Team 1

Window evaluation strategies in Flink using the RocksDB state backend

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

RocksDB

Project Goals

Project Objective

- 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.

Project Motivation

- 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, the 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.

Main Achievements

01

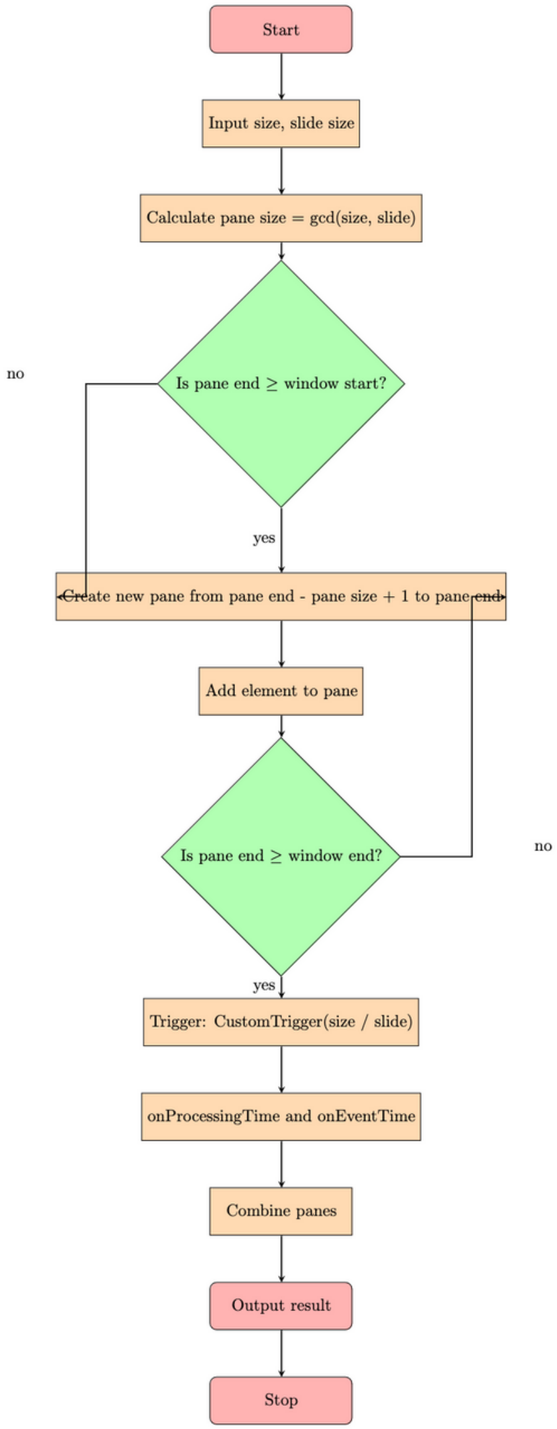
The Tumbling Window we designed with the respective trigger is much more efficient than the default one provided by the Flink as our implementation has *lower latency and higher throughput*.

02

The Sliding Window we implemented has *lower latency than the one provided by Flink*.

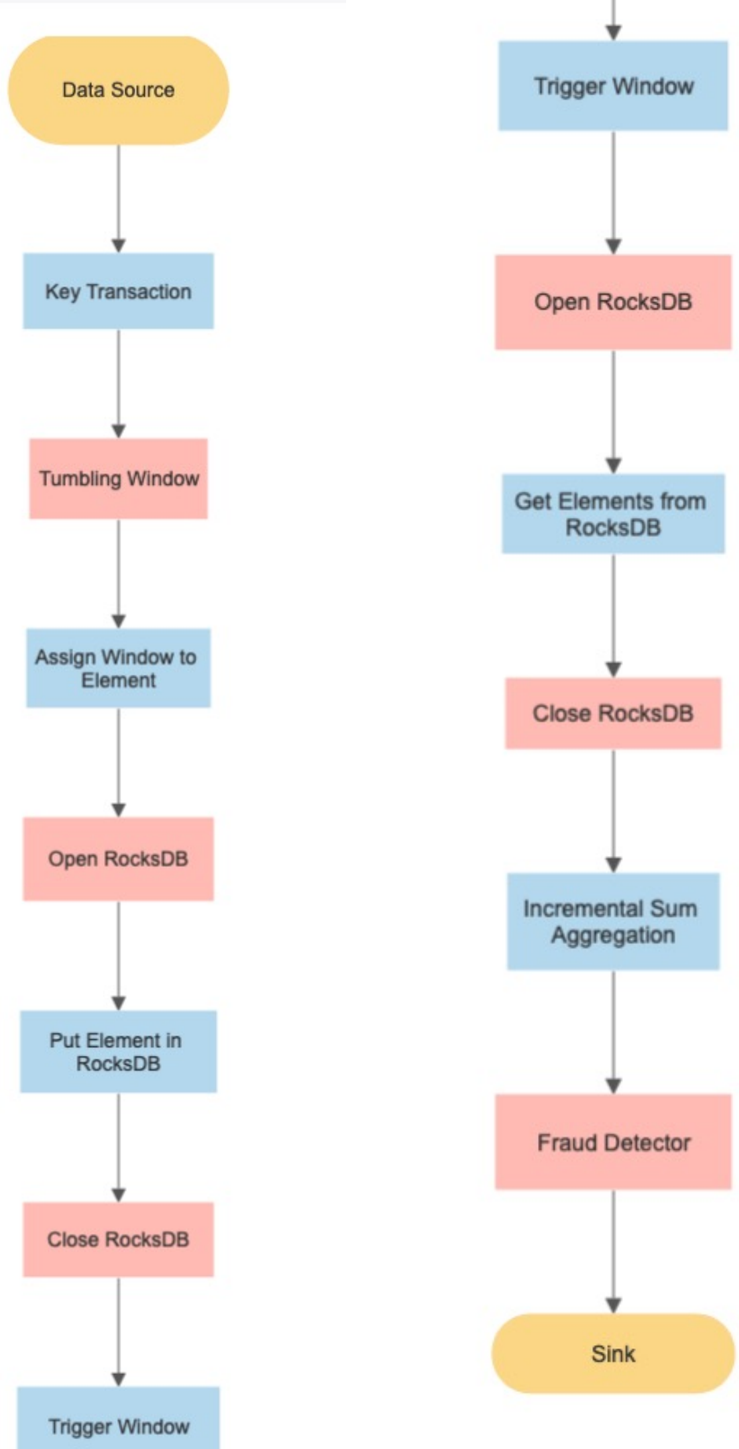
03

We were able to *incorporate parallelism* with our Tumbling Window and Sliding Window approaches to see the difference in performance.



Sliding Window Approach

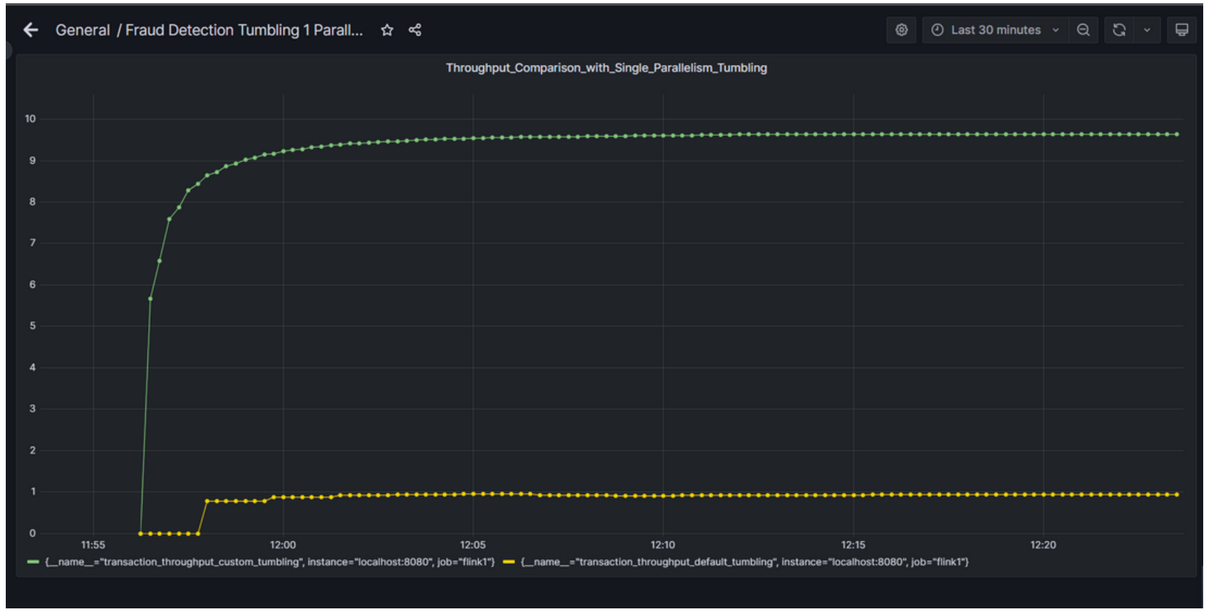
Tumbling Window Approach



Experimental Results

Flink's Tumbling Window approach vs Custom Tumbling Window approach

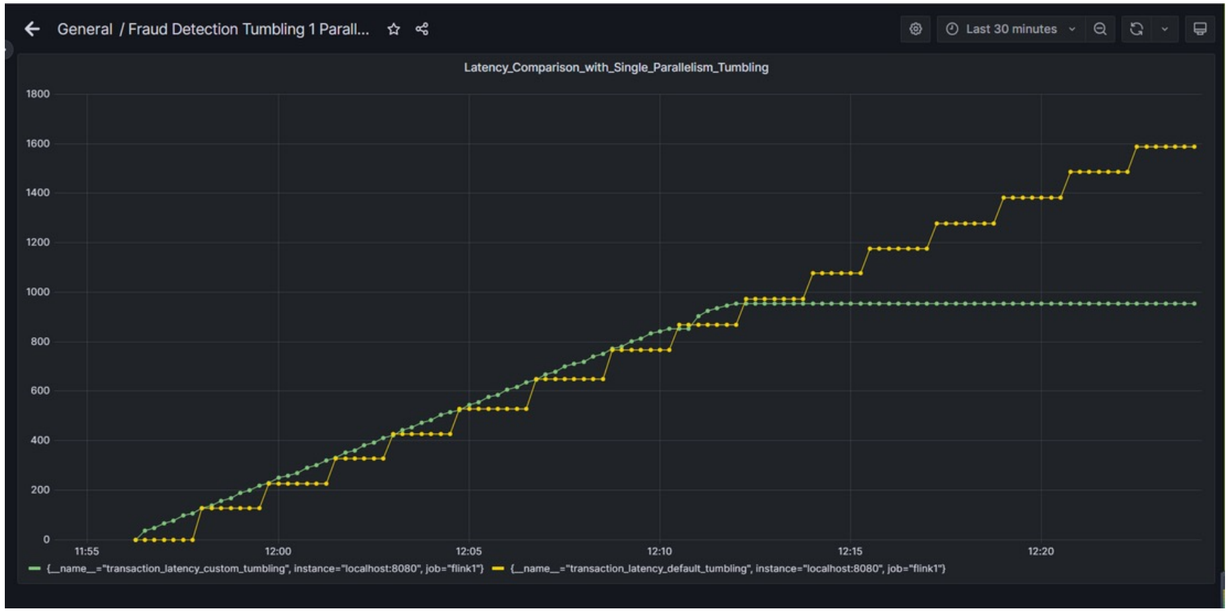
Throughput



Our implementation of the Tumbling Window with incremental sum aggregation has better throughput than Flink's default Tumbling Window implementation with incremental sum aggregation.

Flink's Tumbling Window approach vs Custom Tumbling Window approach

Latency

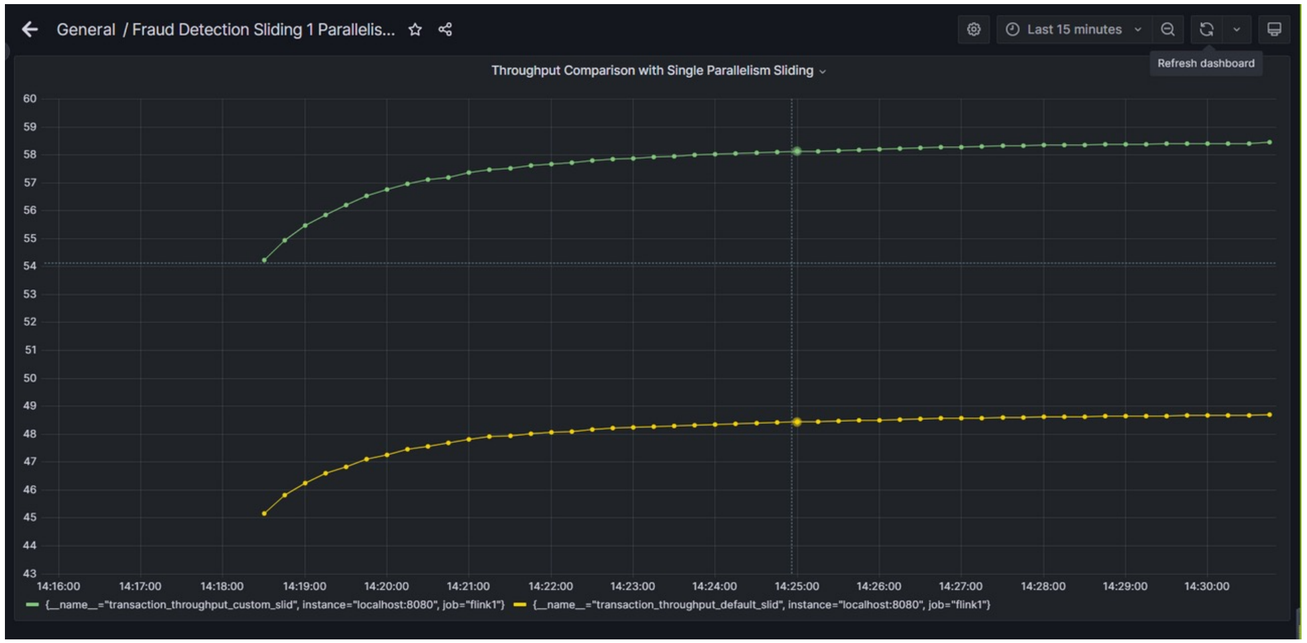


Our implementation of the Tumbling Window with incremental sum aggregation has lower latency than Flink's default Tumbling Window implementation with incremental sum aggregation.

Experimental Results

Flink's Sliding Window approach vs Custom Sliding Window approach

Throughput



Our Implementation of the Sliding Window with incremental mean aggregation and slicing strategy has better throughput than Flink's default Sliding window implementation with incremental mean aggregation and slicing strategy.

Flink's Sliding Window approach vs Custom Sliding Window approach

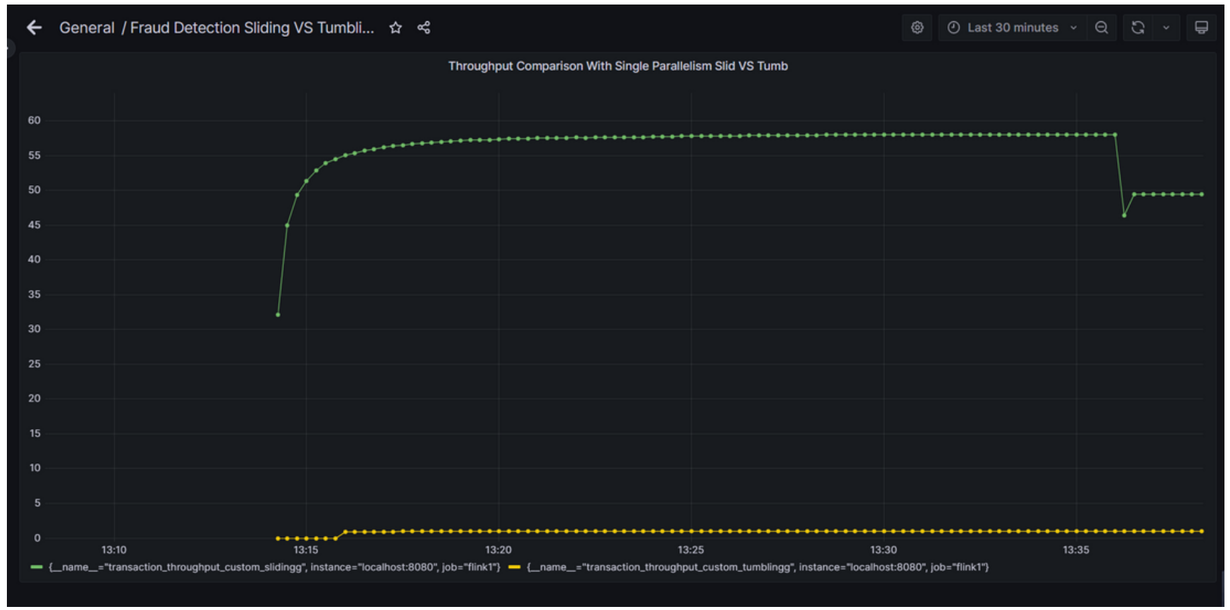
Latency

Our Implementation of the Sliding Window with incremental mean aggregation and slicing strategy has lower latency than Flink's default Sliding Window implementation with incremental mean aggregation and slicing strategy.

Experimental Results

Custom Sliding Window approach with incremental mean aggregation and slicing strategy vs Custom Tumbling Window approach with incremental Sum Aggregation and record Buffer Strategy

Throughput



We can observe that the Sliding Window approach with incremental mean aggregation and slicing strategy has higher throughput than the Tumbling Window approach with incremental Sum Aggregation and record buffer strategy.

Custom Sliding Window approach with incremental mean aggregation and slicing strategy vs Custom Tumbling Window approach with incremental Sum Aggregation and record Buffer Strategy

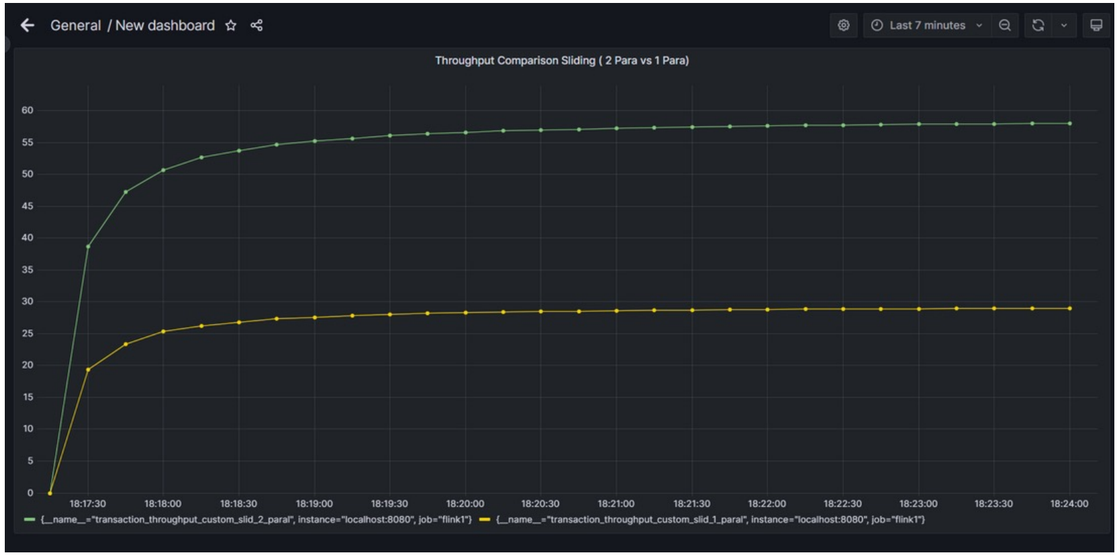
Latency

We can observe that the Sliding Window approach with incremental mean aggregation and slicing strategy has lower latency than the Tumbling Window approach with incremental Sum Aggregation and record buffer strategy.

Experimental Results

Custom Sliding Window approach with parallelism - 1 vs
Custom Sliding Window approach with parallelism - 2

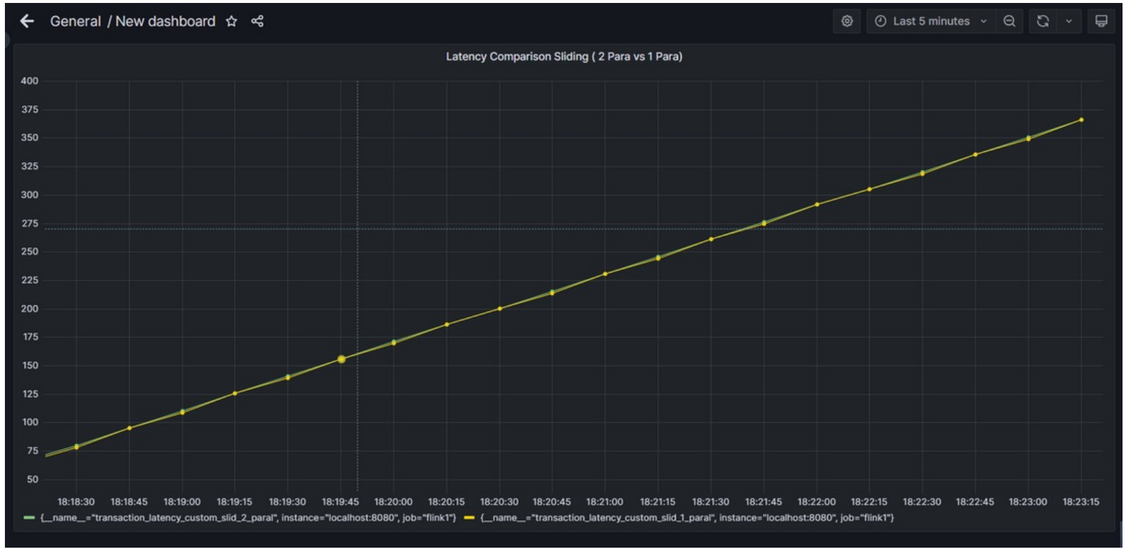
Throughput



We can observe that the Sliding window with parallelism 2 with slicing strategy and incremental mean aggregation has higher throughput than the Sliding window with parallelism 1 with slicing strategy and incremental mean aggregation.

Custom Sliding Window approach with parallelism - 1 vs
Custom Sliding Window approach with parallelism - 2

Latency

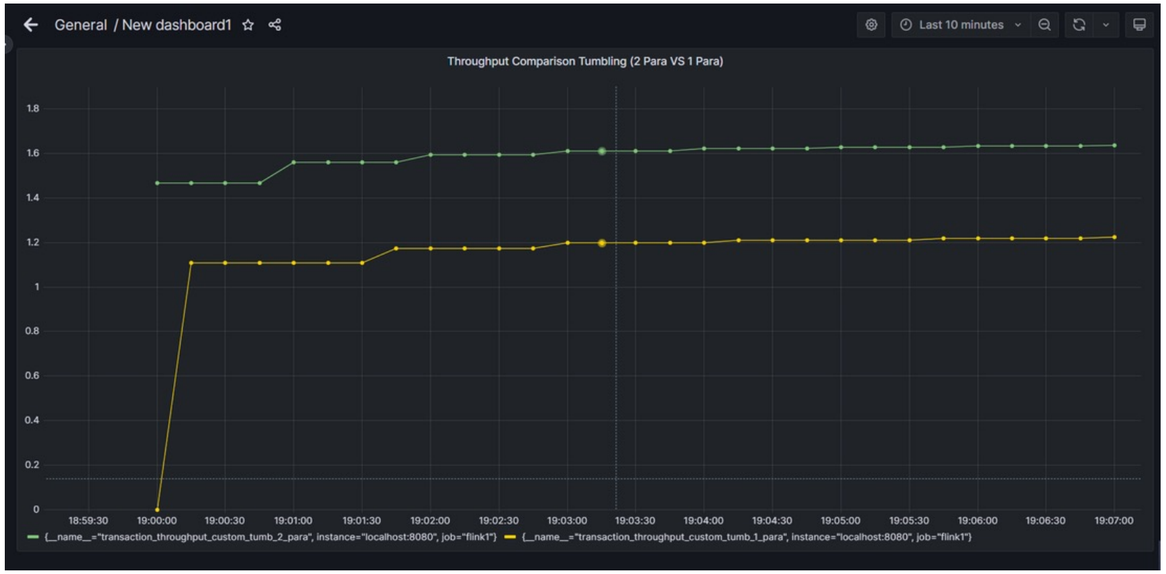


We can observe that the Sliding window with parallelism 2 with slicing strategy and incremental mean aggregation has around the same latency as the Sliding window with parallelism 1 with slicing strategy and incremental mean aggregation.

Experimental Results

Custom Tumbling Window approach with parallelism - 1 vs
Custom Tumbling Window approach with parallelism - 2

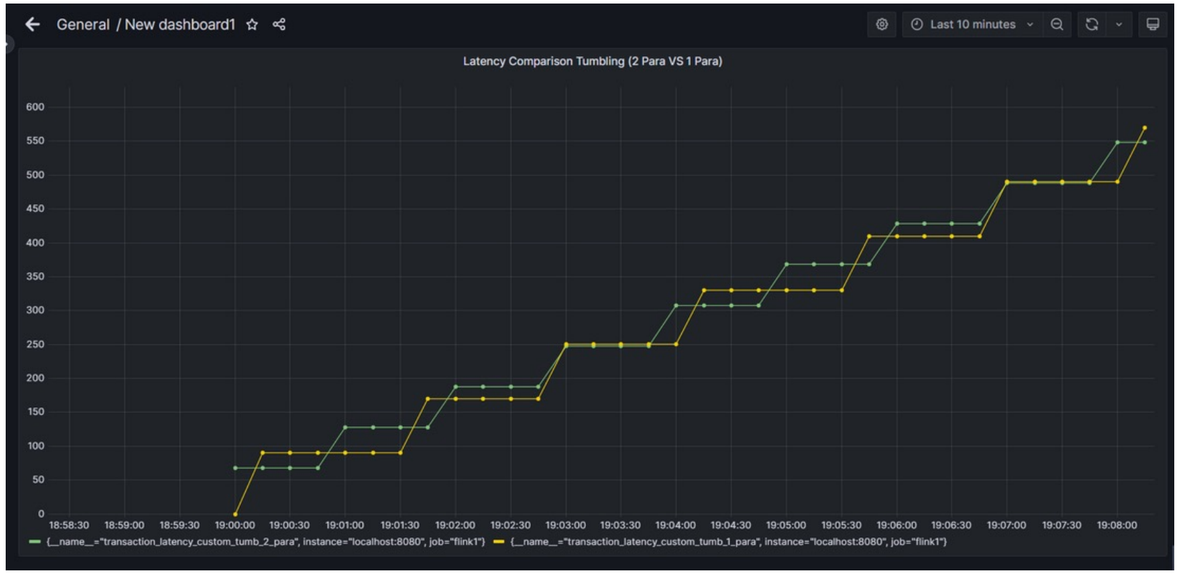
Throughput



We can observe that the Tumbling window with parallelism 2 with record buffer strategy and incremental sum aggregation has higher throughput than the Tumbling window with parallelism 1 with record buffer strategy and incremental sum aggregation.

Custom Tumbling Window approach with parallelism - 1 vs
Custom Tumbling Window approach with parallelism - 2

Latency



We can observe that the Tumbling window with parallelism 2 with record buffer strategy and incremental sum aggregation has lower latency than the Tumbling window with parallelism 1 with record buffer strategy and incremental sum aggregation.

Experience and Challenges

Experience

- Built custom window assigner classes extending Flink's capabilities. This allowed us to implement the Sliding, Tumbling, and other window strategies.
- It was interesting to integrate Tumbling Windows with RocksDB as the record buffer for performing incremental sum aggregation following the record buffer strategy.
- It was fascinating to implement different levels of parallelism for the Tumbling Window approach by incorporating different RocksDB Instances to be accessed by the processes.
- Engaging to implement Sliding Windows with slicing strategy with different levels of parallelism.

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.
- It was difficult to include combined panes for getting the incremental aggregation as a result of the aggregated transaction.
- Incorporating Grafana and Prometheus with Apache Flink to monitor the performance of our project was another challenge. Using Fraud Detector and involving latency and throughput for the pipeline was difficult.
- Managing RocksDB instances to work with higher parallelism for the Tumbling window with incremental sum aggregation.
- Less documentation for setting up parallelism for operators, jobs, and also for enabling state backend with parallelism.

THANK YOU!