Networked Software for Distributed Systems
Dipartimento di Elettronica, Informazione e Bioingegneria
Politecnico di Milano

Project 2: Akka

Carlo Laurenti Argento
Gianluca Scanu
Federico Zanon

A.Y. 2023-2024

Contents

Introduction

Assignment

The project aims to write a simple stream processing system based on a pipeline of three operators. Each message flowing through the pipeline contains a <key, value> pair and each operator exists in multiple instances. Each operator is defined using a window (size, slide) and an aggregation function to process values in the current window. When the window is complete, the operator computes the aggregate value and forwards the result to the next operator. The key space is partitioned across operator instances, and they identify a different data type to be processed. The pipeline must be fault-tolerant and ensure that each <key, value> pair is processed at least once.

Technologies

The primary technology involved is Akka. The project also uses Akka facility, such as the clustering/membership service.

Implementation

Design

The application is built using the Akka actor model: actors communicate exchanging messages and can be distributed on different hosts over the network using the **Akka clustering/membership service**. As shown in Figure 2, a process reads data from an input file and sends them to the pipeline. Each operator of the pipeline has a router that receives data from the previous stage and forwards them to the correct instance based on the key. At the end the third operator instances forward the results to a sink operator that prints them.

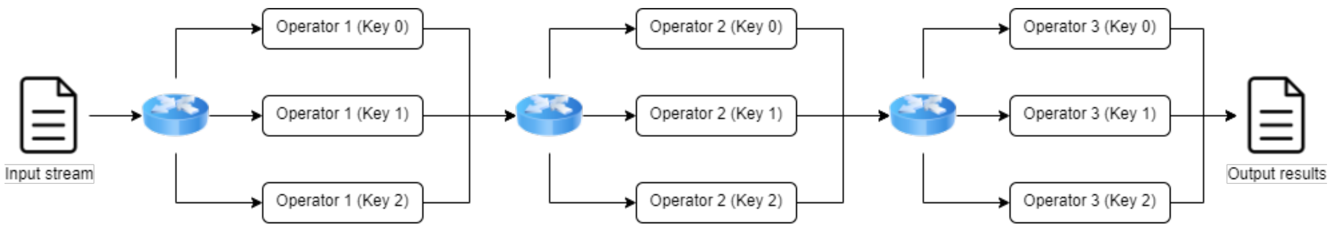


Figure 2: Pipeline design

Configuration

For simplicity we have decided to create a specific configuration of this pipeline:

- **First operator:**
  - Window size: 5
  - Window slide: 3
  - Aggregation function: average (rounded down)
- **Second operator:**
  - Window size: 4
  - Window slide: 2
  - Aggregation function: maximum
- **Third operator:**
  - Window size: 2
  - Window slide: 1
  - Aggregation function: sum

Assumptions

- Since we must show only failures in the pipeline operators, we assume that routers (which act as operator supervisors) cannot fail.
- For simplicity we assume that there is an operator instance per key (data type).
- For simplicity the pipeline manages only integers, all the results are rounded down.
- For simplicity we assume that the data in the input file always respects a specific format, designed to recognize crash messages on the operators.

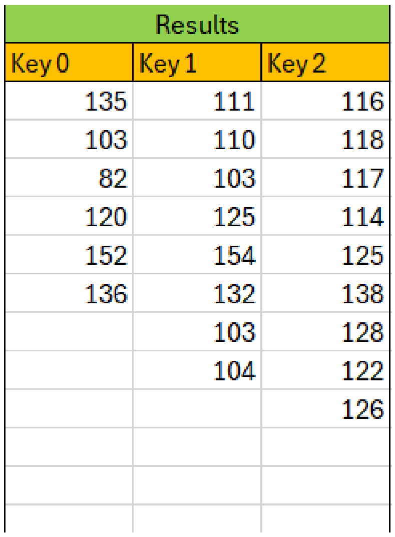
Actors

- **Router:** this actor is instantiated by `OperatorMain` class, and it represents the router for all the instances of the same operator. For this reason, three instances of this actor are instantiated by three different `OperatorMain` processes that represent a specific operator of the pipeline. When a router is created it instantiates all the operator instances (the number of instances is defined by the number of data type of the main process). When it receives a `DataMessage` it forwards it to a specific instance based on the key. When it receives a `CrashMessage` it forwards it to a random instance to make it crash. `Router` acts also as supervisor of all the operator instance it instantiates, and, for this reason, it defines the strategy to resume an operator instance when it throws a `ResumeException`.
- **Operator:** this actor is instantiated by the corresponding router (supervisor) and represents the operator instances. Each operator stores a set of values into its window and the address of the next router. When an operator receives a `DataMessage` it updates the values stored in the window. Then, if the window is completed, it computes the aggregated value and sends the result to the next router. At the end it makes the window slide. When it receives a `CrashMessage` it throws a `ResumeException` to notify the supervisor (router) to resume it back. There are three different version of this actor that vary based on the configuration of the pipeline operator (window parameters and aggregation function).
- **Sink:** this actor is instantiated by its supervisor (through an ask pattern) and represents the end of the pipeline which collects all the results. When it receives a `DataMessage`, it prints the result on the command line; when it receives a `CrashMessage` it throws a `ResumeException` to notify the supervisor to resume it back.
- **Sink Supervisor:** this actor acts as a supervisor of the `Sink` actor, and it is instantiated by the main process. It defines the strategy to resume the sink when it throws a `ResumeException`.

Testing

Since one of the requirements in Akka is knowing the address of the actors, to test the application it's necessary to change the IP address of the machines which will run the system. In particular all the `.conf` files and the references of the actors retrieved with `actorSelection()` must be changed with the correct IP of the hosting machine (in particular the second requirement can be easily done by changing the addresses in the `ConfigUtils` class). Then each part of the pipeline (operators and stream process) can be executed on different machines. It's important to run first the three `OperatorMain` classes to correctly set up the cluster and only after the `StreamProcessingSystem` can be executed. Note that a single machine can also run more parts of the pipeline on its own.

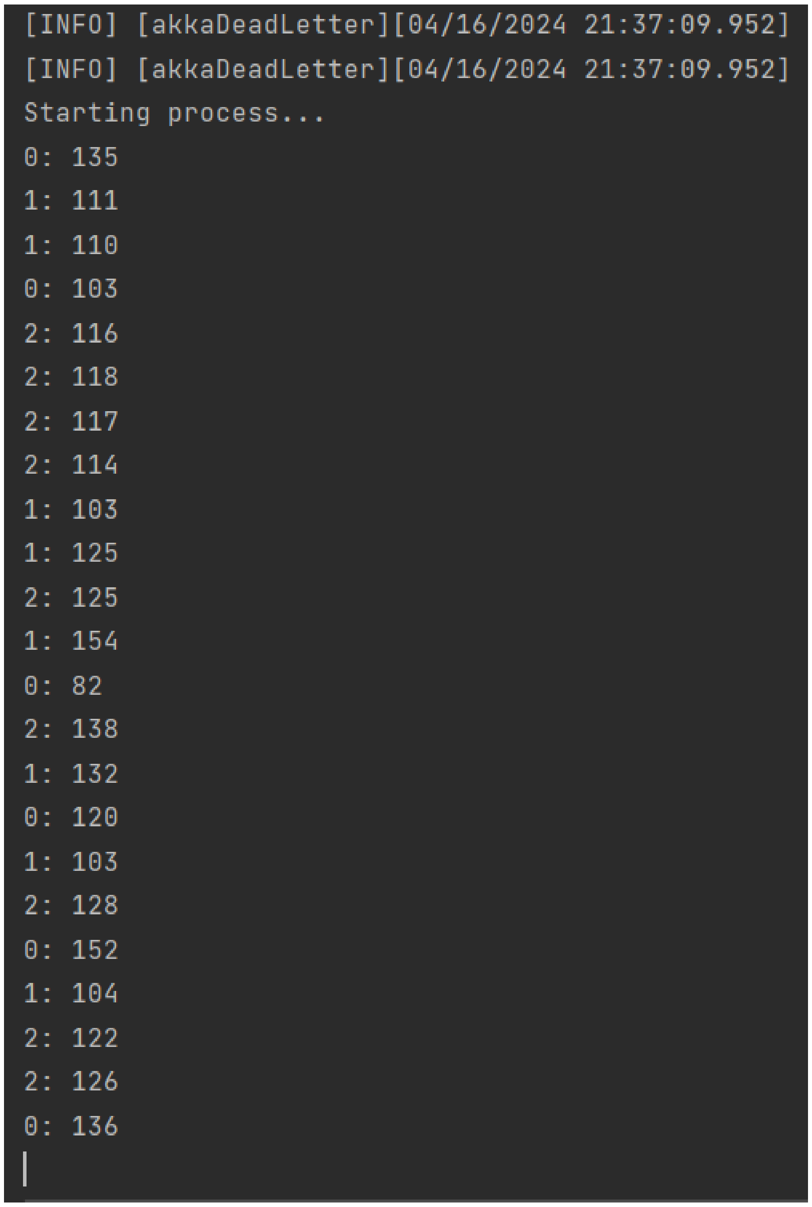
To test the correctness of the pipeline we used a simple example, and we expect the following results (computed with a spreadsheet):



Results		
Key 0	Key 1	Key 2
135	111	116
103	110	118
82	103	117
120	125	114
152	154	125
136	132	138
	103	128
	104	122
		126

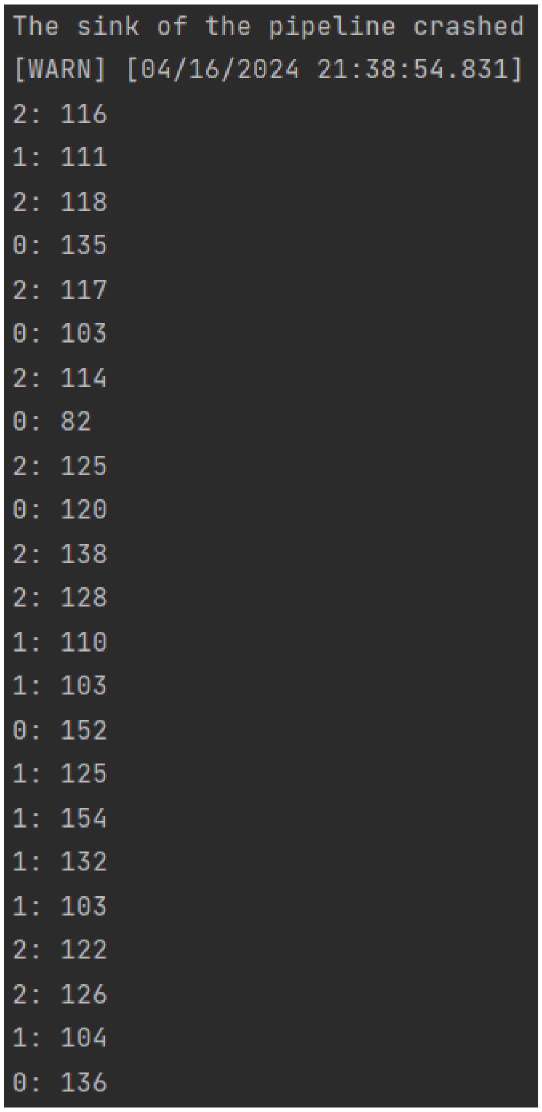
Figure 3: Expected results on the spreadsheet

To test the fault-tolerance we can run the same example but with crashes message at each stage of the pipeline and we expect the same results. Here the results generated with the two input files `noCrashes.txt` and `withCrashes.txt`.



```
[INFO] [akkaDeadLetter][04/16/2024 21:37:09.952]
[INFO] [akkaDeadLetter][04/16/2024 21:37:09.952]
Starting process...
0: 135
1: 111
1: 110
0: 103
2: 116
2: 118
2: 117
2: 114
1: 103
1: 125
2: 125
1: 154
0: 82
2: 138
1: 132
0: 120
1: 103
2: 128
0: 152
1: 104
2: 122
2: 126
0: 136
```

Figure 4: Results using `noCrashes.txt`



```
The sink of the pipeline crashed
[WARN] [04/16/2024 21:38:54.831]
2: 116
1: 111
2: 118
0: 135
2: 117
0: 103
2: 114
0: 82
2: 125
0: 120
2: 138
2: 128
1: 110
1: 103
0: 152
1: 125
1: 154
1: 132
1: 103
2: 122
2: 126
1: 104
0: 136
```

Figure 5: Results using `withCrashes.txt`