TEAM 6

Apache Flink Cloud Bursting Technique

Anwesha Saha, Sakshi Sharma, Sanath Bhimsen, Showndarya Madhavan, Ye Tian, Yujie Yan

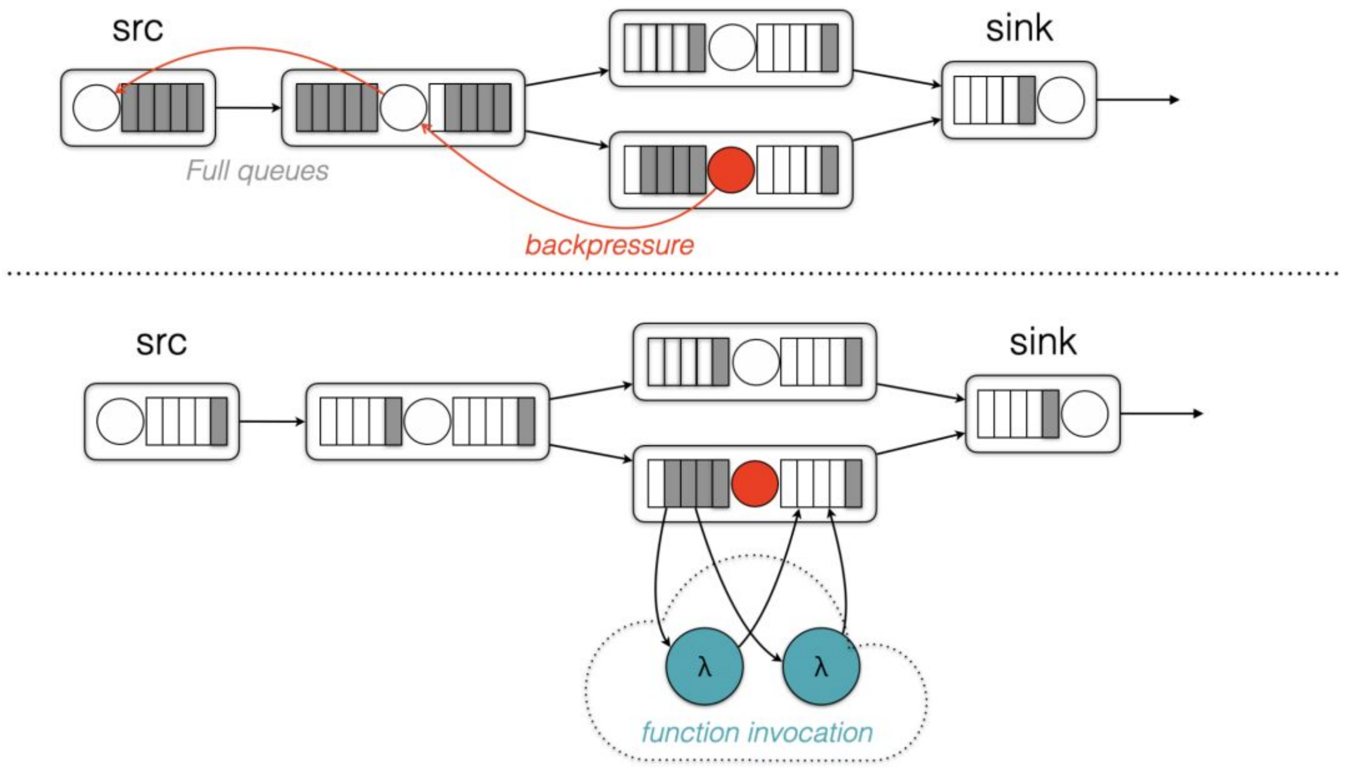
MOTIVATION

The motivation for the project is to address the long-standing research challenge of maintaining quality of service (QoS) in stream processing systems when input rates exceed system capacity.

Existing systems like Flink use flow control mechanisms such as back-pressure, but this can result in performance degradation and the need for a persistent input message queue and adequate storage space. The project aims to investigate an alternative approach using aws Lambda for offloading.

GOAL

The goal of this project is to design and implement an adaptive Flink application that leverages the “cloud bursting” technique as an alternative to back-pressure.



ASSUMPTIONS

- The system is fault tolerant
- The system is very secure and does not need any additional security features

COMPONENTS

1. Source - Nexmark
2. Operator - Process Function
3. Cloud - AWS Lambda Function - Java, Python
4. Sink - Writes data to a file

SOURCE

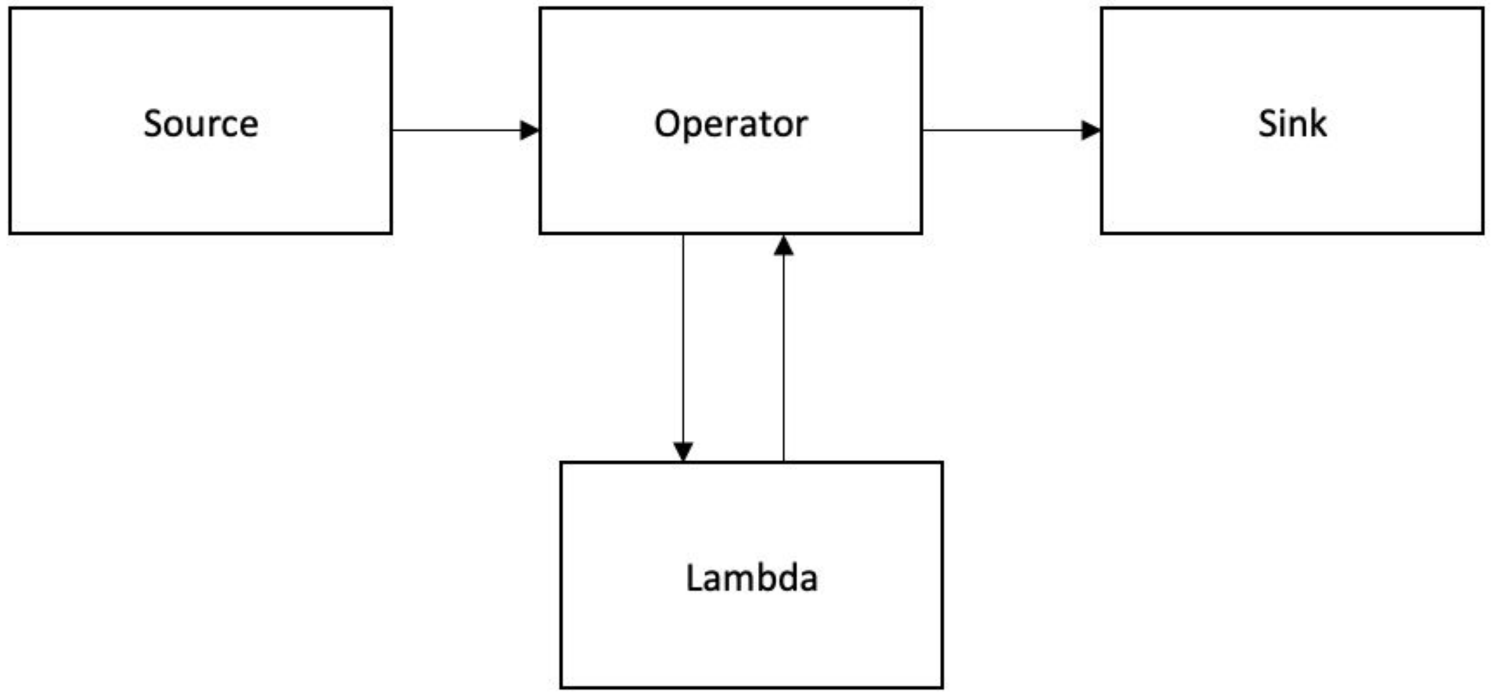
Nexmark Benchmark

online auction system:

- Person
- Auction
- Bid

Full control of events proportions, numbers, intervals etc.

DESIGN



METRICS

<u>Offloading Process:</u>

- Measure Interval: Predefined interval for evaluating offloading decisions.
- During processElement, the current time is checked against the last measure time.
- Threshold: Predefined threshold for offloading.
- records Processed Counter: Flink's built-in counter for tracking the number of processed records.
- If elapsed time >= measure interval, calculate input rate using records Processed counter.
- If input rate > threshold, enable offloading and perform tokenization using AWS Lambda.
- If input rate <= threshold, tokenize within Tokenizer ProcessFunction.

AWS - LAMBDA

- 2 lambda function apps deployed - Java and Python
- Code developed as deployable package for easier CI/CD
- Endpoint URL configured for easy consumption
- Next steps:
  - IAM
  - Context switch between different operators
  - State management when implementing stateful operators

File Sink

- Send the tokenized data into the File Sink.
- Used Lambda expressions to fit the directory path with different systems.
- Encode the String using utf-8.

Next step:

- Use Kafka as downstream

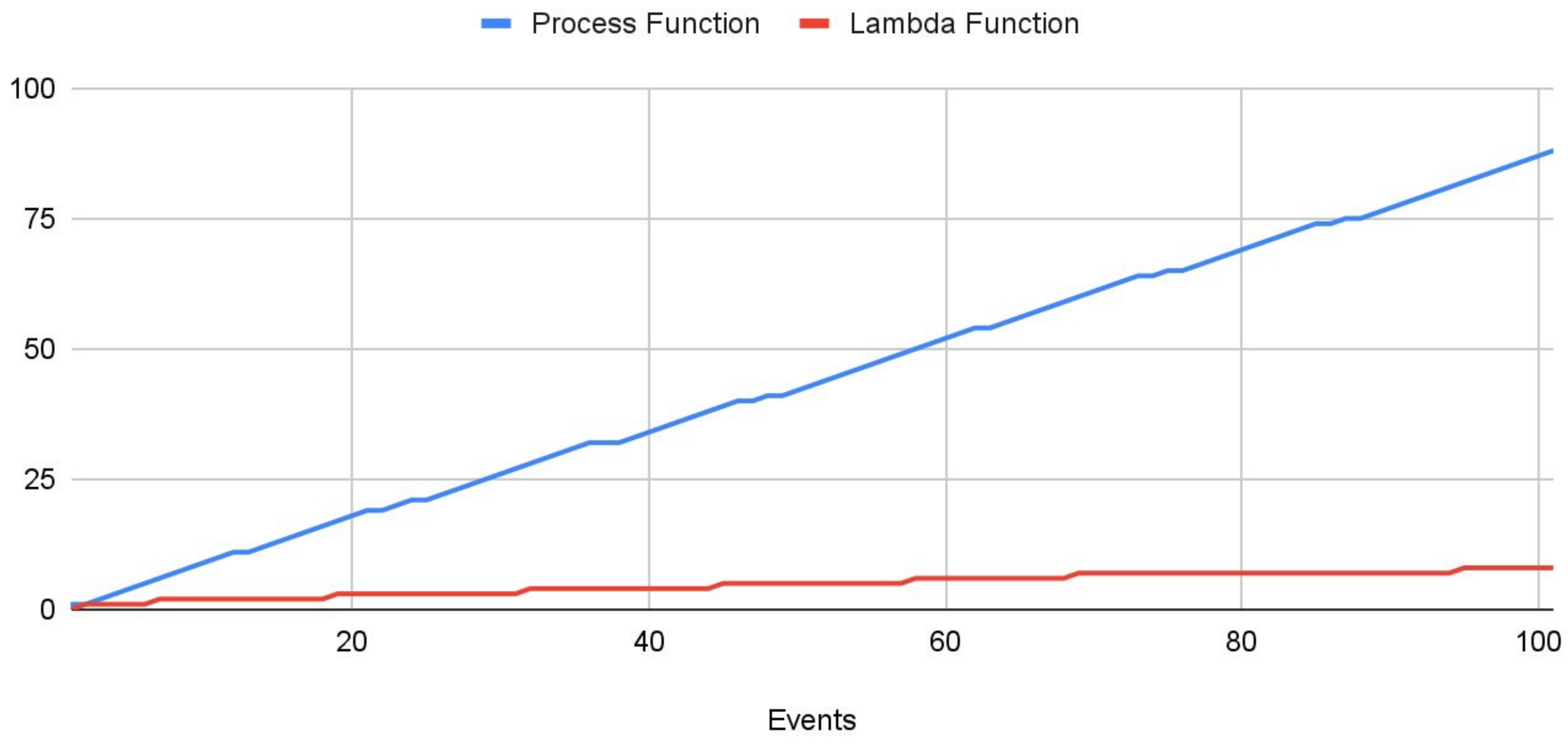
INTEGRATION

- Invoking AWS Lambda from our pipeline achieved using “HttpURLConnection” object
- Use a GET/ POST method as configured in the cloud
- Method writes the JSON payload to the output stream of the HttpURLConnection object and closes stream
- Method then retrieves the response code and the response body from the HttpURLConnection object
- Next Steps:
  - authentication(access key ID & secret access key)
  - use ARN instead of URL (safer)

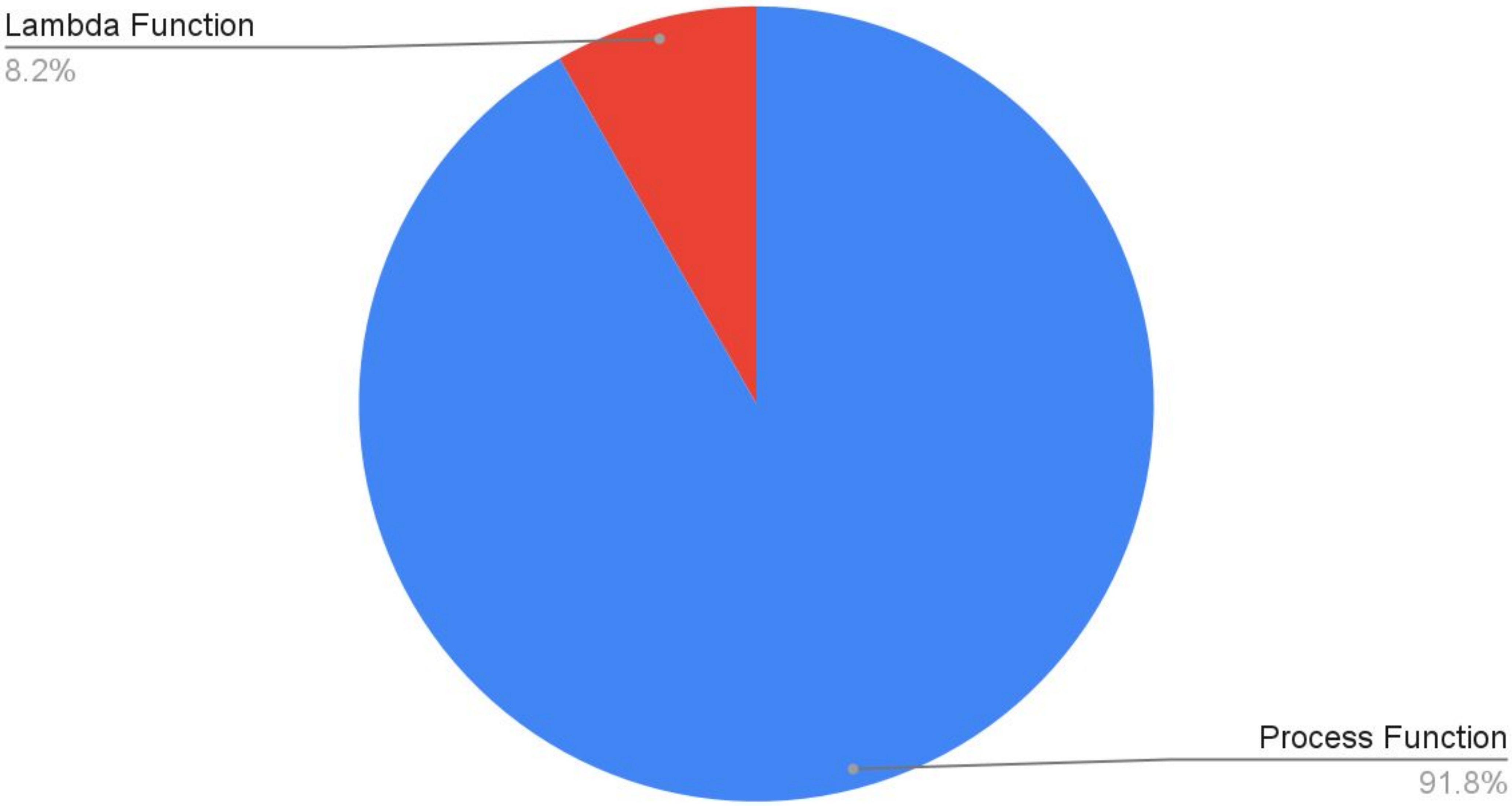
CHALLENGES

- Operator decision - flat Map vs Process Function
- How to setup the flink pipeline
- Figuring out aws Lambda and making it available to external sources
- Connecting to aws Lambda and being able to send it information and retrieving data from the function
- Figuring out metrics for offloading payload to lambda
- Estimating efficiency of approach to offload
- Understanding how backpressure is generated on the Flink UI and how it can be observed in the interface
- Understanding how to connect standard source to pipeline for data generation and control

Process Function and Lambda Function Data rate = 100 events/msec



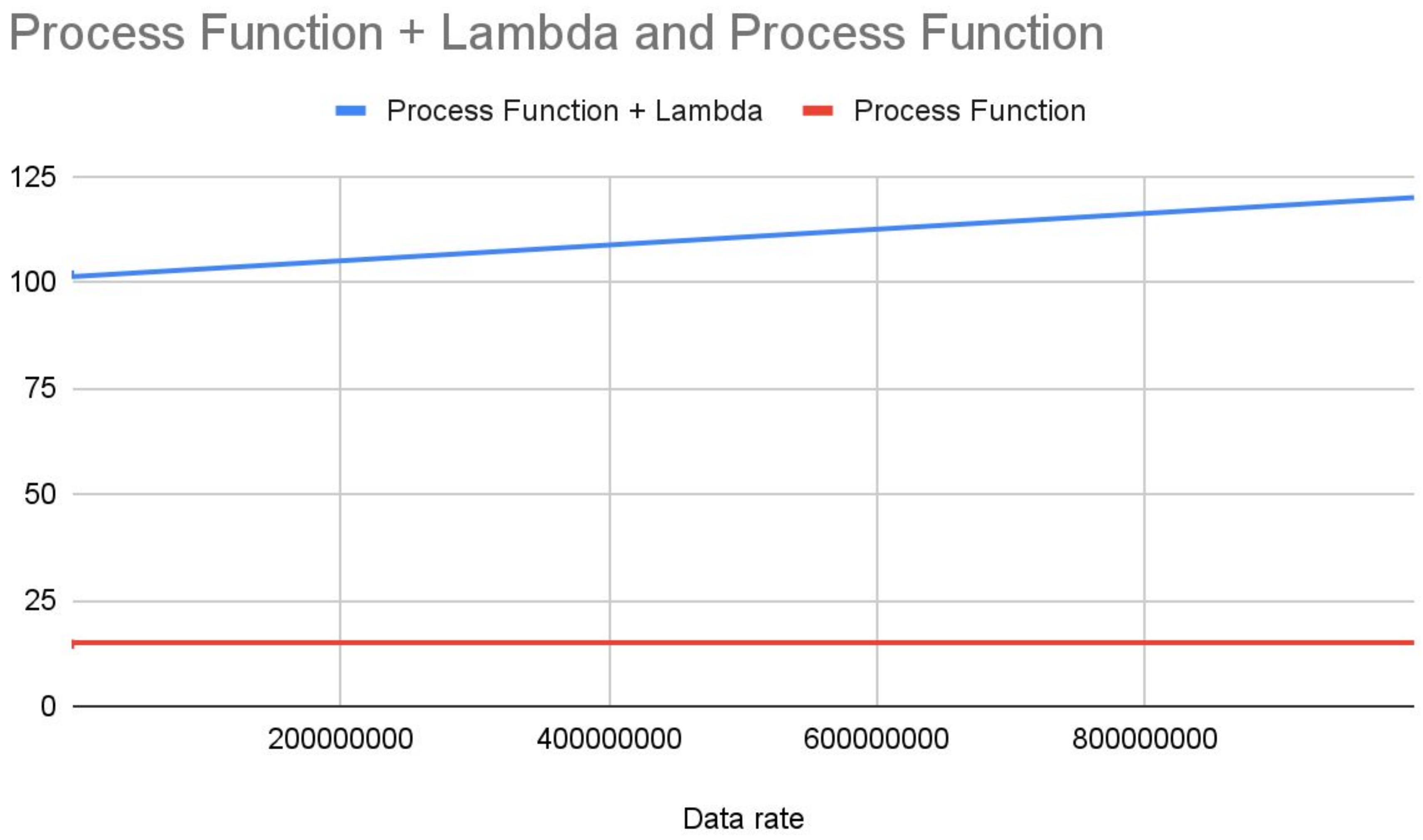
Events



FUTURE WORK

- Test with stateful operator
- Work on state management
- Work on chaining different operators, identifying the bottlenecks
- Work on improving Lambda functions
- Work on improving the metrics to offload
- Experiment and evaluate

QUESTIONS


Process Function + Lambda and Process Function
Process Function + Lambda
Process Function
125
100
75
50
25
0
200000000
400000000
600000000
800000000
Data rate