Revisiting Splittable DoFn in KafkaIO

BEAM SUMMIT NYC 2025

Steven van Rossum
Software Engineer, Google Cloud (Consulting)

KafkaUnboundedSource

- Fixed parallelism
  - May decrease when splits end
- Source matches partitions
  - One or more partitions per split
  - Evaluated during construction
- Polls a consumer on a background thread
- Offsets can be committed in checkpoint finalization

ReadFromKafkaDoFn

- Dynamic parallelism
- Source matches partitions
  - One partition per split
  - Evaluated continuously
- Polls a consumer on the processing thread
- Offsets can be committed in a downstream step

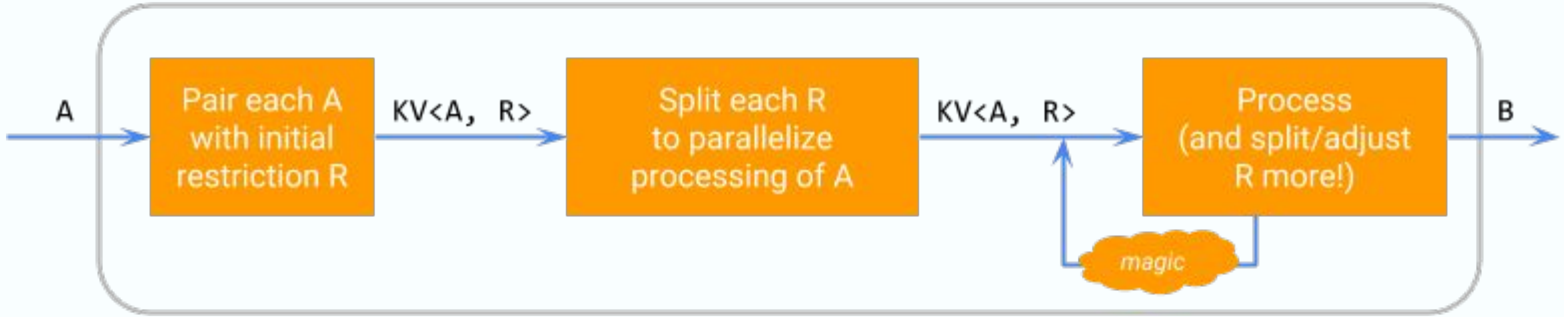
UnboundedReader

```
interface UnboundedReader<OutputT> {
  OutputT getCurrent();
  Instant getCurrentTimestamp();
  boolean advance();
  UnboundedSource.CheckpointMark getCheckpointMark();
  byte[] getCurrentRecordId();
  byte[] getCurrentRecordOffset();
  UnboundedSource<OutputT,?> getCurrentSource();
  long getSplitBacklogBytes();
  long getTotalBacklogBytes();
  Instant getWatermark();
  boolean start();
}
```

Splittable DoFn

```
interface SplittableDoFn<SourceT, OutputT, ...> {
  RestrictionT getInitialRestriction(SourceT);
  TrackerT newTracker(SourceT, RestrictionT);
  void processElement(SourceT, OutputReceiver<OutputT>);
  double getSize(RestrictionT, TrackerT);
  void splitRestriction(RestrictionT, TrackerT);
  TruncateResult<> truncateRestriction(RestrictionT, TrackerT);
  WatermarkEstimatorStateT getInitialWatermarkEstimatorState();
  WatermarkEstimatorT newWatermarkEstimator();
}
```

Splittable DoFn


A
Pair each A with initial restriction R
KV<A, R>
Split each R to parallelize processing of A
KV<A, R>
Process (and split/adjust R more!)
B
magic

KafkaIO on different Dataflow runners

- Frequently reported issues with KafkaIO on Runner V2
    - Low throughput
        - 4-100x variance
    - Resource hungry
        - >100 Kafka client connections per second
            - Schema registries become unreachable
        - Memory running low
        - High CPU utilization

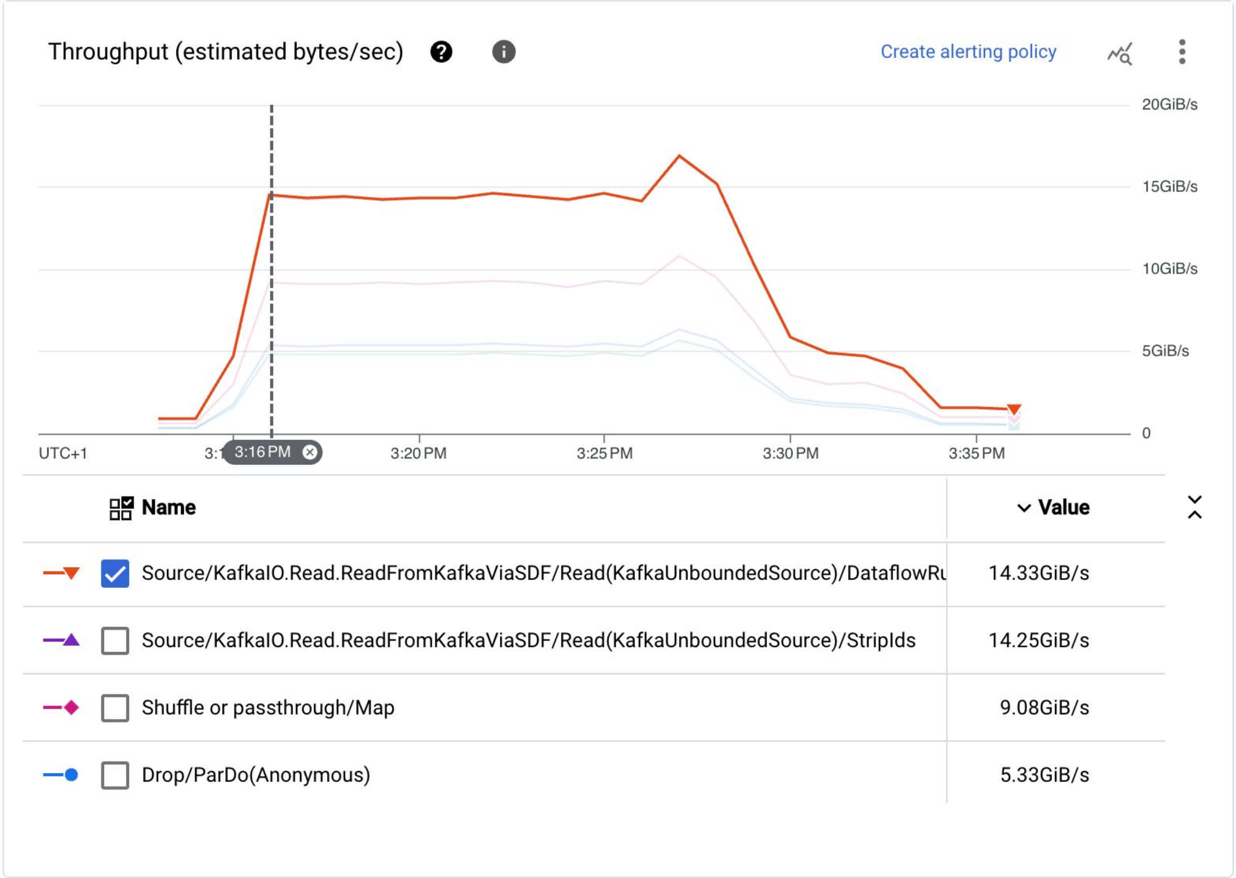
KafkaIO on different Dataflow runners

- >100 connections per second?
  - New Kafka client per call to processElement
  - Caching backlog estimators for all assignments
- Easy fix
  - Cache kafka clients per DoFn

KafkaIO on different Dataflow runners

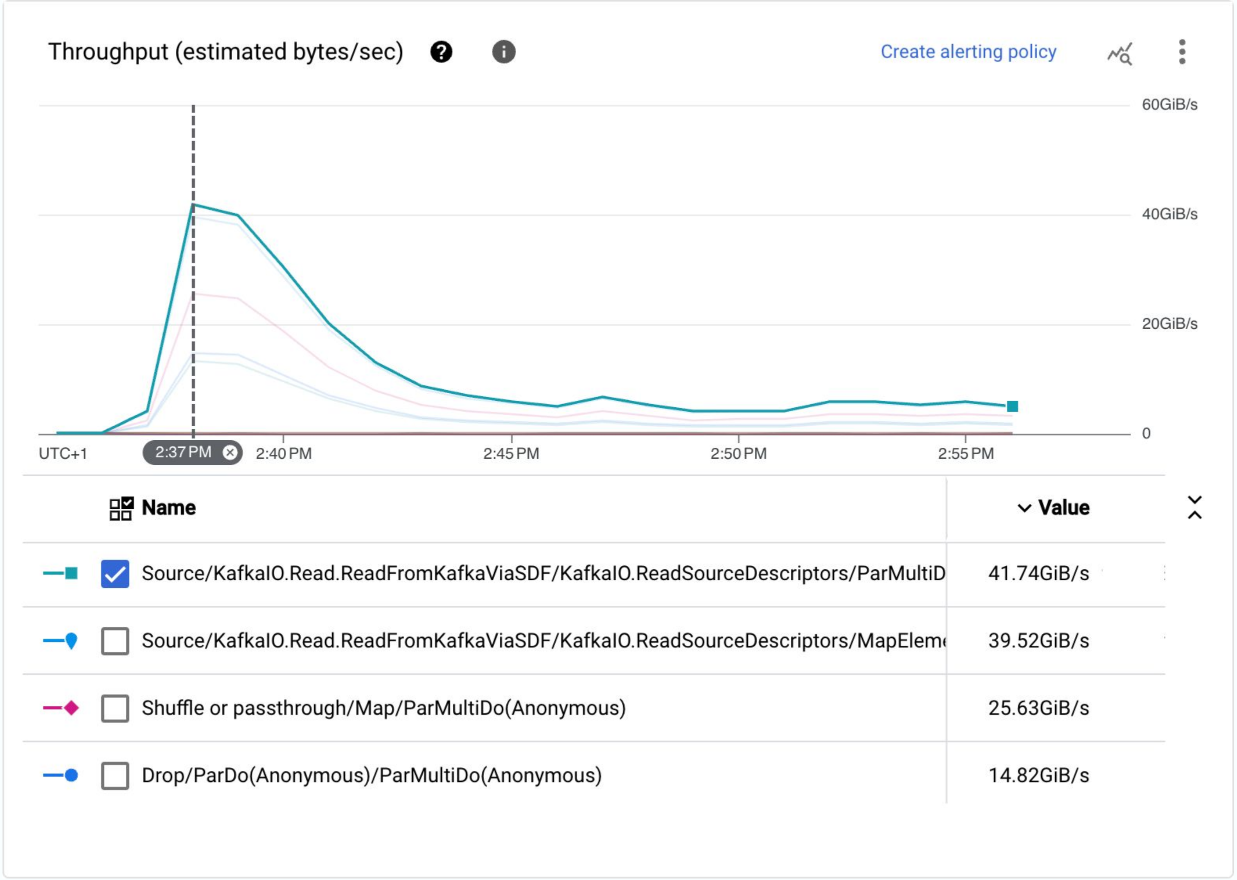
- DoFn instance fields are not shared
  - Every DoFn instance (execution) creates their own cache
- No easy fix
  - Static fields are shared among all instances (construction, execution)
  - Different instances of the same DoFn should have isolated caches
- Solution
  - `MemoizingPerInstantiationSerializableSupplier`
    - Access through group scope assigned at construction

KafkaIO on Dataflow Runner V1



		Name	Value
▼	✓	Source/KafkaIO.Read.ReadFromKafkaViaSDF/Read(KafkaUnboundedSource)/DataflowRu	14.33GiB/s
▲	☐	Source/KafkaIO.Read.ReadFromKafkaViaSDF/Read(KafkaUnboundedSource)/StripIds	14.25GiB/s
◆	☐	Shuffle or passthrough/Map	9.08GiB/s
●	☐	Drop/ParDo(Anonymous)	5.33GiB/s

KafkaIO on Dataflow Runner V2



		Name	Value
■	✓	Source/KafkaIO.Read.ReadFromKafkaViaSDF/KafkaIO.ReadSourceDescriptors/ParMultiD	41.74GiB/s
●	☐	Source/KafkaIO.Read.ReadFromKafkaViaSDF/KafkaIO.ReadSourceDescriptors/MapElem	39.52GiB/s
◆	☐	Shuffle or passthrough/Map/ParMultiDo(Anonymous)	25.63GiB/s
●	☐	Drop/ParDo(Anonymous)/ParMultiDo(Anonymous)	14.82GiB/s

But wait, there's more

- First attempts focused on reusing Kafka clients for multiple active splits
  - ExecutorService
    - Submit blocks of operations per processing thread
    - Splits queue up polls that could have been combined
  - Phaser
    - Join and await arrival at the next phase
    - Once all current parties arrive one of the parties calls poll
    - All parties consume results and leave to emit elements without blocking other parties
    - A split's partition is paused/resumed on join/leave to prevent a call to poll from advancing if the split may end before rejoining

Keeping it simple in 2.65.0

- Cache Kafka clients per partition as weak references
  - Eviction is triggered by GC
- Lazily submit backlog estimator refreshes in the background
  - Carefully order atomic operations (release/acquire)
    - Non-volatile atomic writes to 64-bit primitives require Java 11
- Remove offset gap adjustments
  - Slow readers on a partition’s tail fall behind and report low backlog
- Update tracker and watermark for non-visible progress
  - Poll may return no records while advancing client’s position

Looking ahead

- Store the group scoped cache instead of retrieving it
  - Removes unnecessary overhead while processing elements
- Use unsigned integer to floating point conversions
  - BigDecimal comes with noticeable overhead
- Run metric and internal state updates before emitting elements
  - Emitting elements runs the remainder of a fused stage
- Resolve continuous growth of reported data lag
  - Profiling and debug capture show threads stuck in `Selector.wait`
- Tracing and metrics
  - Work in progress

IO test harness

- Accelerate testing
  - Throughput
  - Concurrency bottlenecks
  - Step/stage lag
  - Source system issues

Steven van Rossum

QUESTIONS?

BEAM SUMMIT NYC 2025