Deterministic Model for Distributed Speculative Stream Processing

DISCAN 2018

Igor Kuralenok, **Artem Trofimov**, Nikita Marshalkin, and Boris Novikov

JetBrains Research, Saint Petersburg State University

Outline

- **Deterministic computations**
- Stream processing computational model
- Optimistic determinism: drifting state
- Experiments
- Exactly-once on top of determinism
- Yet another experiment

JET BRAINS RESEARCH

Deterministic computations

Given a particular input, the same output will be produced after any number of reruns

For streaming it means:

$$F(I_n \dots I_0): \forall k, F(I_k I_{k-1} \dots I_0) = J_k$$

Usually, it is considered as:

$$F(I_n, S_n): \forall k, \exists S_k = S(I_{k-1} \dots I_0), F(I_k, S_k) = J_k$$

JET BRAINS RESEARCH

Why is determinism important?

Determinism is a desired property in many CS areas

- Natural for users (people got used to think *sequentially)*
- Computations are reproducible and predictable
- Implies consistency [*Stonebreaker et al. The 8 requirements of real-time stream processing.* ***ACM SIGMOD Record 2005***]

JET BRAINS RESEARCH

Determinism: simple way

Determinism can be easily achieved if

- All computations are sequential
- All transformations are pure functions

Stream processing

- Shared-nothing distributed runtime
- Record-at-a-time model
- Latency is a key performance metric

Cloud Dataflow

Flink

samza

distributed stream processing

APACHE STORM™

Distributed • Resilient • Real-time

JET BRAINS RESEARCH

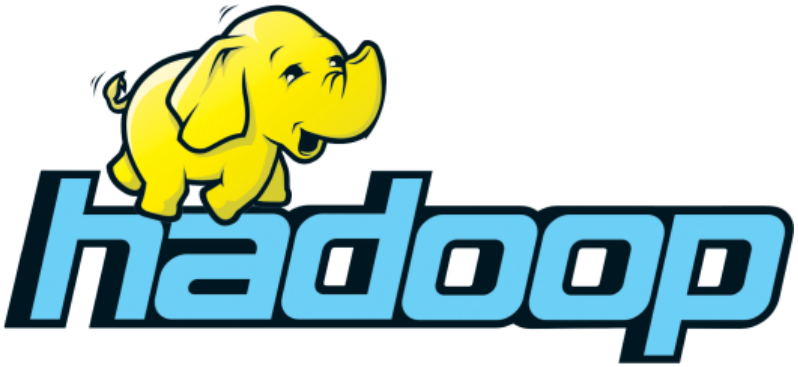
Determinism in stream processing

- It is considered that determinism is too difficult too achieve
- Systems usually provide low-level interfaces, which do not guarantee any level of determinism
- Trade-off between determinism and latency
  [Zacheilas et al. Maximizing Determinism in Stream Processing Under Latency Constraints. ***DEBS 2017****]*

JET BRAINS RESEARCH

What is about batch processing?

- MapReduce is usually implemented deterministically
- Micro-batching (spark streaming, storm trident) is also deterministic



The ultimate question of life, the universe, and everything

Is it possible to combine low-latency and determinism within distributed stream processing?

JET BRAINS RESEARCH

The ultimate question of life, the universe, and everything

Is it possible to combine low-latency and determinism within distributed stream processing?

- In spite of asynchronous distributed processing

JET BRAINS RESEARCH

The ultimate question of life, the universe, and everything

Is it possible to combine low-latency and determinism within distributed stream processing?

- In spite of asynchronous distributed processing
- Avoiding input buffering

JET BRAINS RESEARCH

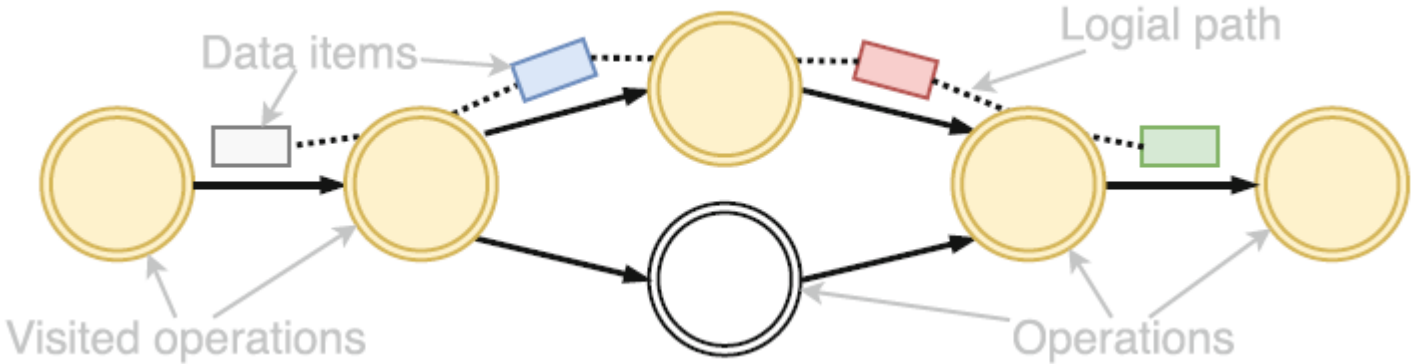
Outline

- Deterministic computations
- **Stream processing computational model**
- Optimistic determinism: drifting state
- Experiments
- Exactly-once on top of determinism
- Yet another experiment

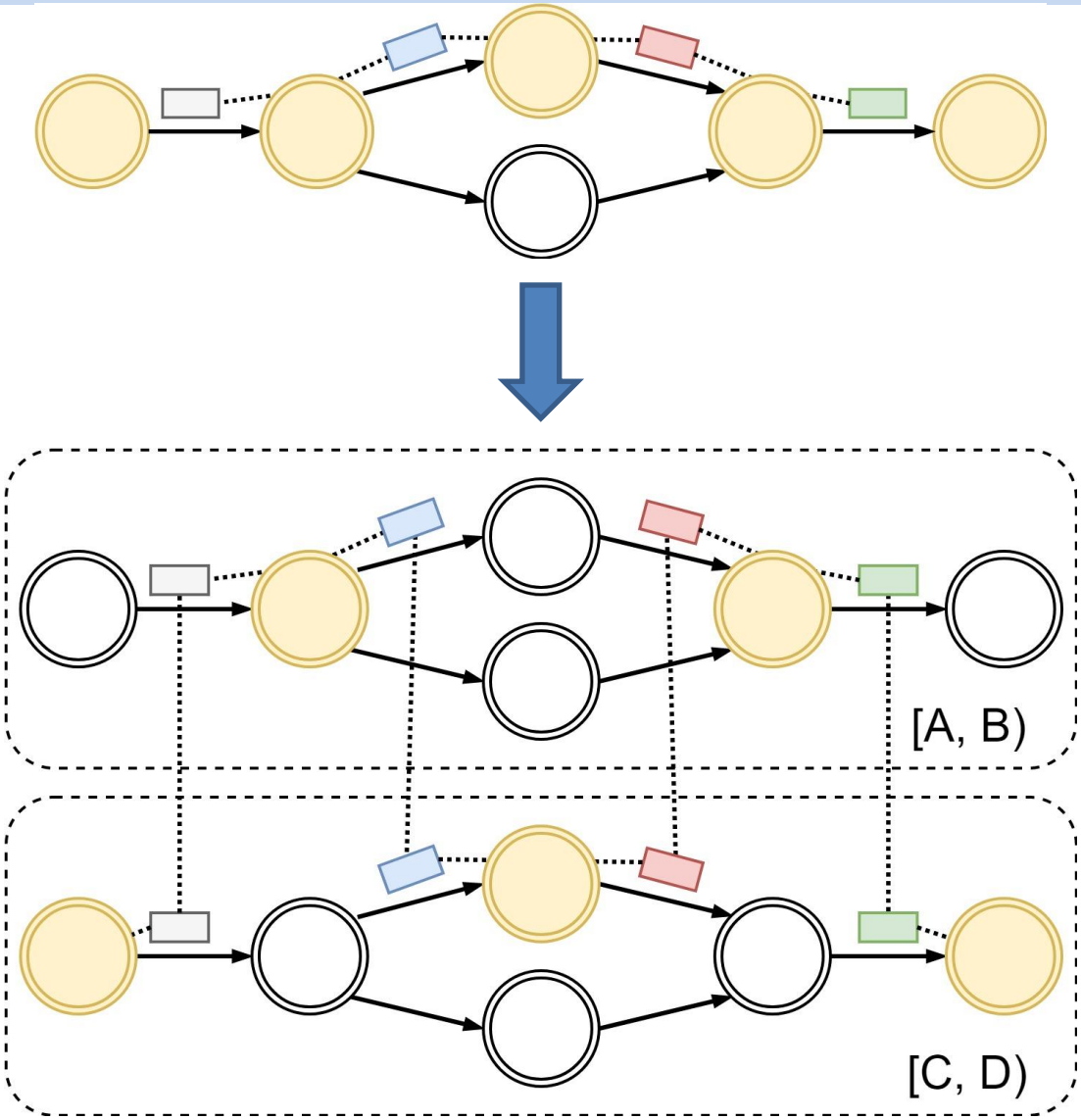
JET BRAINS RESEARCH

Dataflow

- Dataflow is a potentially unlimited sequence of data items
- Timestamps can be assigned to data items to define an order
- Dataflow is expressed in the form of a graph
- Vertices are operations, which are implemented by user-defined functions
- Edges declare an order between operations



Physical deployment



JET BRAINS RESEARCH

Outline

- Deterministic computations
- Stream processing computational model
- **Optimistic determinism: drifting state**
- Experiments
- Exactly-once on top of determinism
- Yet another experiment

JET BRAINS RESEARCH

What do we require to achieve determinism?

- Total order and transformations as pure functions
  - We can define synthetic order by assigning timestamps at system entry

What do we require to achieve determinism?

- Total order and transformations as pure functions
  - We can define synthetic order by assigning timestamps at system entry
- We need to care about the order only in the operations that are order-sensitive and before output

What do we require to achieve determinism?

- Total order and transformations as pure functions
  - We can define synthetic order by assigning timestamps at system entry
- We need to care about the order only in the operations that are order-sensitive and before output
- Calculations are partitioned, and order between items from different partitions does not influence the result (if they will not be merged)

Unrealistic requirement

- Total order preservation

Unrealistic requirement

- Total order preservation
- Let’s try to rethink streaming computations

Drifting state: idea

$$I_k \to \mathrm{Op}(I_k, S_k) \to J_k$$
$$\uparrow\downarrow$$
$$S_k \; S_{k+1}$$

Drifting state: idea

$$I_k \rightarrow \mathrm{Op}(I_k, S_k) \rightarrow J_k$$

$$\uparrow\downarrow$$

$$S_k \; S_{k+1}$$

```
newState = combine(prevState, newItem)
handler.update(newState)
return newState
```

Drifting state: idea

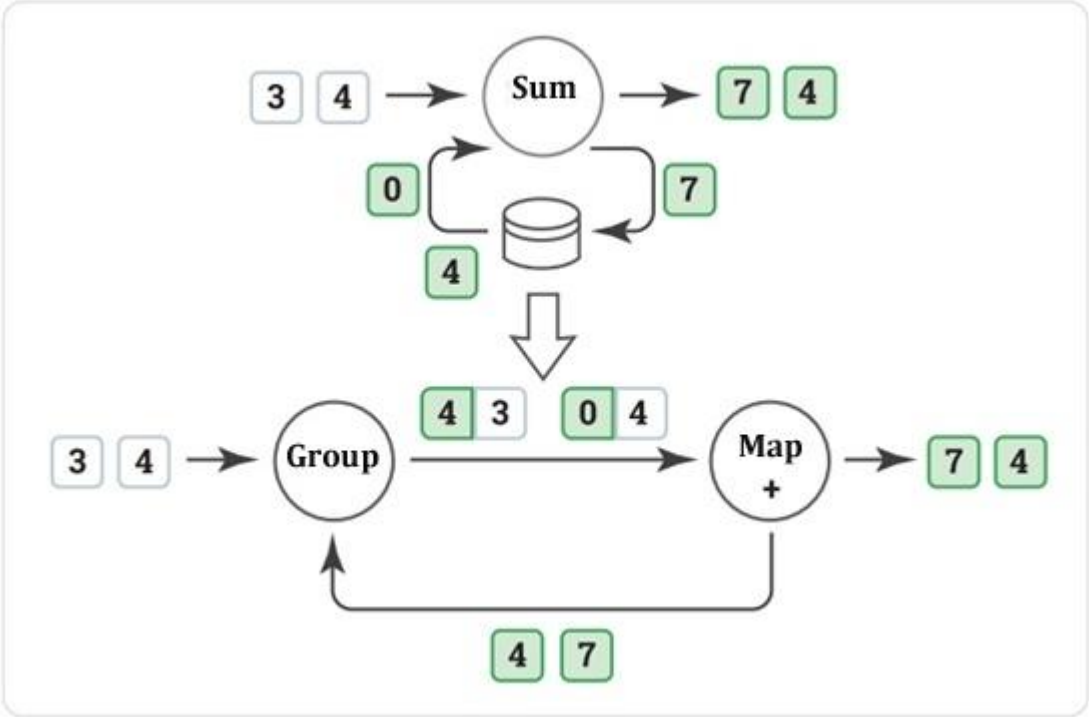
$$I_k \rightarrow \mathrm{Op}(I_k, S_k) \rightarrow J_k$$
$$\uparrow\downarrow$$
$$S_k \; S_{k+1}$$

What if we put state directly into the stream?

$$I_k, S_k \rightarrow \mathrm{Op}(I_k, S_k) \rightarrow J_k, S_{k+1}$$

Drifting state: implementation

- Any stateful transformation can be decomposed into map and windowed grouping operation with a cycle
- Map operation is stateless == order insensitive and pure
- Grouping operation is pure (and even does not contain user-define logic)

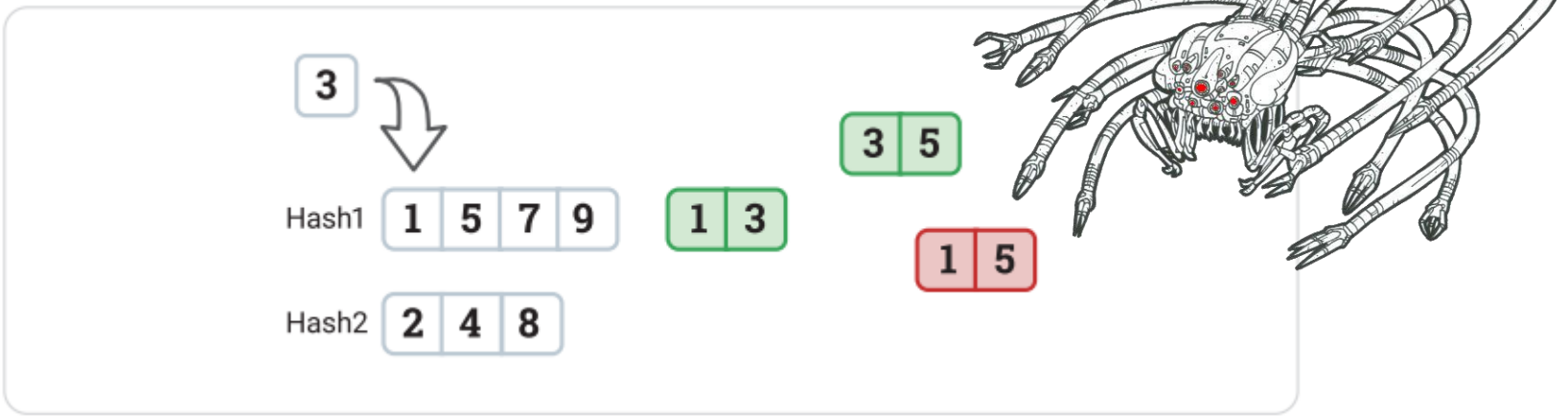


Drifting state: optimistic grouping

- Grouping operation is pure, but order-sensitive
- Buffers before each grouping can increase latency [*Li et al. Out-of-order processing: a new architecture for high-performance stream systems.* ***VLDB 2008***]
- Grouping can be implemented optimistically without blocking

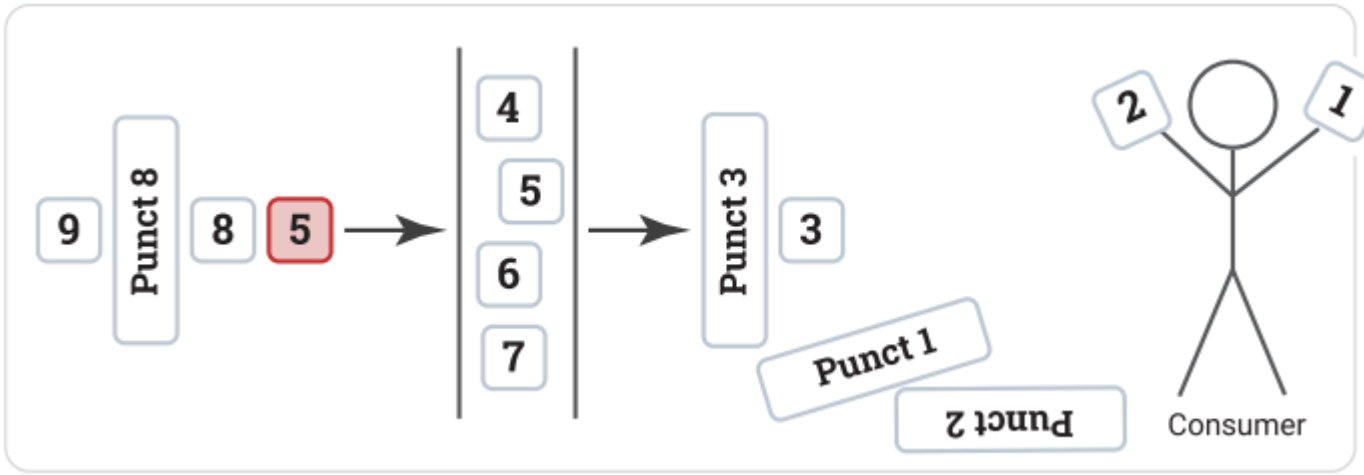
Drifting state: optimistic grouping

- Grouping operation is pure, but order-sensitive
- Buffers before each grouping can increase latency [*Li et al. Out-of-order processing: a new architecture for high-performance stream systems.* ***VLDB 2008***]
- Grouping can be implemented optimistically without blocking



Drifting state: the only buffer

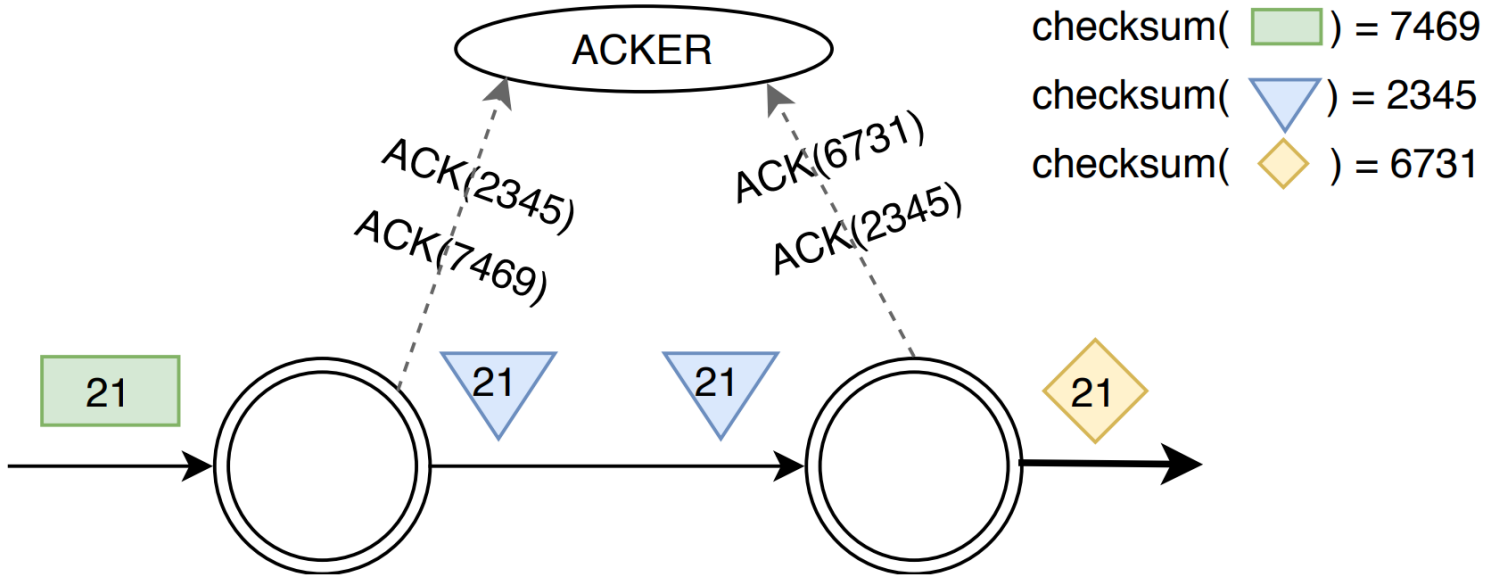
- Optimistic approach produces invalid items
- Invalid items must be filtered out before they are sent to consumer
- Punctuations (low watermarks) allow releasing items



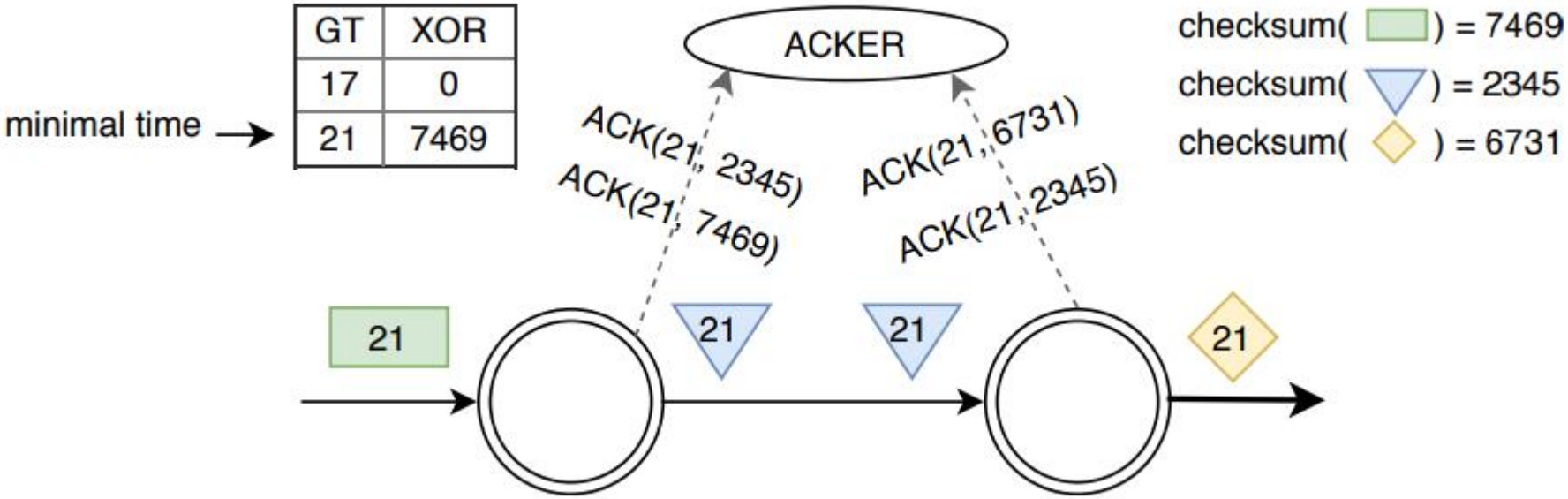
Something is wrong here

- Dataflow graphs are cyclic
- It is unclear how to send low watermarks through the cycles

Implementation notes: acker



Implementation notes: modified acker



Discussion: drifting state pros

- Determinism is closer than you think!
- We moved from “state as a special item” to “state as an ordinary item”
  - Business-logic becomes stateless
  - All guarantees that system provides regarding ordinary items are satisfied for state
  - Any stateful dataflow graph can be expressed using drifting state model
  - Transformations can be non-commutative, but should be pure
  - **Single buffer before output per dataflow graph**

JET BRAINS RESEARCH

Discussion: drifting state cons

- It is harder to write code
  - A need for a convenient API
- Optimistic technique can potentially generate a lot of additional items
- How does drifting state behaves within real-world problem?

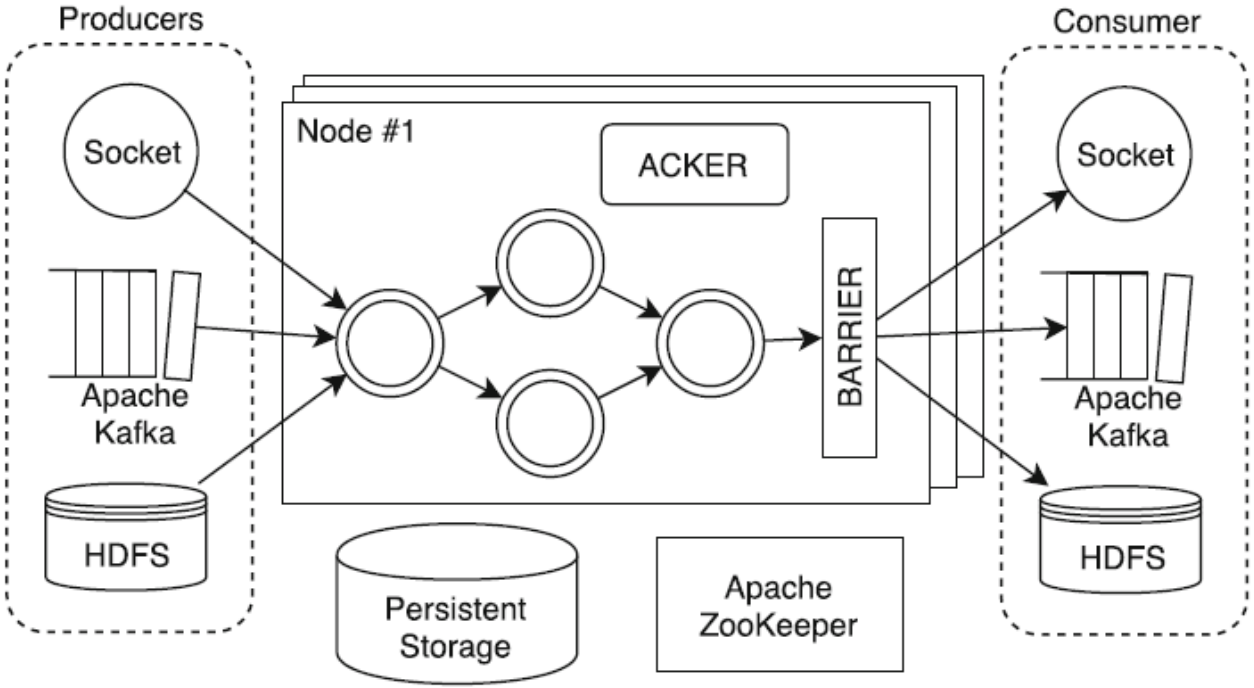
Outline

- Deterministic computations
- Stream processing computational model
- Optimistic determinism: drifting state
- **Experiments**
- Exactly-once on top of determinism
- Yet another experiment

JET BRAINS RESEARCH

Experiments: prototype

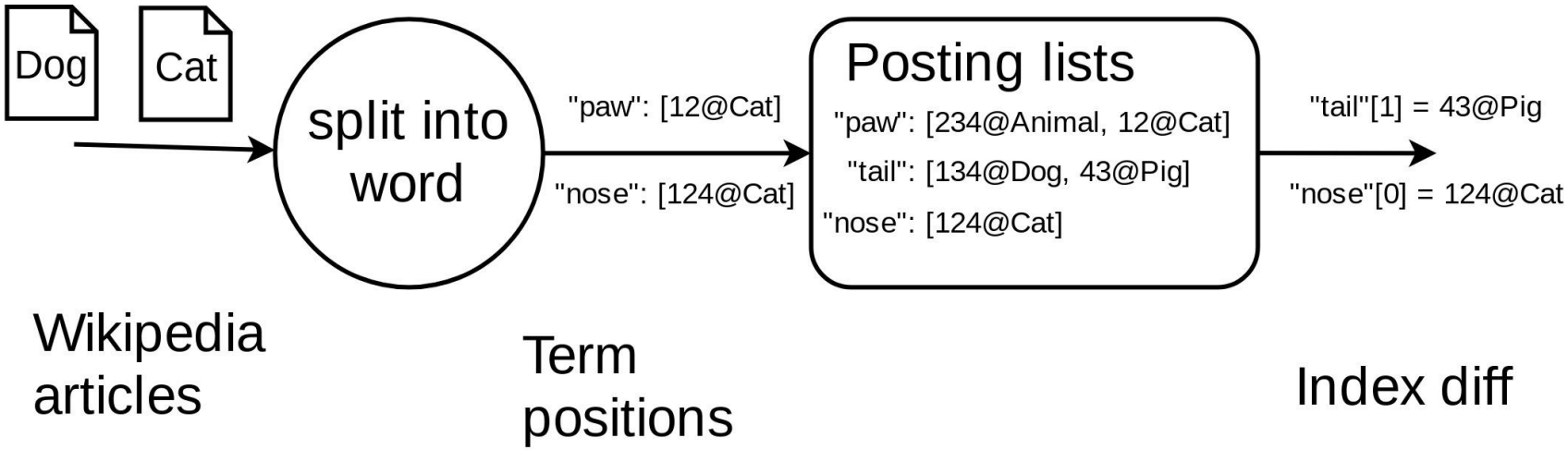
- FlameStream [https://github.com/flame-stream]
- Java + Akka, Zookeper



Experiments: task

Incremental inverted index building

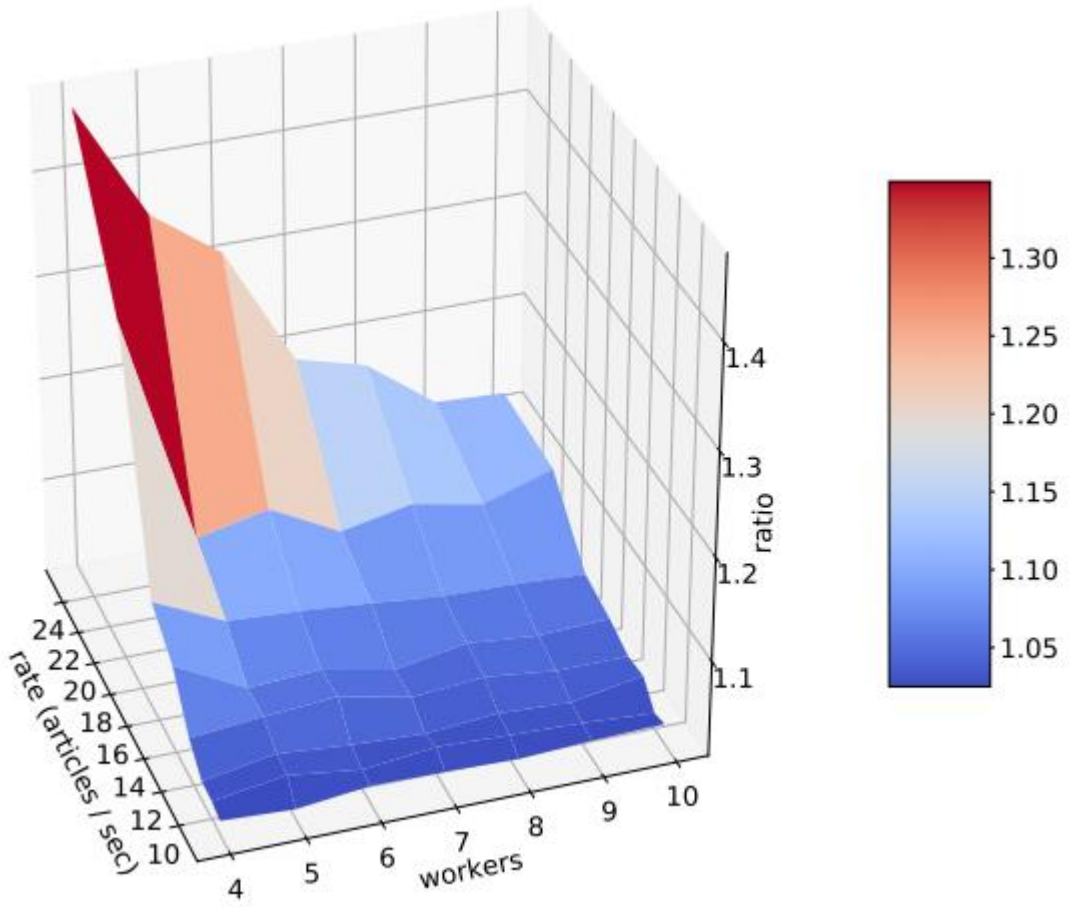
- Requires stateful operations
- Contains network shuffle
- Workload is unbalanced due to Zipf's law



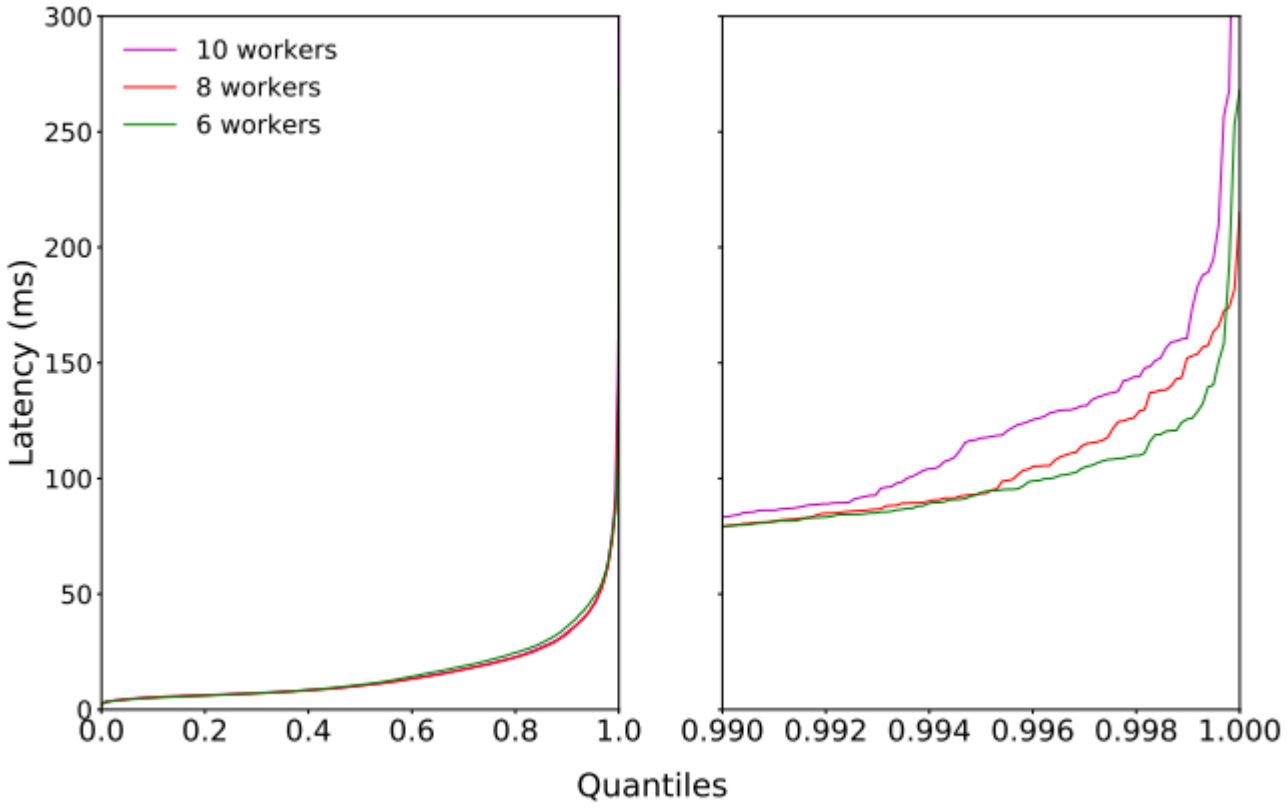
Experiments: setup

- 10 EC2 micro instances
  - 1 GB RAM
  - 1 core CPU
- Wikipedia documents as a dataset

Experiments: overhead

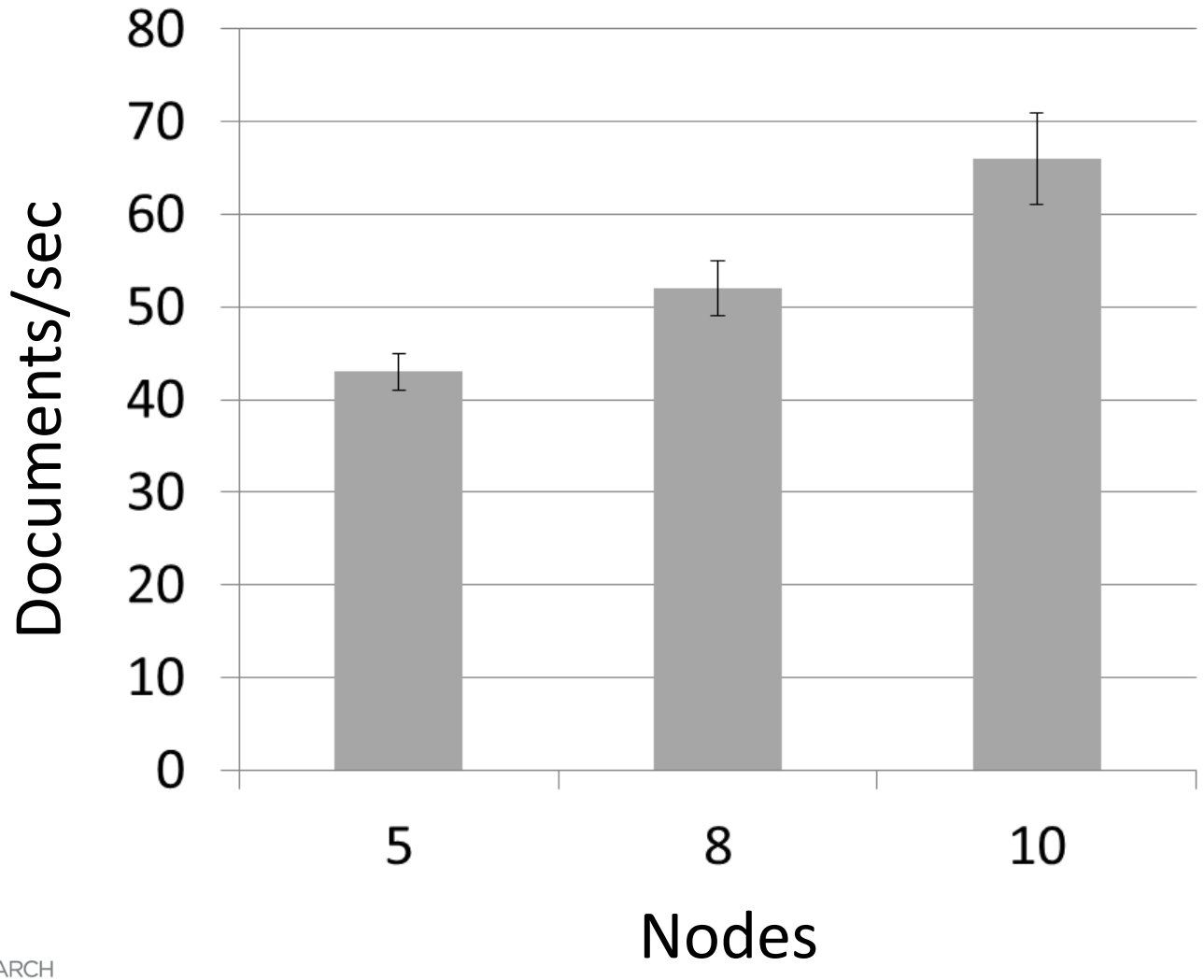


Experiments: latency scalability



JET BRAINS RESEARCH

Experiments: throughput scalability



JET BRAINS RESEARCH

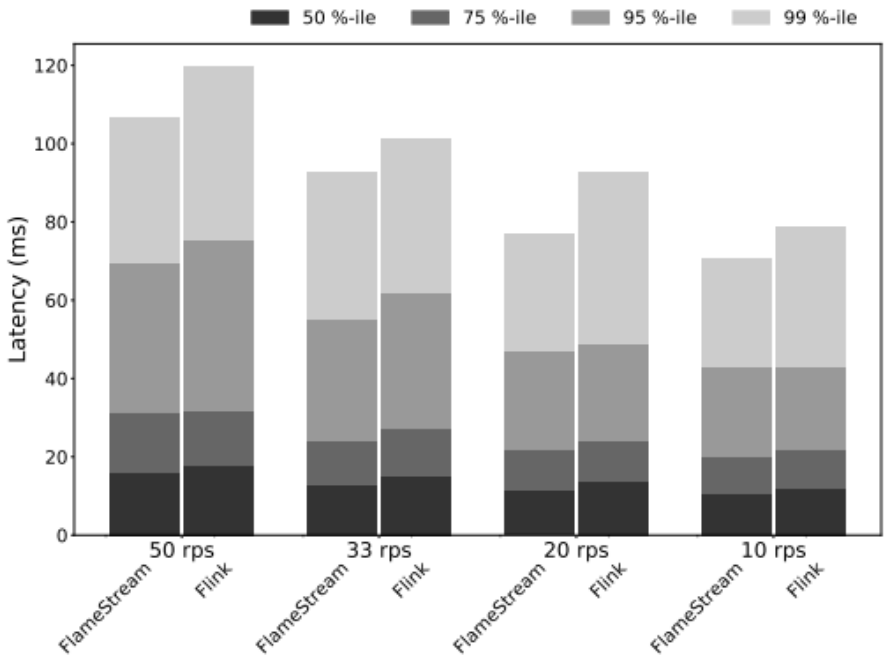
Experiments: comparison with conservative approach

- Posting lists update is order-sensitive operation
- Buffer elements before this operation
- Buffer is flushed on low watermarks
- Low watermarks are sent after each input element to minimize overhead
- [*Li et al. Out-of-order processing: a new architecture for high-performance stream systems.* ***VLDB 2008***]
- Apache Flink as stream processing system

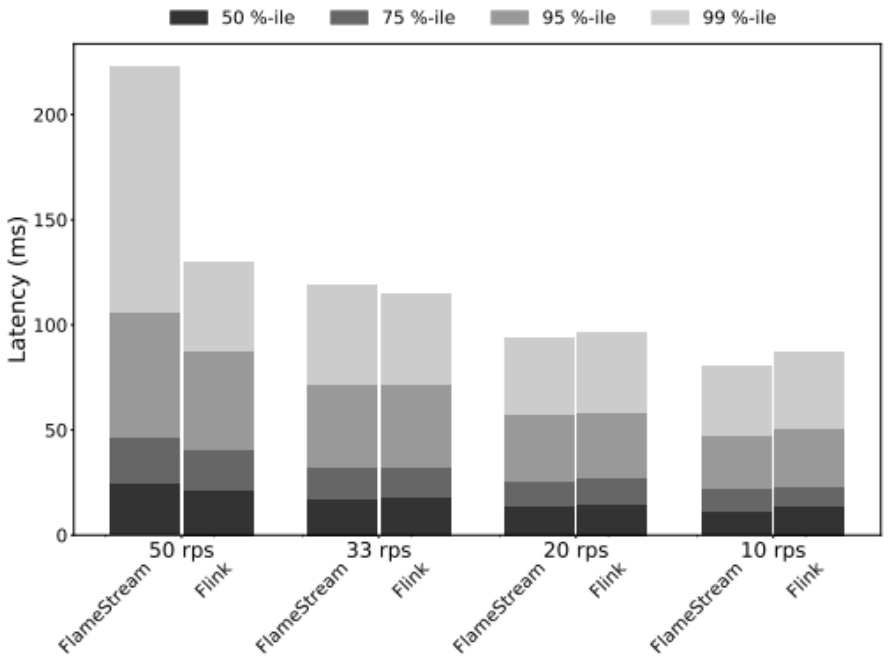
JET BRAINS RESEARCH

Experiments: comparison with conservative approach (at most once)

10 nodes

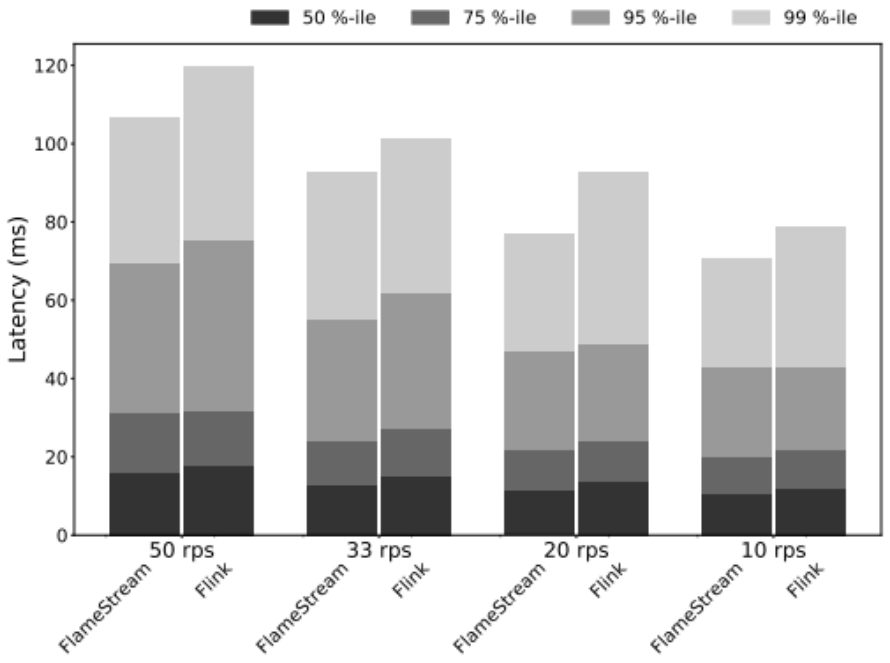


5 nodes

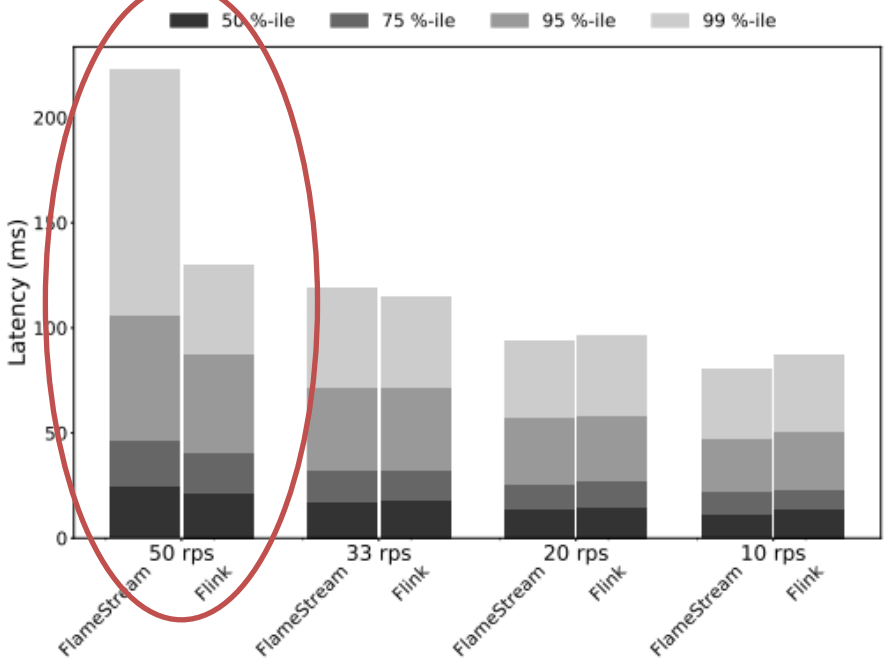


Experiments: comparison with conservative approach (at most once)

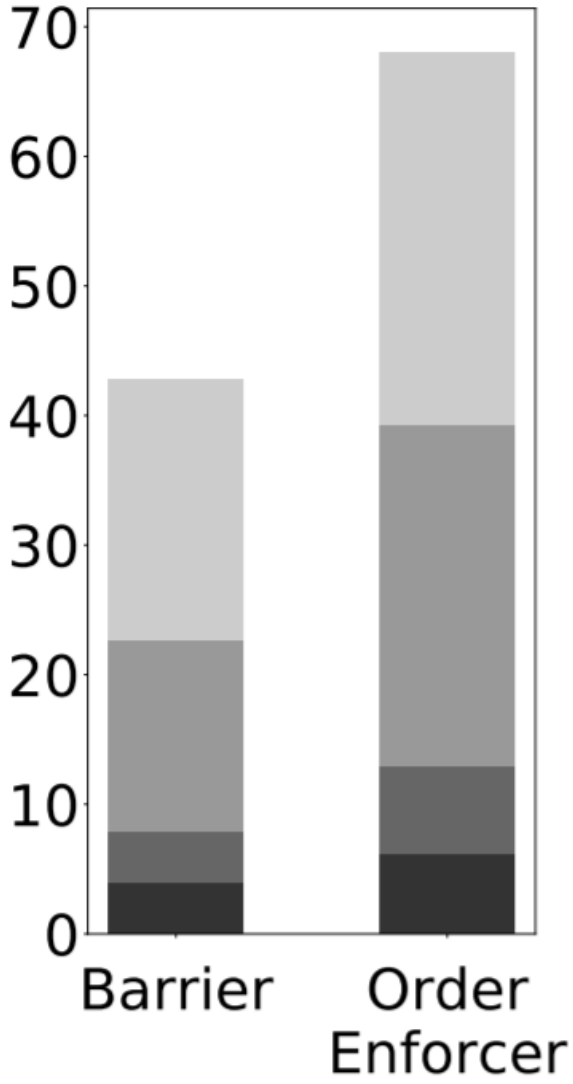
10 nodes



5 nodes

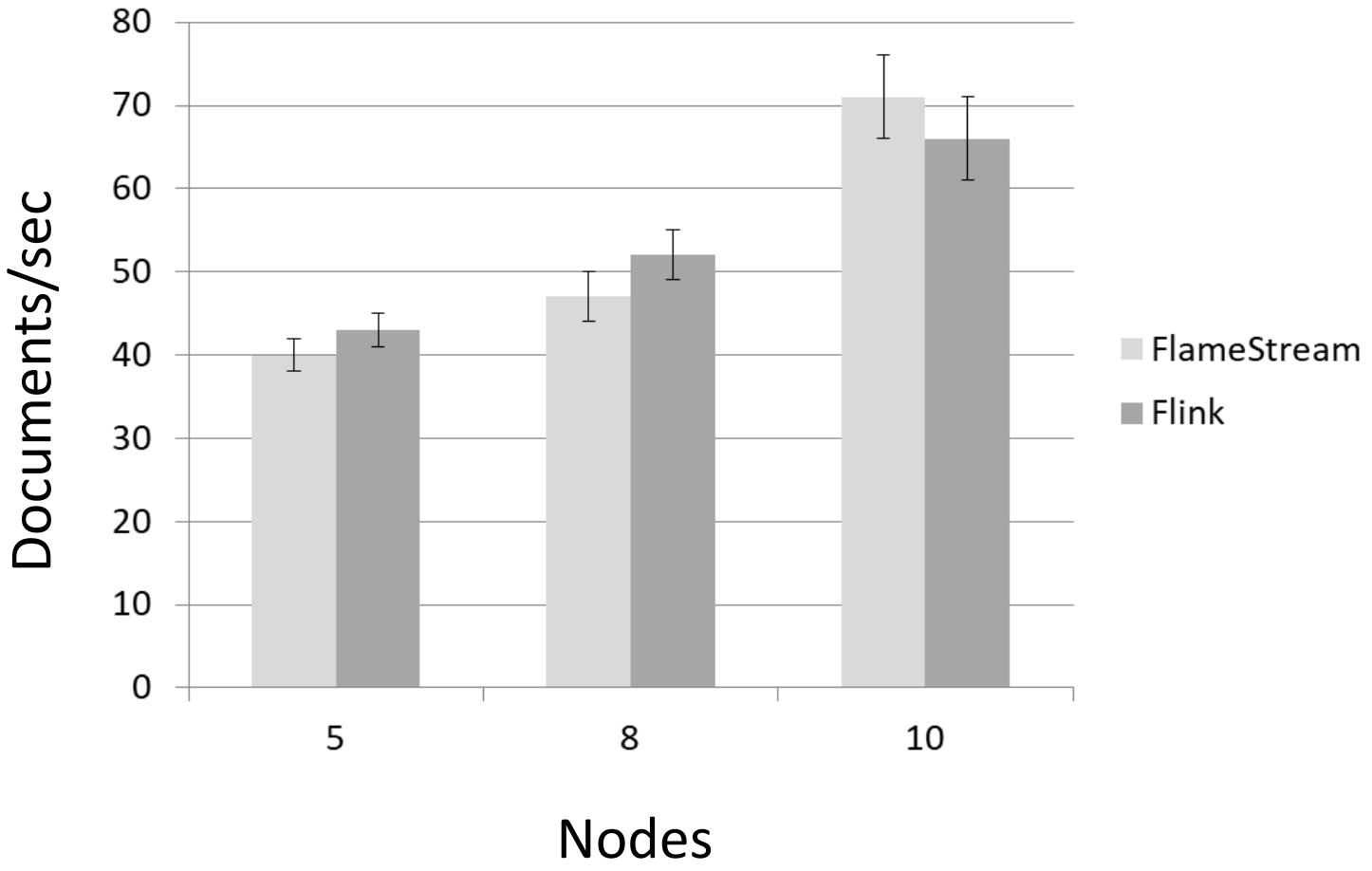


Experiments: comparison with conservative approach



JET BRAINS RESEARCH

Experiments: comparison with conservative approach



Drifting state: conclusion

- Determinism and low-latency is achieved
- Overhead is low
- Throughput is not significantly degraded
- Model is suitable for any stateful dataflows
  - If all transformations are pure

JET BRAINS RESEARCH

What if…

- Map is not pure?
  - *Determinism is lost by definition*
  - *Correctness is lost*

- Multiple input nodes?
  - Timestamp = *timestamp@node_id*
  - Latency ≥ out of sync time
  - Possible to sync in ~10ms
- Acker fails?
  - Replication
  - Separate ackers for timestamp ranges

Outline

- Deterministic computations
- Stream processing computational model
- Optimistic determinism: drifting state
- Experiments
- **Exactly-once on top of determinism**
- Yet another experiment

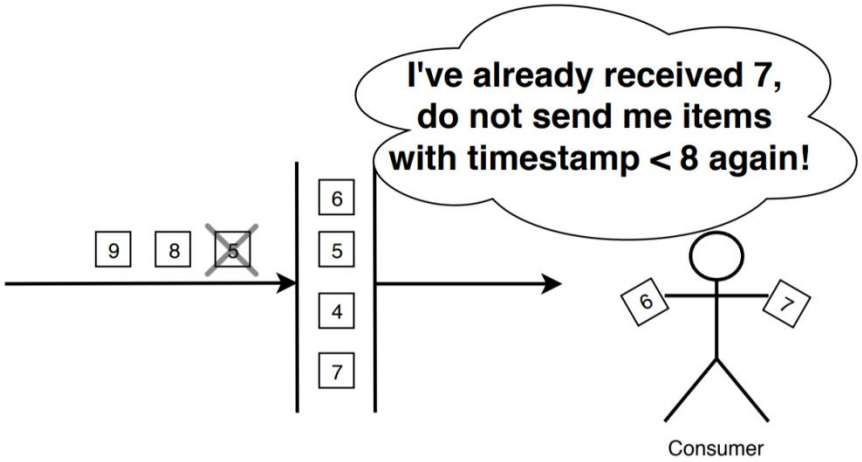
JET BRAINS RESEARCH

What do we need to add to achieve exactly-once?

- Input replay
- Restore consistent state in groupings
- Deduplicate items only at the barrier

What do we need to add to achieve exactly-once?

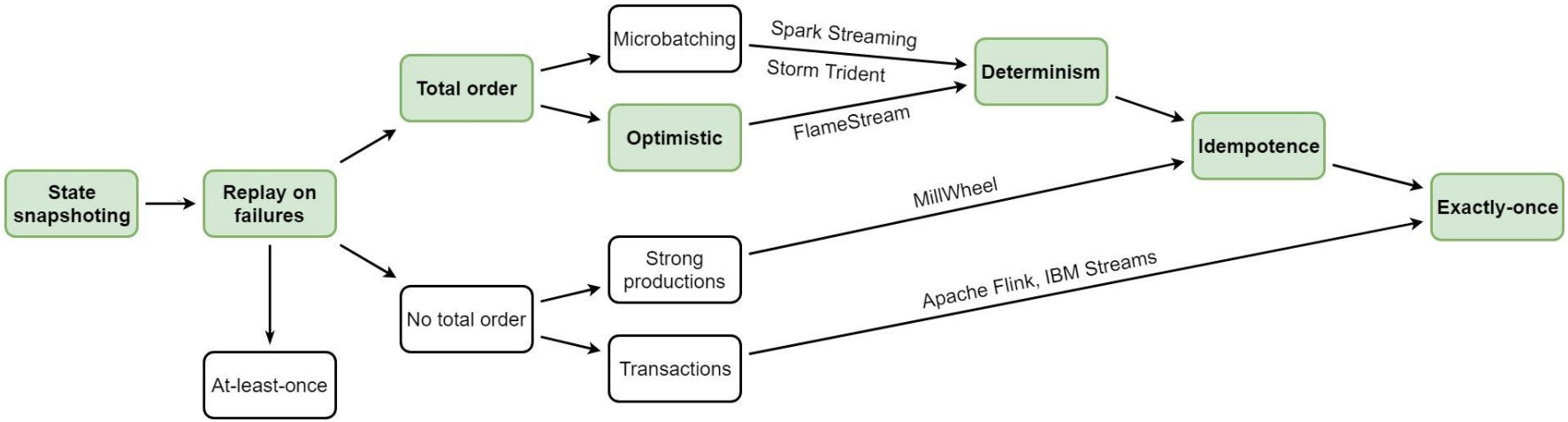
- Input replay
- Restore consistent state in groupings
- **Deduplicate items only at the barrier**
  - Output items atomically
  - Sink stores timestamp of the last received item
  - Simply compare timestamps!



Exactly-once: discussion

- Pure streaming
- Deduplication only at the barrier
- Snapshotting and outputting are independent (are not connected into transaction)

Exactly-once: roadmap

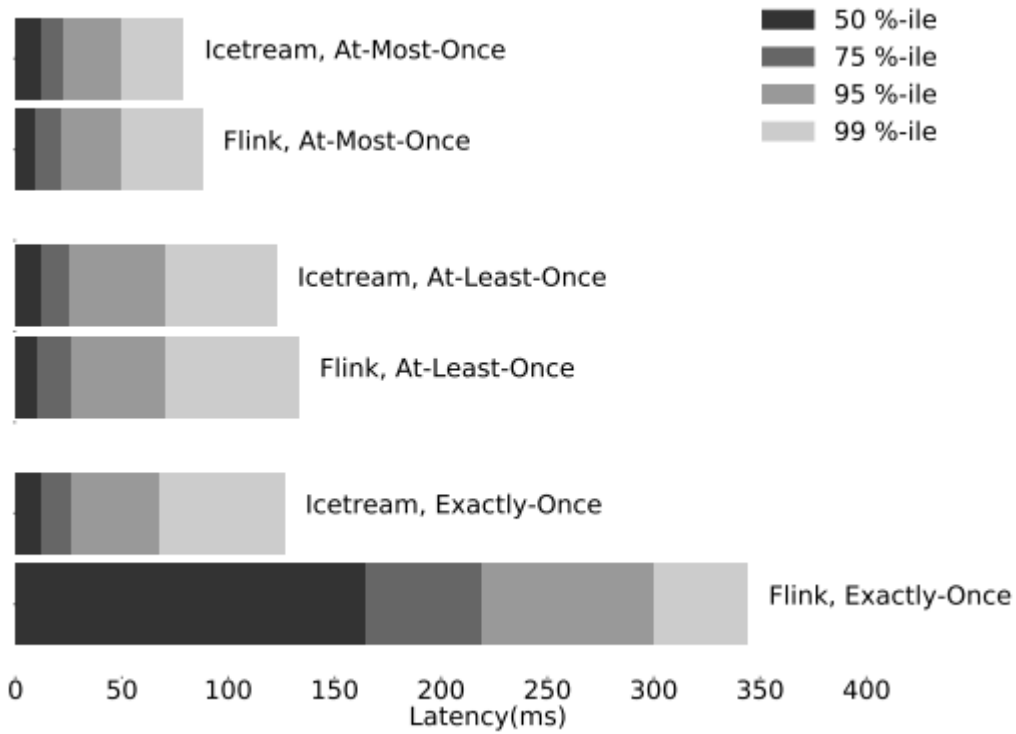


Outline

- Deterministic computations
- Stream processing computational model
- Optimistic determinism: drifting state
- Experiments
- Exactly-once on top of determinism
- **Yet another experiment**

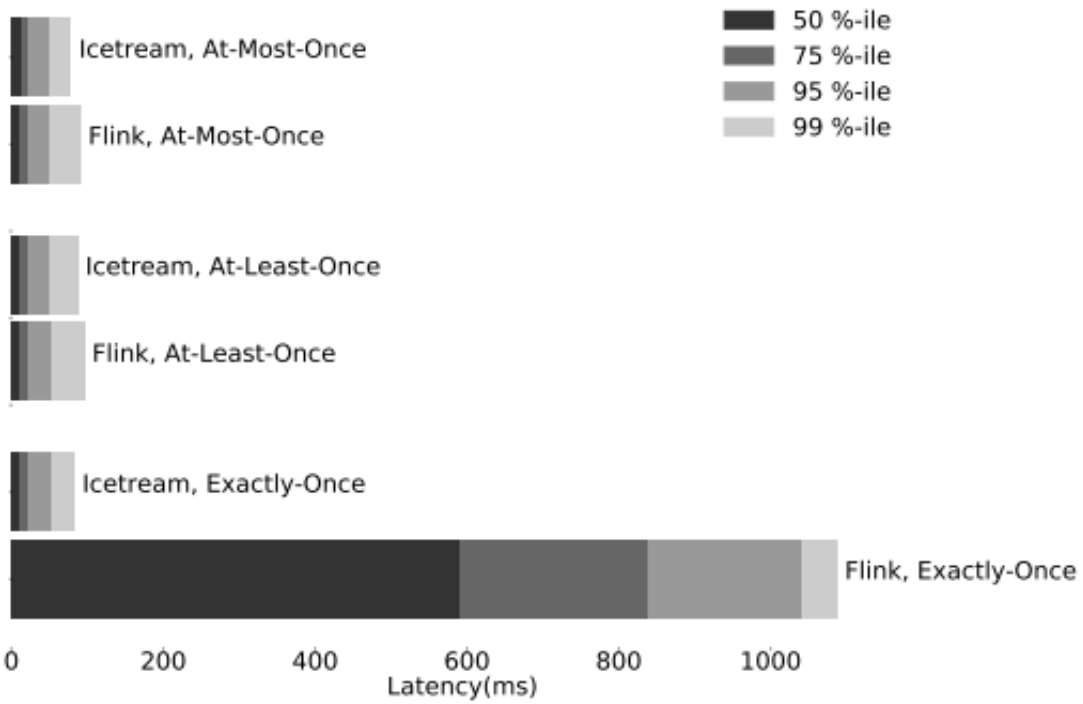
JET BRAINS RESEARCH

Experiments: latency (50 ms between checkpoints)



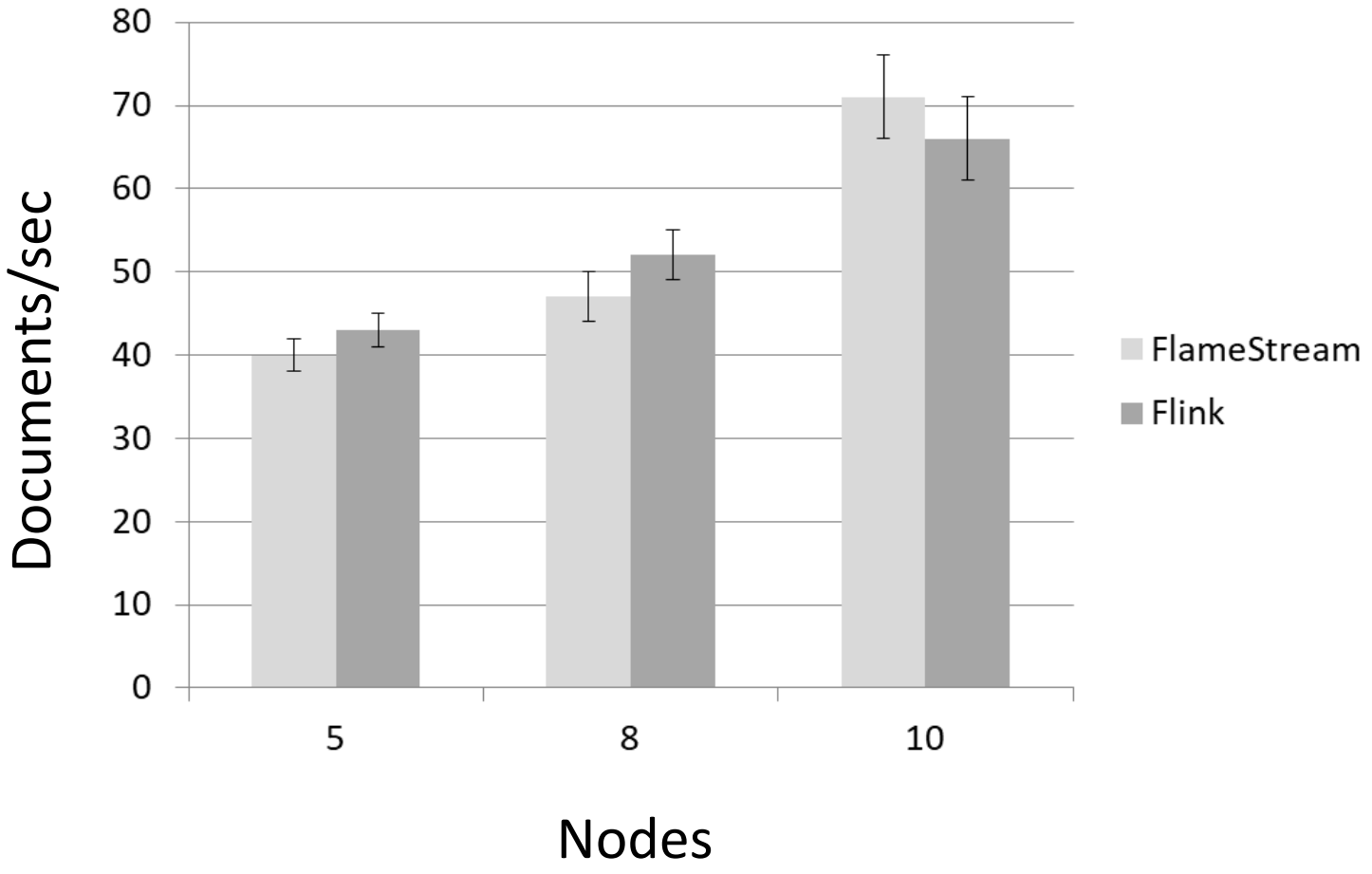
50 ms between state snapshots

Experiments: latency (1000 ms between checkpoints)



1000 ms between state snapshots

Experiments: throughput



JET BRAINS RESEARCH

Conclusions

- Single extra requirement: all transformations are pure
- Results look promising
- A lot of work
  - Understand properties and limitations
  - Real-life deployment
- We are open for collaboration

JET BRAINS RESEARCH

Future work

- Real-life deployment
- Efficient determinism and exactly-once can be used for system-level acceptance testing
  - [*Trofimov. Consistency maintenance in distributed analytical stream processing.* ***ADBIS DC 2018***]

Papers

- *Kuralenok et al. FlameStream: Model and Runtime for Distributed Stream Processing.* **BeyondMR@SIGMOD 2018**
- *Kuralenok et al. Deterministic model for distributed speculative stream processing.* **ADBIS 2018**
- Long paper about exactly-once is on the anvil

JET BRAINS RESEARCH