Team 1

Window evaluation strategies in Flink using the RocksDB state backend

Nirbhay Malhotra
Aman Ahmed
Rhythm Somaiya
Bhargav Ram Reddy Pattiputtur
Yuesi Liu

Project Overview

In this project, We intend to design and implement two alternative window operators in Flink using the Record buffer and Slicing strategies. In our approach, we have opted to include one commutative operator and one holistic operator. Our objective is to design efficient operators so that they provide low latency and high throughput. Here are some necessary background.

- The record buffer strategy buffers incoming records until a specific condition is met, such as the arrival of a certain number of records or the passage of a certain amount of time.
- The slicing strategy slices the stream into consecutive time intervals of equal length, or slices, and applies the window operation on each slice.

- A commutative operator (sum) is an operator where the order of the operands does not affect the final result. In other words, the operation can be applied in any order and the result will be the same length, or slices, and applies the window operation on each slice.
- A holistic operator (mean) is an operator in which the order of the incoming events does affect the final result of the operation.

- RocksDB is an embedded key-value database engine that is designed for h performance, low-latency data storage and retrieval operation.

Project Overview

The motivation behind this project is to explore and develop alternative window operators in Apache Flink in order to improve its performance and scalability for handling large volumes of streaming data.

- Design and implement two alternative window operators in Flink using the record buffer and slicing strategies
- Optimize the representation of the state of these operators as key-value pairs in RocksDB
- Evaluate the performance and scalability of the new operators compared to Flink's default implementation, with respect to window length, ratio of length to slide, and using different window evaluation functions
- Analyze the results of the evaluation and identify any trade-offs or limitations of the new operators

By the end of the semester, we hope to achieve these goals and have a working implementation of the new window operators in Flink, along with a thorough evaluation of their performance and scalability.

Status update

- Implemented
- Work-in-progress
- Future work

1. Construction of a base streaming framework with architecture permeable to the project's vision (initial setup in Flink).
2. Implementation of Record Buffer for Holistic and Commutative.
3. Implementation of Slicing Operator for Holistic and Commutative.
4. Implementation of the Functions for Commutative and Holistic.
5. Integrate RocksDB backend as a way to store states (which involves connectors).
6. Develop the watermarks and sliding/tumbling window metadata.
7. Develop the Test cases from the datasets and use them.
8. Exploiting RocksDB's Features to increase the performance of the application
9. Brief comparison of our model against Flink's default implementation mentioned in the paper.
10. Analysis and Optimization. (Current Ideas: Extension to Out of order windows and exploiting state compression feature of RocksDB)
11. Shuffle Load Balancing. Parallelism distribution between operators, snapshot state.
12. Slicing and Window Manager needs to be constructed for the operators.

Demonstration of Working Components

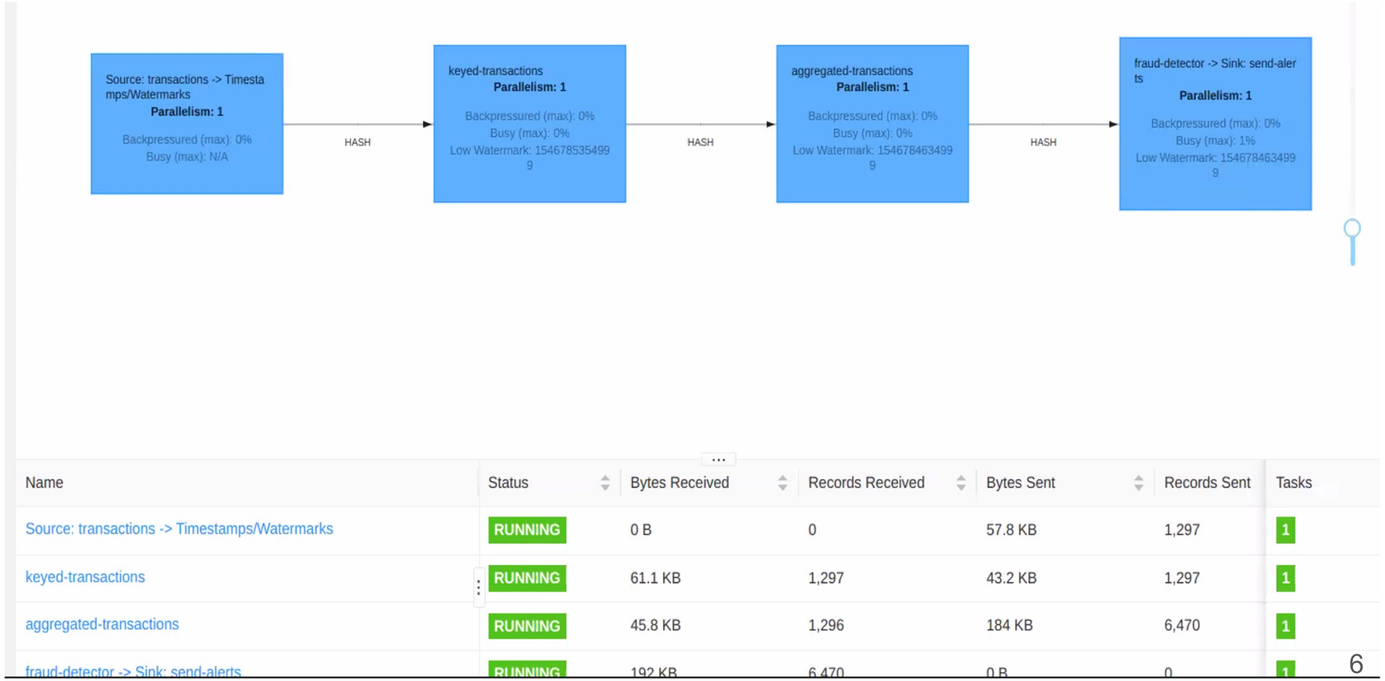
Preliminary Analysis of the Dataset:

- Our group uses Apache Flink’s Fraud Detection project as a foundation to work on our project.
- Apache Flink's Fraud Detection is a real-time streaming data processing application that detects fraudulent transactions in financial transactions in real-time.
- Although we don’t have access to the data emitted from the TransactionSource() in Apache Flink. However, based on our own experience with transactions, Each Transaction event is a custom class with attributes such as transaction id, account id, transaction amount, and timestamp.
- It should look something like this:

```
Transaction {
  transactionId: "12345",
  accountId: "56789",
  amount: 1000.0,
  timestamp: 1647901000000
}
```

For Sliding Windows:

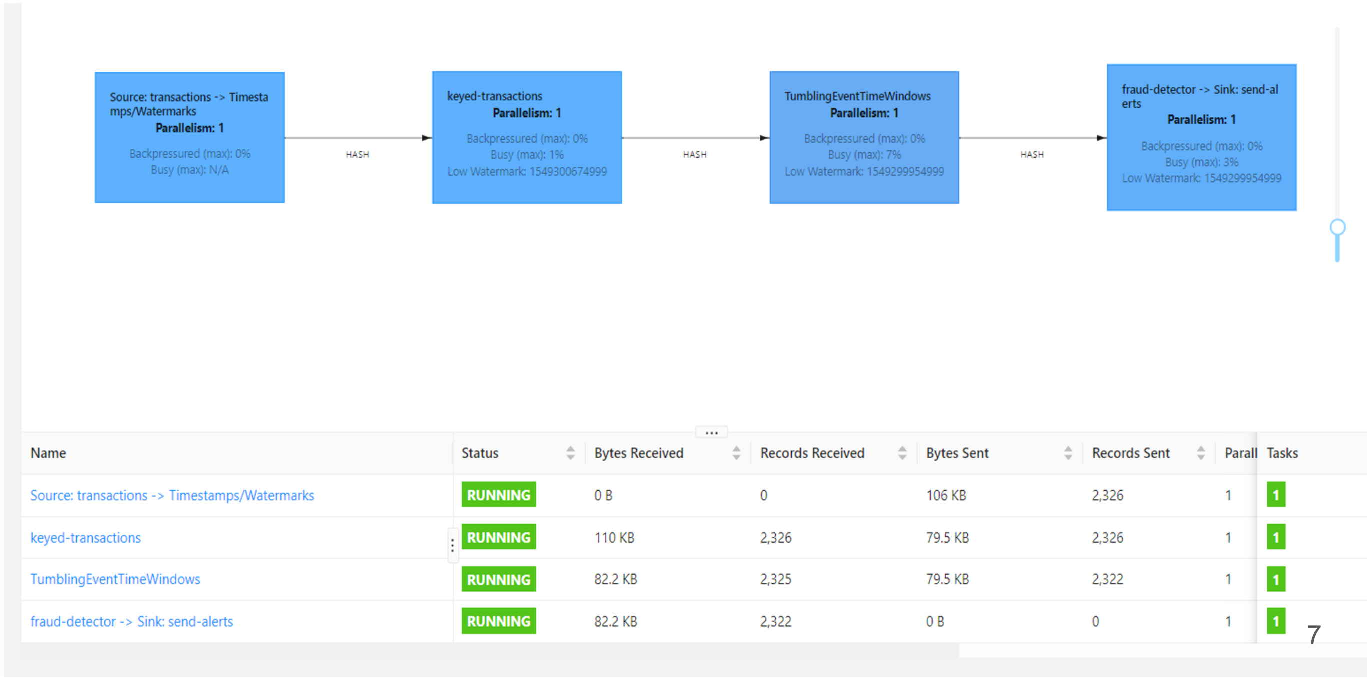
The program sets up the Flink execution environment, uses a sliding window and custom AggregateFunction to aggregate transactions, and applies a FraudDetector KeyedProcessFunction to detect fraud where the stream is out of order coming from the source. The program outputs alerts using an AlertSink and collects metrics such as the number of events processed and first event latency using Prometheus libraries.



Name	Status	Bytes Received	Records Received	Bytes Sent	Records Sent	Tasks
Source: transactions -> Timestamps/Watermarks	RUNNING	0 B	0	57.8 KB	1,297	1
keyed-transactions	RUNNING	61.1 KB	1,297	43.2 KB	1,297	1
aggregated-transactions	RUNNING	45.8 KB	1,296	184 KB	6,470	1
fraud-detector -> Sink: send-alerts	RUNNING	192 KB	6,470	0 B	0	1

For Tumbling Windows:

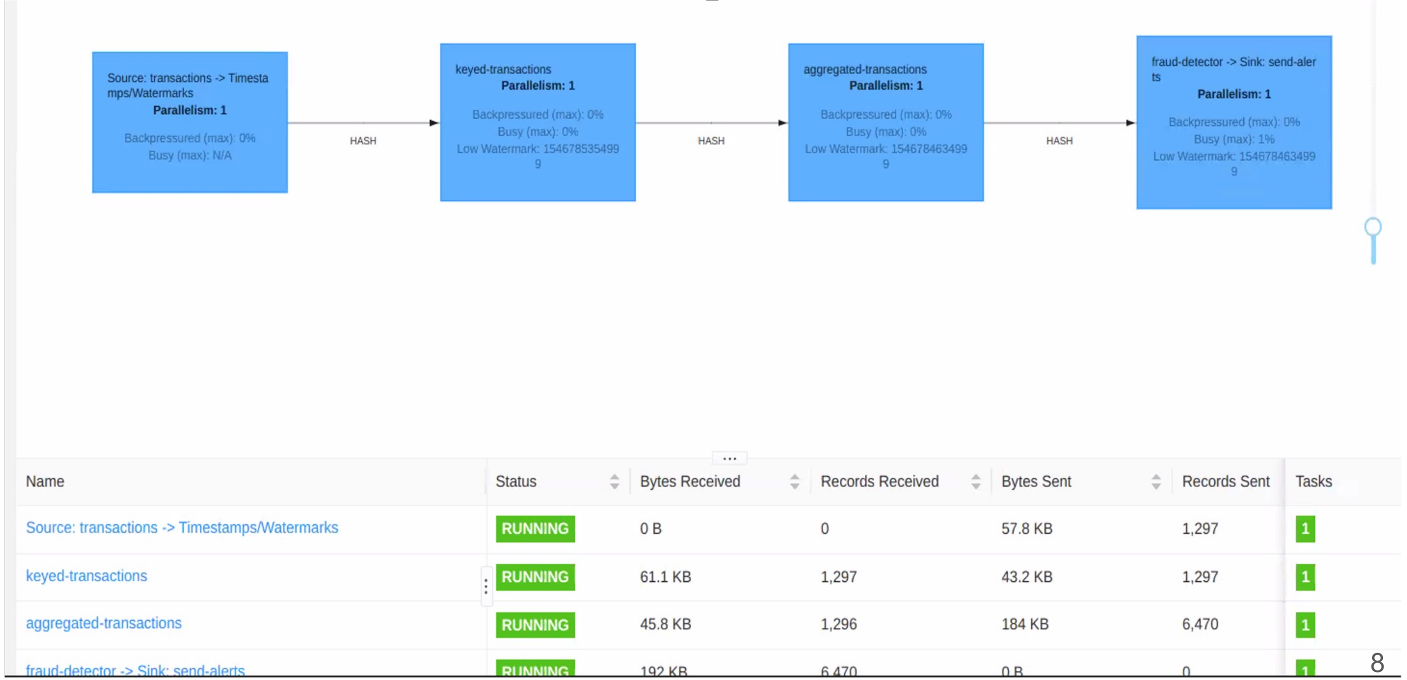
The transactions are keyed by the account ID, and a keyed process function is used to assign manual timestamps based on the account ID. The transactions are then aggregated using a tumbling window of 5 minutes with a sum aggregation function, and a FraudDetector KeyedProcessFunction is applied to detect fraud and emit alerts for suspicious transactions where the stream is out of order coming from the source. Finally, the alerts are sent to an AlertSink for output.



Name	Status	Bytes Received	Records Received	Bytes Sent	Records Sent	Parall	Tasks
Source: transactions -> Timestamps/Watermarks	RUNNING	0 B	0	106 KB	2,326	1	1
keyed-transactions	RUNNING	110 KB	2,326	79.5 KB	2,326	1	1
TumblingEventTimeWindows	RUNNING	82.2 KB	2,325	79.5 KB	2,322	1	1
fraud-detector -> Sink: send-alerts	RUNNING	82.2 KB	2,322	0 B	0	1	1

Inorder Sliding Windows

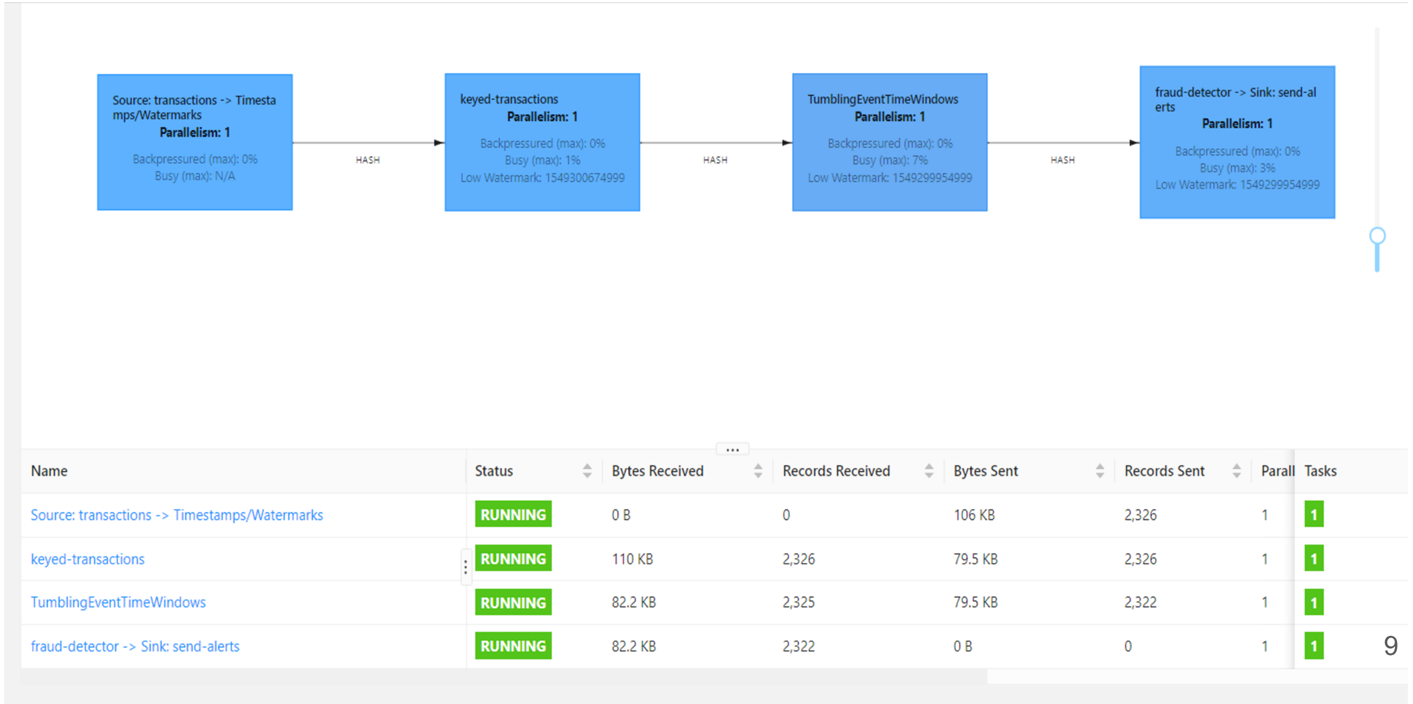
The system aggregates and detects fraud in a stream of transactions and sends alerts to an alert sink. The program utilizes Flink's RocksDB state backend for state management and checkpointing where we have in order stream of data coming from source. It assigns manual timestamps based on account ID and uses a sliding window of 5 minutes with a slide of 1 minute to aggregate transactions per account using an incremental mean aggregate function. Finally, the program executes the Flink job and computes performance metrics and includes Prometheus metrics for monitoring and visualization.



Name	Status	Bytes Received	Records Received	Bytes Sent	Records Sent	Tasks
Source: transactions -> Timestamps/Watermarks	RUNNING	0 B	0	57.8 KB	1,297	1
keyed-transactions	RUNNING	61.1 KB	1,297	43.2 KB	1,297	1
aggregated-transactions	RUNNING	45.8 KB	1,296	184 KB	6,470	1
fraud-detector -> Sink: send-alerts	RUNNING	192 KB	6,470	0 B	0	1

Inorder Tumbling Windows

The system assigns manual timestamps based on the account ID, aggregates transactions per account using a 5-minute tumbling window, applies fraud detection logic on the aggregated transactions, and sends alerts to an alert sink where the stream of data is in order coming out from the source. The program uses Flink's RocksDB state backend for checkpointing and state management, includes Prometheus metrics for monitoring and visualization of performance metrics, and computes some performance metrics such as the number of checkpoints, number of events processed, and latency and throughput of the system.



Name	Status	Bytes Received	Records Received	Bytes Sent	Records Sent	Parall	Tasks
Source: transactions -> Timestamps/Watermarks	RUNNING	0 B	0	106 KB	2,326	1	1
keyed-transactions	RUNNING	110 KB	2,326	79.5 KB	2,326	1	1
TumblingEventTimeWindows	RUNNING	82.2 KB	2,325	79.5 KB	2,322	1	1
fraud-detector -> Sink: send-alerts	RUNNING	82.2 KB	2,322	0 B	0	1	1

Challenges

- Setting up RocksDB as the state backend was a technical challenge that required careful configuration and tuning. Especially for M1/M2 Macs, RocksDB couldn't load its native library into the Apache Flink Cluster, so using a VM was the only solution.
- Creating a custom assigner function capable of mapping incoming records to panes was also a challenge in implementing sliding and tumbling window operators with the slicing approach.
- Incorporating Grafana and Prometheus with Apache Flink to monitor the performance of our project was another challenge. By utilizing Prometheus with Grafana, we were able to collect metrics such as latency, throughput, in and out-of-order tuples, etc.
- Challenge in converting a normal aggregation to incremental aggregation for reducing latency and throughput.
- Less documentation for setting up parallelism for operators and jobs, also for enabling state backend with parallelism

Immediate Future Work

- Shuffle Load meeting and enabling parallelism distribution between operators and snapshot state.
  Estimated effort: 2 weeks
- Optimize the record buffer strategy for the two window operators. This could involve exploring different buffer sizes and designing more efficient data structures for storing the buffer.
  Estimated effort: 2 weeks
- Optimize the slicing strategy for the two window operators. This could involve optimizing the slicing function, exploring different slice sizes, and designing efficient data structures for storing the slice state.
  Estimated effort: 2 weeks
- Implement the optimized window operators and integrate them into the Flink framework. This step will involve testing and validating the performance improvements achieved by the optimization.
  Estimated effort: 2 weeks
- Document the optimization process and prepare a report outlining the performance improvements achieved. The report should also provide insights into the design decisions made during the optimization process.
  Estimated effort: 1 week