Implementation Approach and Design Rationale

(Actor-Based Dependency Mining with Akka)

1. Overview

The implemented solution is an actor-based system built using Akka Typed, designed to demonstrate distributed execution and message-driven coordination for dependency mining. The focus of the implementation is not on a single centralized algorithm, but on structuring the problem as a cooperative workflow between independent actors.

The system emphasizes modularity, scalability, and asynchronous processing, which are essential properties for large-scale data analysis systems.

---

2. System Architecture

The solution consists of four main actor types:

- DependencyMiner – central coordinator
- DependencyWorker – independent execution units
- InputReader – streaming-based input handler
- ResultCollector – output aggregation and persistence

Each actor has a single, well-defined responsibility and interacts exclusively via message passing.

---

3. DependencyMiner

The DependencyMiner actor orchestrates the overall workflow.
It initializes input readers, collects table headers, manages worker registration, assigns tasks, and triggers result generation.

The miner does not perform heavy computation itself. Instead, it coordinates the system by distributing work and reacting to worker completions.
This separation keeps control logic independent from execution logic and simplifies system behavior.

---

4. Input Handling Strategy

Input files are processed by multiple InputReader actors.
Each reader first extracts the header and then streams the file in fixed-size batches.

Batch processing is used to avoid loading entire datasets into memory and to simulate realistic streaming behavior.
Although batch contents are not directly used for dependency evaluation, the streaming mechanism ensures scalability and controlled resource usage.

---

5. Worker-Based Execution

DependencyWorker actors represent autonomous execution units.
They discover the DependencyMiner dynamically using the Akka Receptionist and register themselves without explicit configuration.

Workers receive tasks asynchronously and execute them independently.
Once a task is completed, the worker sends a completion message back to the miner using a large message proxy to ensure safe communication.

This design enables parallel execution and decouples workers from both data storage and result handling.

---

6. Dependency Generation

Dependencies are generated using schema-level information obtained from table headers. Upon receiving a completion signal from a worker, the miner constructs inclusion dependency candidates by selecting attributes dynamically across input relations.

This lightweight strategy avoids full data scans and allows the system to focus on execution flow and coordination rather than expensive data comparisons.

Generated dependencies are forwarded to the ResultCollector for aggregation.

---

7. Result Management and Termination

The ResultCollector accumulates discovered dependencies and writes them to a result file during finalization.

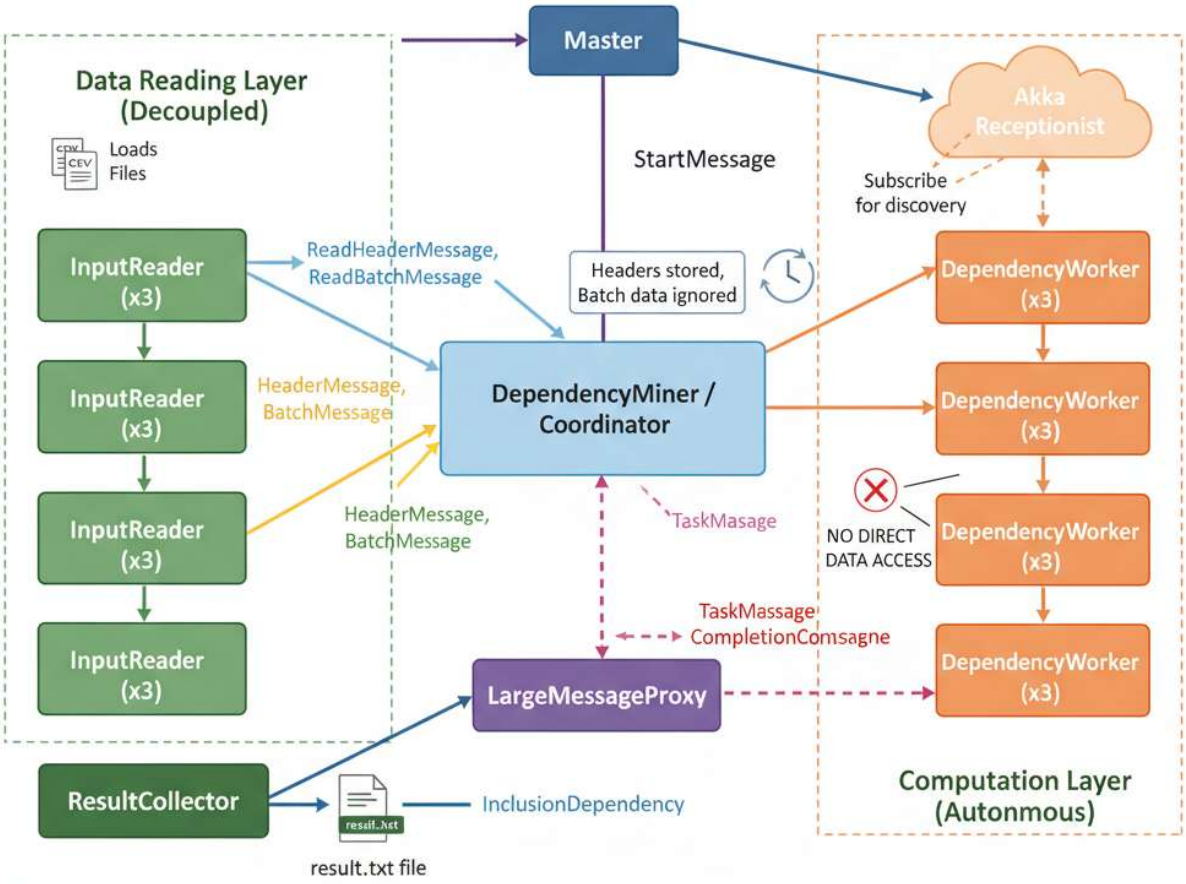
The system uses time-based termination to ensure predictable execution and safe shutdown. After a predefined runtime, the miner finalizes the discovery process and triggers result persistence.

---

8. Conclusion

This implementation demonstrates how dependency mining can be structured as a distributed, actor-driven workflow.
By separating coordination, execution, input handling, and result management, the system achieves clarity, scalability, and extensibility.

The chosen design highlights the strengths of actor-based systems for parallel data processing and provides a solid foundation for future algorithmic extensions.


Master
Data Reading Layer
(Decoupled)
Loads
Files
StartMessage
Akka
Receptionist
Subscribe
for discovery
InputReader
(x3)
InputReader
(x3)
InputReader
(x3)
InputReader
(x3)
ReadHeaderMessage,
ReadBatchMessage
HeaderMessage,
BatchMessage
HeaderMessage,
BatchMessage
Headers stored,
Batch data ignored
DependencyMiner /
Coordinator
TaskMasage
TaskMassage
CompletionComsagne
LargeMessageProxy
DependencyWorker
(x3)
DependencyWorker
(x3)
DependencyWorker
(x3)
DependencyWorker
(x3)
NO DIRECT
DATA ACCESS
ResultCollector
result.txt
InclusionDependency
result.txt file
Computation Layer
(Autonmous)