StreamScope: Continuous Reliable Distributed Processing of Big Data Streams

Wei Lin*, Haochuan Fan*, Zhengping Qian (ZP)*, Junwei Xu, Sen Yang, Jingren Zhou*, Lidong Zhou

Microsoft
*Now with Alibaba Group

@NSDI’16
A new transaction got reflected in the output within 3s
The system processed up to 50 million events/s
High complexity
48 stages
18 joins of 5 different types
21.3 TB in-memory state

Massive scalability
Reads 61TB + Write 61TB
7 billions of input events
6 billions of output events
3000+ long-running tasks

Fault tolerance
Handles both planned failures and unplanned outages automatically
Streaming dataflow

Input streams

Streaming events

Channels

Vertices

Output streams
Streaming dataflow

Input streams

Vertices

Output streams

R

M

X

Replay of upstream events

Rebuild the state

Missing or duplicate events
Decoupling

rStream Provides the illusion of reliable and asynchronous communication channels


**rStream**

- **Properties**
  - There is a unique value associated with each sequence number
  - A read returns only after a successful write, for the same seq
  - If a write of \((seq, e)\) succeeds, then for the following reads that reach position \(seq\), they eventually return \((seq, e)\)
Execution of a vertex

**Snapshot**
- Sequence numbers of its input streams
- Sequence numbers of its output streams
- Computation state

$s_1 = \langle \{2\}, \{1\}, t_1 \rangle$
rVertex

Timeline

Restart from a snapshot

\[
s_1 = \langle 2, \{1\}, t_1 \rangle \\
\]

\[
s_2 = \langle 3, \{2\}, t_2 \rangle \\
\]

\[
s_3 = \langle 4, \{4\}, t_3 \rangle \\
\]
Failure recovery

\[ s_1 = \langle 2, \{1\}, t_1 \rangle \]
\[ s_2 = \langle 3, \{2\}, t_2 \rangle \]
\[ s_3 = \langle 4, \{4\}, t_3 \rangle \]
Optimization

• Naïve implementation of rStream: writing events to reliable store
  • Synchronous writes introduce significant latencies

• Uses a hybrid scheme that moves writes out of the critical path while providing the illusion of reliable channels

1. Buffered in memory
2. Asynchronously flushed to reliable store
3. If lost, recomputed when requested
Different failure recovery strategies

- Recomputation using dependency tracking at runtime
- Checkpoint/log replay
- Persistent state/streams
- Hybrid
Development/debugging

• Greatly leveraged and tightly integrated with existing system
  • Integrated language, optimizer, scheduling, etc.

• Distributed streaming made easy
  • Off-line mode: starting with finite inputs with minimum resources to validate/debug a streaming application
  • Later switched to on-line, live execution transparently
  • Greatly improves developer productivity in lifecycle of an application
    • E.g., Can even debug/profile a vertex without impacting the running job
Deployment

• Re-examination of segments of execution in the past for auditing

• Dynamic scaling and robustness to load fluctuation

• Continuous operation during system maintenance

• Straggler handling

• Dynamic reconfiguration/patching to resolve data anomalies
Conclusion

• Cloud-scale stream computation is challenging due to the complexity of dependencies

• StreamScope introduces two new abstractions, rVertex and rStream, to manage the complexity through decoupling

• The abstractions separate system properties from the actual implementation to,
  • Enable powerful optimizations
  • Develop different failure recovery strategies
  • Better support the lifecycle of streaming applications in production