BluePrint for Understanding the Differences in Apache Flink's Concrete and Conceptual Architecture

Discrepancy Analysis
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 discrepancy analysis report provides a detailed description of the difference between Apache Flink's concrete and conceptual architecture. The version of Apache Flink in context is 1.17.1. This report aims to uncover and comprehend differences at the top subsystem level as well as the detailed design level for the Cluster Manager through a thorough reflective analysis that contrasts the conceptual architecture created in Assignment 1 with the concrete architecture in Assignment 2. The report explores the causes of these differences and suggests adjustments to bring the two designs into line. Our analysis reveals subtle changes in the overall structure and subsystem relationships at the top subsystem level. At the design level, the Cluster Manager is examined in detail to identify particular differences between the conceptual and concrete architectures. These variations show how inconsistencies must be addressed and corrected reflexively. The report demonstrates the importances of the new dependencies discovered to each subsystem and what those dependencies provide for that particular subsystem. The research presented here suggests specific changes to the conceptual architectures to address the found disparities. Rethinking inter-subsystem dependencies and streamlining system needs are some of the changes made to the conceptual architecture, to bring the two architectures together. The report includes clear visualisations of the differences in the top-level and design level subsystems. The report also discusses any changes made to the use cases and sequence diagrams created in previous assignments. Moreover, it mentions the lessons learned and limitations found when conducting the reflexion analysis. Through comprehending the causes of deviations and suggesting focused solutions, the report supports the continuous development and enhancement of the architectural framework, promoting a stronger and more coherent system architecture.

Introduction and Overview

In our preceding tasks, we deeply investigated the intricate operations of the Cluster Manager, a pivotal component comprising various subsystems such as JobManager, JobMaster, TaskManager, and ResourceManager. Together, these elements play a crucial role in facilitating the smooth deployment and operation of jobs within our system. The objective of today's assignment is to delve deeper into the architecture of these subsystems and suggest modifications to the conceptual diagrams based on our insights and discoveries.
This report offers an in-depth reflection analysis focusing on the Apache Flink Runtime, comparing and contrasting its Conceptual Architecture with the Concrete Architecture Diagram. The investigation examines the subsystems within the Cluster Manager, identifying disparities and proposing modifications to enhance the conceptual diagram's accuracy. Examining both high-level and design-level subsystems, this report highlights discrepancies, particularly noting the absence of dependencies such as Checkpoint, State, and TaskExecutor in the initial Conceptual Architecture Diagram. Through our analysis and code exploration aided by GitLens, the report sheds light on the intricacies of the architecture, showing the interdependencies among subsystems like JobManager, TaskManager, ResourceManager, and others. The proposed modifications aim to streamline and make the Conceptual Architecture Diagram more parallel and fit closely with the Concrete Architecture, integrating newly discovered dependencies and enhancing readability for a comprehensive representation. The

report covers the process of identifying discrepancies, proposing modifications, and emphasising the significance of a thorough understanding of subsystem interdependencies. Lessons learned emphasise the necessity of a Concrete Architecture Diagram to capture intricate dependencies missed in the Conceptual Architecture. However, it also acknowledges potential limitations due to subjective interpretation among team members, influencing the accuracy and completeness of identified differences. In summary, this report serves as a comprehensive guide to understanding and aligning the Conceptual and Concrete Architectural diagrams for Apache Flink, emphasising the significance of continuous review, documentation, and adaptability in architectural development.

Detailed Descriptions of Reflection Analysis Process

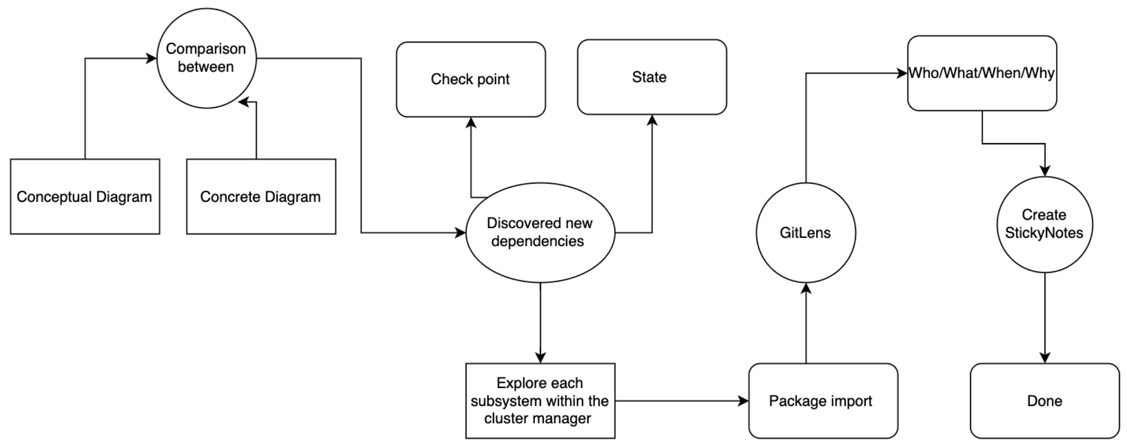

Figure 1 - Reflection Analysis Process

To conduct this reflexion analysis, we isolated the Runtime part of Apache Flink as the subject of our analysis.We compared the Conceptual Architecture of the Runtime to the Concrete Architecture Diagram.We reported our findings and investigated the gaps between the two diagrams..We improved the dependency readability to more clearly visualise them in LSEdit. We highlighted the new dependencies noticed in the Concrete Architecture Diagram.Through this analysis, we discovered that the **Checkpoint** and **State** dependencies were not accounted for in the conceptual diagram in each of the design level subsystems discussed. **TaskExecutor** was also not discovered in the Conceptual Architecture Diagram as a subsystem itself. We imported the package and with the help of GitLens we were able to get a comprehensive set of tools for exploring and understanding the history of our codebase. We used the Who/What/When/Why method to gather a more detailed understanding of the findings and helped in drawing conclusions or lessons learned from the process.

Conceptual Architecture vs. Concrete Architecture

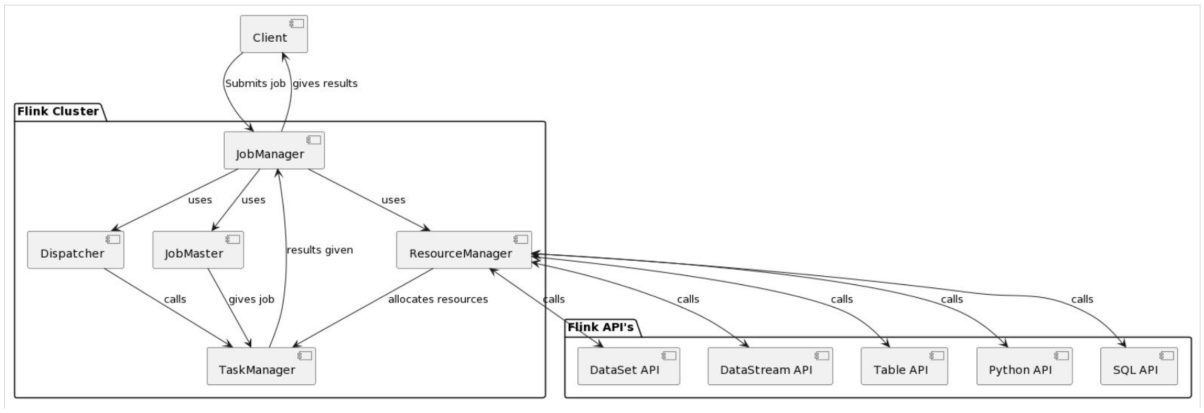

Figure 2 - Top-Level Conceptual Architecture of Apache Flink
The image above shows our conceptual architecture of Apache Flink, and the dependencies we believed to exist from reading the documentation. As shown in the Conceptual Architecture diagram, this initially led us to believe that the **JobManager** directly communicates with the mentioned sub-systems: **Dispatcher**, **JobMaster** and **ResourceManager** and **TaskManager**, and these components made up the "ClusterManager". The diagram also shows how the **ResourceManager** interacts with the various involved API's. This diagram shows a very over-the-top relationship of the involved dependencies as we were not aware of any intricacies of the system.

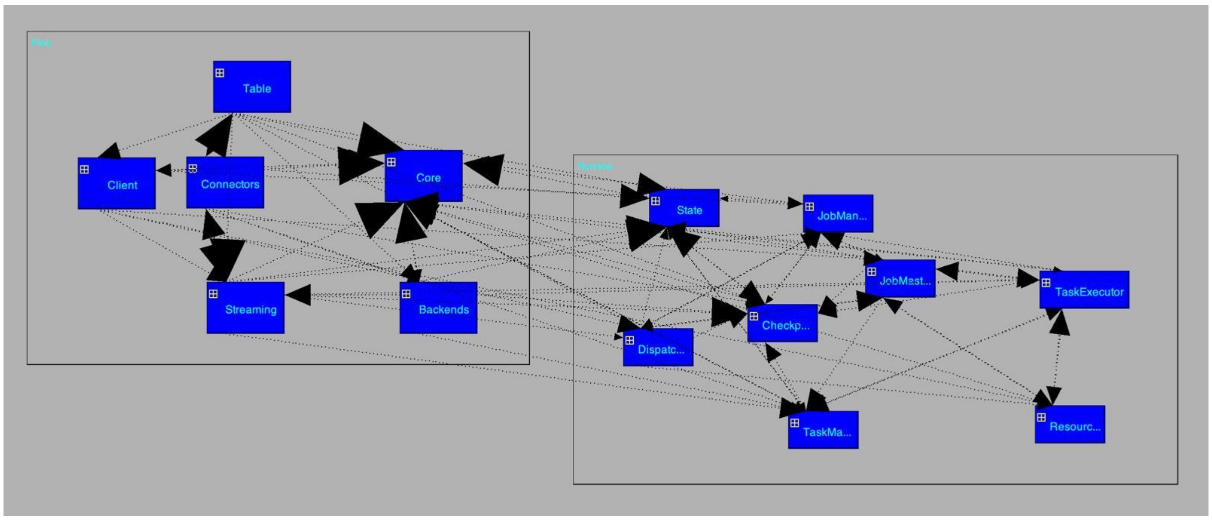

Figure 3 - Top-level Concrete Architecture of Apache Flink

The diagram above shows the top-level concrete architecture of Flink we obtained through LSEdit. This is when we became aware that there were more subsystems than expected, and the image shows all the other subsystems in Flink that make up the entirety of the application. The subsystems on the right are encapsulated within the **Runtime** subsystem, which will be our focus, while the others are external to the core functionality of Flink, e.g. the client, backend and streaming APIs, as well as the Table API.

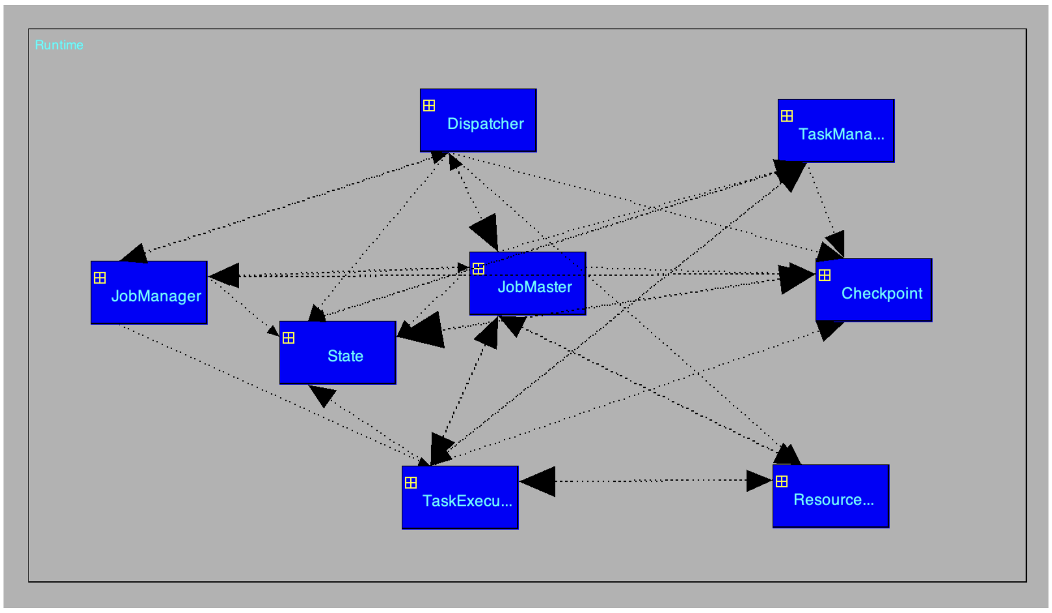

Figure 4 - Top-level Subsystem: Runtime

Above is a diagram displaying the concrete top-level architecture of the Flink **Runtime** subsystem in the Flink application. This is the subsystem where we assumed the ClusterManager would be located. This is where we realised that the "ClusterManager" subsystem does not exist, but is made up of all the **Runtime** subsystems seen above, that are involved in managing the Flink clusters. The main components of the "ClusterManager" in the **Runtime** subsystem are **Dispatcher**, **JobManager**, **JobMaster**, **TaskManager**, **TaskExecutor**, **ResourceManager**, **Checkpoint**, and **State**, as seen above, and we will go into further detail into some of the subsystems below.

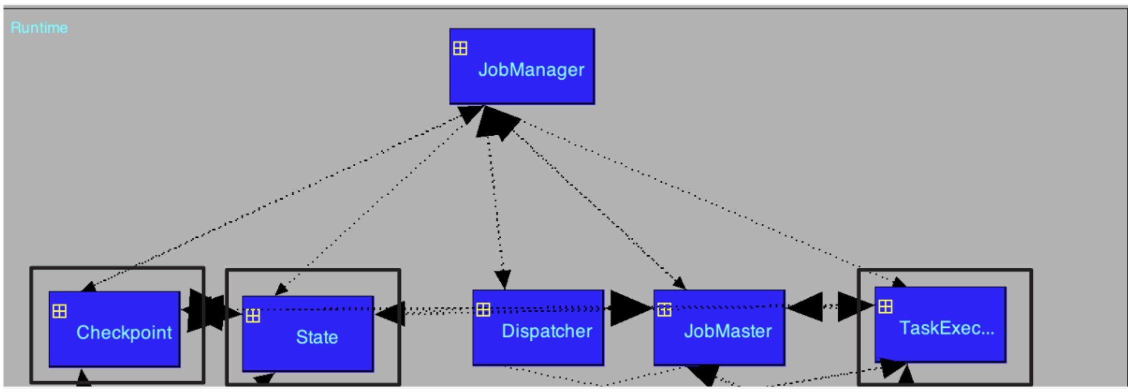

Figure 5 - Design-level Subsystem Architecture: JobManager

The image above shows the design-level architecture of the **JobManager** subsystem with every other subsystem in **Runtime** that is dependent on, or a dependent of, **JobManager**. The **JobManager** has many roles in the Flink application, from deciding when to schedule the next task or next set of tasks to TaskExecutors, therefore knowing when tasks are completed or fail, dealing with the tasks once they fail, and checkpoint coordination. The documentation states that it completes these jobs through using the **Runtime** subsystems **ResourceManager**, **Dispatcher,** and **JobMaster**[3]. When doing the comparison of this architecture to our conceptual architecture, we realised that there was no direct dependency between **JobManager** and the **TaskManager** and **ResourceManager** subsystems, but there were dependencies with the newly found **Checkpoint** and **State** subsystems which the **JobManager** relies on to make sure proper progress is made and saved, respectively. One job of the **JobManager** we learned was to find which **TaskExecutors** are assigned to a specific **TaskManager** and schedule them a job using its internal **Slots** and **Scheduler** subsystems. We noted that although there is no direct dependency link between the **JobManager** and the **TaskManager**, this behaviour through the **TaskExecutor** is the connection between the two subsystems. We also learned that the connection between the **JobManager** and the **ResourceManager** is indirect and through the **Dispatcher** subsystem.

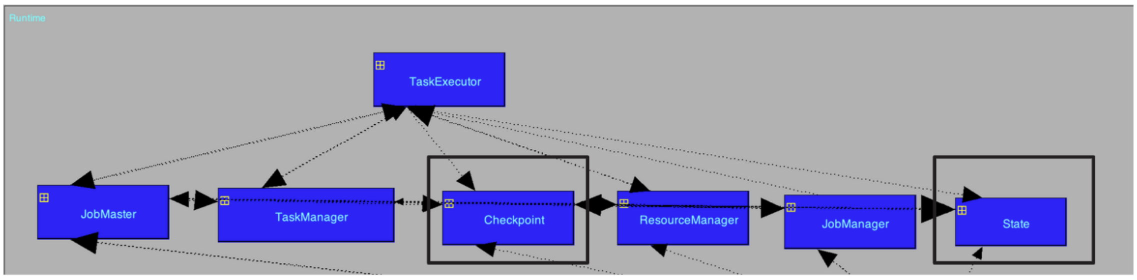

Figure 6 - Design-level Subsystem Architecture: TaskExecutor

The image above shows the design-level architecture of the **TaskExecutor** with every other subsystem in **Runtime** it interacts with. The main job of the **TaskExecutor** is to execute the tasks assigned to the Flink software by the previously mentioned **JobManager** subsystem, as it is the base unit of execution for the application. This is a subsystem that we did not discern during our conceptual architecture analysis as we thought it fell under the **TaskManager** subsystem. We noted that this subsystem has dependencies with every other subsystem of the "**ClusterManager"** except **Dispatcher**. It has dependencies through its internal **RPC** subsystem with **Checkpoint** and **State** subsystems, which it uses to take snapshots of the progress and store them for later, respectively.

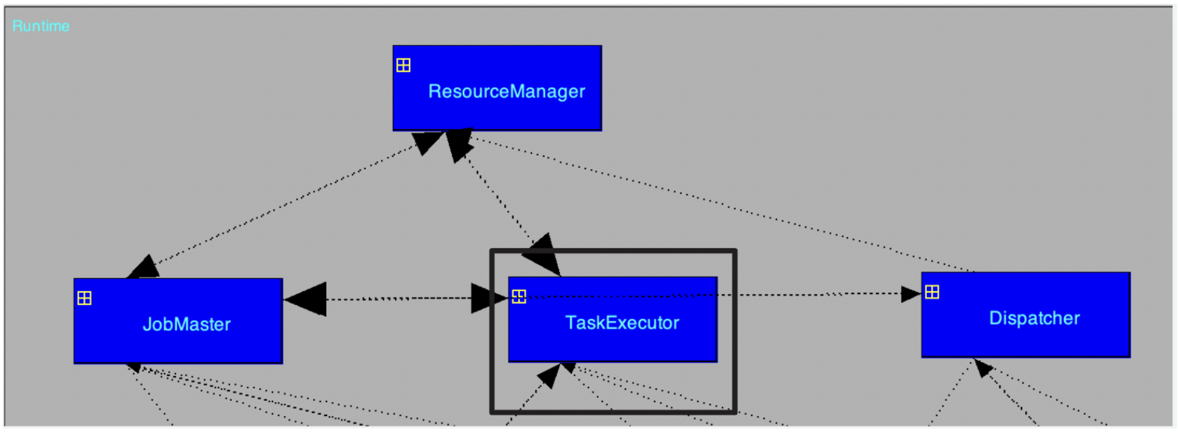

Figure 7 - Design Level Subsystem Architecture: ResourceManager

The **ResourceManager** is responsible for distributing and managing resources within the Cluster. The **ResourceManager** also communicates with the Cluster's nodes to distribute resources to the **JobManagers** and **TaskManagers**. In essence, the **ResourceManager** is in charge of managing and assigning resources throughout the whole distributed Cluster, while the **TaskExecutor** is in charge of carrying out certain tasks on a designated node. Some of these tasks include, but they are not limited to, processing data or carrying out actions. The **TaskExecutor** is also responsible for overseeing a particular node's local storage, memory and CPU [2]. The **TaskExecutor** greatly relies on the **ResourceManager** to find the necessary resources in order to carry out tasks effectively and efficiently. In our conceptual model created Assignment 1, we did not discover the **TaskExecutor** subsystem in general, let alone depend on the **ResourceManager.**

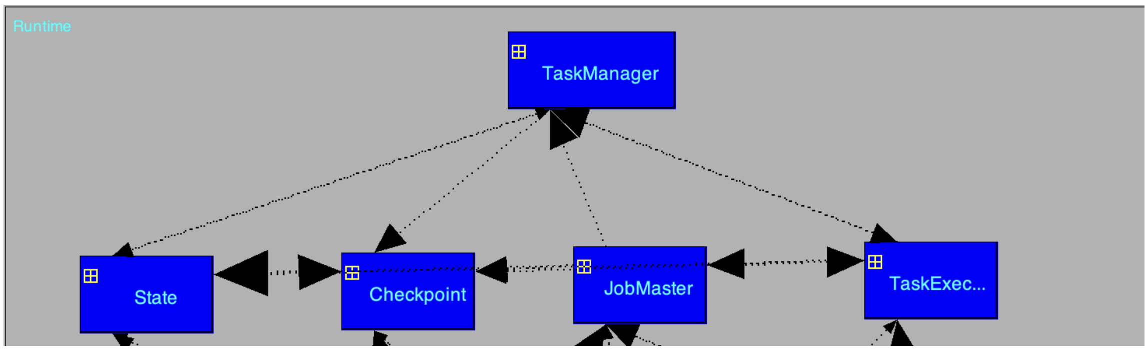

Figure 8 - Design Level Subsystem Architecture: TaskManager

The **TaskManager** subsystem of the runtime is responsible for the receival of scheduled tasks from the **JobManager**, and passing them to **TaskExecutor** instances for execution. There are usually multiple instances of the **TaskManager**, to help handle multiple tasks concurrently. It is important to stress the juxtaposition of the **TaskManager** and **TaskExecutor**, as the latter was one of the subsystems that we discovered in the process of generating the concrete architecture diagram. As opposed to the general tasks scheduling (and inter-process communication) carried out by the **TaskManager**, the **TaskExecutor** is saddled with the responsibility of executing a specific task passed to it by the **TaskManager**.Dependencies on the **State** and **Checkpoint** subsystems were also noticed during the reflection analysis process. From perusing the code, we deciphered that the **TaskManager** requires these dependencies, to update the **State** subsystem of the status of the task(s) being executed. In its **CheckpointResponder** class, it also exposes methods to report the checkpoint metrics for a given checkpoint and job.

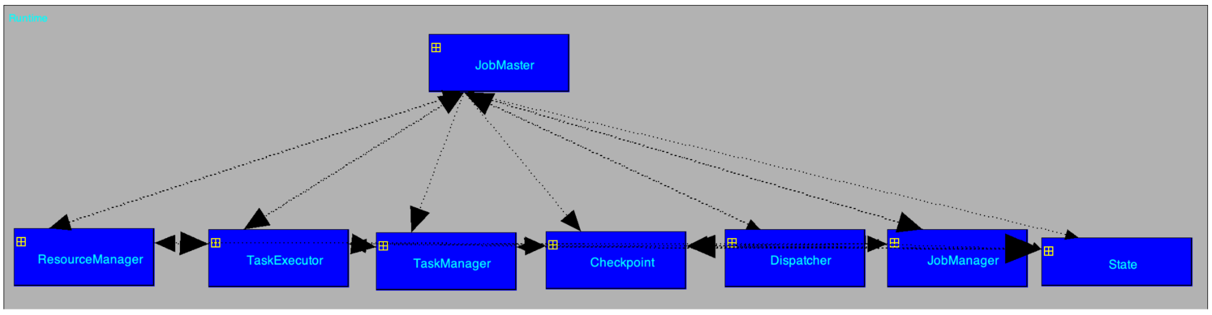

Figure 9 - Design Level Subsystem Architecture: JobMaster

The **JobMaster** subsystem bears the responsibility of managing the execution of a single **JobGraph**. While this was a part of our initial conceptual architecture diagram, we were only able to truly recognize the significance of this subsystem after observing the true nature of its dependencies on other subsystems in the runtime, with the aid of our concrete architecture diagram and analysis of the code. Interestingly enough, it shares a dependency with every other major constituent subsystem of the runtime, a pointer to its importance in the smooth functioning of the Flink system.

Initially a monolithic part of Flink's architecture, Flink 1.11 saw the decomposition of the **JobManager** to a much smaller and manageable **JobMaster**, **ResourceManager** and **Dispatcher**. For this reason, there are still a number of inconsistencies in the naming of related class names and methods in the Flink codebase.

Notably, the **JobMaster** depends on the **TaskExecutor** for frequent updates, which are referred to internally as heartbeats. It also depends on the **Checkpoint** subsystem to acknowledge and report checkpoint metrics. Its **State** dependency also enables it to register states that can be queried in the future.

Proposed Changes to Conceptual Diagram

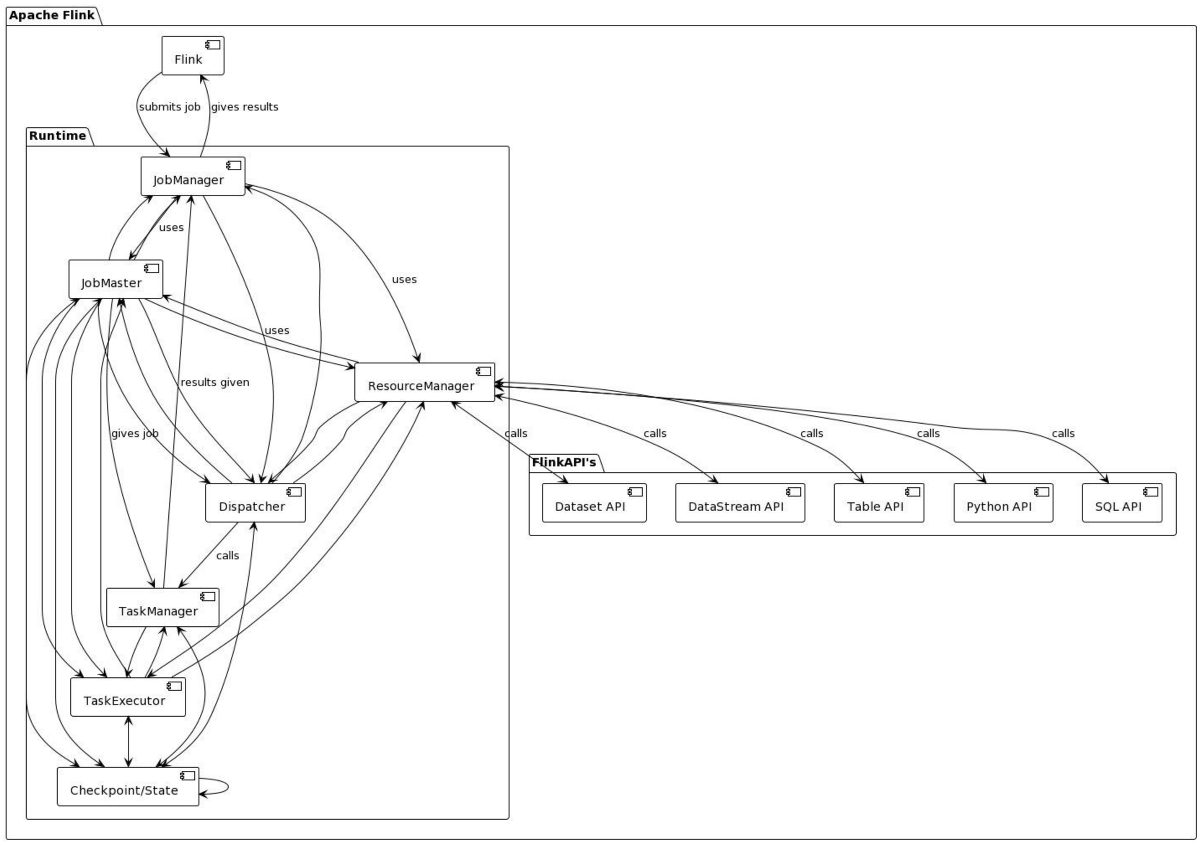

Figure 10 - Final Consolidated Top-level Architecture Diagram

To mitigate the discrepancies, we modified our Conceptual Architecture Diagram. We combined both Conceptual and Concrete Architecture Diagrams to make dependencies more obvious. Our proposed design combines the Checkpoint and State subsystems. Both the subsystems are dependent on each other and are invoked simultaneously. We took our Conceptual Architecture diagram and designed it to look more like our Concrete Architecture diagram.

Concurrency

IIt appears that our initial understanding was that the JobManager interacts directly with the ResourceManager, JobMaster, and Dispatcher to execute tasks in the TaskManager. However, upon closer examination of the concrete architecture, it seems that the JobManager doesn't communicate directly with the ResourceManager. Instead, it is still involved in utilising the ResourceManager, possibly through interactions with the Dispatcher, TaskExecutor, and JobMaster subsystems.The communication between the JobManager and the ResourceManager appears to be mediated through these intermediary components, which are all somehow connected. Therefore, we need to investigate the precise mechanisms through which the JobManager communicates with the ResourceManager. This task may pose a greater challenge for us, as we are dealing with multiple components, each playing a role in the communication flow. In contrast, other groups may have a simpler task, focusing on a single component.To gain a clearer understanding, we must delve into the specifics of how the JobManager, Dispatcher, TaskExecutor, and JobMaster subsystems are interconnected and how they collaborate to facilitate communication with the ResourceManager. This exploration will help us uncover the intricate details of the communication flow within the system.

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 utilisation, 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 centralised location for further processing or analysis.
- **Monitoring System:** The cluster manager integrates with external monitoring systems such as Apache Metric for monitoring and visualisation 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.

Conclusions

To conclude, the reflective analysis carried out for this report has revealed significant differences between the conceptual and concrete architecture diagrams created in Assignments 1 and 2. This analysis has provided a comprehensive understanding of the variables affecting the evolution of architecture. The report provides a description on our derivation process, in order to conduct this reflection analysis. By examining both the top-level and design-level subsystems of the Cluster Manager in detail, we were able to address the inconsistencies between the conceptual and concrete diagrams. The inconsistencies were also highlighted for more of a visually appealing comparison. Moreover, we were able to propose possible modifications to the conceptual diagram to bring both designs to line. In addition, proper rationale was provided on why these changes were made. This report acts as a guide for both novices and experts in the field looking to take advantage of Apache Flink's features.

The following are our proposals for future directions:

1. Future directions for architectural progress should include an ongoing, reflective process. Both the conceptual and concrete design should be reviewed and updated on a regular basis, as technology advances.
2. It is important to establish thorough documentation procedures in order to record the architectural development, the reasoning behind design choices, and the lessons discovered from past endeavours.
3. Future architectural improvements should require an emphasis on adaptability and scalability. Creating architectures with built-in flexibility guarantees that they may change to meet project requirements as they change.

The following is a summary of our key findings:

1. Got familiar with the Reflection Analysis
2. Analysed top-level subsystems and discovered the subtle discrepancies
3. Analysed design-level subsystems and discovered the detailed discrepancies
4. Researched the discrepancies that were newly discovered and importance of them in each respective subsystem
5. Created a modified version of the conceptual architecture diagram that combines diagrams from assignments 1 and 2, as well as the newly discovered dependencies
6. Report a rationale on why a modified version of the conceptual architecture was created

7. Identified differences in past sequence and use case diagrams created

Lessons Learned

Some of the lessons learned in this assignment was that Checkpoint, State and TaskExecutor were the most common subsystems with dependencies that were missed in the Conceptual Architecture Diagram. It was also Important to have a Concrete Architecture Diagram because dependencies such as the ones mentioned above were missed in the Conceptual Architecture Diagram.

Some of the limitations: The same architectural decisions may be interpreted differently by different team members, resulting in differing justifications. This subjectivity may have an impact on the completeness and correctness of the stated causes of differences.

References

1. TomTom 5 and David AndersonDavid Anderson 40.1k44 gold badges3333 silver badges6161 bronze badges, “Confused with jobmanager and JobMaster,” Stack Overflow, https://stackoverflow.com/questions/63669973/confused-with-jobmanager-and-jobmaster (accessed Nov. 10, 2023).
2. “Back to Flink website,” TaskExecutor (Flink : 1.19-SNAPSHOT API), https://nightlies.apache.org/flink/flink-docs-master/api/java/org/apache/flink/runtime/taskexecutor/TaskExecutor.html. (accessed Nov. 8, 2023).
3. “Flink architecture,” Flink Architecture | Apache Flink, https://nightlies.apache.org/flink/flink-docs-master/docs/concepts/flink-architecture/ (accessed Nov. 16, 2023).