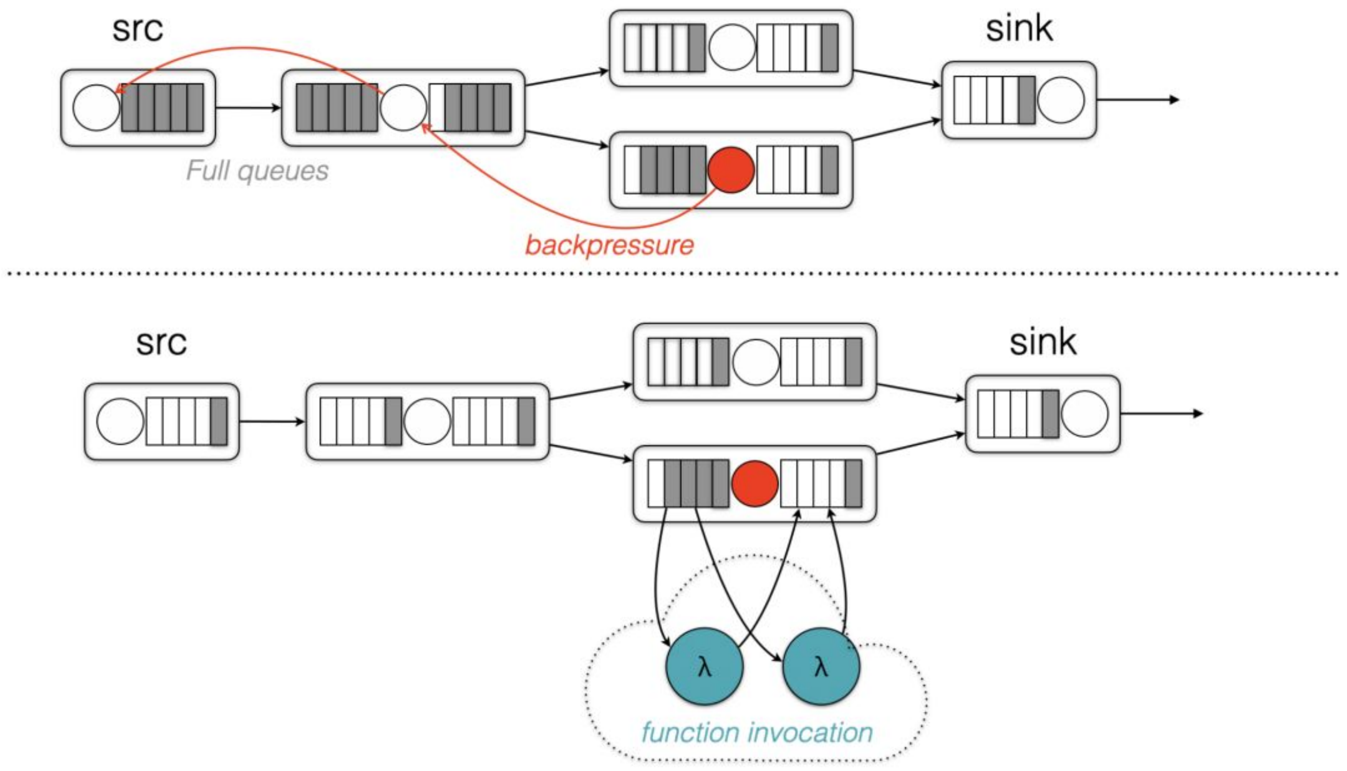
APACHE FLINK CLOUD BURSTING

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

Project 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.



Main achievements

- Implemented an Fault tolerant adaptive Flink application leveraging "cloud bursting" technique.

- Developed a controller that periodically collects execution metrics in order to invoke cloud bursting when workload variations are detected.

- Integrated with AWS lambda functions written in Python and Java with code package deployment.

- Implemented a custom source with dynamic input rate and spikes generation.

- Tested and evaluated the system's performance under different workload conditions to demonstrate its effectiveness in handling workload spikes and as an alternative to back-pressure.

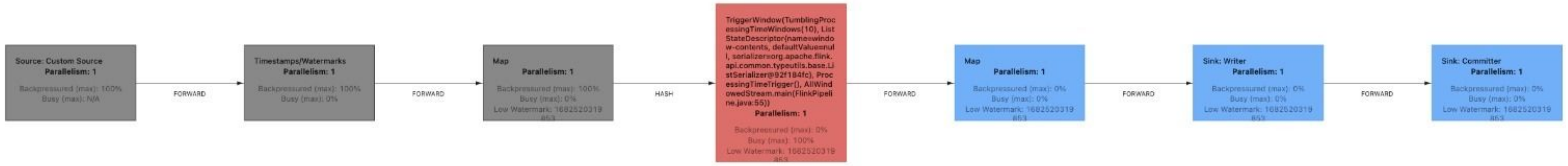
- Documented the implementation details and provided clear instructions for reproducing the results.

Demo

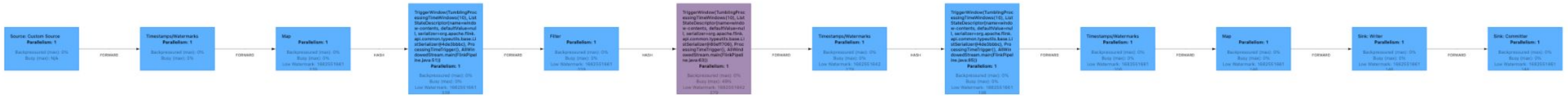
Experimental Setup

Ran our experiments with **3** different pipelines

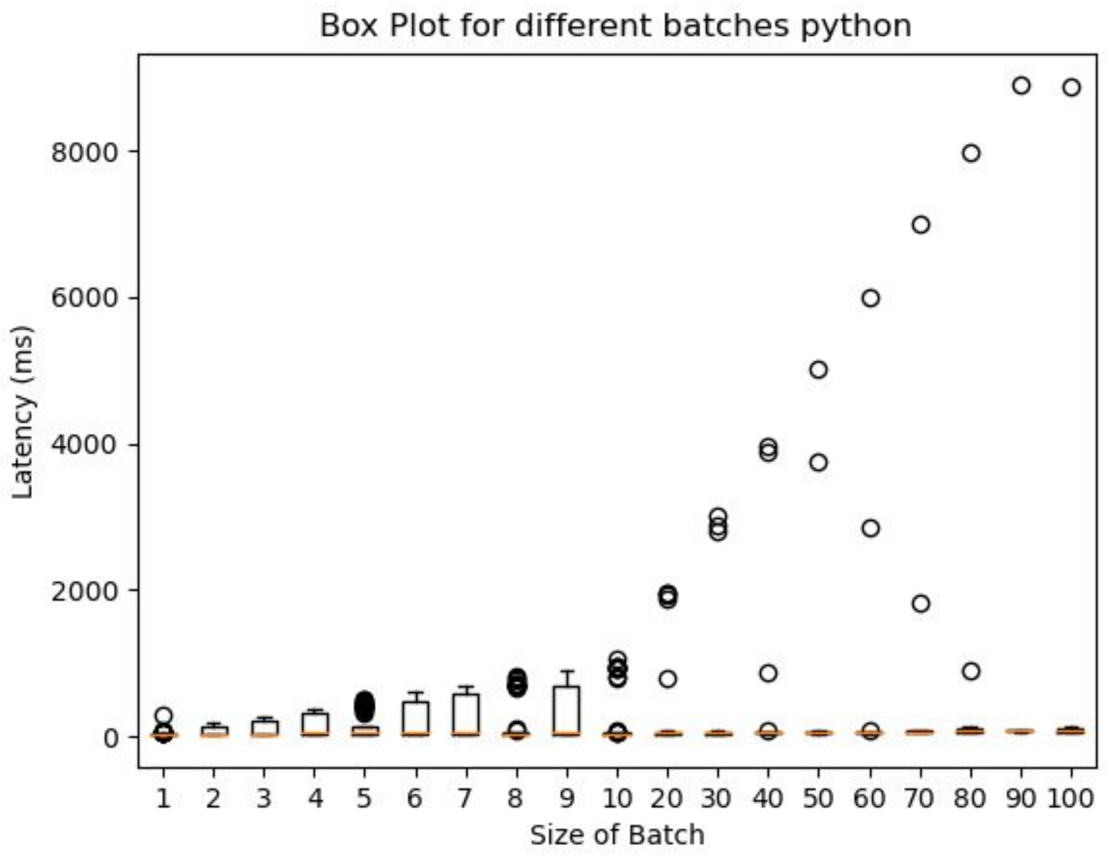
Vanilla Pipeline

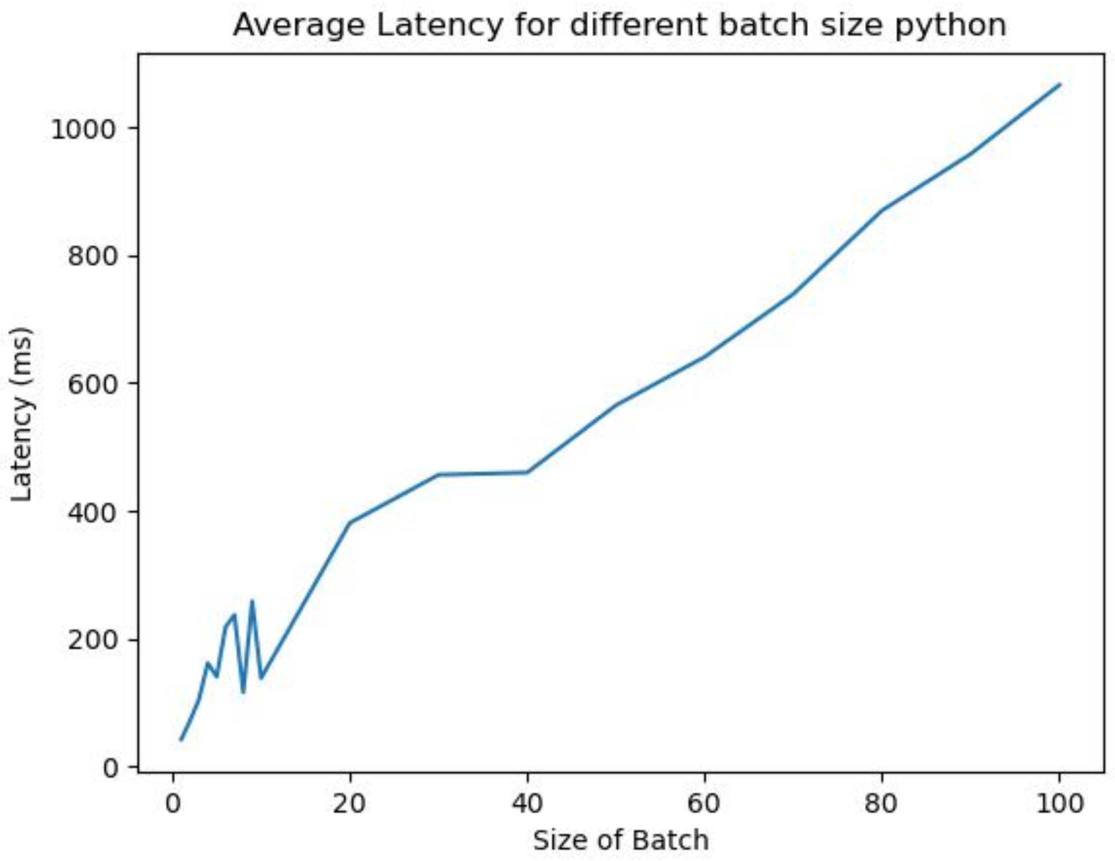

Source: Custom Source
Parallelism: 1
Backpressured (max): 100%
Busy (max): N/A
FORWARD
Timestamps/Watermarks
Parallelism: 1
Backpressured (max): 100%
Busy (max): 0%
FORWARD
Map
Parallelism: 1
Backpressured (max): 100%
Busy (max): 0%
Low Watermark: 1682520319853
HASH
TriggerWindow(TumblingProcessingTimeWindows(10), ListStateDescriptor{name=window-contents, defaultValue=null, serializer=org.apache.flink.api.common.typeutils.base.ListSerializer@92f184fc}, ProcessingTimeTrigger(), AllWindowedStream.main(FlinkPipeline.java:55))
Parallelism: 1
Backpressured (max): 0%
Busy (max): 100%
Low Watermark: 1682520319853
FORWARD
Map
Parallelism: 1
Backpressured (max): 0%
Busy (max): 0%
Low Watermark: 1682520319853
FORWARD
Sink: Writer
Parallelism: 1
Backpressured (max): 0%
Busy (max): 0%
Low Watermark: 1682520319853
FORWARD
Sink: Committer
Parallelism: 1
Backpressured (max): 0%
Busy (max): 0%
Low Watermark: 1682520319853

Modified Pipeline without Lambda

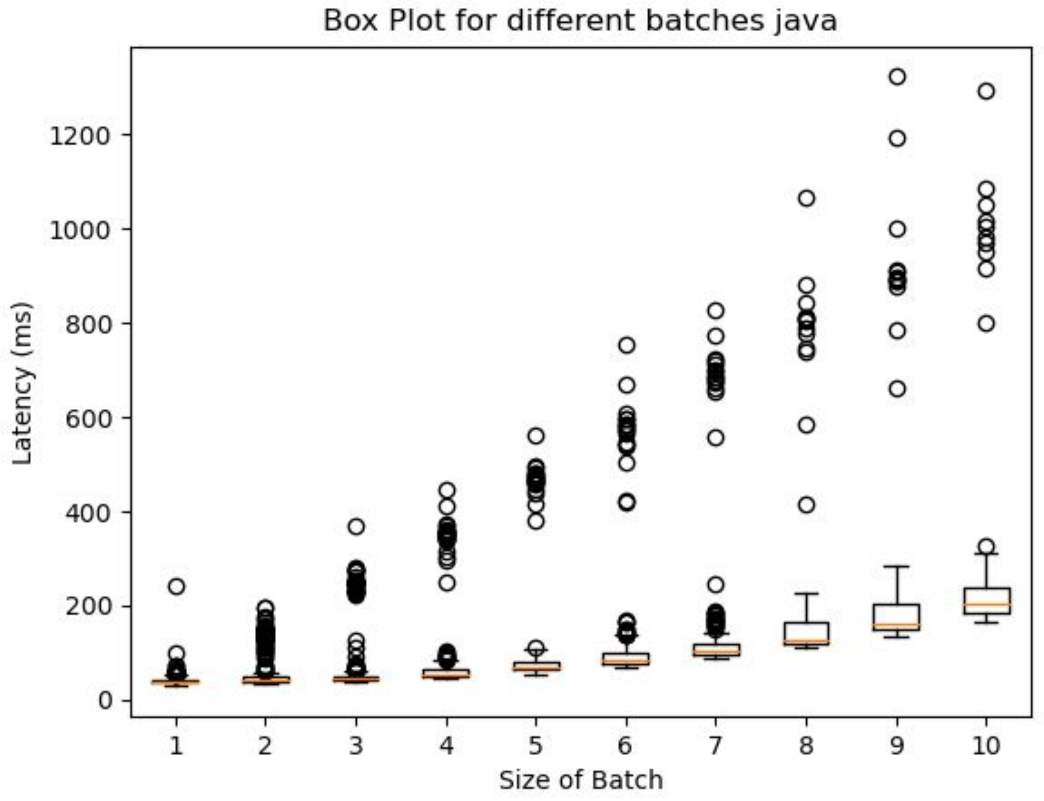

Source: Custom Source
Parallelism: 1
FORWARD
Timestamps/Watermarks
Parallelism: 1
FORWARD
Map
Parallelism: 1
HASH
TriggerWindow(TumblingProcessingTimeWindows(10), ListStateDescriptor{name=window-contents, defaultValue=null, serializer=org.apache.flink.api.common.typeutils.base.ListSerializer@4de3bbbc}, ProcessingTimeTrigger(), AllWindowedStream.main(FlinkPipeline.java:51))
Parallelism: 1
FORWARD
Filter
Parallelism: 1
HASH
TriggerWindow(TumblingProcessingTimeWindows(10), ListStateDescriptor{name=window-contents, defaultValue=null, serializer=org.apache.flink.api.common.typeutils.base.ListSerializer@d0eff706}, ProcessingTimeTrigger(), AllWindowedStream.main(FlinkPipeline.java:63))
Parallelism: 1
FORWARD
Timestamps/Watermarks
Parallelism: 1
HASH
TriggerWindow(TumblingProcessingTimeWindows(10), ListStateDescriptor{name=window-contents, defaultValue=null, serializer=org.apache.flink.api.common.typeutils.base.ListSerializer@4de3bbbc}, ProcessingTimeTrigger(), AllWindowedStream.main(FlinkPipeline.java:85))
Parallelism: 1
FORWARD
Timestamps/Watermarks
Parallelism: 1
FORWARD
Map
Parallelism: 1
FORWARD
Sink: Writer
Parallelism: 1
FORWARD
Sink: Committer
Parallelism: 1

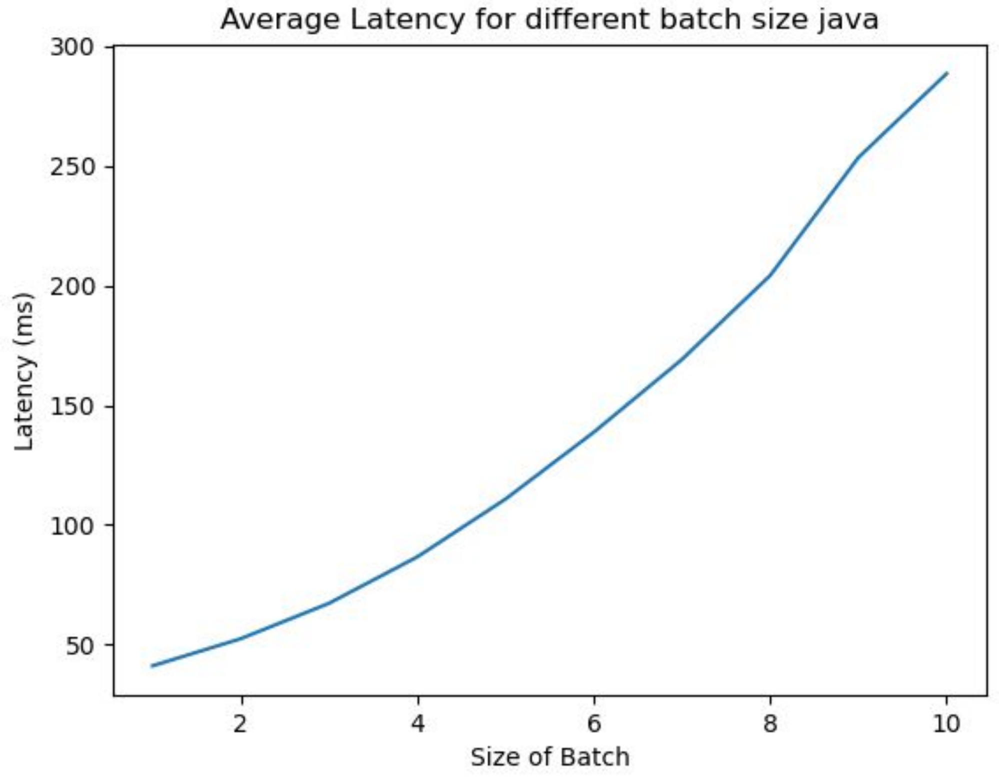
Experimental results



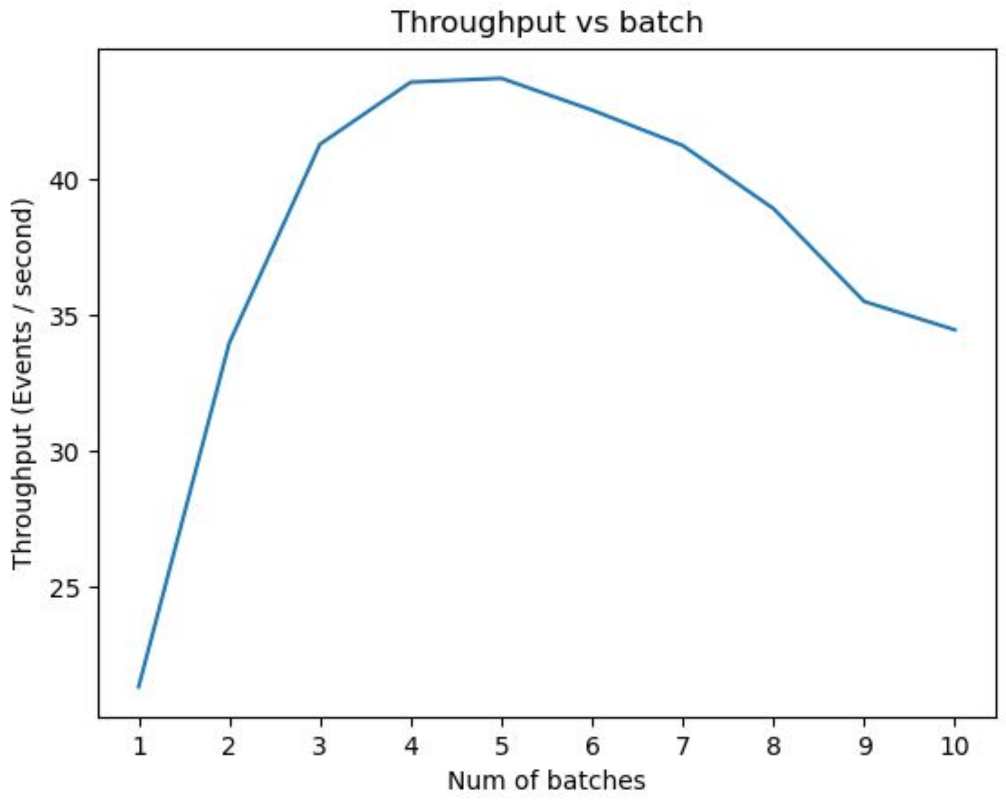


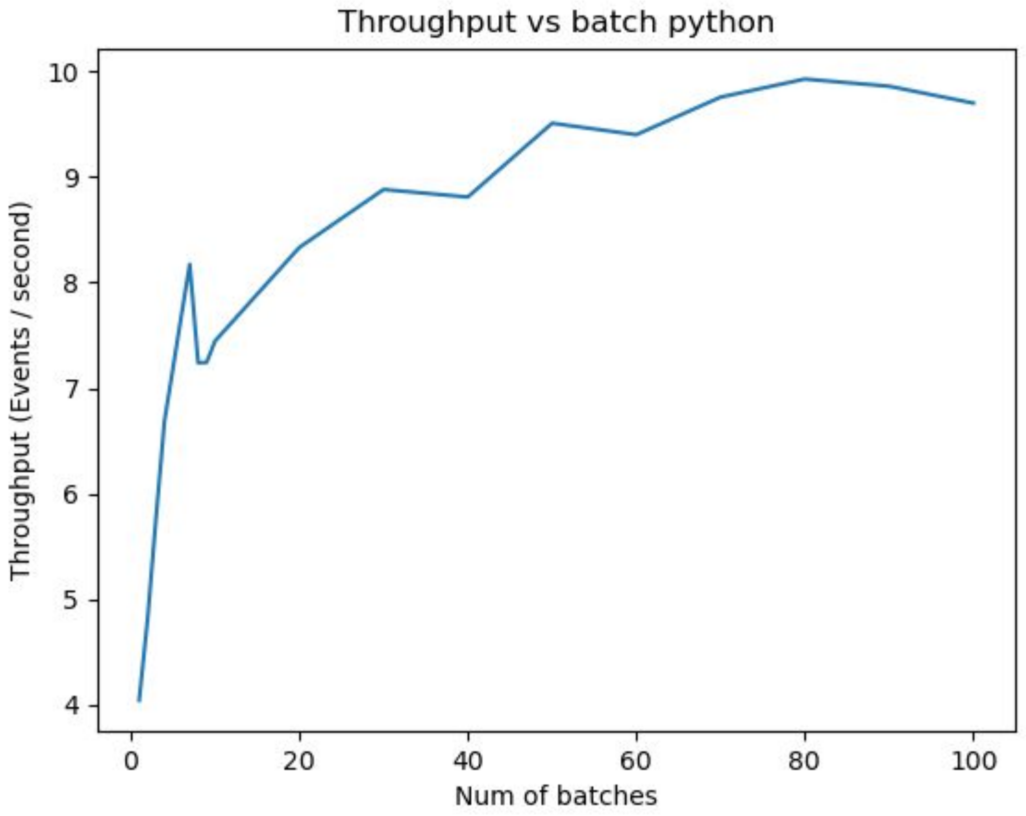
Experimental results





Experimental results

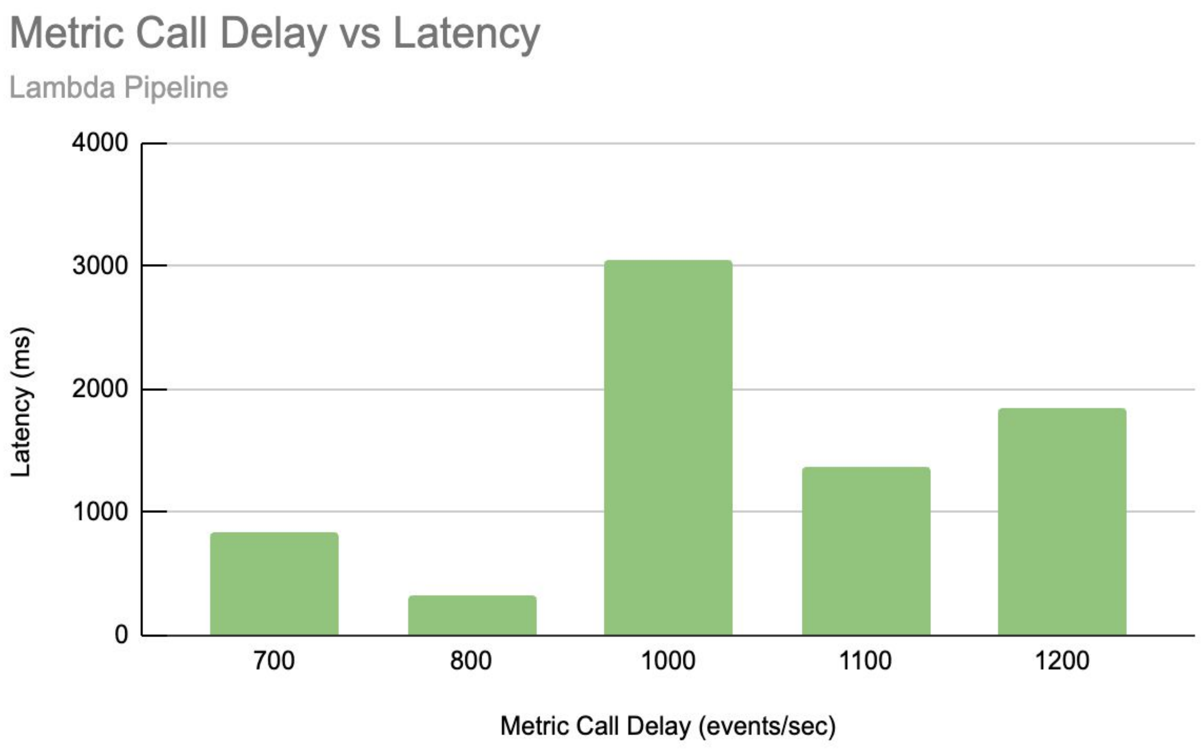


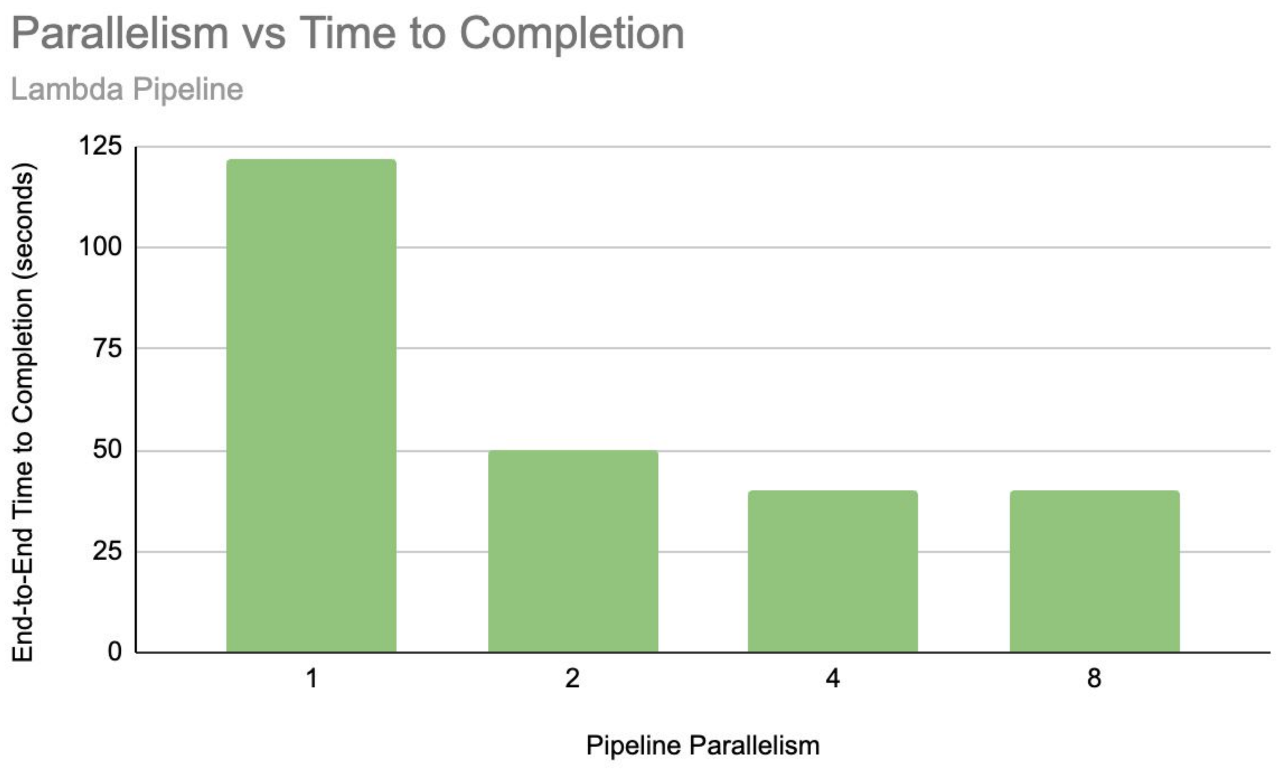


Experimental results

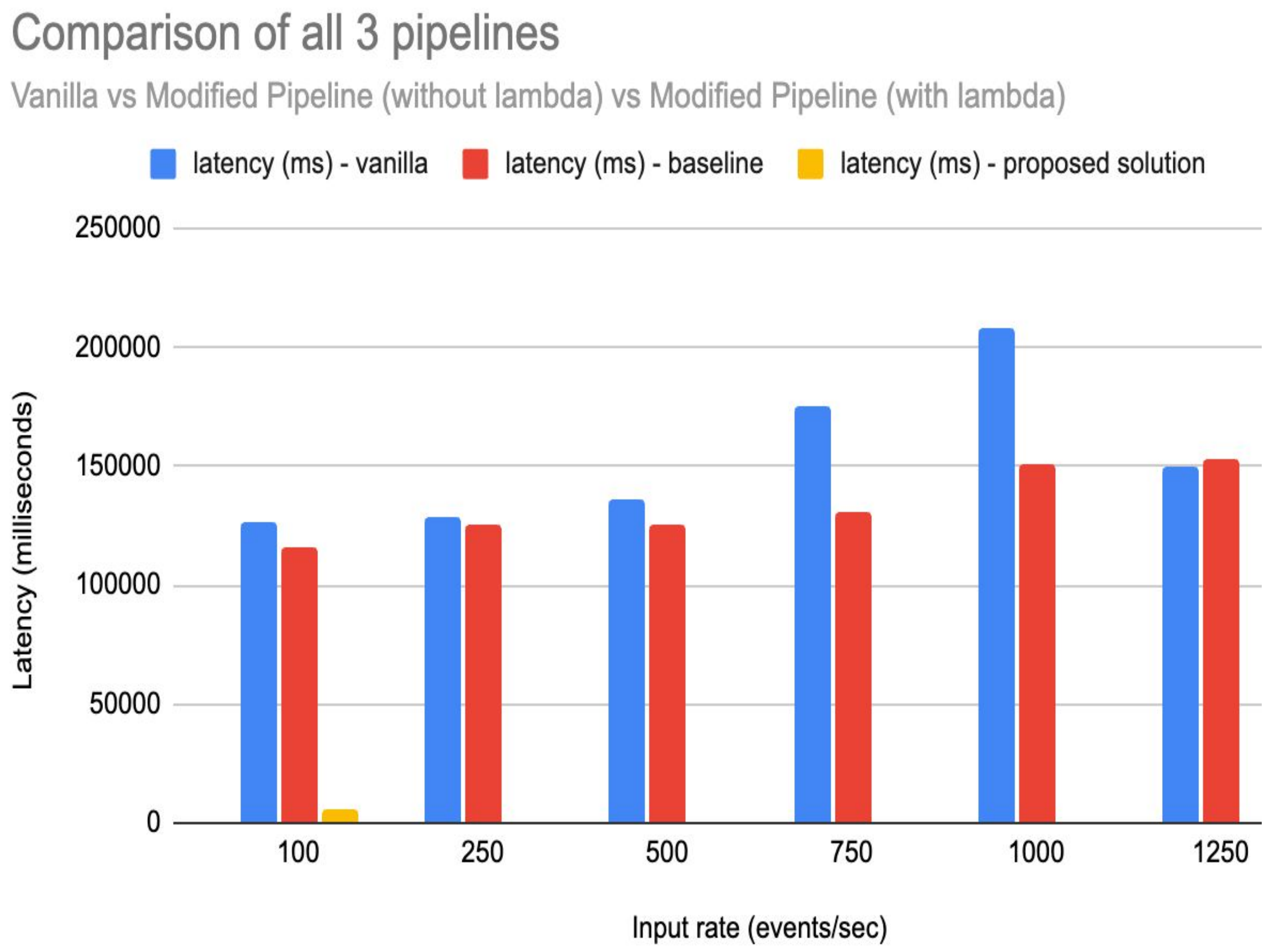
Input Rate = 1000 events/s
Duration of Experiment = 5mins
Parallelism = 1
Max Records Sent = unlimited,
Batch size change-every 8 secs

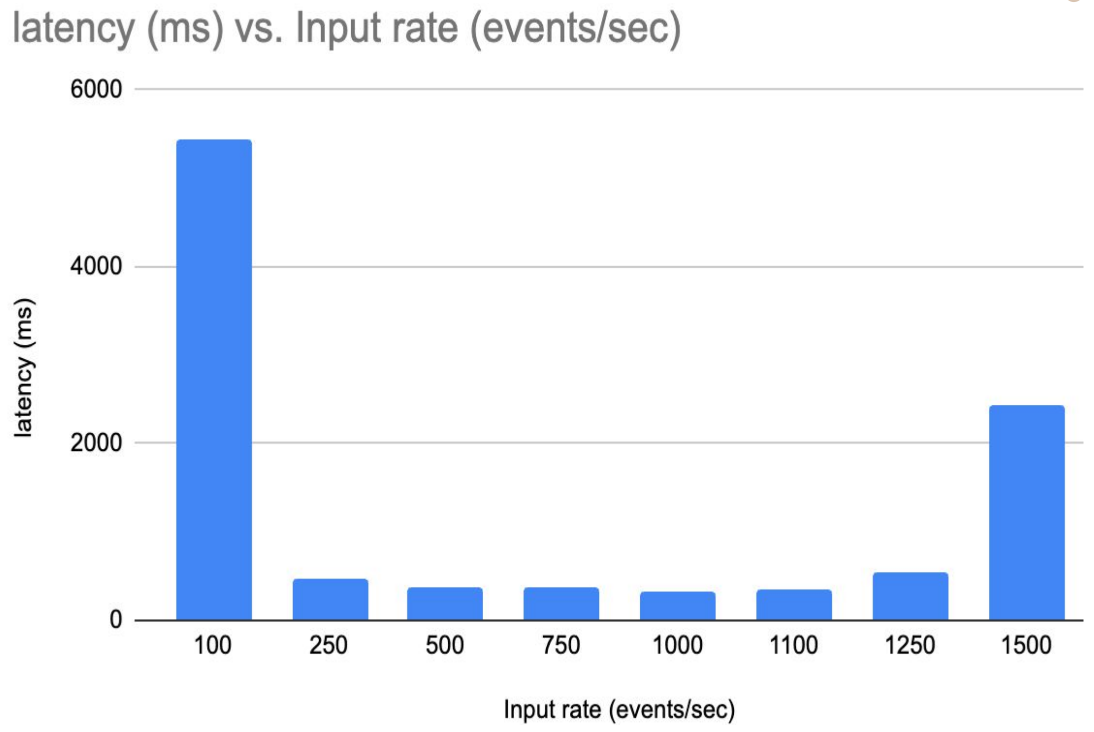
Input Rate = 1000 events/s
Duration of Experiment = 5mins
Metric Call Latency = 800 ms
Max Records Sent = 100000





Experimental results





Experience, Challenges and Future work

Challenges

- Hyper parameter tuning
- Integration with lambda
- Policy metrics calculation
- Integration
- Flink documentation redirects to erroneous URL at many instances.

Future work

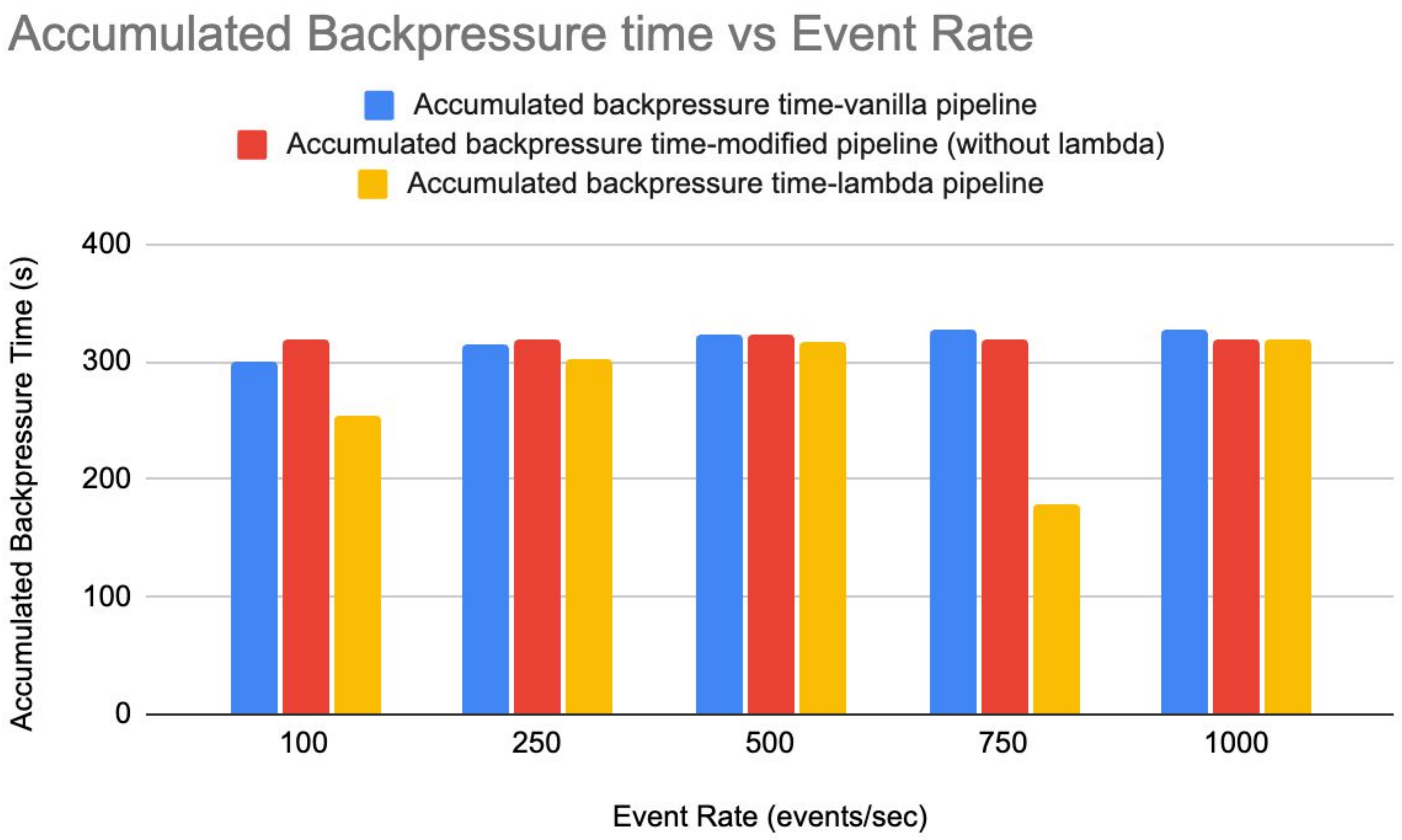
- Handling stateful operations
- Security with lambda
- Fault tolerance
- Fetch the flink metrics directly from code instead of getting it from the flink web UI

Key takeaways

Overhead increases a little, but we save on processing time and reduce back pressure on the pipeline

thank you
danke
gracias
obrigado
teşekkür ederim
go raibh maith agat
eskerrik asko
dziękuję
sukriya
kop khun krap
mochchakkeram
ngiyabonga
格雷西
terima kasih
bedankt
dakujem
toda
takk
хвала
дзякуй
고맙습니다
ありがとう
ευχαριστώ
dankewol
тападh leat
hvala
mauruuru
sagolun
salamat
дякуємо
ти благодарам
благодаря
рахмат
баярлалаа
спасибо
dank je
mersi
barka
welalin
tack
misaotra
blagodaram
faafetai lava
kiitos
dankie
dhanyavad
enkosi
sobodi
děkuji
mèrsi
najis tuke
kam sah hamnida
tanemirt
rahmet
diolch
dhanyavadagalu
shukriya
trugarez
djiere dieuf
mamnun
asante
manana
mahalo
paldies grazzi
matondo
sulpáy
aitäh
taiku
obrigado
grazie
gràcie
nandri
dankon
obrigada

Duration of Experiment = 5mins
Parallelism = 1
Max Records Sent = unlimited,
Batch size change-every 8 secs
Metric Call Latency = 1000 ms


Accumulated Backpressure time vs Event Rate
Accumulated backpressure time-vanilla pipeline
Accumulated backpressure time-modified pipeline (without lambda)
Accumulated backpressure time-lambda pipeline
Accumulated Backpressure Time (s)
Event Rate (events/sec)