BEAM SUMMIT
Beam IO: CDAP and SparkReceiver IO Connectors Overview

Alex Kosolapov & Elizaveta Lomteva
Agenda

➔ Introduction
➔ Developing an IO
➔ CDAP IO Overview
➔ Streaming Source IO – SparkReceiver
➔ Testing IO
➔ Akvelon Data Analytics and ML Accelerators demo
AKVELON

1200+ technology experts
23+ years of expertise
150+ clients
15 offices in 11 countries
24/7 operations support
Developing Beam IO (Java)

- **Starting point:** Developing a new I/O connector
- **Design:**
  - Define the input/output format
  - Read - Splittable DoFn (SDF), Write - ParDo
  - Determine target pipeline configuration parameters
- **Develop:**
  - DoFn to process an element
  - Read/Write PTransforms
- **Test IO:**
  - Unit testing, Integration, Performance testing
- **Release:** IO Documentation and examples
CDAP cdap.io
An open-source platform for data applications in hybrid and multi-cloud environments

Google Cloud Data Fusion
Visual point-and-click interface enabling code-free deployment of ETL/ELT data pipelines

Ecosystem of plugins, including business applications connectors
CDAP IO

Provides transforms for reading and writing data via CDAP plugins

Connects Apache Beam with a variety of business applications like Salesforce, Hubspot, ServiceNow and Zendesk

Uses CDAP plugin definition
CDAP IO Workflow
SparkReceiver IO

SparkReceiverIO provides transforms to read data via Apache Spark Receiver

Prerequisites:

- Spark Receiver provides HasOffset interface.
- Records have a numeric field that represents record offset.
SparkReceiver IO Workflow

Transform definition

SparkReceiverIO.Read

ReadFromSparkReceiver DoFn

SDF execution

Spark Consumer

Receiver Supervisor

Spark Receiver
Beam Parallelism & IO

**Input** parallelism – reading from bounded and unbounded sources, i.e. data source parallelism

**Inter-stage** parallelism – splitting processing across workers, e.g. key-based data partitioning

**Intra-stage** parallelism – splitting element processing within transforms, e.g. Splittable DoFns, bundle processing
Data Source Parallelism

Refers to the parallelism achieved by reading data from multiple sources or partitions of a single source concurrently.

(E.g. Kafka topic partitions)

SparkReceiverIO

Each receiver builder can be associated with single source object and create multiple receivers during processing.
Inter-stage parallelism

Refers to the parallelism between different transforms (or stages) within a Beam pipeline.

Achieved by runner implementation

(E.g. key-based operations in Beam)

SparkReceiverIO

Achieved by supported runners – Direct runner and Dataflow runner v1 and v2
Intra-stage: Splittable DoFn (SDF)

Executing an SDF follows the following steps:

1. Each **element** is paired with a **restriction** (e.g. filename is paired with offset range representing the whole file).
2. Each element and restriction pair is **split** (e.g. **offset** ranges are broken up into smaller pieces).
3. The runner redistributes the element and restriction pairs to several workers.
4. Element and restriction pairs are processed in parallel (e.g. the file is read). Within this last step, the element and restriction pair can pause its own processing and/or be split into further element and restriction pairs.
Agenda

➔ Introduction
➔ Developing an IO
➔ CDAP IO Overview
➔ Streaming Source IO – SparkReceiver
➔ Testing IO
➔ Akvelon Data Analytics and ML Accelerators demo
Testing IO and Release

IO Testing
- testing guide, IO transforms testing
- Unit, integration and performance test
- Created RabbitMQ SparkReceiver on-demand source in Apache Beam that generates streaming data according to provided profile

Release
- Beam website IO Connectors
- Documentation & Readmes
- Complete examples
Akvelon Data and Analytics Accelerators

Akvelon is a digital product and software engineering company that empowers strategic advantage and accelerates your path to value in Data and Analytics, AI/ML, MLOps, Application development, and more with innovation and predictable delivery. Akvelon is providing this collection of accelerators as a reference and easy customizations for developers looking to build data, machine learning, and visualizations.

- Get in touch about Data and Analytics and Data Migrations projects.
- Get in touch about ML projects.
- Get in touch about Google Cloud projects.

Learn more about all our ML and software engineering services at our website akvelon.com.

Accelerators

ML, Streaming and Batch Data Processing

Apache Beam and Google Cloud Dataflow

Apache Beam provide unified streaming and batch processing to power ML and streaming analytics use cases. Google Cloud Dataflow is a managed service to run Apache Beam in cloud with minimal latency and costs, and integrations with other Google Cloud products like Vertex AI and TensorFlow TFX. Akvelon, a Google Cloud Service Partner, and an active Apache Beam contributor and Beam Summit partner, presents several of our favorite accelerators for Dataflow.

- Salesforce to Txl - Flex templates for batch and streaming Salesforce data processing with Google Cloud Dataflow, using Apache Beam CDAP IO.
- Salesforce to BigQuery - Flex templates for batch and streaming Salesforce data processing with Google Cloud Dataflow and BigQuery, using Apache Beam CDAP IO. Flex templates provide a comprehensive example of using Machine Learning (ML) to process streaming data in Dataflow, using Java multilanguage pipeline with Python transforms to run custom TFX and PyTorch ML models. This complete Flex template example also demonstrates creating and setting up Expansion Service in Dataflow to enable running custom Python transforms within a Java pipeline.
- TensorFlow TFX model training with Apache Beam - a Python notebook and Python Beam pipeline that demonstrates both Jupyter notebook to train a TensorFlow TFX ML model and the converted Python pipeline ready for Expansion Service use.
- PyTorch ML model training and Expansion Service for multilanguage pipelines with Apache Beam - a complete example to train a PyTorch ML model using Apache Beam, convert the notebook to the Python pipeline, create custom Python Transforms and deploy as Apache Beam Expansion Service for Google Cloud Dataflow.

Custom Visualizations

Akvelon has accumulated vast experience with data analytics, custom visualizations, dashboards, and reports for a wide range of industries and use cases. Here are some of our favorite visualization accelerators.

Looker Visuals
Google Cloud Dataflow Accelerators

Apache Beam provides unified streaming and batch processing to power ML and streaming analytics use cases. Google Cloud Dataflow is managed to run Apache Beam in the cloud with minimal latency and costs, and integrates with other Google Cloud products like Vertex AI and Tensorflow TFX. Akvelon, a Google Cloud Service Partner, and an active Apache Beam contributor and Beam Summit partner, presents several of our favorite accelerators for Dataflow.

Akvelon, a Google Cloud Partner, is providing this open-source collection of Dataflow Flex templates as a reference and easy customizations for developers looking to build streaming, batch, multilanguage data pipelines with ML processing in Google Cloud Dataflow.

Flex Templates for Google Cloud Dataflow

Google Cloud Dataflow Flex Templates are a powerful way to build and run data pipelines on Google Cloud Platform. With Flex Templates, you can package your pipeline code and dependencies as a Docker image, and then run it on Dataflow with just a few clicks. This makes it easy to build and deploy complex pipelines quickly and reliably.

- Salesforce to Txt - Flex templates for batch and streaming Salesforce data processing with Google Cloud Dataflow, using CDAP IO.
- Salesforce to BigQuery - Flex templates for batch and streaming Salesforce data processing with Google Cloud Dataflow, using Apache Beam CDAP IO. Flex templates provide a comprehensive example of using Machine Learning (ML) to process data in Dataflow, using Java multilanguage pipeline with Python transforms to run custom TFX and PyTorch ML models. Flex template example also demonstrates creating and setting up Expansion Service in Dataflow to enable running custom transforms within a Java pipeline.

Machine Learning with Google Cloud Dataflow

- Tensorflow TFX model training with Apache Beam - a Python notebook and Python Beam pipeline that demonstrate both training and serving TensorFlow models.
Summary

Developing Beam IOs

Machine Learning

Multilanguage pipelines

https://github.com/akvelon/DnA_accelerators
AKVELON

https://github.com/akvelon/DnA_accelerators

https://akvelon.com

Questions?

https://www.linkedin.com/in/akosolapov
https://www.linkedin.com/in/elizaveta-lomteva
Meeting Security Requirements for Apache Beam Pipelines on Google Cloud

Lorenzo Caggioni
Google
linkedin.com/in/lcaggio/
Agenda

Securing a Beam Pipelines on Google Cloud

- Private resources
- Role separation and least privileges
- Data Encryption at rest
Customer requirements

1. Internal addressment of tenants must be private.

2. Every tenants must be isolated and dedicated to a specific system of services.

3. All data must have encryption at-rest with keys managed by ACME's security team.
1. Internal addressment of tenants must be private.

1. Set `disable-public-ips` when deploying the pipeline
2. Enable `Private Access` on the subnet to access GoogleAPIs
3. Network: shared-VPC
VPC Service Controls helps preventing data exfiltration and controlling access to Google APIs.

Isolate resources of multi-tenant Google Cloud services to mitigate data exfiltration risks.
IAM and Service Accounts

2. Tenants must be isolated

Cloud Storage -> Dataflow -> BigQuery

Service Account

Dataflow Service Agent
roles/dataflow.serviceAgent
roles/compute.networkUser

Worker Service Account
roles/storage.objectAdmin
roles/dataflow.worker
roles/bigquery.dataEditor

Job orchestrator
role/iam.serviceAccountUser
role/dataflow.admin
2. Tenants must be isolated

Project separation

- **Landing**
  - Service Account
  - Cloud Storage

- **Processing**
  - Service Account
  - Dataflow

- **Curated**
  - Service Account
  - BigQuery
3. At rest encryption

Data at rest are encrypted on GCP:

1. Data split in chunk and encrypted with a key: Data Encryption Key (DEK)
2. DEK encrypted with Key Encryption Key (KEK)
3. Chunk stored with encrypted DEK

Options

- Default Google encryption
- Customer-managed encryption keys
  - Cloud KMS
- Customer-managed encryption keys
  - Cloud HSM
- Customer-managed encryption keys
  - Cloud EKM
3. At rest encryption

- Security
  - Cloud KMS
- Landing
  - Service Account
  - Cloud Storage
- Processing
  - Service Account
  - Dataflow
- Curated
  - Service Account
  - BigQuery

roles/cloudkms.cryptoKeyEncrypterDecrypter
1. Every tenants must be isolated and dedicated to a specific system of services.
2. Internal addressment of tenants must be private.
3. All data must have encryption at-rest with keys managed by ACEME's security team.

Recap

End to end example
QUESTIONS?

Contact info
https://twitter.com/lcaggio
https://www.linkedin.com/in/lcaggio
https://github.com/lcaggio
Simplifying Speech-to-Text Processing with Apache Beam and Redis

Pramod Rao & Prateek Sheel
Simplifying Speech-to-Text Processing with Apache Beam and Redis

Pramod Rao
Cloud Data Engineer
Google Cloud Consulting

Prateek Sheel
Data & Analytics Consultant
Google Cloud Consulting
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overview</td>
<td>01</td>
</tr>
<tr>
<td>Design Journey</td>
<td>02</td>
</tr>
<tr>
<td>Lessons Learned</td>
<td>03</td>
</tr>
</tbody>
</table>
Overview
Business Process
So, what’s the problem?

Multiple Call-Transfer Scenarios

- First Agent Joins
- Second Agent Joins
- First Agent Drops
- Second Agent Drops

Events
- Key1
- Key2

Transcript
- Key1|Key2

Metadata
- Key2

Wait Time
- Warm Transfer
- Overlap
- Conversation

User Joins

Plus, additional business rules

[DUPLICATES]
[OUT-OF-ORDER DATA]
[MISSING DATA]
02
Design Journey
Design Approach # 1

Events | Transcript | Metadata
---|---|---
Key1 | Key1|Key2 | Key2

Session Window

Key1

- Conversation 1

Key1|Key2

- Conversation 2

Re-key, and Window with key1 | key 2

Set 2 expiry timers $t_1 = t_0 + X_s$, $t_2 = t_0 + Y_s$

T1 expired

T2 expired

$\text{t0} + X_s$ output

$\text{t0} + Y_s$ output

$\text{t0} + 60s$ payload

$\text{t0} + 180s$ payload
Design 1 Trade Offs

**Dependencies**
No state external to Dataflow. No external service dependencies.

**Latency**
Need to wait for the session to end and the timers to expire before the output payloads can be produced. Not ideal based on the business SLO.

**Completeness**
In some cases all of the information required to creating the output payloads may not be available when the timers expire. This is due to the uncertain ordering of events.

**Code Complexity**
Windowing allows for relatively simpler business logic implementation for creating the output payloads since re-keying produces outputs at the required granularity.
Design Approach #2

**Events** Key1

**Transcript** Key1|Key2

**Metadata** Key2

---

**Session Window**

- **Conversation 1**
  - T1 expired
  - Set 2 expiry timers \( t_1 = t_0 + Xs, t_2 = t_0 + Ys \)

- **Conversation 2**
  - T2 expired

---

- **Events**
- **Transcripts**
- **Metadata**

- t0 + 60s payload
- t0 + 180s payload
### Design 2 Trade Offs

<table>
<thead>
<tr>
<th>Dependencies</th>
<th>Latency</th>
<th>Completeness</th>
<th>Code Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>No state external to Dataflow. No external service dependencies.</td>
<td>Need to wait for the session to end and the timers to expire before the output payloads can be produced. Not ideal based on the business SLO.</td>
<td>In some cases all of the information required to creating the output payloads may not be available when the timers expire. This is due to the uncertain ordering of events.</td>
<td><strong>Granularity</strong> of outputs doesn’t match the inputs thereby increasing the business logic <strong>complexity</strong> required to produce the output payloads</td>
</tr>
</tbody>
</table>
Design Approach #3

**Events**

- **Key1**

**Transcript**

- **Key1|Key2**

**Metadata**

- **Key2**

**Dataflow**

- **Redis**
  - Sorted Sets
  - Lettuce 6.1.8

**Session Window**

- Conversation 1
- Conversation 2

**Set 2 expiry timers**

- $t_1 = t_0 + X_s$, $t_2 = t_0 + Y_s$

**Events**

- T1 expired
- T2 expired

**Conversation 2**

- Lettuce 6.1.8
  - $t_0 + 60s$ payload
  - $t_0 + 180s$ payload
Latency
Low latency data store that dovetails well with streaming use cases

Order
We rely on Redis sorted sets for accumulating the speech transcripts, we are able to maintain the order of the conversation as well as deduplicating the transcripts automagically

Data Lifecycle
Redis offers a simple approach to manage cleanup of stale data
# Design 3 Trade Offs

<table>
<thead>
<tr>
<th>Dependencies</th>
<th>Latency</th>
<th>Completeness</th>
<th>Code Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Dependency</strong> on a managed Redis instance. This also results in additional <strong>costs</strong> to host a Redis instance in the Cloud environment.</td>
<td>No need for any additional wait time over and above the required timers. <strong>Subsecond end-to-end latency for ML predictions.</strong></td>
<td>Least chance of incomplete outputs due to the <strong>ordering</strong> provided by Redis</td>
<td>Much <strong>simpler processing</strong> because complicated scenarios related to cross-referencing the three data sources are eliminated. Only need to “act” on events.</td>
</tr>
</tbody>
</table>
## Latency Metrics*

<table>
<thead>
<tr>
<th>Machine Type</th>
<th>Dataflow</th>
<th>PreProcessing Avg. (ms)</th>
<th>Redis Avg. (ms)</th>
<th>Predictions Avg. (ms)</th>
<th>End-To-End Avg. (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>n1-standard-2</td>
<td>t0+60s</td>
<td>1210.90</td>
<td>20.84</td>
<td>204.83</td>
<td>1441.75</td>
</tr>
<tr>
<td>n1-standard-2</td>
<td>t0+180s</td>
<td>1155.52</td>
<td>18.62</td>
<td>260.33</td>
<td>1441.72</td>
</tr>
<tr>
<td>n2d-standard-4</td>
<td>t0+60s</td>
<td>580.38</td>
<td>9.84</td>
<td>198.68</td>
<td>796.10</td>
</tr>
<tr>
<td>n2d-standard-4</td>
<td>t0+180s</td>
<td>596.54</td>
<td>9.98</td>
<td>260.54</td>
<td>874.35</td>
</tr>
</tbody>
</table>

*Excluding the wait time to accumulate data for each event type*
Final Solution

Speech-to-text Processing with Apache Beam and Redis

Google Cloud

Context Pub/Sub
Label Pub/Sub
Transcript Pub/Sub
Pre-Processing Dataflow
Summary Event Pub/Sub
Audit Info BigQuery
redis
Prediction Dataflow
Vertex AI
On-Prem Predictions
Lessons Learned
Lessons Learned

**Functional**

- **Order of data**
  Real world scenarios include out-of-order data, duplicates, and missing elements

- **Granularity of inputs**
  Business logic is greatly simplified if all inputs are at the same level of "granularity"

- **Latency**
  Latency requirements dictate the nature of the final solution

**Operational**

- **Observability**
  Non functional requirements such as operational metrics and dead-letter queues are essential to gain insights into the processing state at any time

- **Configurability**
  Levers should be provided to change the processing characteristics without changing any code

- **Representative test data**
  "Good" test data is imperative to shorten the development lifecycle and can be tricky to generate or acquire
Thank you!

https://cloud.google.com/consulting
Hot Key Detection and Handling in Apache Beam Pipelines

Shafiqaa Iqbal & Ikenna Okolo
To this
How stragglers can look like
WordCount

Pipeline p = Pipeline.create(options);
p.apply(TextIO.Read.from("gs://dataflow-samples/shakespeare/*"))
    .apply(FlatMapElements.via(
        word → Arrays.asList(word.split("[^a-zA-Z]")))
    .apply(Filter.byPredicate(word → !word.isEmpty()))
    .apply(Count.perElement())
    .apply(MapElements.via(
        count → count.getKey() + " : " + count.getValue())
    .apply(TextIO.Write.to("gs://.../..."));
p.run();
Primitives to keep in mind

MapReduce = ParDo + GroupByKey + ParDo
How a ParDo would work

shrek is the greatest movie ever

to be, or not to be

for to be made

shard 1

DoFn

(shrek, 1)
(is, 1)
(movie, 1)
(the, 1)
(ever, 1)

(greatest, 1)

(shrek, 1)
(is, 1)
(movie, 1)
(the, 1)
(ever, 1)

(to, 2)
(be, 2)
Gantt charts

Workers

shard N

Time
What is a straggler, really?

Slower than perfectly-parallel:

$$t_{end} > \frac{\text{sum}(t_{end})}{N}$$
Amdahl’s law: it gets worse at scale

\[
\text{Speedup} = \frac{N}{1 + \left(N - 1\right) \cdot S}
\]

#workers
serial fraction

Higher scale \(\Rightarrow\) More bottlenecked by serial parts.
# Reasons for Stragglers

**Uneven partitioning**
- Process dictionary in parallel by first letter -> 6x speedup only by ahmdahl’s law

**Uneven Complexity**
- Join keys with some external input values

**Uneven resources**
- Bad machines, network or resource contention

**Bugs**
- Slow RPCs or bugs
Reasons for Stragglers

- Uneven partitioning
  - Process dictionary in parallel by first letter -> 6x speedup only by Amdahl's law

- Uneven Complexity
  - Join keys with some external input values

- Uneven resources
  - Bad machines, network or resource contention

- Bugs
  - Slow RPCs or bugs
What are hotkeys

A hot key is a key with enough elements to negatively impact pipeline performance. These keys limit a Pipeline's ability to process elements in parallel, which increases execution time.

Think about hotkeys in this way. Let's imagine there's a room filled with 150 Red, 30 Blue and 20 Green unsorted plates and there are 3 students who are to arrange those plates in sorted orders (as seen here to the right).

Let's assume that student 1 will sort the Red plates, student 2 will sort the blue and the last student will sort the green plates.
From the illustration in the previous slide, students 2 and 3 will finish before student 1. Though the second and third students had already completed sorting their respective colored plates, they have to wait for the first student to complete theirs before the task can be termed as completed. This delay by student 1 is due to the larger number of plates they need to sort. In parallel processing, this is referred to as hotkeys.

If we replace the students with workers and the unsorted-plates with work-items to be processed, we can apply the same thinking to Dataflow pipelines. If the work-items are not evenly distributed, then there’s bound to be an issue of hotkeys which obviously would impact the performance of the Pipeline.

In subsequent slides, we will explain this using a Key Value pair to represent individual work-items.
How do Hotkeys cause problems?

Source Data (KeyValues) → GroupByKey → Transformation

- `<K1, V>`
- `...`
- `<K1, V>`
- `<K2, V>`
- `<K2, V>`
- `<K3, V>`
- `...`
- `<Kn, V>`

1,000,000 records

Note that the dataset is heavily imbalanced.

`K1` has broken the uniformity and thus is called the "hotkey."

- `<K1, (V, V, V, ..., V)>` → Worker 1
  - 100% CPU utilization rate.

- `<K2, (V, V)>` → Worker 2
  - Idle. Done processing data.

- `<Kn, (V, V, V)>` → Worker n
  - Idle. Done processing data.

Problem: The next job will not start until Worker 1 finishes its transformations.

Parallelism (Number of active workers) vs. Time (t)
One of the quickest ways to identify a Job that is impacted by hotkeys is by taking a quick look at the worker CPU utilisation. While some workers are maxing out at about 90% utilisation, some are idle at about <5% utilisation. This truly indicates that there is a possibility that the Job is stuck due to hotkeys.
What can you do?

- **Uneven partitioning**: Oversplit, Hand-tune, Use data statistics
- **Uneven complexity**: Predictive
- **Uneven resources**: Backups, Restarts
- **Noise**: Weak

Data Monitoring, key partitioning, iterative optimization

Using statistical analysis to pre-detect the hot key
To resolve this issue, you may have to check that your data is evenly distributed. If a key has disproportionately many values, consider the following courses of action:

- Rekey their data. Apply a ParDo transform to output new key-value pairs.
- Autosharding
- Combine.Globally withFanout(int fanout)
- Java jobs should consider using the Combine.PerKey.withHotKeyFanout transform.
- Python jobs should consider using the CombinePerKey.with_hot_key_fanout transform.
- Finally, consider enabling Dataflow Shuffle (if using dataflow).
Job not impacted by hotkeys anymore!

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>dna-integration-gsi-edd-p-02270430-b79j-harness-0100</td>
<td>1.66%</td>
</tr>
<tr>
<td>dna-integration-gsi-edd-p-02270430-b79j-harness-05z9</td>
<td>1.61%</td>
</tr>
<tr>
<td>dna-integration-gsi-edd-p-02270430-b79j-harness-0g4x</td>
<td>2.08%</td>
</tr>
<tr>
<td>dna-integration-gsi-edd-p-02270430-b79j-harness-0kjp</td>
<td>1.81%</td>
</tr>
</tbody>
</table>
From this
To this
Can we assign a more powerful machine to the worker that is processing the hotkey (i.e. Worker 1)?

>> Unfortunately, you cannot. Dataflow, by design, assigns the same machine to all of its workers.

In that case, if all workers run with powerful machines, the pipeline will finish quicker.
+ It will be cheap, since most of them will be idle anyways.

>> This will not speed up the process. A powerful machine will still use up only one of its cores. Imagine a giant for-loop to better understand -- cores do not split the work of a for loop.

I enabled autoscale, but my job doesn’t finish any faster. Why?

>> You will see in monitoring that the average CPU utilization rate is far below 20%; therefore, Dataflow will not bring in more workers. Even if it does, it won’t help -- remember that you already have n-1 idle workers. Surely n idle workers won’t make a difference.

Root cause: dataset is imbalanced.
Fix the root cause: balance the dataset.

Solution: Classify the imbalanced key and break them down into smaller pieces.
Troubleshooting Slow Running Beam Pipelines

By Mehak Gupta
Google Cloud, Canada
Hello!

I’m Mehak

Technical Solutions Specialist at Google Cloud
Goals

- Apache Beam pipeline troubleshooting techniques that would empower professionals to research and resolve Beam issues.
- Self service skills would reduce MTTR (Mean Time To Recover) from a job failure significantly.
- Share some tricks and samples of troubleshooting slow running beam pipelines using Dataflow as an example.
How to identify if the beam pipeline is slow/stuck

- Pipeline is running from a long time without reporting results
- Increased data watermark or system latency
- Pipeline is not consuming input
Troubleshooting Slow Running Beam Pipelines

**Troubleshooting Workflow**

- **Check job errors**
- **View step logs**
- **Check quotas and sinks**

**Worker code exceptions**

- **Batch pipeline**
  - Use the execution details tab to check for slow or stuck stages

- **Find the slow or stuck stage**

- **Look for parallelism bottlenecks**

- **Streaming pipeline**
  - Use the Cloud Monitoring for Dataflow view to see metrics
### Troubleshoot slow/stuck dataflow jobs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>End time</th>
<th>Elapsed time</th>
<th>Start time</th>
<th>Status</th>
<th>SDK version</th>
<th>ID</th>
<th>Region</th>
<th>Insights</th>
</tr>
</thead>
<tbody>
<tr>
<td>wordcount6</td>
<td>Streaming</td>
<td>Apr 11, 2023, 3:08 PM</td>
<td>48 days 19 hr</td>
<td>Running</td>
<td>2.43.0</td>
<td></td>
<td>2023-04-11-17_08_12-00-000020137046300790566</td>
<td>us-east1</td>
<td></td>
</tr>
<tr>
<td>wordcount5</td>
<td>Batch</td>
<td>May 28, 2023, 12:31 PM</td>
<td>21 hr 30 min</td>
<td>Succeeded</td>
<td>2.46.0</td>
<td></td>
<td>2023-05-27_12_00_28-100407821648944481412</td>
<td>us-east1</td>
<td></td>
</tr>
</tbody>
</table>
Troubleshoot slow/stuck dataflow jobs

Troubleshooting using Logs Explorer View
Troubleshoot slow/stuck dataflow jobs

Check logs here
Troubleshoot slow/stuck dataflow jobs
Troubleshoot slow/stuck dataflow jobs
Troubleshoot slow/stuck dataflow jobs

Log fields

Histogram

Query results: 17 log entries

This query has been updated. Run it to view matching entries.

Run query

Log field: dataflow.googleapis.com/job-message

Worker configuration: nl-standard-4 in us-central1-c.

Worker configuration: nl-standard-4 in us-central1-c.

Your project already contains 100 Dataflow-created metric descriptors, so new user metrics of the form custom.googleapis.com/*/ will not be created. However, all user metri-
Troubleshoot slow/stuck dataflow jobs
Troubleshoot slow/stuck dataflow jobs

Select which logs you want to view from here:
- worker-startup
- worker
- docker & kubelet
- shuffler
Troubleshoot slow/stuck dataflow jobs
Troubleshoot slow/stuck dataflow jobs

Troubleshooting using Job Metrics Tab
Troubleshoot slow/stuck dataflow jobs

Throughput dropping to zero

Check under "Job Metrics" tab for various metrics

Throughput (elements/sec)
Troubleshoot slow/stuck dataflow jobs

High CPU Utilization

CPU utilization (All Workers)
High CPU Utilization

Troubleshoot slow/stuck dataflow jobs
Troubleshoot slow/stuck dataflow jobs

Data Freshness

Data freshness by stages

Create alerting policy

Metric

Value

per_stage_data_watermark_age
1.728d
Troubleshoot slow/stuck dataflow jobs

**System Latency**

System latency by stages

Create alerting policy

- **Metric**: per_stage_system_lag
- **Value**: 1s
Troubleshoot slow/stuck dataflow jobs

Stragglers in batch job
When a batch job takes a long time to process data, it would be best to check on the **Straggler Workers**

**How to check it?**

- Under Execution details, select Stage progress in graph view list.
Troubleshoot slow/stuck dataflow jobs

There can be various causes of stragglers:

- **Hot Keys**: Hot keys can create stragglers because they limit the ability of Dataflow to process elements in parallel.
  
a. Re-key your data. Apply a ParDo transform to output new key-value pairs.

- Re-shuffle your data to avoid a single worker having extra load
Troubleshoot slow/stuck dataflow jobs

Scenario 1: Long active user operation
Troubleshoot slow/stuck dataflow jobs

Processing Stuck/ Operation ongoing

- Operation ongoing in step (step name) for at least (duration)

- OR

- Processing stuck in step (step name) for at least (duration)
Troubleshoot slow/stuck dataflow jobs

Processing Stuck/ Operation ongoing

From Logs Explorer

Query:

<table>
<thead>
<tr>
<th>Query</th>
<th>Saved (0)</th>
<th>Suggested (2)</th>
<th>Library</th>
</tr>
</thead>
<tbody>
<tr>
<td>🕒 Last 14 days</td>
<td>🕒 Search all fields</td>
<td>🕒</td>
<td>🕒</td>
</tr>
<tr>
<td>1. <code>resource.type=&quot;dataflow_step&quot;</code></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2. <code>resource.labels.job_id=$JOB_ID</code></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3. <code>logName:=&quot;/logs/dataflow.googleapis.com%2Fworker&quot;</code></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Results:

```
Operation ongoing in step Write to BQ/BatchLoads/SinglePartitionWriteTables/ParMultiDo(WriteTables) for at least 02h20m00s without outputting or completing in state finish.
```

at java.base@11.0.9/java.lang.Thread.sleep(Native Method)
at app/com.google.api.client.util.Sleepers$1.sleep(Sleepers.java:42)
at app/com.google.api.client.util.BackOffUtils.next(BackOffUtils.java:48)
at app/org.apache.beam.sdk.io.gcp.bigquery.BigQueryHelpers$PendingJobManager.nextBackOff(BigQueryHelpers.java:162)
at app/org.apache.beam.sdk.io.gcp.bigquery.BigQueryHelpers$PendingJobManager.waitForDone(BigQueryHelpers.java:148)
at app/org.apache.beam.sdk.io.gcp.bigquery.WriteTables$WriteTablesDoFn.finishBundle(WriteTables.java:380)
at app/org.apache.beam.sdk.io.gcp.bigquery.WriteTables$WriteTablesDoFn$DoFnInvoker.invokeFinishBundle(Unknown Source)
```
Troubleshoot slow/stuck dataflow jobs

Processing Stuck/ Operation ongoing

From Logs Explorer

Operation ongoing in step Write to BQ/BatchLoads/SinglePartitionWriteTables/ParMultiDo(WriteTables) for at least 02h20m00s without outputting or completing in state finish

at java.base@11.0.9/java.lang.Thread.sleep(Native Method)
at app/com.google.api.client.util.Sleepers$Sleep$1.sleep(Sleepers.java:42)
at app/com.google.api.client.util.BackOffUtils.next(BackOffUtils.java:48)
at app/org.apache.beam.sdk.io.gcp.bigquery.BigQueryHelpers$PendingJobManager.nextBackOff(BigQueryHelpers.java:162)
at app/org.apache.beam.sdk.io.gcp.bigquery.BigQueryHelpers$PendingJobManager.waitForDone(BigQueryHelpers.java:148)
at app/org.apache.beam.sdk.io.gcp.bigquery.WriteTables$WriteTablesDoFn.finishBundle(WriteTables.java:380)
at app/org.apache.beam.sdk.io.gcp.bigquery.WriteTables$WriteTablesDoFn$DoFnInvoker.invokeFinishBundle(Unknown Source)

Troubleshoot slow/stuck dataflow jobs

Processing Stuck/ Operation ongoing
Troubleshoot slow/stuck dataflow jobs

Apache Beam Issues/Feature Request

- Performance Regression or Improvement: Pytorch image classification on 50k images of size 224 x 224 with resnet 152 with Tesla T4 GPU:mean_load_model_latency_milli_secs (awaiting triage) [#27077]
- Performance Regression or Improvement: Pytorch image classification on 50k images of size 224 x 224 with resnet 162 with Tesla T4 GPU:mean_inference_batch_latency_micro_secs (awaiting triage) [#27076]

[Feature Request]: BigqueryOJ Java WriteTableRows RangePartitioning support (awaiting triage) [#27047]

[Bug]: Kafkato read transform is inefficient when using the commit_offsets_in_finalization option (awaiting triage) [#27051]

[Failing Test]: BigQueryJOWriteTest.testWriteFileSchemaUpdateOptionAllowFieldAddition (awaiting triage) [#27040]

[Bug][Go]: Metrics incremented in Setup methods are not recalled. (good first issue) [#27036]

[Bug]: beam.transforms.util.LogElements(with_timestamp=True, with_window=True) does not work with GlobalWindows (good first issue) [#27036]
Scenario 2: GC Thrashing/OOM
Troubleshoot slow/stuck dataflow jobs

GC Thrashing/OOM: Diagnostics Tab

<table>
<thead>
<tr>
<th>Occurrences</th>
<th>Count</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>8</td>
<td>Shutting down JVM after 8 consecutive periods of measured GC thrashing. Memory is used/total/max = 7904/20103/37513 MB, GC last/max = 90.03/95.7... The worker was shut down after a long period of high memory pressure.</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>StatusRuntimeException: UNAVAILABLE: keepalive watchdog timeout</td>
</tr>
</tbody>
</table>
Troubleshoot slow/stuck dataflow jobs

**GC Thrashing/OOM**

Memory utilization

Max worker memory utilization (estimated bytes/sec)

<table>
<thead>
<tr>
<th>Metric</th>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max worker memory capacity</td>
<td>29.39GB</td>
<td></td>
</tr>
<tr>
<td>Max worker memory usage</td>
<td>0.63GB</td>
<td></td>
</tr>
</tbody>
</table>
Troubleshoot slow/stuck dataflow jobs

General Recommendations

- Use machine types with higher memory
  - Link: goo.gle/45USWe3

- Decrease the parallelism of processing by reducing the number of worker harness threads
  - Link: goo.gle/45RM6WT

- Do vertical autoscaling (Enable Dataflow Prime)
  - Link: goo.gle/3r3KZjv
Performance Optimization using Dataflow profiling

- Cloud Profiler is available for Dataflow pipelines written in Apache Beam SDK for Java and Python, version 2.33.0 or later.

- It can be enabled at pipeline start time

- E.g. For Java SDK, to enable CPU profiling, start the pipeline with the following option:
  --dataflowServiceOptions=enable_google_cloud_profiler
QUESTIONS?

mhkgupta@google.com

linkedin.com/in/mhkgupta
Resolving out of memory issues in Beam Pipelines

Zeeshan Khan
Benchmarking Beam pipelines on Dataflow

Pranav Bhandari