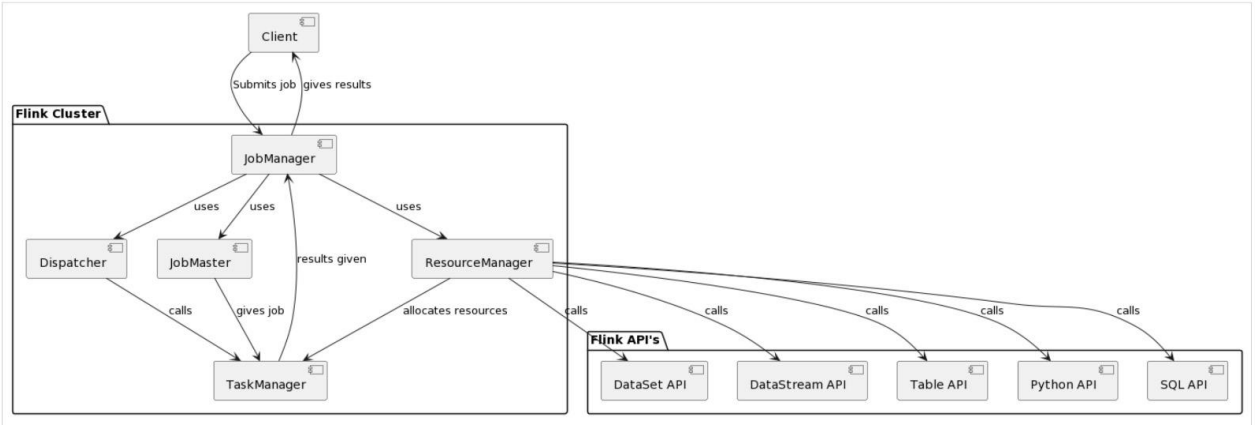

Figure 7- Conceptual Architecture of Apache Flink from Assignment 1

Figure 7 from above is the conceptual architecture that was presented in the Conceptual Architecture Assignment (Assignment 1). The conceptual architecture diagram was created after studying the Apache Flink documentation. The documentation also included information in terms of its components and the responsibilities they fulfilled.

Noteworthy Aspects

The cluster manager is essential to Apache Flink's functionality. Within Apache Flink, some of the cluster manager's duties comprise of the following: Resource Allocation, Task Scheduling, Resource Cleanup, TaskManager Health Monitoring, and Failure Recovery. It is possible to think of the cluster manager as a higher-level system that does not always include explicit subsystems, unlike Apache Flink. Rather, the cluster manager usually manages resources for Flink operations and communicates with its underlying components. Some of these components include: ResourceManager, Scheduler, Health Monitoring and Reporting components and Resource Cleanup components. [1]

The ResourceManager is responsible for the distribution of resources and management. It communicates with the cluster's nodes to distribute resources to the JobManagers and TaskManagers for Flink job executions. The cluster manager ensures that the necessary CPU, memory, and other resources are available for completing tasks. This is an essential component of the cluster manager because the effectiveness and scalability of Flink tasks depend on the efficient allocation of resources. Jobs might not operate as efficiently without adequate resource management, which could result in resource contention and subpar job performance. The cluster manager's Scheduler component decides when and where to conduct Apache Flink jobs. It considers job requirements, load balance, and resource availability before doing this. It is an essential component of the cluster manager because the Scheduler speeds up work completion times and helps prevent overloading specific nodes. The Health Monitoring and Reporting components involve evaluating the condition of the cluster and its individual components. Monitoring contributes to the real-time identification and resolution of problems, maintaining the stability and responsiveness of the Flink cluster. Moreover, the Resource Cleanup components are responsible for releasing unused resources when a project is finished or resources need to be scaled back. By ensuring that resources are not wasted, they can be made available for use in other projects or tasks. It facilitates cost control and resource optimization. [2]

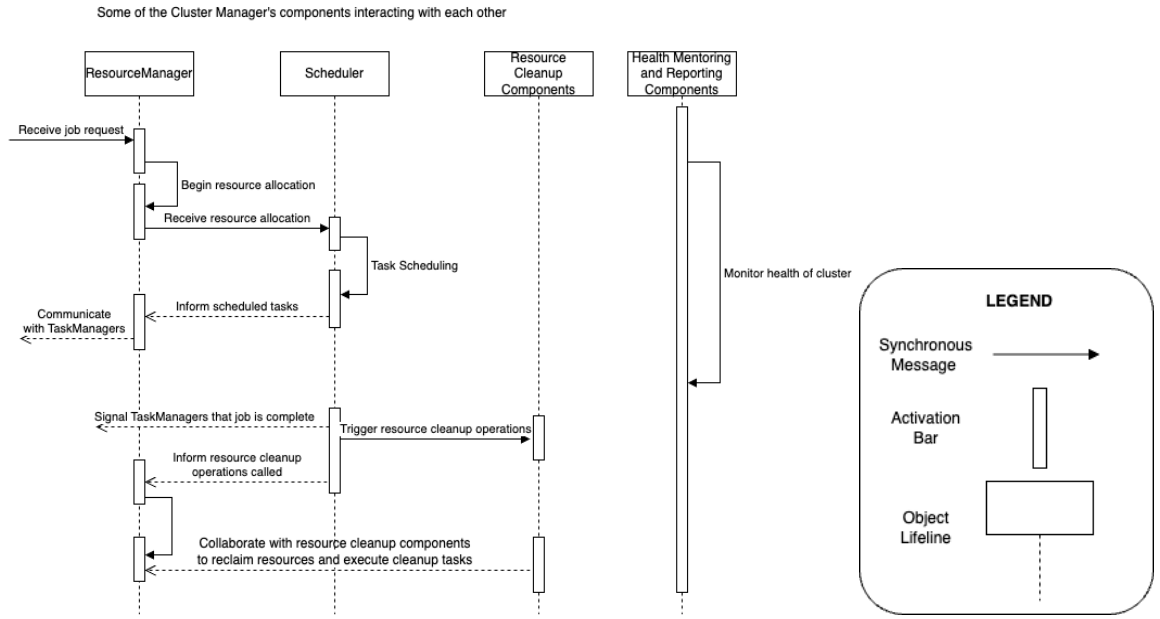

Figure 8- Sequence Diagram of Cluster Manager's components

The sequence diagram above (Figure 8) demonstrates some of the Cluster Manager's components interacting with each other. The Cluster Manager receives a job request from the user. After receiving the job request, the ResourceManager starts allocating resources. It looks for resources that are available to do the task. The Scheduler component determines task scheduling after resources are assigned, taking load balance, resource availability, and job requirements into consideration. The ResourceManager contacts TaskManagers to start task execution after the Scheduler notifies it of the tasks that are scheduled. The Scheduler notifies the TaskManagers that the work has ended when it is completed. Simultaneously, the Scheduler initiates resource cleanup procedures and notifies the resource manager of their call. To recover resources and carry out cleanup operations, the ResourceManager works in collaboration with the Resource Cleanup Components. The cluster's health is continuously assessed by the monitoring and health components during the entire execution in real-time.

External Interfaces

The cluster manager provides various external interfaces. Some of the key external interfaces are:

- **Web UI**: Cluster manager encompasses a Web UI for monitoring job progress, resource utilization, and system metrics. This is also responsible for managing the overall performance and health of the flink cluster in a more front-end manner.
- **REST API**: The flink cluster encompasses a "Dispatcher" which uses a REST API to submit a Flink Job and attach a Job master to it.
- **Log**: The cluster also offers a log for the developer to see the system logs and send the output data streams to a centralized location for further processing or analysis.
- **Monitoring System:** The cluster manager integrates with external monitoring systems such as Apache Metric for monitoring and visualization of the clusters.

Use Cases

1. An ecommerce website like Ali Baba that needs to process real-time customer clickstream data for personalised recommendations. This can be done using apache Flink with resource manager which allocates resources throughout the day depending on different loads. This ensures that the cluster is scaled up or down depending on the incoming data volume and provides a seamless experience for users.
2. Apache Flink is used in many financial departments as a fraud detector tool. These institutions require large amounts of transaction data to be processed in real time. Apache Flink allows them to submit fraudulent transactions into the cluster which ensures the job Manager executes the fraud detected activities while also detecting fraudulent activities efficiently.
3. Uber is a ride service provider which uses Apache Flink to process real-time location data to match users to the nearest riders. Apache Flink allows the expansion of the cluster by introducing additional task managers in order to handle the increasing processing capabilities during peak hours.

Concurrency

In terms of the concurrency within the Cluster Manager, there are many instances where multiple tasks are executing simultaneously.

1. The Cluster Manager uses concurrency for allocating resources. To facilitate effective parallel execution, it divides resources, including CPU cores, memory, and network bandwidth, among several running tasks or jobs.
2. The concurrent scheduling of jobs throughout the cluster is managed by the Scheduler of the Cluster Manager. It effectively handles the distribution of jobs and the duration of their completion, taking into account a number of variables such as resource availability and load balance.
3. Under the Cluster Manager, components for health monitoring and reporting run in parallel to continuously check on node health and provide real-time information about system performance and possible problems.

Conclusions

In conclusion, Apache Flink's concrete architecture offers a concise framework for understanding Apache Flink's Cluster Manager architecture that plays a vital role in distributing tasks within the system. In this report we have provided a deep analysis of the important components, architectural styles and the design patterns that are used to make the Cluster Manager efficient. It shows the role of the Cluster Manager in job submissions and scheduling, resource allocation, fault tolerance, scaling, monitoring and logging. Moreover, the report includes important sequence diagrams to visualise the important components that interact with each other. The report has noteworthy aspects of the Cluster Managers responsibilities which are thoroughly examined through the sequence diagrams.This report acts as a guide for both novices and experts in the field looking to take advantage of Apache Flink's features. The report gives an overview of external interfaces that are provided by the Cluster Manager which allows for web-based monitoring, REST API interactions, log management and integration with external monitoring systems.

The following are our proposals for future directions:

1. Optimising Scheduler Performance: add smarter job scheduling techniques for maximising load balancing throughout the Cluster.
2. Improved Real-Time Health Monitoring: Enhance the Cluster's alerting and monitoring features to identify and notify possible problems within the cluster.

The following is a summary of our key findings:

1. The Importance of the Cluster Manager's Role: The Cluster Manager serves an important role within Apache Flink, which has important responsibilities such as resource distribution, task scheduling, health tracking, and failure recovery.
2. Using Dependency Extraction Tools like Understand: Through this assignment, we got familiar with utilising tools such as Understand and LSEdit in order to extract dependencies and for creating containment files.
3. The Cluster Managers ability to handle multiple tasks and jobs concurrently by assigned and managing tasks while making sure parallel execution and monitoring job progress in real-time.

Lessons Learned

As a group, we wished that we had prior knowledge of the analytic tools and their features in order to expedite the process of extracting architectural insights. By having improved proficiency with tools like Understand, we could have sped up the Cluster Manager's top-level entity mapping procedure to subsystem mapping. Moreover, a more thorough examination of the Cluster Manager's design would have been facilitated by a cooperative strategy incorporating cooperation and knowledge exchange. It's possible that team members' ideas, varied viewpoints, and cross-verification of results would have revealed more subtle information and provided all-encompassing viewpoints. Something that we would have done differently was rather than depending only on Understand to extract the architecture, we could have investigated a variety of different analytic tools or alternative software could have provided a more thorough understanding of the architecture of the subsystem. Different insights from each technique could have improved the understanding of the Cluster Manager and its subsystems.

References

1. “Improvements in task scheduling for batch workloads in Apache Flink 1.12,” Apache Flink, https://flink.apache.org/2020/12/02/improvements-in-task-scheduling-for-batch-workloads-in-apache-flink-1.12/ (accessed Oct. 26, 2023).
2. “Flink architecture,” Apache Flink 1.11 Documentation: Flink Architecture, https://nightlies.apache.org/flink/flink-docs-release-1.11/concepts/flink-architecture.html (accessed Oct. 26, 2023).
3. “Creational Design Patterns,” Refactoring.Guru, https://refactoring.guru/design-patterns/creational-patterns (accessed Oct. 27, 2023).
4. Resource manager design pattern, https://www.eventhelix.com/design-patterns/resource-manager/#:~:text=The%20resource%20manager%20design%20pattern,resource%20usage%20in%20the%20system. (accessed Oct. 30, 2023).
5. Master-worker pattern, https://docs.gigaspaces.com/solution-hub/master-worker-pattern.html#:~:text=The%20Master%2DWorker%20Pattern%20 (accessed Oct. 29, 2023).
6. “Code style and quality guide - java,” Apache Flink, https://flink.apache.org/how-to-contribute/code-style-and-quality-java/ (accessed Oct. 28, 2023).