Project Report: Inclusion Dependencies Identification using Akka Actors

Department of Computer Science and Mathematics
Philipps University of Marburg

Course: Big Data Systems

Instructor: Prof. Dr. Thorsten Papenbrock

Submitted by:
Yasmine Batteieb (MTR: 3795910)
Ugur Taysi (MTR: 3806159)
Seyyed Mostafa Rahimi (MTR: 3800582)

January 7th, 2024

1 Introduction

This project implements a distributed system to identify inclusion dependencies in datasets using the Akka actor model framework in Java. It aims to efficiently detect all inclusion dependencies (INDs) within a set of relational tables, fundamental for data integration, cleaning, and schema redesign. The system leverages the Akka Actor model to distribute tasks among multiple actors, allowing parallel computation and improving overall system scalability and efficiency.

2 System Overview

The system is architected using the Akka actor model, capitalizing on lightweight, event-driven processes. The model allows the system to handle high volumes of data by distributing tasks across multiple actors, parallelizing the workload. The core components of the system include:

2.1 Actors Overview

- **Guardian:** The main actor overseeing the entire operation, responsible for creating and managing other actors and handling system-level messages.
- **Master:** Initiates the dependency mining process by creating a DependencyMiner actor and sending it a StartMessage.
- **DependencyMiner:** Orchestrates the process, dividing the data into batches, and assigning them to Dependency Workers. It collects and aggregates the results upon completion.
- **DependencyWorker:** Performs the computation of inclusion dependencies, processing tasks assigned by DependencyMiner and returning results.
- **Reaper:** Handles system shutdown when all tasks are completed, ensuring proper termination.

2.2 Operational Flow

The operational flow involves the Guardian actor creating and starting system actors, managing the initiation of the process by the Master actor, and overseeing the entire task distribution and result collection process. DependencyMiner reads and distributes data, while DependencyWorkers compute INDs. The result is collected and finalized by the ResultCollector, followed by system shutdown through the Reaper.

2.3 System Interaction Diagram

The following diagram illustrates the interactions between the various actors in the system:

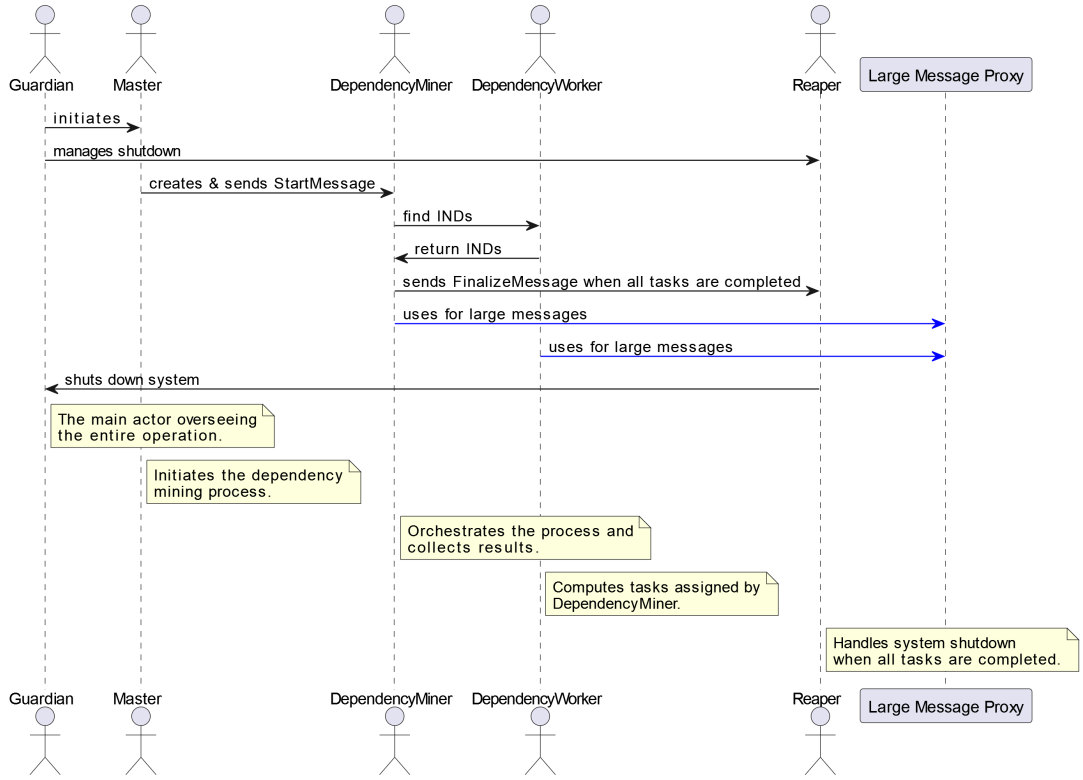

Figure 1: UML Diagram of Actor Interactions

3 Implementation Details

The project is implemented in Java, using the Akka framework and Maven build system. It's structured around the actor model, with components being actors with specific roles. Here are some key implementation details:

- **Content Class**: Represents data content, managing different data types and storage mechanisms.
- **Task Management**: Efficient task management and distribution are achieved using Akka's messaging system, which handles communication between actors.
- **Handling of Large Messages**: Implemented using a proxy pattern to manage large messages effectively, segmenting them into smaller units for transmission and reassembly.
- **Error Handling**: Robust error handling mechanisms are in place, with the Reaper actor ensuring the system shuts down correctly in case of actor failures, preventing incomplete or incorrect results.
- **Utilities**: Includes serialization classes, configuration singletons, and helper methods for tasks such as data input and result output.

3.1 Sequence Diagram for evaluateInclusionDependency

To detail the computational steps taken by the `DependencyWorker` class in evaluating inclusion dependencies, the following sequence diagram is provided. It illustrates the methodical interaction between the `DependencyWorker`, `LargeMessageProxy`, and `DependencyMiner`, delineating the flow of data and the decision-making process involved in the task.

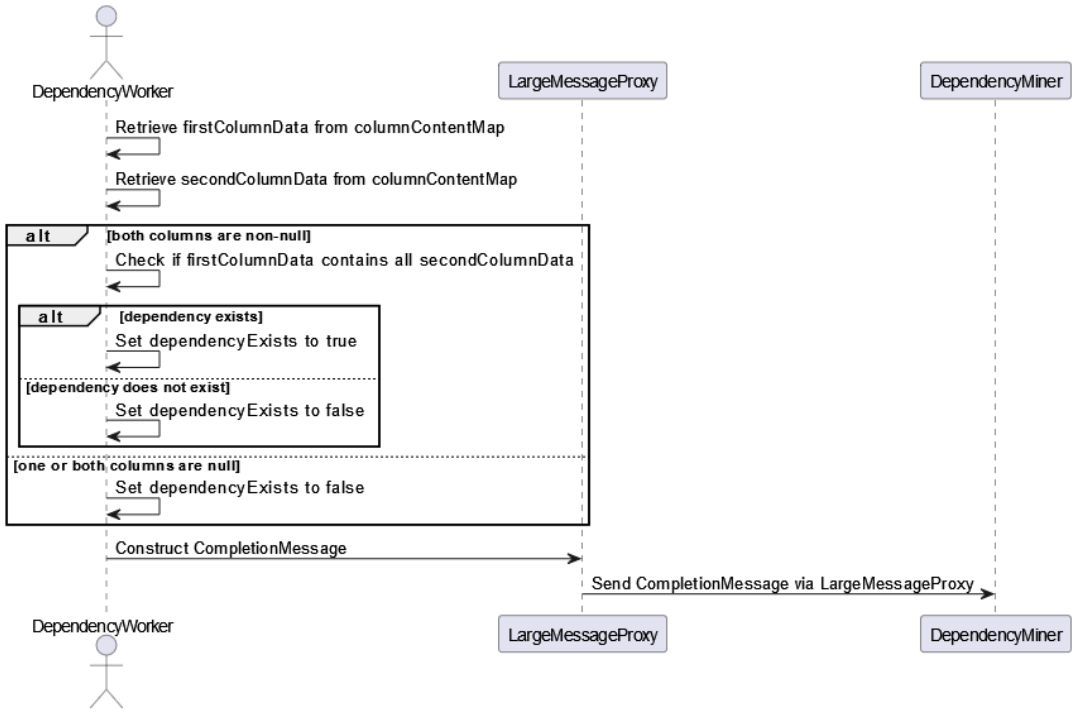

Figure 2: Sequence Diagram for the method `evaluateInclusionDependency` within the `DependencyWorker` class

4 Results and Discussion

The system has been tested with the TCP-H dataset to evaluate its performance and accuracy. It demonstrates significant improvements in speed and scalability compared to traditional approaches. The Akka actors efficiently distribute workloads and handle large datasets, reducing computation time and ensuring robustness through parallel computation and effective task management.

5 Conclusion

The project successfully demonstrates the use of the Akka actor model in a real-world application of identifying inclusion dependencies in large datasets. It highlights the potential of distributed systems in handling large-scale data processing tasks efficiently and robustly. Future work may extend this project to handle more complex dependencies, improve algorithm efficiency, and explore integration with other big data tools.