Cost of Fault-tolerance on Data Stream Processing

Valerio Vianello, Marta Patiño-Martínez, Ainhoa Azqueta-Alzúaz
{vvianello,mpatino,aazqueta}@fi.upm.es
Universidad Politécnica de Madrid, Madrid, Spain.

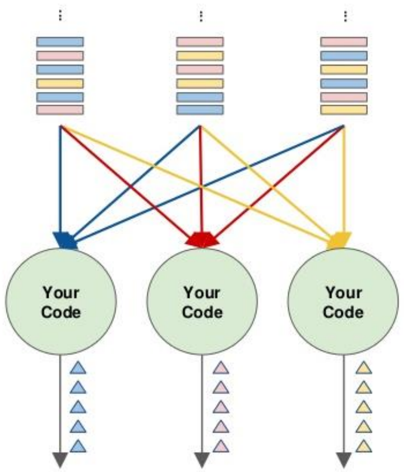
Ricardo Jimenez-Péris
rjimenez@leanxcale.com
LeanXcale

Outline

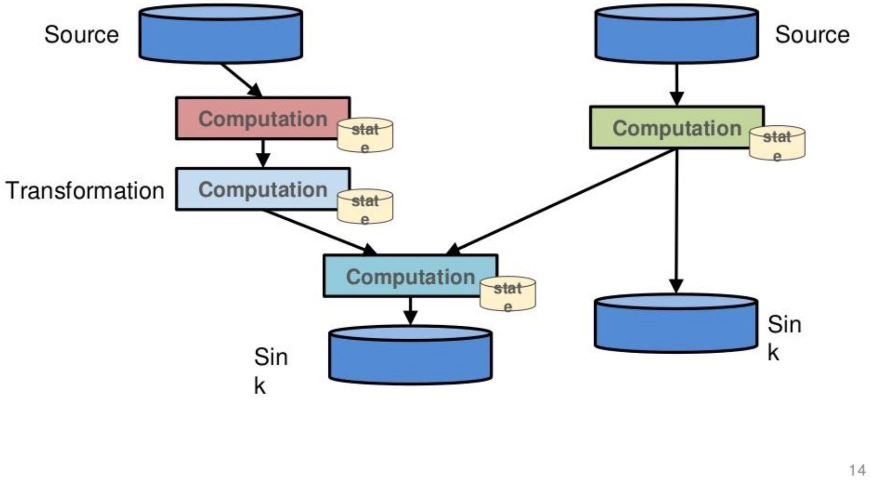
- Introduction.
- Flink Architecture.
- Fault Tolerance in Flink.
- Performance Evaluation.
- Conclusions and Future work.

Introduction

- Apache Flink is an open source platform for scalable stream processing.
- A Flink program transforms incoming data streams and returns results through sinks that can write to different destinations.

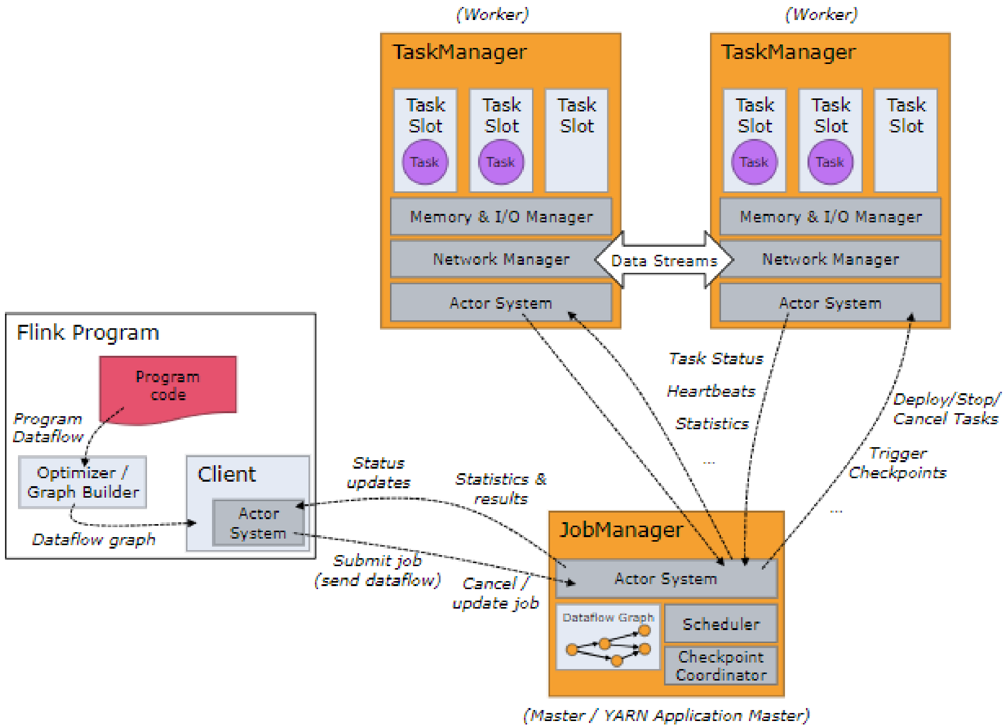


- It partitions input streams by some key.
- It distributes computation across multiple instances.
- Each instance is responsible for some key range



Flink Architecture

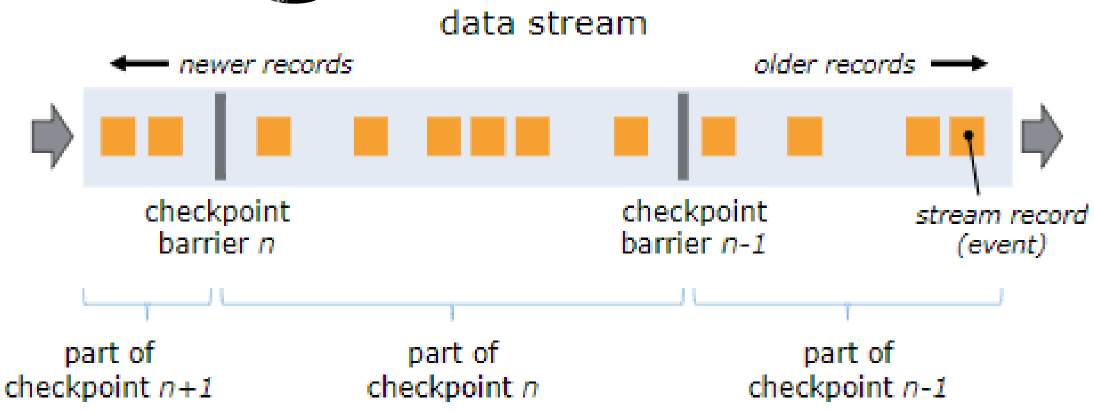
- JobManager: master of a Flink cluster, it is in charge of coordinating the distributed execution.
- TaskManager: runs topologies (or part of them) and manages the data exchange using streams.
- Task slots are used to split and isolate TaskManager dedicated memory for different topologies.



Computation Blocks

- Stateless: each event is processed independently.
  - Ex: Filtering, Projection..
- Stateful: operators store data across the processing of individual events.
  - Ex: Aggregation, Joins..
- Operator state:
  - Each operator instance holds a part of the whole state.
  - Managed State: data structure controlled by Flink.
  - Raw State: user data structures.
- State is made fault tolerant by using checkpointing mechanism.

State Checkpointing



- Burrier tuple: separates records in a data stream into sets.

- A snapshot (serialized state) of an operator is taken when a barrier tuple is received from all its input streams.
- Then, the operator sends the barrier in all its outgoing streams.
- When a sink receives barrier ‘n’ from all its incoming streams, it informs the snapshot coordinator.
- When the coordinator receives this message from all the sinks in the topology, the n-th snapshot is completed.

Fault Tolerance

- Mechanism to consistently recover the state of data streaming applications.
- Flink continuously draws snapshots of the distributed streaming data flow.
- In case of a program failure:
  - Flink stops the distributed streaming dataflow.
  - Then, it restarts the operators and resets them to the latest successful checkpoint.
  - The input streams are reset to the point of the state snapshot.
  - Any records that are processed as part of the restarted parallel dataflow are guaranteed not to have been part of the previously checkpointed state.

FT Settings

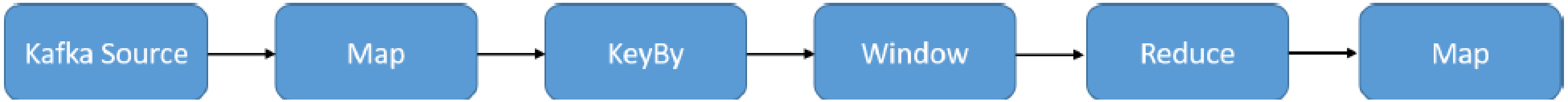
- state.backend:
  - MemoryStateBackend
    - Data are stored in the JAVA heap space available in the JobManager process.
  - FsStateBackend
    - Data are stored as files. Filesystem must be accessible by each and any component => HDFS
  - RocksDBStateBackend
    - Holds in-flight data in a RocksDB database that is (per default) stored in the TaskManager data directories. Upon checkpointing, the whole RocksDB database will be checkpointed into the configured file system and directory.
- state.backend.incremental: only a diff from the previous checkpoint is stored.
- state.checkpoints.num-retained: maximum number of completed checkpoints to retain.
- Minimum time between checkpoints: define checkpoint interval.

Performance Evaluation

- Goal:
  - Evaluate the overhead that fault-tolerance introduces in Flink regular processing and the cost of recovery.
- Benchmark
  - Intel HiBench suite.
- Tested:
  - Distributed tested with 6 powerful machines.
- Fault Injection:
  - Execute command to kill one TaskManager.
- Parameters:
  - State size, input load, state backend.

Benchmark

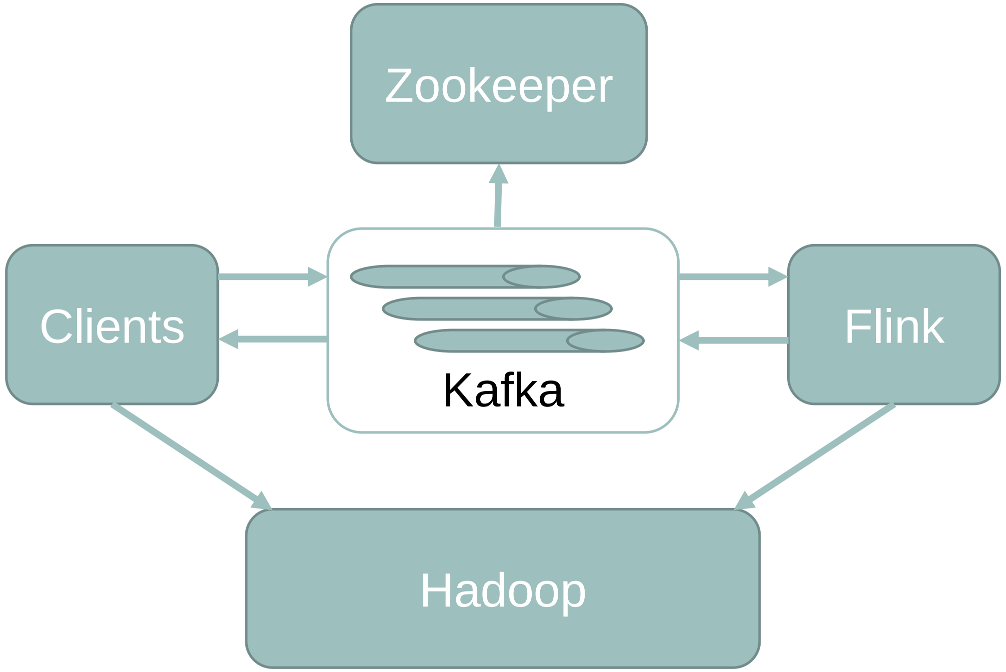
- Intel HiBench Suite.
- Fix-window micro-benchmark
  - The workload performs a window based aggregation. It tests the performance of window operation in the streaming frameworks.



- Load is generated by multiple instances of the benchmark executed in parallel.

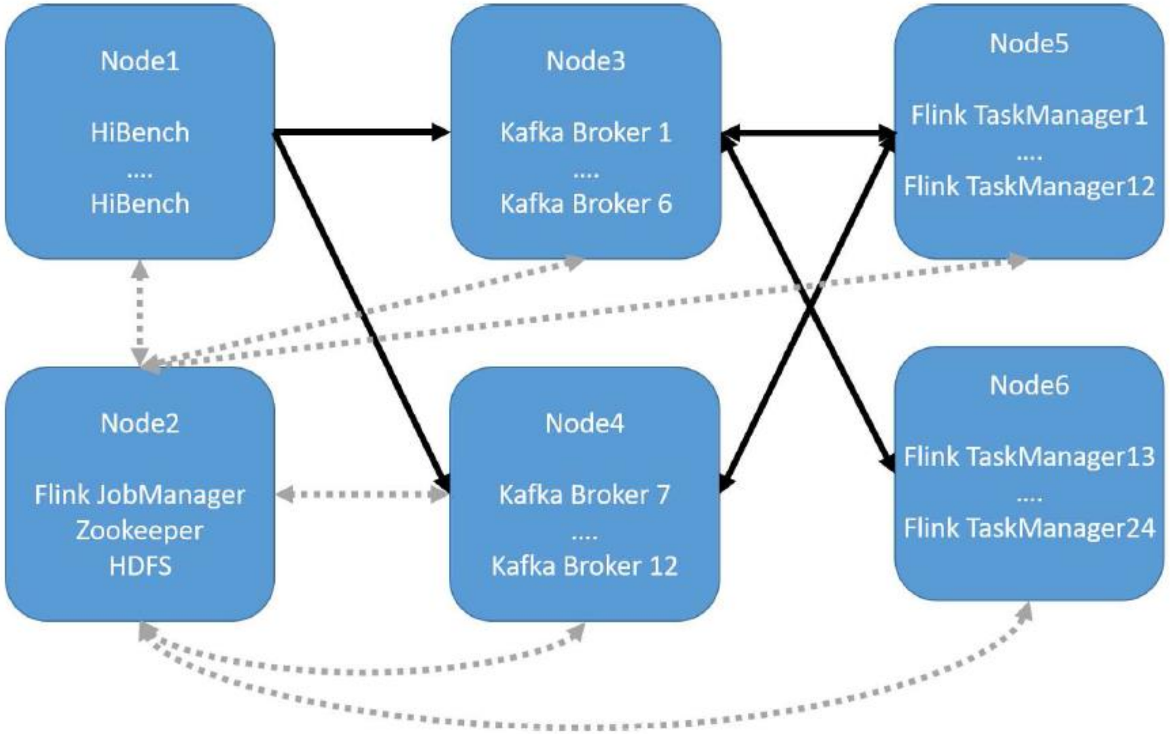
Testbed

- 6 homogeneous nodes. Each node:
  - 2 CPU sockets with Intel XEON E5-2620 v3 with 6 cores each ➔ 24 virtual cores
  - 128 GB RAM divided into 8 slots.
  - Disk: SSD Intel SD3510 480GB.
  - Network: 1Gbit Ethernet.
- Software versions:
  - Flink 1.4.2
  - Intel HiBench 7.0
  - Kafka 2.10-0.8.2.2
  - Hadoop 2.6.5
  - Zookeeper 3.4.8.



Deployment

- Benchmark: from 2 to 5 instances.
- Kafka Cluster: 12 brokers
- Flink Cluster: 24 TaskManagers with 2 slots each ➔ 48 task slots.

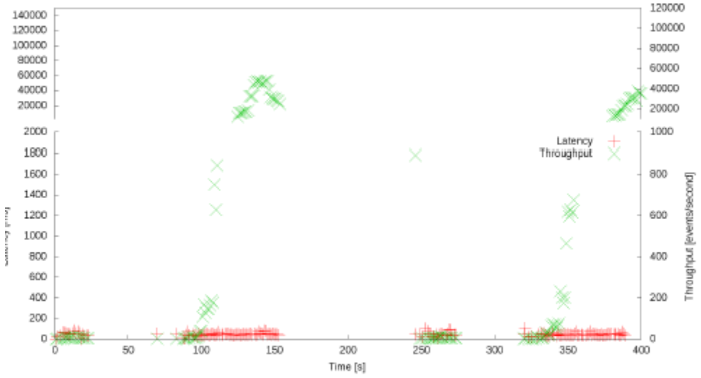


Evaluation Configuration

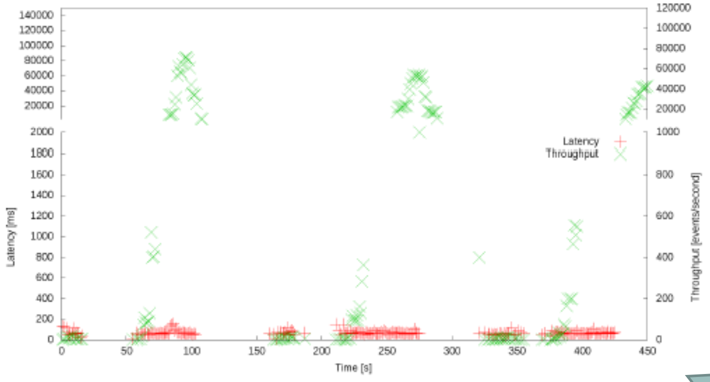
Input Load (r/sec)	Window Size	Checkpointing	Fault Injection
200k - 500k	50 Records	No	No
200k - 500k	30 to 50 Records	HDFS	No
200k - 500k	30 to 50 Records	HDFS	Yes
200k - 500k	50 Records	HDFS + RocksDB	No
200k - 500k	50 Records	HDFS + RocksDB	Yes

Table 1: Experiments configurations.

Results: checkpointing disabled – w50

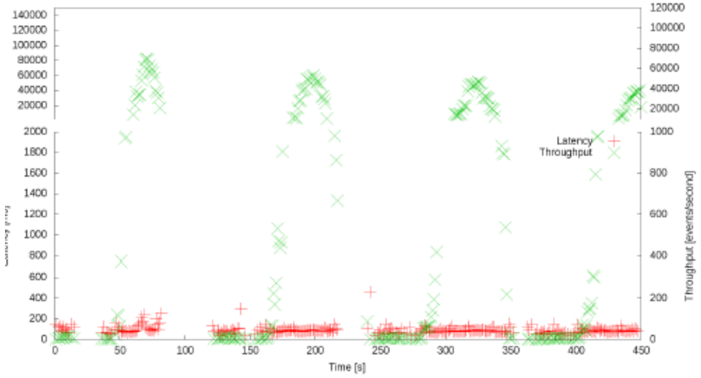


(a) Input load: 200,000 records/second

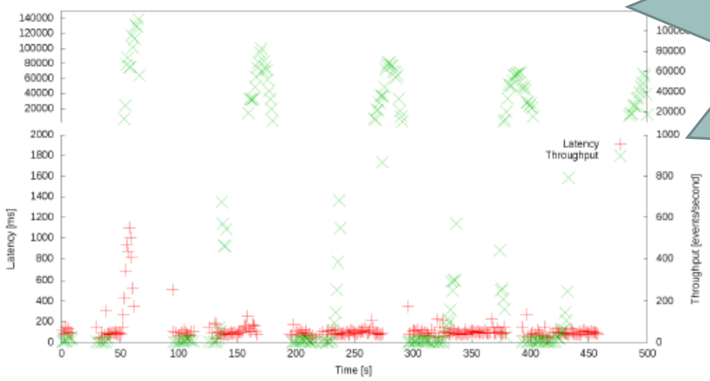


(b) Input load: 300,000 records/second

Very low latency (<200ms) with any load

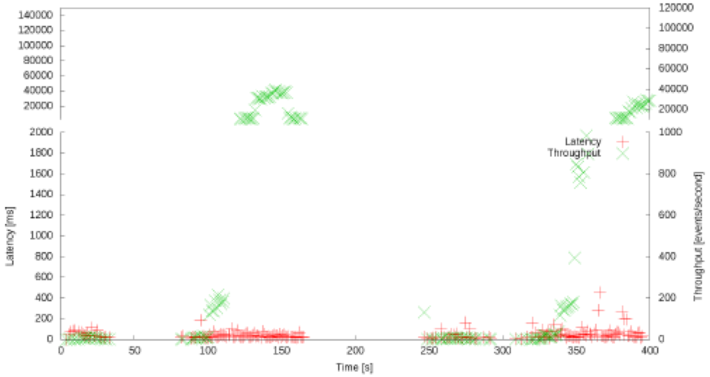


(c) Input load: 400,000 records/second

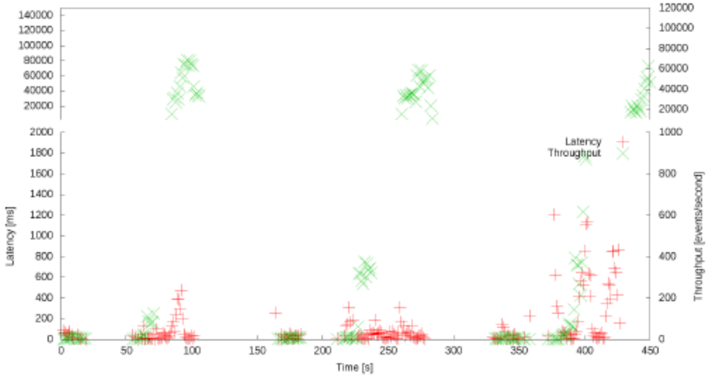


(d) Input load: 500,000 records/second

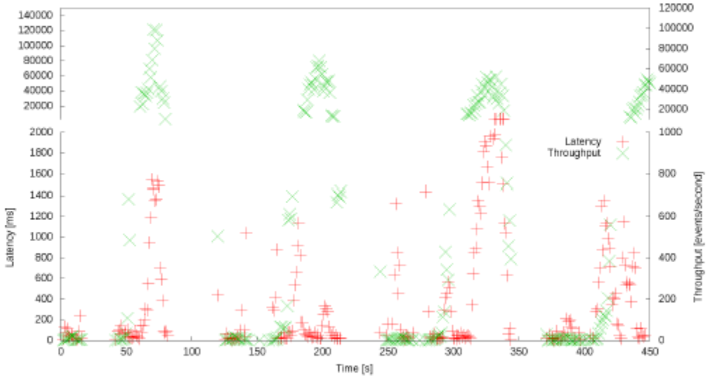
Results: checkpointing on HDFS – w50



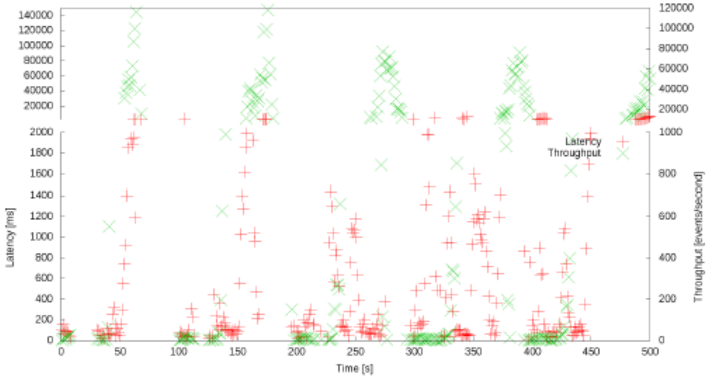
(a) Input load: 200,000 records/second



(b) Input load: 300,000 records/second



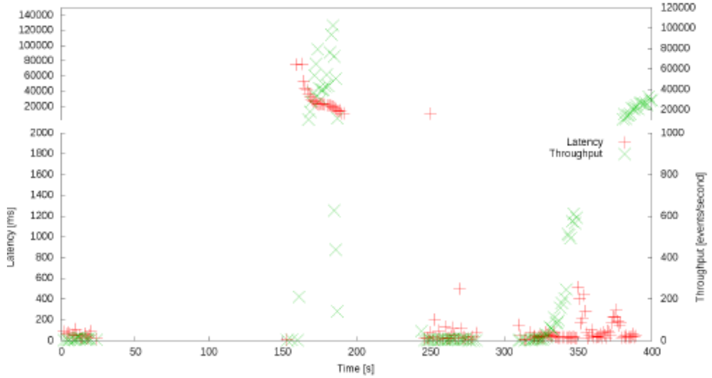
(c) Input load: 400,000 records/second



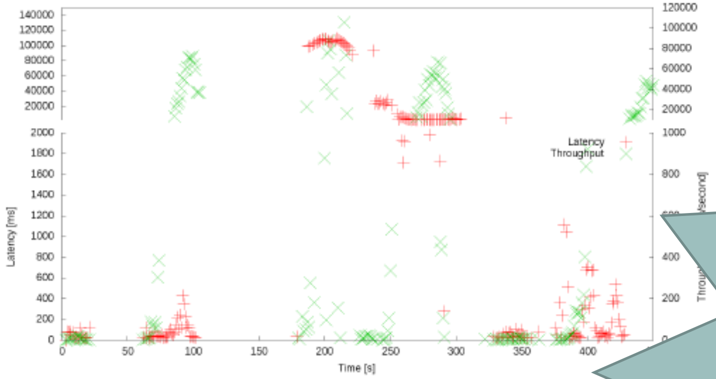
(d) Input load: 500,000 records/second

The system is always able to process the workload showing peak in latency (>10s) with highest workloads

Results: fault injection – w50

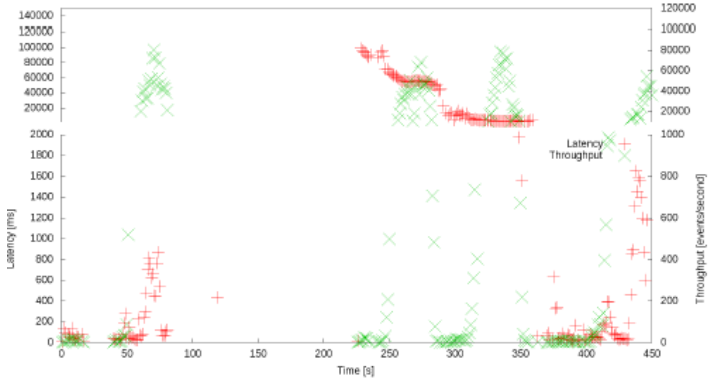


(a) Input load: 200,000 records/second

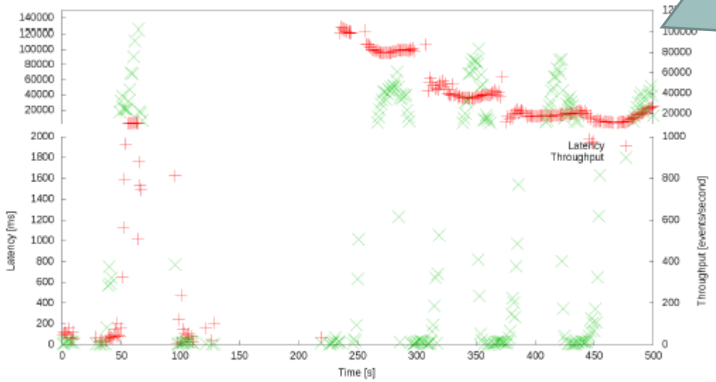


(b) Input load: 300,000 records/second

The system takes from
90 to 170 secs to recover in
the first 3 workloads.
With the highest workload
it is not able to recover during the
experiment evaluation.



(c) Input load: 400,000 records/second

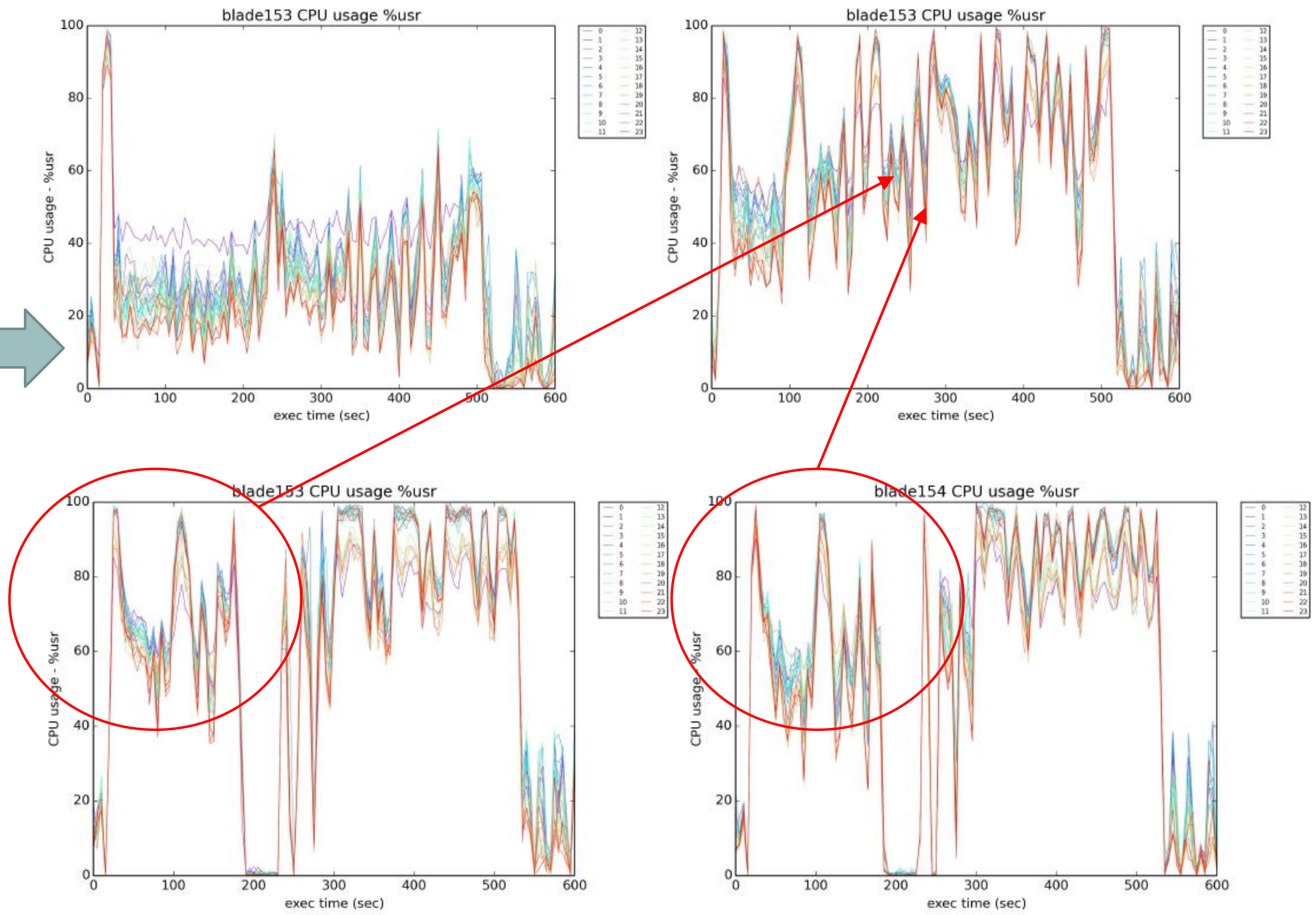


(d) Input load: 500,000 records/second

CPU Utilization

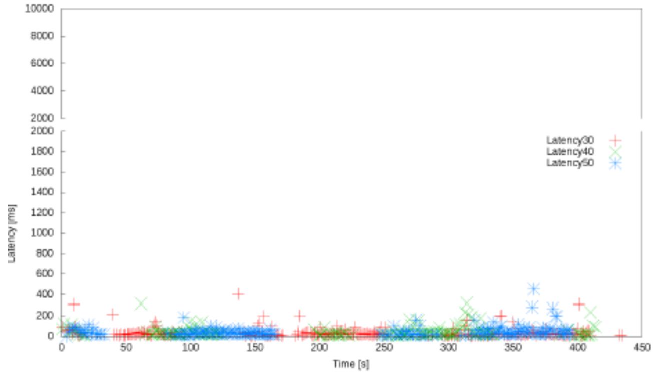
- With checkpointing enabled, the system consumes much more recourses.

No CP

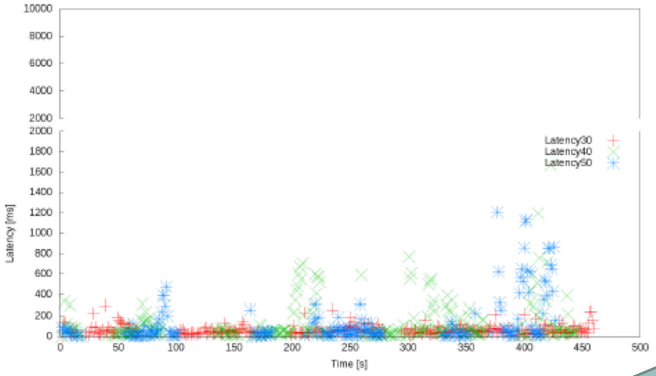


- With the highest workload (500k t/s) it would need more resources to process the pending load in order to recover.

CP Latency / State size

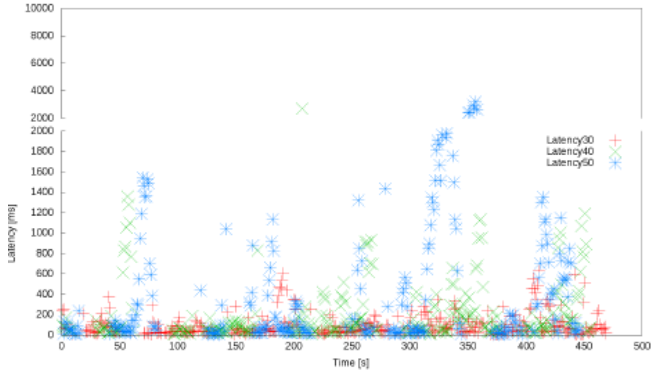


(a) Input load: 200,000 records/second

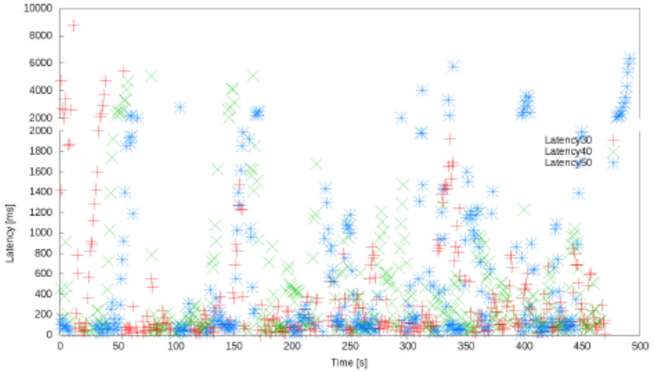


(b) Input load: 300,000 records/second

State size has a strong impact on the latency as expected.



(c) Input load: 400,000 records/second



(d) Input load: 500,000 records/second

Conclusion & Future Work

- In presence of failures the system is able to recover quickly if it has enough available resources.
- Latency:

Input Load (r/sec)	window size 30	window size 40	window size 50	window size 50 no chekpointing
200k	25-113 ms	32-281 ms	34-286 ms	73-158 ms
300k	60-279 ms	125-855 ms	172-947 ms	89-215 ms
400k	127-622 ms	391-2062 ms	768-4186 ms	108-282 ms
500k	681-2499 ms	885-3194 ms	1922-4982 ms	212-1060 ms

Table 2: Latency. Percentiles 75% and 95%

- Network bottleneck: -The network was not able to process higher workloads.
- Incremental checkpointing with RocksDB:
  - In the tested scenario, incremental checkpointing was a drawback for performance.
- Future work:
  - Evaluate the performance with multiple queries deployed at the same time.
  - Evaluate overhead of the new exactly once end-to-end protocols.
  - Compare with other frameworks.