Dealing with order in streams using Apache Beam

Israel Herraiz
Getting started with Apache Beam Quest

Svetak Sundhar
Too Big to Fail - A Pattern for Enriching a Stream using State and Timers

By Tobi Kaymak & Israel Herraiz
Enrich me, if you can - A Pattern for Enriching a Stream using State and Timers

By Tobi Kaymak & Israel Herraiz
The Problem
Two Streams Need to be Joined
The “Core” one with the core info

```
{
  "id": 123,
  "color": "gold",
  "can_dance": true
}
```
The Second one with “Lookups”

```json
{
   "id": 123,
   "serial_number": 456
}
```
Two Streams Need to be Joined

{ "id": 123,  "color": "gold",  "can_dance": true }

{  "id": 123,  "current_serial": 456 }
Enriching Streaming Data
Enriching Streaming Data

(Slowly) updating side inputs
Enriching Streaming Data
(Batched) RPC calls
Is there another way?
State & Timers
Implementation details
Message Queues

- Kafka
- Google Pub/Sub
1. Preload the Lookup Topic

(Shell) Script

BigQuery

Pub/Sub Topic Lookups
2. Start the Beam Pipeline

(Shell) Script

BigQuery

Core

Lookups
The Beam Pipeline
The Beam Pipeline
The StatefulDoFn

@ProcessElement
@OnTimer

The input needs to be a PCollection of KV

Callback when the timer has expired

Core State

Lookup State

buffer

count

buffer

count

Timer (30s)

GC-Timer (30d)
The StatefulDoFn (2)

class StatefulJoinFn(beam.DoFn):
    BUFFER_TIMER = TimerSpec('expiry', TimeDomain.WATERMARK)
    GC_TIMER = TimerSpec('gc_timer', TimeDomain.WATERMARK)

    CORE_BUFFER_BAG = BagStateSpec('core', coders.registry.get_coder(CoreType))
    CORE_COUNT_STATE = CombiningValueStateSpec('count_core', combine_fn=sum)
    LOOKUP_BUFFER_BAG = BagStateSpec('lookup', coders.registry.get_coder(LookupType))
    LOOKUP_COUNT_STATE = CombiningValueStateSpec('count_lookup', combine_fn=sum)

    def __init__(self):
        self.time_seconds = 30

    def process(
        self,
        input_element: Union[Tuple[str, CoreType], Tuple[str, LookupType]],
        element_timestamp=beam.DoFn.TimestampParam,
        core_count_state=beam.DoFn.StateParam(CORE_COUNT_STATE),
        core_state=beam.DoFn.StateParam(CORE_BUFFER_BAG),
        lookup_count_state=beam.DoFn.StateParam(LOOKUP_COUNT_STATE),
        lookup_state=beam.DoFn.StateParam(LOOKUP_BUFFER_BAG),
        timer=beam.DoFn.TimerParam(BUFFER_TIMER),
        gc_timer=beam.DoFn.TimerParam(GC_TIMER),
    ): [...]
Don’t miss out!

**Talk:** “Design considerations to operate a stateful streaming pipeline as a service” on Wednesday from 12:30-12:55 in Palisades with Bhupinder and Israel!

**Workshop:** “Complex Event Processing With State & Timers” on Thursday from 10:45-12:15 in Palisades with Miren and Israel!
Thank you ❤️
References

- Prathap Reddy – Cache reuse across DoFn’s in Beam:  
  https://medium.com/google-cloud/cache-reuse-across-dofns-in-beam-a34a926db848
- Chirag Shankar – Stateful Processing In Apache Beam/Cloud Dataflow:  
  https://medium.com/google-cloud/stateful-processing-in-apache-beam-cloud-dataflow-109d1880f76a
- Iñigo San Jose – Dataflow Cookbook:  
  https://cloud.google.com/blog/products/data-analytics/introducing-dataflow-cookbook
- Kenneth Knowles – Timely (and Stateful) Processing with Apache Beam:  
  https://beam.apache.org/blog/timely-processing/

(CC-BY) Files by Plastic Donut from the Noun Project https://thenounproject.com/search/?q=batch&i=722276
This presentation has been designed using images from Flaticon.com
(CC 3.0 BY) Icons made by phatplus from https://www.flaticon.com/authors/phatplus
(CC 3.0 BY) Icons made by iconixar from https://www.flaticon.com/authors/iconixar
(CC 3.0 BY) Icons made by Those Icons from https://www.flaticon.com/authors/those-icons
(CC 3.0 BY) Icons made by Smashicons from https://www.flaticon.com/authors/smashicons
(CC 3.0 BY) Icons made by photo3idea_studio from https://www.flaticon.com/free-icon/fire_3163799
(CC 3.0 BY) Icons made by Icons made by Eucalyp from https://www.flaticon.com/free-icon/confidential_2857573
(CC 3.0 BY) Icons made by smalllikeart from https://www.flaticon.com/authors/smalllikeart
Do you have a Question for us?
github.com/tkaymak/beam_summit_2023_talk

Israel Herraiz
ihr@google.com

Tobi Kaymak
kaymak@google.com
Deduplicating and analysing time-series data with Apache Beam & QuestDB

Javier Ramirez
Design considerations to operate a stateful streaming pipeline as a service

Israel Herraiz
& Bhupinder Sindhwan
Parallelizing Skewed Hbase Regions using Splittable DoFn

Prathap Reddy
Google
Agenda

- HBase and BigTable Overview
- HBase Snapshot Storage Structure
- Import Snapshots Pipeline
- Challenges & Resolutions
HBase

- Open Source Distributed Scalable Big Data Store
- Random read/write access patterns
- Automatic sharding of tables across regions
- Server side processing using Coprocessors

Bigtable

- Fully managed by Google
- High availability and automatic replication
- Auto Scaling based on application traffic
- Enterprise grade security and control
Hbase Snapshots

● Representation of table at point in time

● Zero Data Copying

● Minimal impact on region servers

● Creating Snapshot

```
hbase> snapshot 'tableName', 'snapshotName'
```

● Export Snapshot to Google Cloud Storage

```
hbase> hbase \
org.apache.hadoop.hbase.snapshot.ExportSnapshot \
-snapshot $SNAPSHOT_NAME \
-copy-to $BUCKET_NAME$SNAPSHOT_EXPORT_PATH/data \
-mappers $NUM_MAPPERS
```
Hbase Storage Structure

<table>
<thead>
<tr>
<th>Table</th>
<th>(HBase table)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Region</td>
<td>(Regions for the table)</td>
</tr>
<tr>
<td>Store</td>
<td>(Store per ColumnFamily for each Region for the table)</td>
</tr>
<tr>
<td>MemStore</td>
<td>(MemStore for each Store for each Region for the table)</td>
</tr>
<tr>
<td>StoreFile</td>
<td>(StoreFiles for each Store for each Region for the table)</td>
</tr>
<tr>
<td>Block</td>
<td>(Blocks within a StoreFile within a Store for each Region for the table)</td>
</tr>
</tbody>
</table>

* Region represents a key range (startKey - endKey) and may live on a different region server

* Store Files are also known as Hfiles
Importing to BigTable (v1)

- Build Snapshot Config
- Read Snapshot (HadoopFormatIO)
- Create Mutation
- Write to Bigtable

* Pipeline Source
Challenges

❖ Skewed regions

❖ Single Table Snapshots
Importing to BigTable (v2)

- Read multiple Snapshot Configs
- List Regions
- Read Region Splits (in parallel)
- Create mutation
- Write to multiple tables in Bigtable

* Snapshot config provides snapshot name, source path and target table name
Splittable Dofn

- Powerful abstraction with support to split each element of work
  
  \((\text{element}, \text{restriction}) \rightarrow (\text{element}, \text{restriction}_1) + (\text{element}, \text{restriction}_2)\)

- Dynamic rebalancing to avoid stragglers
Splittable Dofn

- Restriction represents a portion of work (e.g. OffsetRange, ByteKeyRange)

- Similar Syntax as DoFn with an additional `RestrictionTracker` parameter to `@ProcessElement` method

- `@GetInitialRestriction` - Represents the complete work for a given element

- `@SplitRestriction` (Optional) - Supports pre-splitting initial restriction
Execution of Splittable Dofn
@GetInitialRestriction

```java
public ByteKeyRange getInitialRange(@Element RegionConfig regionConfig) {
    return ByteKeyRange.of(
        ByteKey.copyFrom(regionConfig.getRegionInfo().getStartKey()),
        ByteKey.copyFrom(regionConfig.getRegionInfo().getEndKey()));
}
```
@SplitRestriction

public void splitRestriction(@Element RegionConfig regionConfig,
                              @Restriction ByteKeyRange range,
                              OutputReceiver<ByteKeyRange> outputReceiver) {

    int numSplits = (int) Math.ceil((double) regionConfig.getRegionSize() / BYTES_PER_SPLIT);
    if (numSplits > 1) {
        RegionSplitter.UniformSplit uniformSplit = new RegionSplitter.UniformSplit();
        byte[][] splits =
            uniformSplit.split(
                range.getStartKey().getBytes(),
                range.getEndKey().getBytes(),
                getSplits(regionConfig.getRegionSize()),
                inclusive: true);

        IntStream.range(0, splits.length - 1).forEach((int i) ->
            outputReceiver.output(
                ByteKeyRange.of(ByteKey.copyFrom(splits[i]), ByteKey.copyFrom(splits[i + 1]))));
    } else {
        outputReceiver.output(range);
    }
}
```java
@ProcessElement
public void processElement(
    @Element RegionConfig regionConfig,
    OutputReceiver<KV<SnapshotConfig, Result>> outputReceiver,
    RestrictionTracker<ByteKeyRange, ByteKey> tracker)
    throws Exception {
    try (ResultScanner scanner = newScanner(regionConfig, tracker.currentRestriction())) {
        for (Result result : scanner) {
            if (tracker.tryClaim(ByteKey.copyFrom(result.getRow()))) {
                outputReceiver.output(KV.of(regionConfig.getSnapshotConfig(), result));
            } else {
                break;
            }
        }
    }
    tracker.tryClaim(ByteKey.EMPTY);
}
```
Dynamic Splitting

- Splits current processing element into primary and residual parts
- Runners schedules residual part onto another instance
```java
public class HbaseRegionSplitTracker extends RestrictionTracker<ByteKeyRange, ByteKey>
        implements RestrictionTracker.HasProgress {

    public HbaseRegionSplitTracker(boolean enableDynamicSplitting) {
        this.enableDynamicSplitting = enableDynamicSplitting;
    }

    public SplitResult<ByteKeyRange> trySplit(double fractionOfRemainder) {
        return enableDynamicSplitting ? this.byteKeyRangeTracker.trySplit(fractionOfRemainder) : null;
    }
}
```
Benchmark Tests

❖ Snapshot Datasets

➢ 104 GB with 19 regions (6 regions of 3.5 GB in size and remaining 13 regions are approximately 7 GB)

➢ 875 GB with 14 regions (Mixed region sizes varying from 30GB to 98 GB)

❖ Enabled and Disabled Dynamic Splitting

❖ 10 - 30% improvements in Job Duration with reduced VCPU Consumption

* Beyond Initial splits enabling further splitting didn’t yield significant differences
QUESTIONS?

@prathapreddy017

https://github.com/prathapreddy123

https://www.linkedin.com/in/prathapparvathareddy
Case study: Using statefulDoFns to process late arriving data

Amruta Deshmukh
CI CD for Dataflow with Flex Templates and Cloud Build

Mazlum Tosun
The Future of the Apache Beam Community

Apache Beam Community