Portable stateful big data processing in Apache Beam

Aljoscha Krettek/Kenneth Knowles
Apache Beam/Flink PMC
Software Engineer @ dataArtisans
aljoscha@apache.org / @aljoscha

https://s.apache.org/stateful-beam-strata-london-2017

Strata London 2017
What I’d like to talk about

1. Why stateful stream processing?
2. What is Apache Beam?
3. State
4. Timers
Why stateful stream processing?
What is stream processing?

Computations on never-ending “streams” of data records (“events”)
Distributed stream processing

Computation spread across many machines
Stateful stream processing

State is usually “scoped” to some key in the data: event color
Timely, stateful stream processing

*Timers are a special kind of state
Use cases that require stateful stream processing

- Operations and manufacturing
- Mobile gaming
- Wearables
- Automotive
- Solar power/power grid
- Network monitoring
- (Mobile) banking
Common Requirements

Stateful processing

Event-time processing*

Handling of out-of-order events*

*check out watermarks
Example: mobile gaming

First event comes in
  -> put into state AND register session timeout timer

Process events as they come and update state

Timer fires
  -> compute result for session and delete state
What is Apache Beam?
DAGs, DAGs, DAGs

- Apache Hadoop
- MapReduce (paper)
- Apache Spark
- Apache Flink
- Apache Storm
- FlumeJava (paper)
- MillWheel (paper)
- Apache Samza
- Apache Apex
- Apache Gearpump (incubating)
- Heron
- Cloud Dataflow
- Apache Beam
- Dataflow Model (paper)

Years:
- 2004
- 2005
- 2006
- 2007
- 2008
- 2009
- 2010
- 2011
- 2012
- 2013
- 2014
- 2015
- 2016
The Beam Model

(Flink draws it more like this)
The Beam Model
input.apply(
    Sum.integersPerKey())

Python

input | Sum.PerKey()
Python

```python
input | KafkaIO.read()
```

Java

```java
class KafkaIO extends UnboundedSource { ... }
```
The Beam Model

**What** are you computing? (read, map, reduce)

**Where** in event time? (event time windowing)

**When** in processing time are results produced? (triggers)

**How** do refinements relate? (accumulation mode)

The focus of today
What are you computing?

Read
Parallel connectors to external systems

ParDo
Per element "Map"

Grouping
Group by key, Combine per key, "Reduce"

Composite
Encapsulated subgraph
State
What are you computing?

Read
Parallel connectors to external systems

ParDo
Per element "Map"

Grouping
Group by key, Combine per key, "Reduce"

Composite
Encapsulated subgraph
Stateful ParDo

ParDo(DoFn)
Stateful ParDo "Example"

```java
ParDo.of(new DoFn<...>(){
    // declare some state
    @StateId("quantiles")
    private final StateSpec<...>
        quantilesState = StateSpecs.combining();

    @ProcessElement
    public void process(..., @StateId("quantiles") state) {
        // update quantiles
        state.add(...)
        // output if needed
    }
});
```
Partitioning for efficient parallel execution

Quantiles

Quantiles

Quantiles
new DoFn<foo, Quantiles<foo>>() {
    @StateId("quantiles")
    private final StateSpec<...> quantilesState = StateSpecs.combining();
    ...
}
Windowed State

Window into Fixed windows of one hour

Expected result: Quantiles for each hour
Windowed State

Window into windows of 30 min sliding by 10 min

Expected result: Quantiles for 30 minutes sliding by 10 min
State is per key and window

Bonus: automatically garbage collected when a window expires (vs manual clearing of keyed state)

<table>
<thead>
<tr>
<th></th>
<th>( &lt;k, w&gt;_1 )</th>
<th>( &lt;k, w&gt;_2 )</th>
<th>( &lt;k, w&gt;_3 )</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;x&quot;</td>
<td>3</td>
<td>7</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>&quot;y&quot;</td>
<td>&quot;fizz&quot;</td>
<td>&quot;7&quot;</td>
<td>&quot;fizzbuzz&quot;</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Kinds of state

- **Value** - just a mutable cell for a value
- **Bag** - supports "blind writes"
- **Combining** - has a CombineFn built in; can support blind writes and lazy accumulation
- **Set** - membership checking
- **Map** - lookups and partial writes
Timers
new DoFn<...>() {
    @TimerId("timeout")
    private final TimerSpec timeoutTimer = TimerSpecs.timer(TimeDomain.PROCESSING_TIME);

    @OnTimer("timeout")
    public void timeout(...) {
        // access state, set new timers, output
    }
    ...
}
Timers in processing time

"call me back in 5 seconds"

On Element

On Timer

output based only on incoming element

buffer request

batched RPC

output return value of batched RPC
Timers in event time

"call me back when the watermark hits the end of the window"

output speculative result immediately

output replacement value only if changed

store speculative result

On Element

On Timer
More example uses for state & timers

- Per-key arbitrary numbering
- Output only when result changes
- Tighter "side input" management for slowly changing dimension
- Fine-grained combine aggregation and output control
- Per-key "workflows" like user sign up flow w/ expiration
- Low-latency deduplication (let the first through, squash the rest)
Performance considerations (cross-runner)

- Shuffle to collocate keys
- Linear processing of elements for key+window
- Window merging
- Storage of state and timers
- GC of state
**State and Timers in Beam...**

- ... unlock new uses cases
- ... they "just work" with event time windowing
- ... are portable across runners (implementation ongoing)
Thank you for listening!

This talk:
- Me - @aljoscha / aljoscha@apache.org / @KennKnowles / kenn@apache.org
- These Slides - https://s.apache.org/stateful-beam-strata-london-2017

Go Deeper
- Design doc - https://s.apache.org/beam-state
- Blog post - https://beam.apache.org/blog/2017/02/13/stateful-processing.html

Join the Beam community:
- User discussions - user@beam.apache.org
- Development discussions - dev@beam.apache.org
- Follow @ApacheBeam on Twitter

https://beam.apache.org
Backup Slides
What about Combine?

*Window into Fixed windows of one hour*

Quantiles Combiner

Expected result: Quantiles for each hour
Combine vs State (naive conceptualization)

Expected result: Quantiles for each hour
**Combine vs State (likely execution plan)**

- **Quantiles Combiner**
- **Quantiles Combiner**
- **Shuffle Accumulators**
- **Quantiles Combiner**
- **Shuffle Elements**
- **Quantiles DoFn**

**Associative**
- commutative
- single-output
- enables optimizations (engine is in control)

**Non-associative**
- non-commutative
- multi-output
- side outputs (user is in control)

**DoFn**
Combine vs State

output governed by trigger
(data/computation unaware)

"output only when there's an interesting change"
(data/computation aware)
Example: Arbitrary indices

Assign indices

non-associative
non-commutative
non-deterministic
and totally fine!