

Maximizing Performance, Reliability, and Scalability with Dataflow Streaming

Apache Beam Summit

1:00 PM Tuesday, 23 June 2026

Tom Stepp & Ryan Wigglesworth, Google



Table of Contents

Streaming ML	01
High Availability	02
Iceberg Ingestion	03
Offset Deduplication	04
High Cardinality	05
Shuffle Enhancements	06
Parallel Updates	07
Streaming Engine Reliability	08
Observability	09
AI Pipeline Writing/Debugging	10

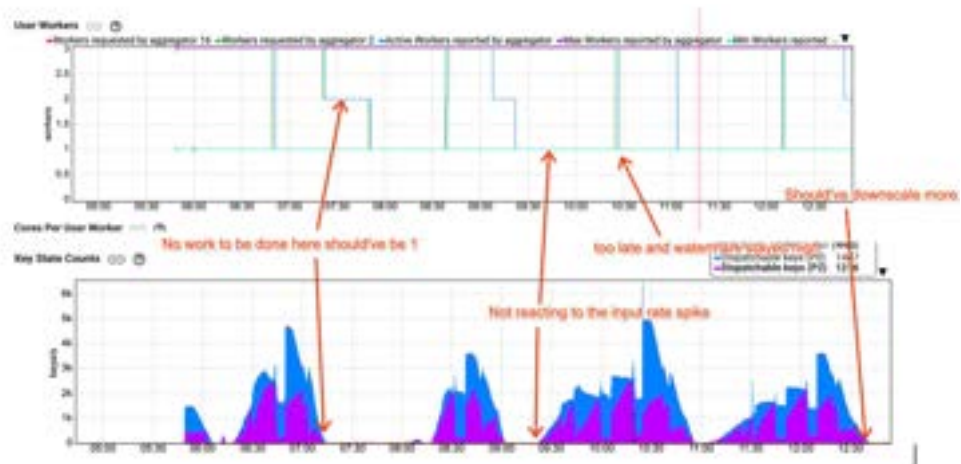
Streaming ML

Async DoFn

- For long running Python or Java Transforms
- Prevents retries allowing long running transforms to finish without premature interruptions from requests hedging
- Also added asyncio support in Python to allow reusing threads waiting on remote calls, increase parallelism by up to 500x

GPU Autoscaling

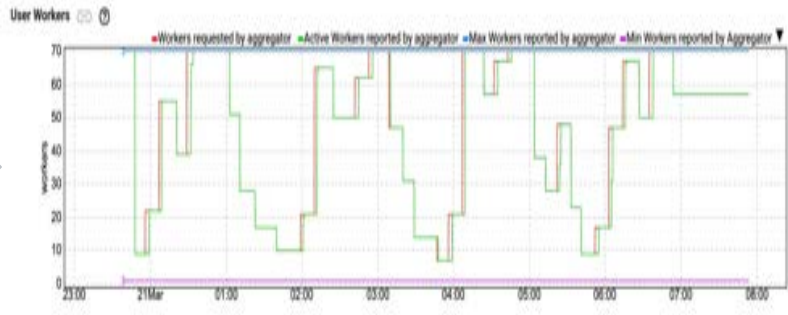
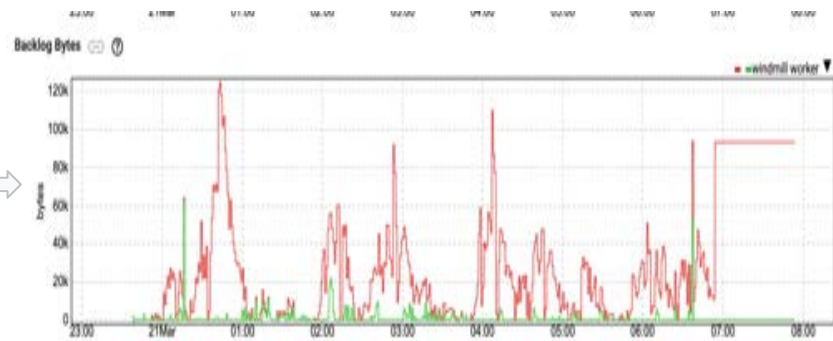
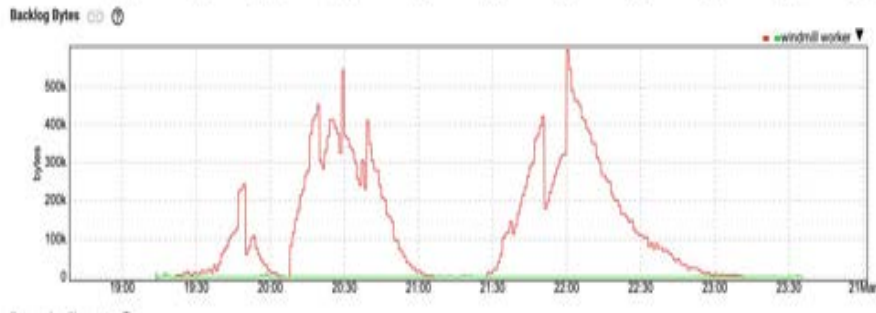
- Legacy autoscaling is designed for operations that take milliseconds to seconds, rather than seconds to minutes like many ML steps do.
- Autoscaling needs special tuning for this case



How to use GPU Autoscaling

- Marking computation
 - Using resource hints and passing in `max_parallelism_per_worker` at the same time.
 - Example usage:
 - `pcoll | MyPTransform().with_resource_hints(
 min_ram="4GB",
 accelerator="type:nvidia-tesla-l4;count:1;install-nvidia-driver",
 max_parallelism_per_worker=2,
)`

Experiment

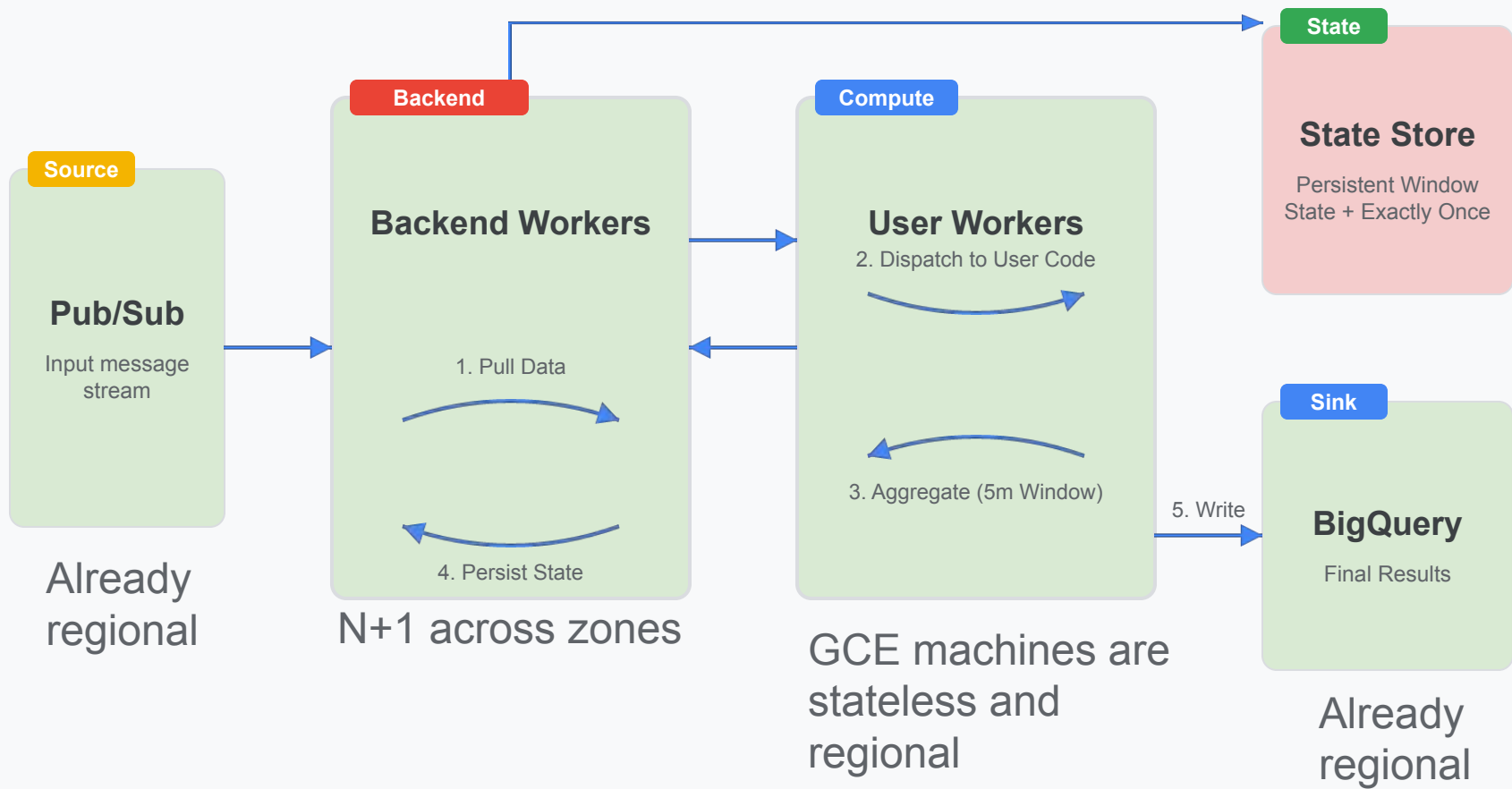


Cores Per User Worker

Cores Per User Worker

High Availability

Jobs are at risk of state store zonal failure



Dataflow HA Advantages

Fully Managed

Fits with our fully managed position versus competitors.

- Single setup option
- Automatic failover
- No data loss
- ~1 min pause only

Cost Efficient

Only replicates the necessary portions of the infrastructure.

- Just replication
- ~10% of total costs
- Avoids full pipeline duplication

Deep Integration

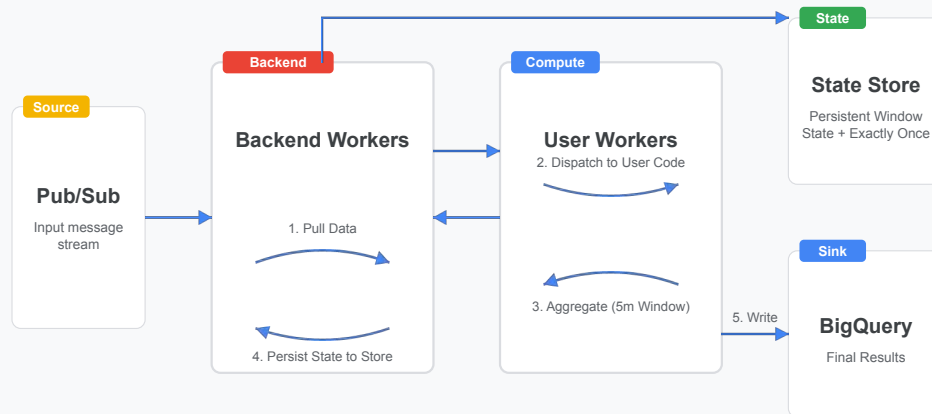
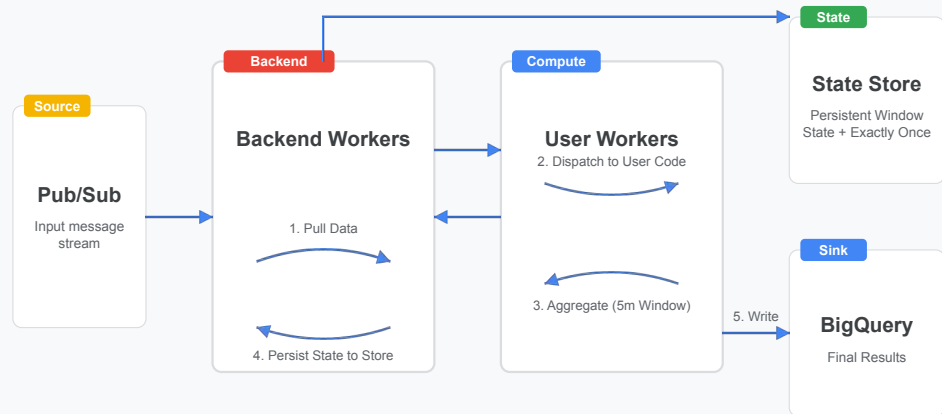
Works seamlessly with Google Cloud's regional ecosystem.

- Regional sources/sinks
- Fits better with the rest of BQ system

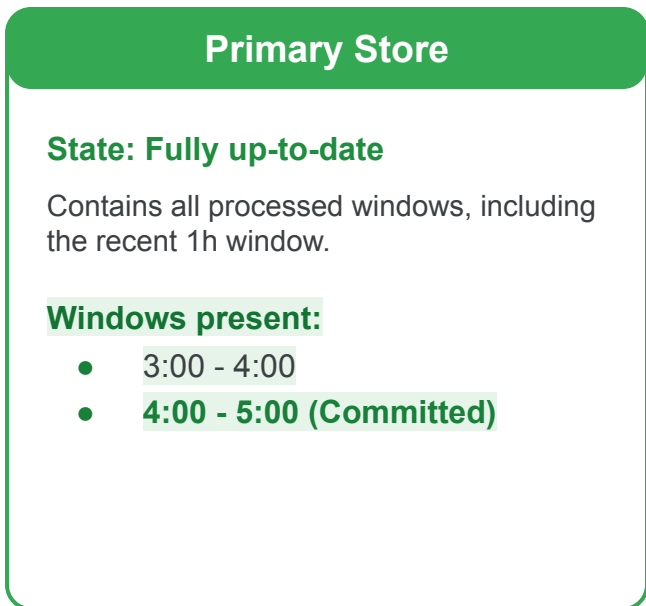
Targeting GA in Q4

Customer Implementation Requires Duplicating Resources

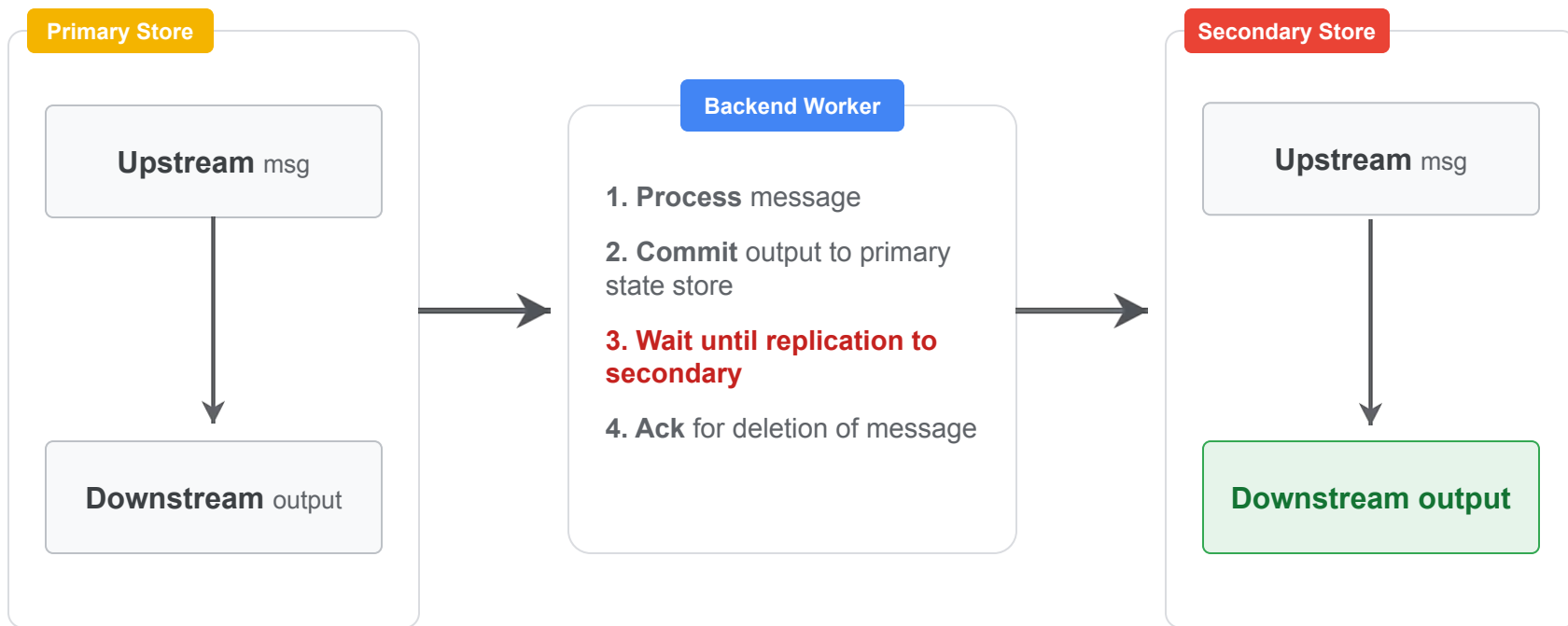
- Customers who want regional availability have to duplicate all resources.
- Both the Dataflow pipeline itself but also the source and sink data.
- Managing the failover between the primary and secondary is non-trivial as well.



We can replicate the state but not lossless



Dataflow exactly once



To achieve zero data loss, Dataflow ensures replication before issuing upstream deletes. If failover occurs, we are guaranteed that input exists or output was replicated. If both exist then standard deduplication resolves the conflict.

Iceberg Ingestion

Benchmarking Results

- **Throughput:** Dataflow Streaming initially took 3 hours to clear 1 TB of backlog.
- **File Size:** Originally produced parquet files of average 38 MB per file.
 - Note: Writing large files (e.g. 500 MB) is desirable since they are more optimal for reads.
- **Resources:** Dataflow had poor CPU utilization versus similar workloads with other IOs.

Improvements

- **Compression** → Use table-defined compression algorithm (defaulting to better zstd over gzip).
- **Metadata Caching** → Improve inefficient metadata lookups & prevent quota exhaustion for BigLake Metastore.
- **Direct Writes** → Avoid expensive GroupIntoBatches (GIB) by directly writing large bundles.
- **Autosharding** → Default range-based algorithm forces many shards, resulting in undesirable small files.

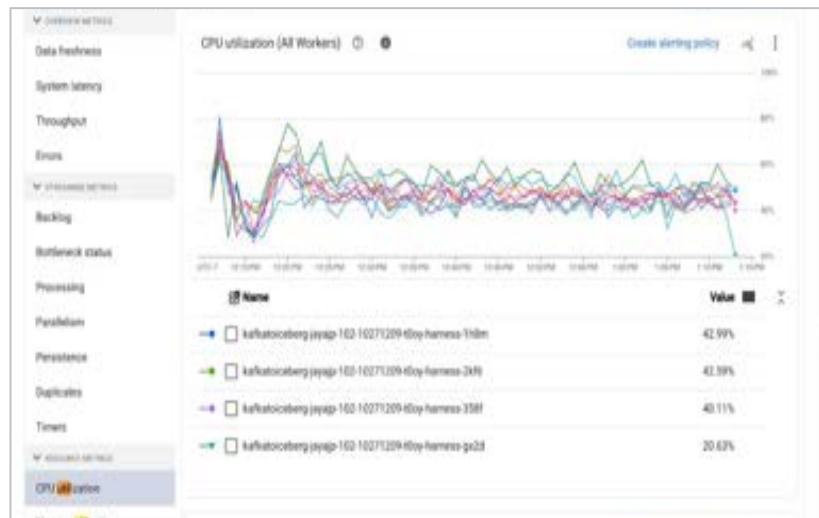
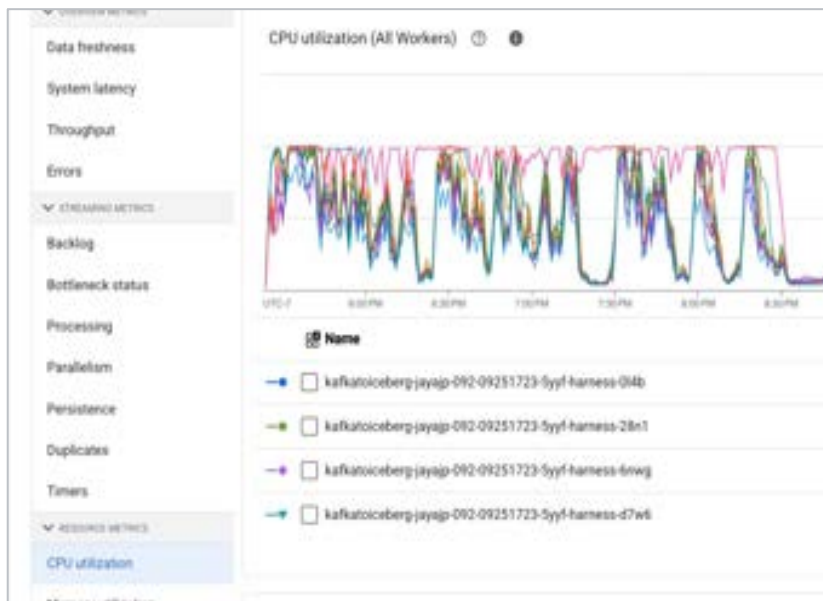
Compression: Overview

	Prior: gzip	Current: zstd
Year Released	1992	2016
Performance	Most reports show that zstd outperforms gzip on the following factors - compression ratio, CPU utilization, wall-time.	
Behavior with incompressible data	Still attempt to apply its compression algorithms leading to increased CPU usage and potentially larger files.	Zstd can detect incompressible data and switch to a "passthrough" mode.
Usage	Used on Beam before 2.70.0	Used on Beam after 2.70.0

<https://github.com/apache/beam/pull/36542>

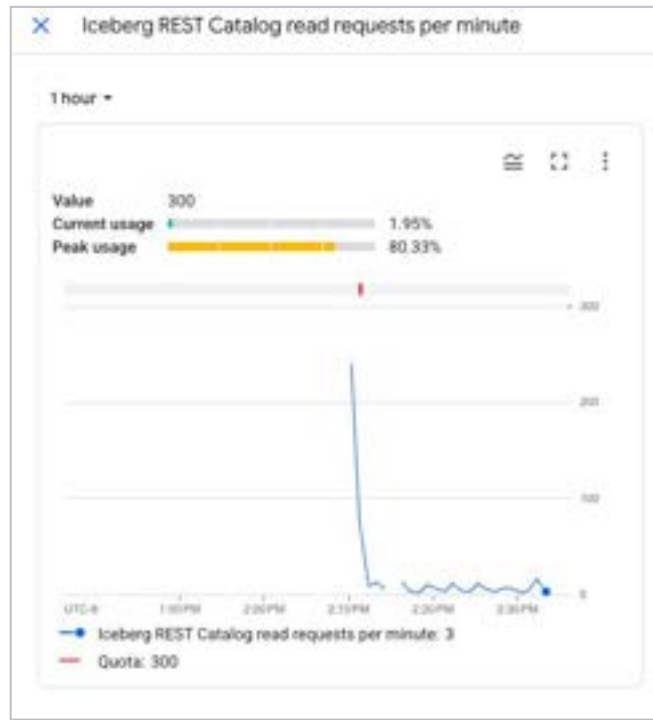
Compression Load Tests

Time to clear backlog 1.2tb Kafka backlog: 3 hours -> 1 hour 40 minutes

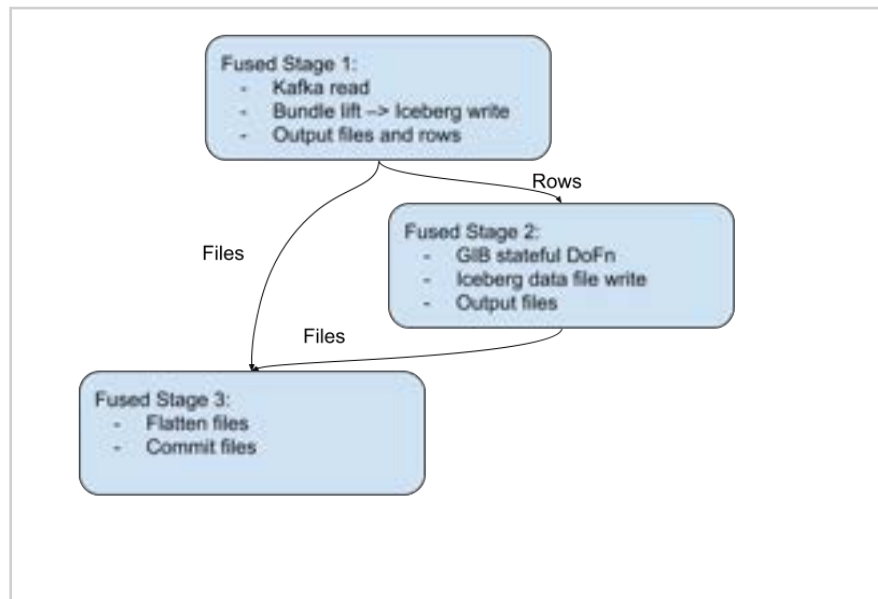
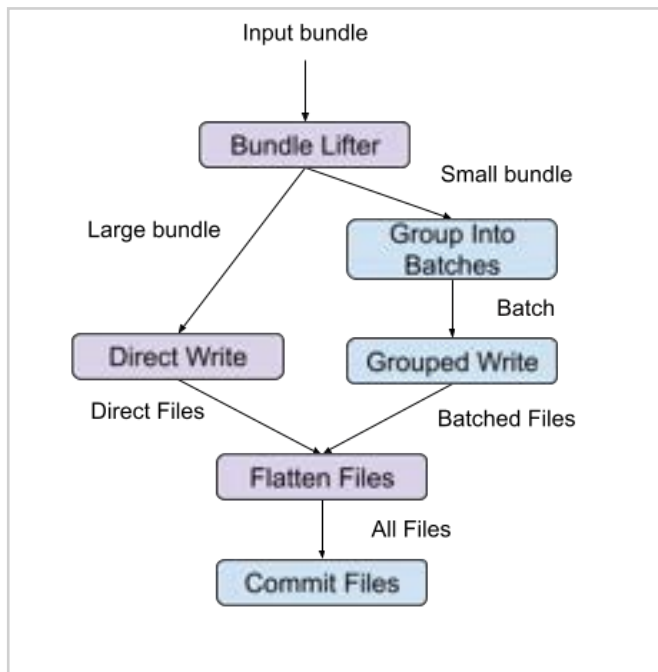


Metadata Caching

- Testing quickly exhausted BigLake Metastore Read Quota due to refreshing table metadata on each input.
- This blocked processing and reduced performance.
- Optimized by caching and refreshing periodically.



Direct Writes: Overview



Direct Writes: File Sizes

Custom Counters (Approximate)

Filter file Filter by counter name, value or step

Counter name	Value
committedDataFileByteSize_COUNT	475,385
committedDataFileByteSize_MAX	308,118,929
committedDataFileByteSize_MEAN	28,445,140
committedDataFileByteSize_MIN	410,619
committedDataFileRecordCount_COUNT	475,385
committedDataFileRecordCount_MAX	30,070
committedDataFileRecordCount_MEAN	2,775
committedDataFileRecordCount_MIN	40
snapshotsCreated	360

Baseline

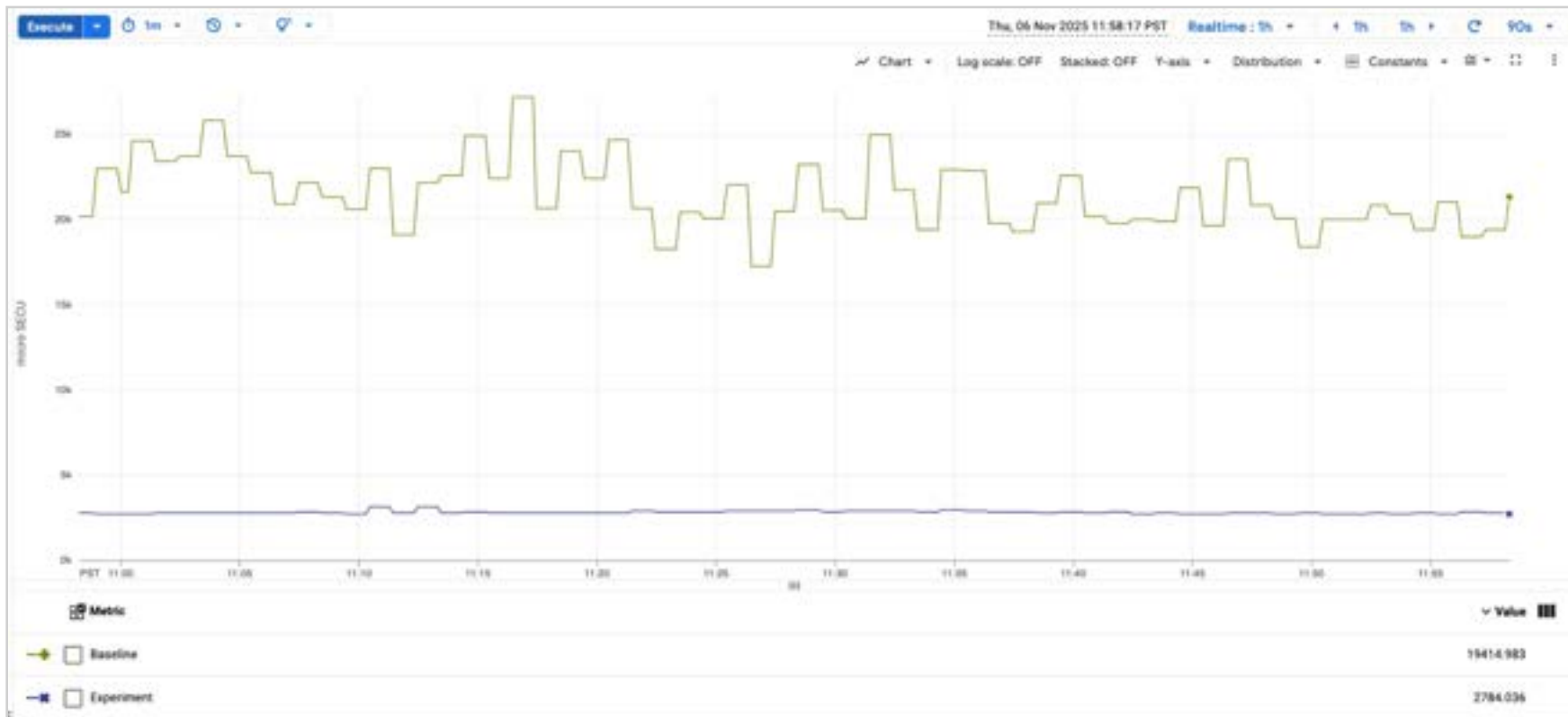
Custom Counters (Approximate)

Filter file Filter by counter name, value or step

Counter name	Value
committedDataFileByteSize_COUNT	34,188
committedDataFileByteSize_MAX	512,329,635
committedDataFileByteSize_MEAN	396,509,967
committedDataFileByteSize_MIN	11,036
committedDataFileRecordCount_COUNT	34,188
committedDataFileRecordCount_MAX	50,000
committedDataFileRecordCount_MEAN	38,697
committedDataFileRecordCount_MIN	1
snapshotsCreated	360

Experiment


Direct Writes: Cost




Motivation for Autosharding in Iceberg

- Iceberg Sink is Over-Parallelized
 - Each worker tries to 40 parallel files at once. Results in small files & slow queries.
- Throughput-based autosharding is a better fit.
 - Allows Iceberg sink to dynamically adjust parallelism.
 - It is already used for BigQuery and GCS IO sinks.

Iceberg Autosharding: File Sizes

Custom Counters (Approximate)	
 Filter Filter by counter name, value or step	
Counter name	Value
committedDataFileByteSize_COUNT	196,056
committedDataFileByteSize_MAX	48,638,921
committedDataFileByteSize_MEAN	3,946,035
committedDataFileByteSize_MIN	10,996
committedDataFileRecordCount_COUNT	196,056
committedDataFileRecordCount_MAX	4,747
committedDataFileRecordCount_MEAN	385
committedDataFileRecordCount_MIN	1
snapshotsCreated	46
activeIcebergWriters	15
dataFilesWritten	196,885

Baseline

Custom Counters (Approximate)	
 Filter Filter by counter name, value or step	
Counter name	Value
committedDataFileByteSize_COUNT	3,697
committedDataFileByteSize_MAX	536,856,634
committedDataFileByteSize_MEAN	212,985,254
committedDataFileByteSize_MIN	7,705,777
committedDataFileRecordCount_COUNT	3,697
committedDataFileRecordCount_MAX	52,396
committedDataFileRecordCount_MEAN	20,786
committedDataFileRecordCount_MIN	752
snapshotsCreated	46
activeIcebergWriters	13
dataFilesWritten	3,848

Experiment

Iceberg Autosharding: File Size Sanity Check

Baseline

Name	Size	Created
gcs-bucket-tomtepp-8f56eb-e02-4346-956d-c19279fbc6d1-hive-warehouse-1f_tomtepp_gd01_base-1.parquet	8.4 MB	Jan 6, 2025, 12:33:28 PM
gcs-bucket-tomtepp-8f56eb-e02-4346-956d-c19279fbc6d1-hive-warehouse-1f_tomtepp_gd01_base-2.parquet	15.1 MB	Jan 6, 2025, 12:33:12 PM
gcs-bucket-tomtepp-8f56eb-e02-4346-956d-c19279fbc6d1-hive-warehouse-1f_tomtepp_gd01_base-3.parquet	21.1 MB	Jan 6, 2025, 12:34:54 PM
gcs-bucket-tomtepp-8f56eb-e02-4346-956d-c19279fbc6d1-hive-warehouse-1f_tomtepp_gd01_base-4.parquet	28.5 MB	Jan 6, 2025, 12:33:22 PM
gcs-bucket-tomtepp-8f56eb-e02-4346-956d-c19279fbc6d1-hive-warehouse-1f_tomtepp_gd01_base-5.parquet	25.1 MB	Jan 6, 2025, 12:33:12 PM
gcs-bucket-tomtepp-8f56eb-e02-4346-956d-c19279fbc6d1-hive-warehouse-1f_tomtepp_gd01_base-6.parquet	32 MB	Jan 6, 2025, 12:34:06 PM
gcs-bucket-tomtepp-8f56eb-e02-4346-956d-c19279fbc6d1-hive-warehouse-1f_tomtepp_gd01_base-7.parquet	79 MB	Jan 6, 2025, 12:32:56 PM
gcs-bucket-tomtepp-8f56eb-e02-4346-956d-c19279fbc6d1-hive-warehouse-1f_tomtepp_gd01_base-8.parquet	30.3 MB	Jan 6, 2025, 12:33:54 PM
gcs-bucket-tomtepp-8f56eb-e02-4346-956d-c19279fbc6d1-hive-warehouse-1f_tomtepp_gd01_base-9.parquet	21.2 MB	Jan 6, 2025, 12:22:08 PM
gcs-bucket-tomtepp-8f56eb-e02-4346-956d-c19279fbc6d1-hive-warehouse-1f_tomtepp_gd01_base-10.parquet	15.4 MB	Jan 6, 2025, 12:33:30 PM
gcs-bucket-tomtepp-8f56eb-e02-4346-956d-c19279fbc6d1-hive-warehouse-1f_tomtepp_gd01_base-11.parquet	7.8 MB	Jan 6, 2025, 12:32:13 PM
gcs-bucket-tomtepp-8f56eb-e02-4346-956d-c19279fbc6d1-hive-warehouse-1f_tomtepp_gd01_base-12.parquet	6.9 MB	Jan 6, 2025, 12:32:17 PM

Experiment

Name	Size	Created
gcs-bucket-tomtepp-8f56eb-e02-4346-956d-c19279fbc6d1-hive-warehouse-2p01_exp-1.parquet	129.2 MB	Jan 6, 2025, 12:34:19 PM
gcs-bucket-tomtepp-8f56eb-e02-4346-956d-c19279fbc6d1-hive-warehouse-2p01_exp-2.parquet	130.2 MB	Jan 6, 2025, 12:34:10 PM
gcs-bucket-tomtepp-8f56eb-e02-4346-956d-c19279fbc6d1-hive-warehouse-2p01_exp-3.parquet	129.7 MB	Jan 6, 2025, 12:34:16 PM
gcs-bucket-tomtepp-8f56eb-e02-4346-956d-c19279fbc6d1-hive-warehouse-2p01_exp-4.parquet	133.8 MB	Jan 6, 2025, 12:34:17 PM
gcs-bucket-tomtepp-8f56eb-e02-4346-956d-c19279fbc6d1-hive-warehouse-2p01_exp-5.parquet	130.7 MB	Jan 6, 2025, 12:34:04 PM
gcs-bucket-tomtepp-8f56eb-e02-4346-956d-c19279fbc6d1-hive-warehouse-2p01_exp-6.parquet	536.9 MB	Jan 6, 2025, 12:32:53 PM
gcs-bucket-tomtepp-8f56eb-e02-4346-956d-c19279fbc6d1-hive-warehouse-2p01_exp-7.parquet	460.9 MB	Jan 6, 2025, 12:33:31 PM
gcs-bucket-tomtepp-8f56eb-e02-4346-956d-c19279fbc6d1-hive-warehouse-2p01_exp-8.parquet	536.9 MB	Jan 6, 2025, 12:32:53 PM
gcs-bucket-tomtepp-8f56eb-e02-4346-956d-c19279fbc6d1-hive-warehouse-2p01_exp-9.parquet	468.8 MB	Jan 6, 2025, 12:33:28 PM
gcs-bucket-tomtepp-8f56eb-e02-4346-956d-c19279fbc6d1-hive-warehouse-2p01_exp-10.parquet	126.1 MB	Jan 6, 2025, 12:34:16 PM
gcs-bucket-tomtepp-8f56eb-e02-4346-956d-c19279fbc6d1-hive-warehouse-2p01_exp-11.parquet	536.9 MB	Jan 6, 2025, 12:32:43 PM
gcs-bucket-tomtepp-8f56eb-e02-4346-956d-c19279fbc6d1-hive-warehouse-2p01_exp-12.parquet	394.1 MB	Jan 6, 2025, 12:33:15 PM

Offset Deduplication

Offset Deduplication

- Problem:
 - Redistribute is expensive due to exactly-once cost and latency.
 - Can't use allow duplicates optimization because you need exactly-once.
- Solution:
 - Utilize source metadata, such as message offsets, to make shuffle cheaper.
 - Enabled for `KafkaIO.Read.withRedistribute()` on Beam 2.70+

Implementation

- Beam SDK
 - Reader interface to provide per record offsets to Dataflow runner.
 - Source interface to provide offset limit for checkpoints.
- Beam Runner
 - Message offsets for records committed to backend.
 - Offset limit for source states committed to backend.

Results

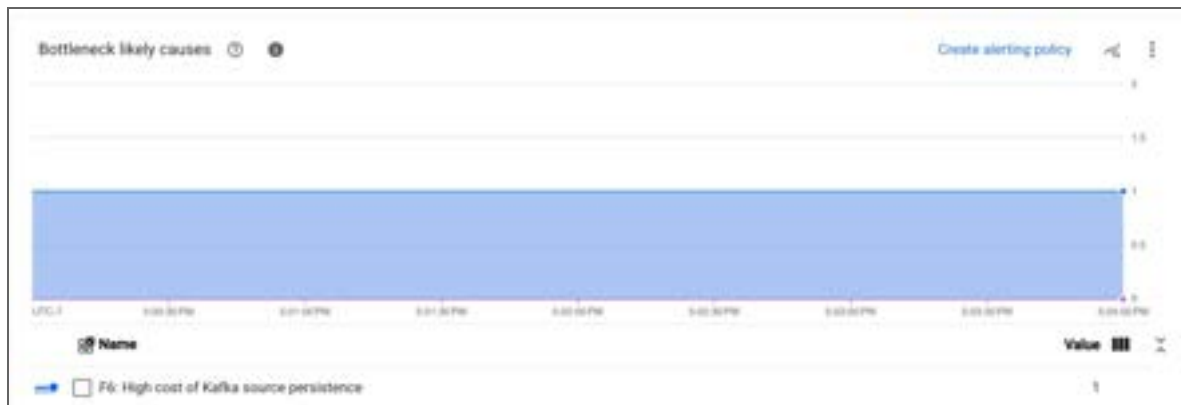


Cost



Latency

Observability



Bottleneck Detector

Helps identify when the Kafka source persistence is slow and the docs linked below help guide to use the new feature.

Recommendations

Recommendation also shown based on the analysis of bottleneck detector.

The screenshot shows the 'Recommendations (14)' tab in the Dataflow UI. The top navigation bar includes 'kafkareadwrit...', 'Stop', 'Create Snapshot', 'Archive', 'Import as pipeline', 'Share', and 'Send feedback'. Below the navigation bar, there are tabs for 'Job Graph', 'Execution Details', 'Job Metrics', 'Cost', 'Recommendations (14)', 'Autoscaling', and 'Advanced'. The 'Recommendations (14)' tab is selected and highlighted with a red box. Below the tabs, there is a list of recommendations. One recommendation is highlighted with a red box and contains the following text:

Jan 29, 2026, 2:50:18 PM Redistribute requires persistence of Kafka outputs as part of the shuffle phase, consider reducing latency and cost with offset deduplication mode.

Stage F0 has high volume of Kafka outputs. To minimize the latency and cost of the shuffle, use offset deduplication mode. For additional information about Kafka Read best practices for parallelism, see <https://cloud.google.com/dataflow/docs/guides/read-from-kafka#parallelism>. [Learn more](#)

High Cardinality

High Cardinality Improvements

Strengths

Dataflow Streaming's ability to handle high scale and cardinality is a key competitive advantage.

Key Customers

- Top customers run large stateful jobs.
- **Throughput:** Millions of messages/s
- **Cardinality:** 100s of million keys

Challenges

- Some of these jobs stress the limits of our backend system.
- Resource utilization on large jobs is a concern.
- Customers want better performance at a lower cost

High Cardinality Improvements - Projects



Multi-Key Bundles

Small bundle sizes hinder performance of streaming pipelines.

Enable large cross key bundles, significantly improving batching and combiner lifting efficiency.



Reduce IO & Resource usage

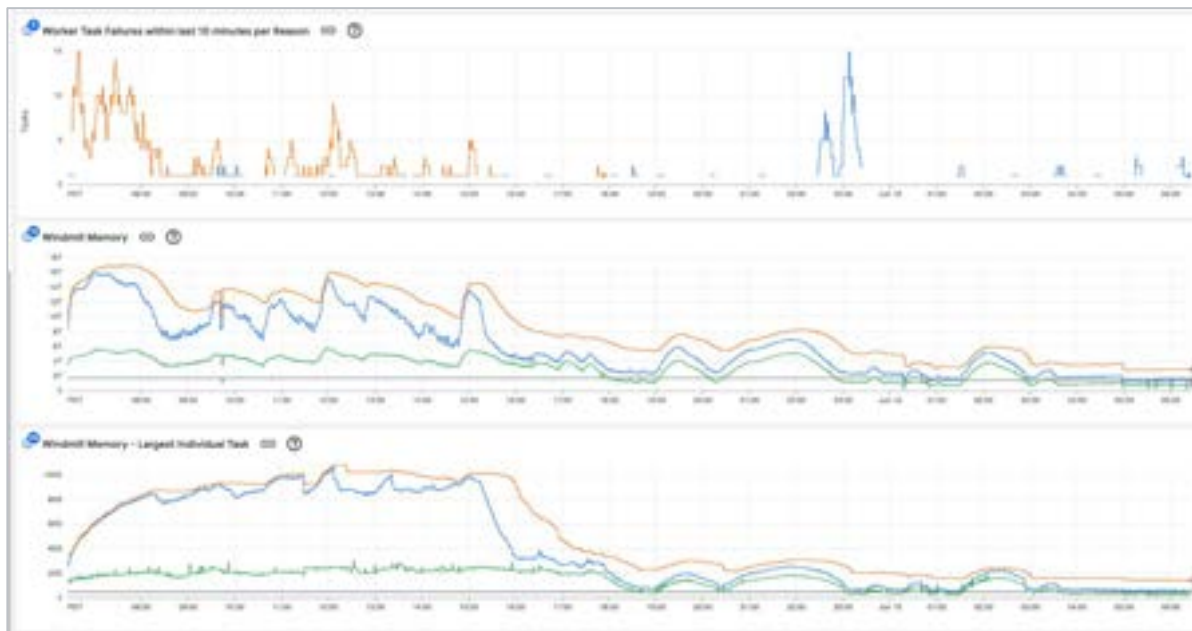
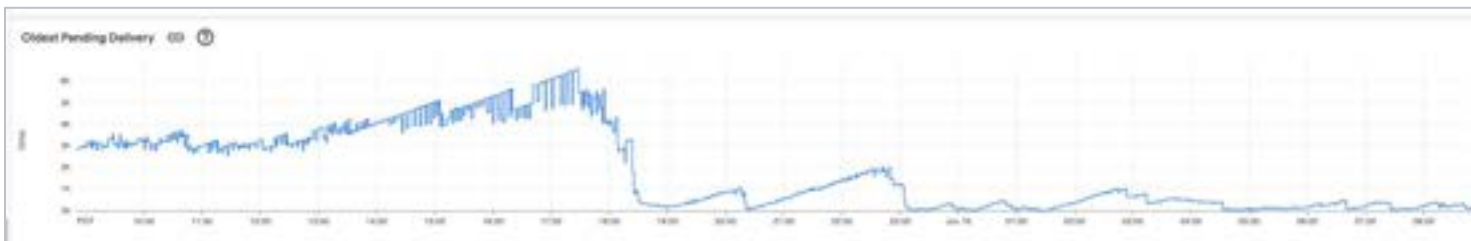
- Reducing state reads by improving state cache.
- Optimizing user worker budgeting for better utilization.

Shuffle Enhancements

Shuffle Enhancements

- Problem:
 - Upstream backend worker not aware of downstream worker resource limitations.
 - Sending individual requests on new RPCs use more resources.
 - Ordering limitations with individual requests.
- Solution:
 - Using long-lived connections reduces resource usage, provides simple way to communicate budgets between pairs, and allows pipelining

First Milestone Results



Parallel Updates

Streaming Job Updates

Current Challenges in Dataflow Streaming Job Updates:

- **Extended Downtime:** In-place updates currently result in substantial service interruptions, typically requiring a 15-minute maintenance window.
- **Job Incompatibility:** Frequent compatibility conflicts require specialized, manual workarounds during the update process.

Project Objective:

- To optimize customer onboarding by reducing manual intervention and lowering operational overhead. Provide declarative APIs.

Parallel Update Features

- Automated stop and replace pipelines streaming job update.
- Automated parallel pipeline updates have been extended with
 - An option to auto cancel the previous job after a drain timeout.
 - Allow creating new parallel jobs with the same name.
- Ability to choose job update strategy: in place vs parallel job update.

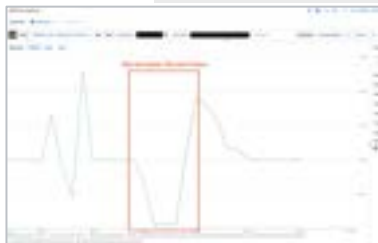
Stop and replace pipelines:

Stop the old job and then start the new job **with** preallocated compute resources

```
parallel_replace_job_max_stop_duration=DRAIN_TIMEOUT_DURATION
```

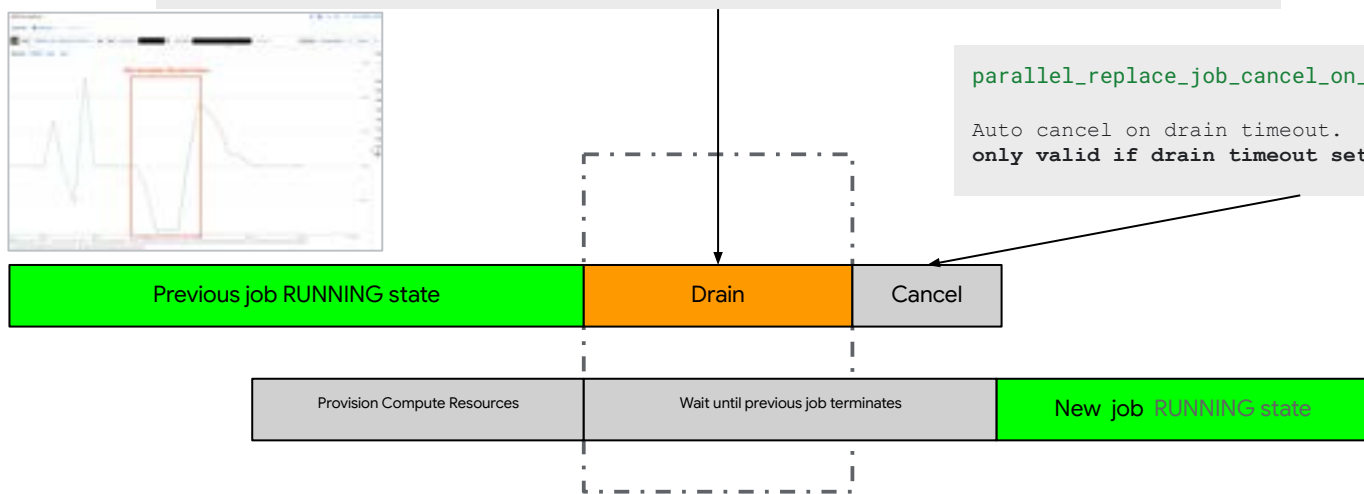
Optional drain timeout, set based on production usage.

Use case: Auto cancel [stuck drain](#) e.g. currently [looping timers](#) can cause drain to never finish.



```
parallel_replace_job_cancel_on_drain_timeout=true
```

Auto cancel on drain timeout. **Default true but only valid if drain timeout set.**



Must not set `parallel_replace_job_min_parallel_pipelines_duration` option.

- For now the `parallel_replace_job_preallocate_compute_resources=true` is default.
- New Job state remains `RUNNING` even during the old job is stopped.

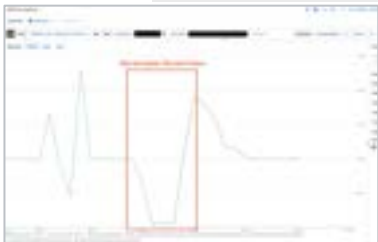
Stop and replace pipelines:

Stop the old job and then start the new job **without** preallocate compute resources

```
parallel_replace_job_max_stop_duration=DRAIN_TIMEOUT_DURATION
```

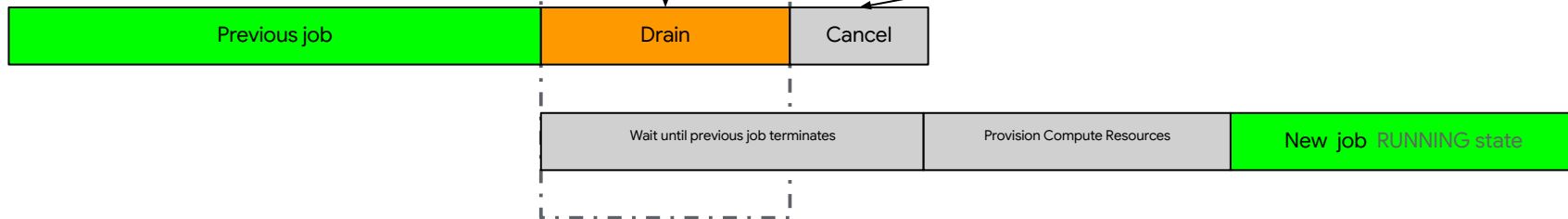
Optional drain timeout, set based on production usage.

Use case: Auto cancel [stuck drain](#) e.g. currently [looping timers](#) can cause drain to never finish.



```
parallel_replace_job_cancel_on_drain_timeout=true
```

Auto cancel on drain timeout. **Default true but only valid if drain timeout set.**



Must not set `parallel_replace_job_min_parallel_pipelines_duration` option.

- Explicitly set the `parallel_replace_job_preallocate_compute_resources=false` option.
- New Job state remains **PENDING** until old job is stopped.

Run parallel pipelines (continued):

Run the new job in parallel with the old one. Stop the old job once the new one is ready

```
parallel_replace_job_max_stop_duration=DRAIN_TIMEOUT_DURATION
```

Optional drain timeout, set based on production usage.

Use case: Auto cancel [stuck drain](#) e.g. currently [looping timers](#) can cause drain to never finish.



```
parallel_replace_job_cancel_on_drain_timeout=true
```

Auto cancel on drain timeout. **Default true but only valid if drain timeout set.**



```
parallel_replace_job_min_parallel_pipelines_duration=DURATION
```

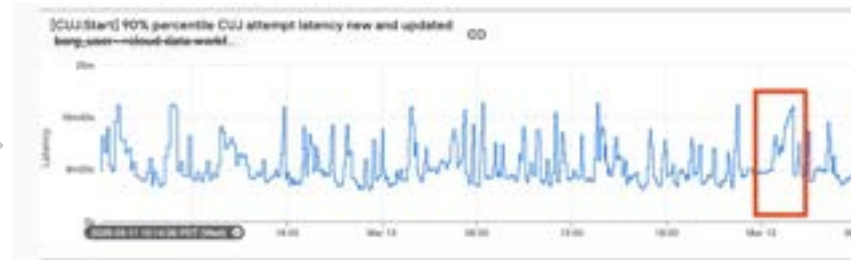
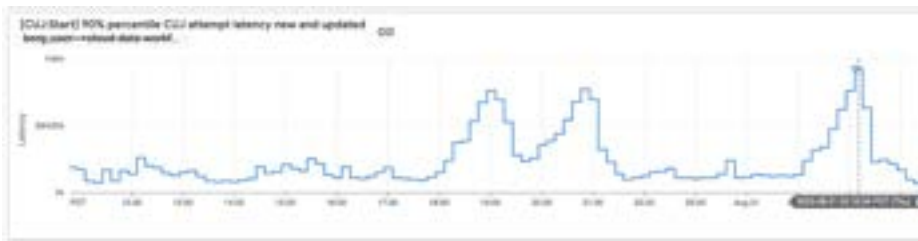
Recommended value `0s`. This example is for non-zero duration.

Two Jobs will be in `running` state for `DURATION` before previous job starts `draining`.

Streaming Engine Reliability

Streaming Engine Reliability Work

- Better support for launching large numbers($O(\text{thousands})$) of jobs at once
 - 73% faster P90 job start time



Observability

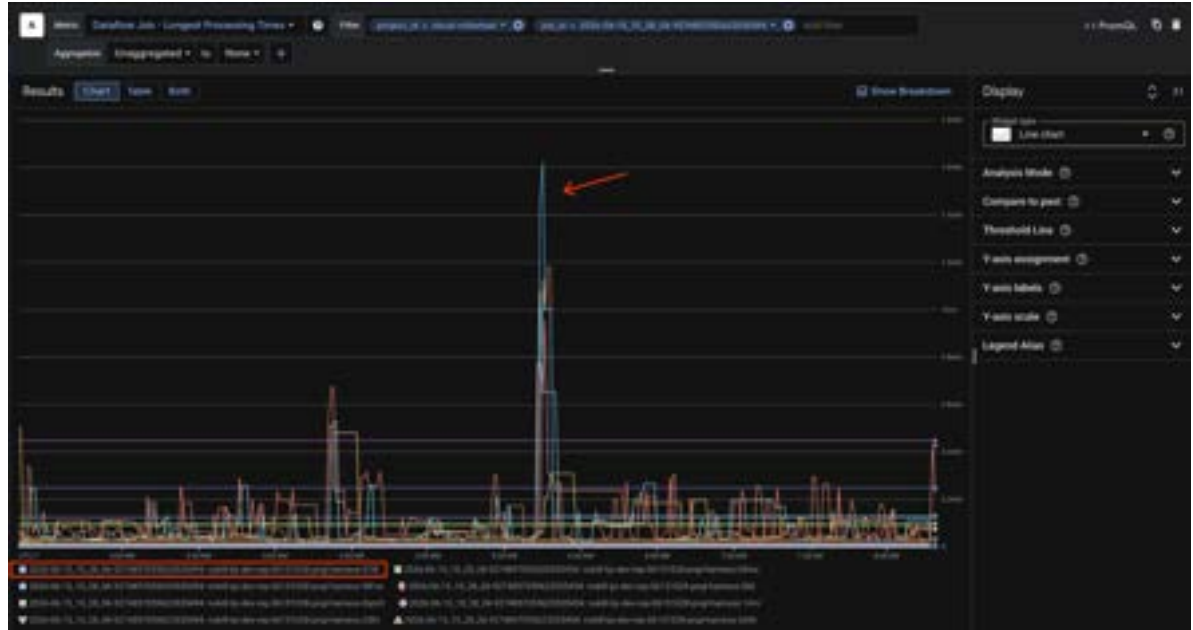
Metrics

Observability: Metrics

- `job/backlogged_keys`: The number of backlogged keys for a bottleneck stage.
- `job/recommended_parallelism`: The recommended parallelism for a stage to reduce bottlenecking.
- `job/is_bottleneck`: Whether a stage is a bottleneck, and the bottleneck kind and likely cause for it.
- `job/longest_processing_time`: The time in microseconds of the longest processing time for a user worker.

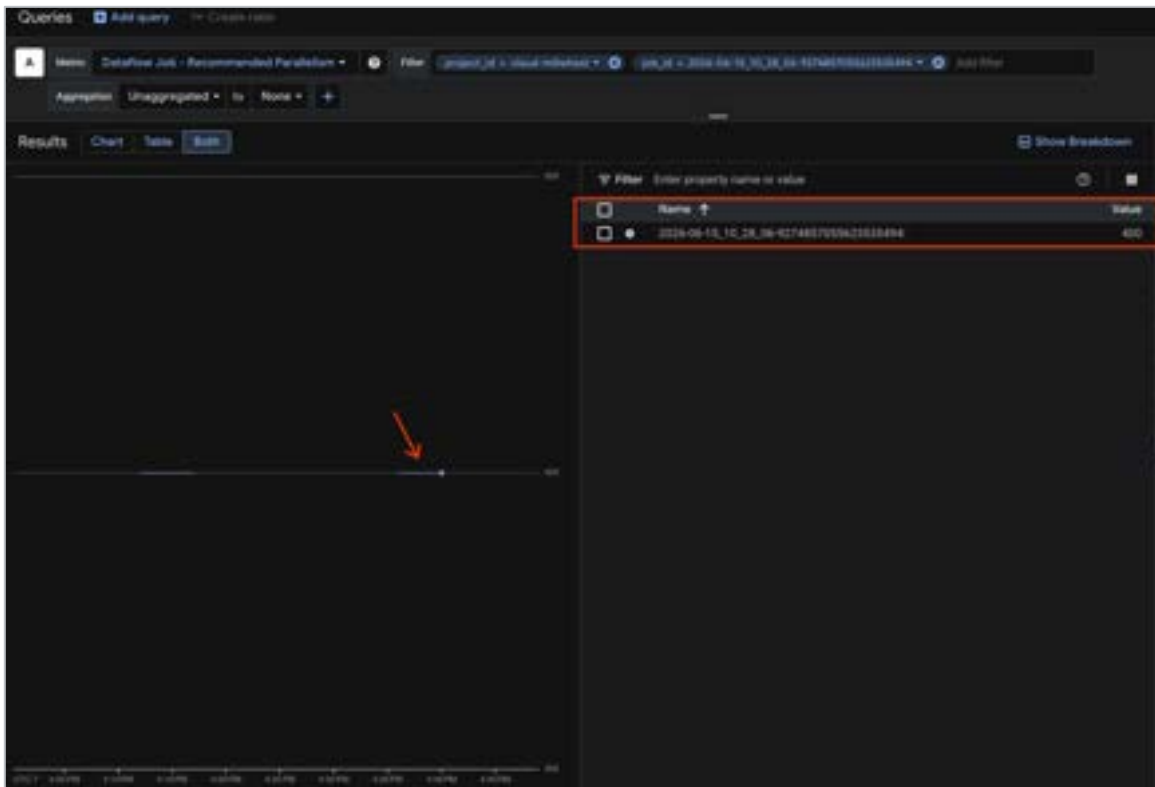
Observability: Metrics

Which worker is slowest?



Observability: Metrics

What parallelism is recommended for the job?



Turnkey Alerts

Observability: Turnkey Alerts

- Predefined suite of alerts to help monitoring your pipelines
- Launching with 2 alerts
 - **Default Policy: Estimated Backlog:** Alerts when the `job/estimated_backlog_processing_time` exceeds a threshold (e.g., 30 minutes).
 - **Default Policy: SECU Usage:** Alerts on significant relative changes (e.g., 50% change) in `job/total_secu_usage` to help monitor costs.
- Alerts can be customized in the Cloud Monitoring console
 - Change parameters or add/subtract new alert queries.
- Launch a job with turnkey alerts via the UI or the cli (`--enable-turnkey-alerts`).
 - Adding turnkey alerts to running jobs is also supported

Advanced Tab

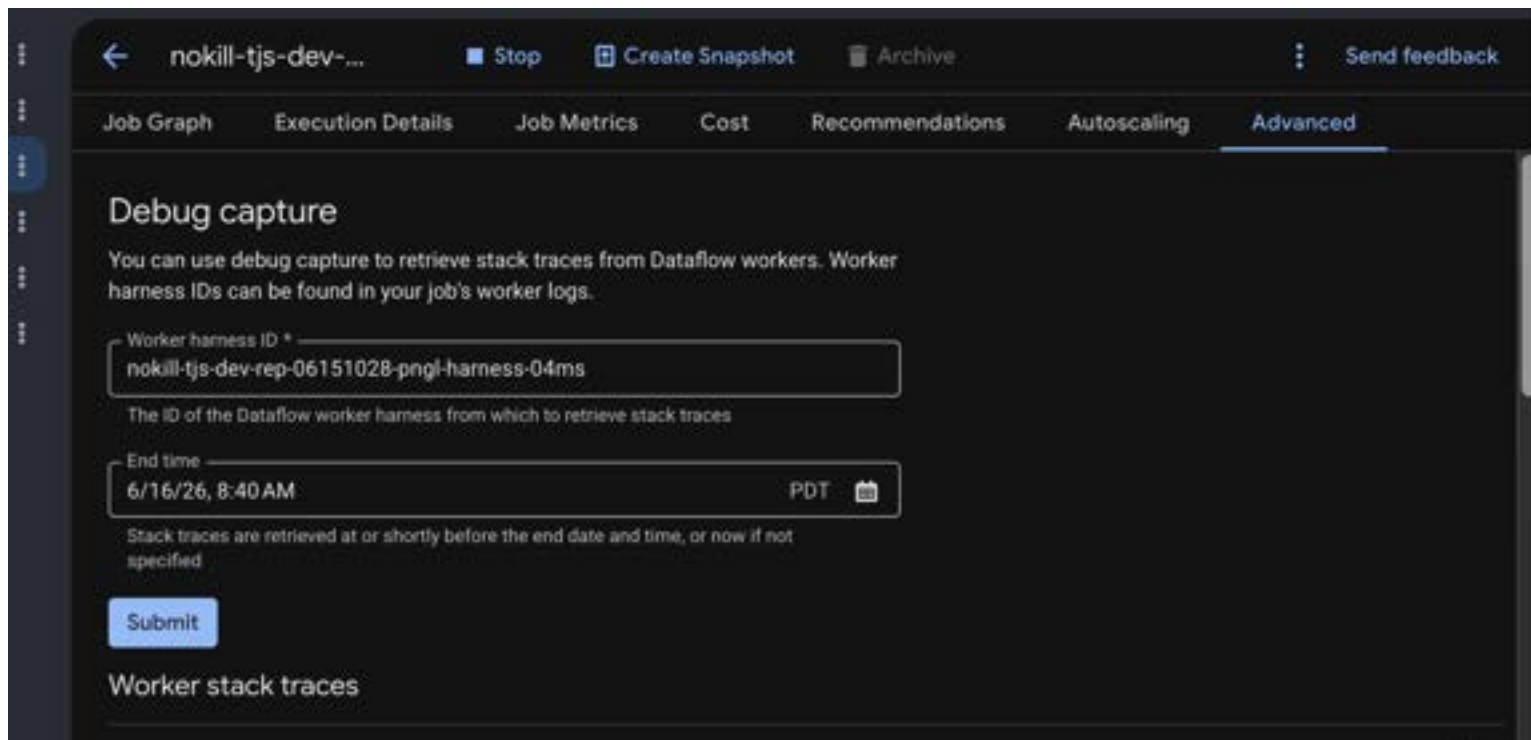
Observability: Advanced Debugging

- Context: Processing was slow by a worker and I want more details.
- Problem: Currently need to dig through logs, only >5m processing is logged.
- Solution: Provide user worker thread stacks in the Job UI

Observability: Advanced Debugging

The screenshot displays a job monitoring interface for a job named "nokill-tjs-dev-...". The top navigation bar includes a back arrow, the job name, and several action buttons: "Stop", "Create Snapshot", "Archive", and "Send feedback". Below this, a secondary navigation bar contains tabs for "Job Graph", "Execution Details", "Job Metrics", "Cost", "Recommendations", "Autoscaling", and "Advanced". An orange arrow points to the "Advanced" tab. On the left, a "Job steps view" dropdown menu is set to "Graph view". On the right, there is a "Clear selection" button. The main area shows a partial view of a job graph with two nodes: "PubsubIO.Read" (Running, Data Lag: 20.59 sec, 22% Total Walltime, Max Op Latency: < 1 sec, 1 stage) and "Window.Into()" (94,544 elements/s, Data Lag: 20.59 sec).

Observability: Advanced Debugging



The screenshot shows the 'Advanced' tab of the Google Cloud Dataflow console for a job named 'nokill-tjs-dev-...'. The navigation bar includes 'Job Graph', 'Execution Details', 'Job Metrics', 'Cost', 'Recommendations', 'Autoscaling', and 'Advanced'. The 'Advanced' tab is selected. The main content area is titled 'Debug capture' and contains the following text: 'You can use debug capture to retrieve stack traces from Dataflow workers. Worker harness IDs can be found in your job's worker logs.' Below this text is a form with two input fields. The first field is labeled 'Worker harness ID *' and contains the text 'nokill-tjs-dev-rep-06151028-pngl-harness-04ms'. Below this field is a tooltip that reads 'The ID of the Dataflow worker harness from which to retrieve stack traces'. The second field is labeled 'End time' and contains the text '6/16/26, 8:40 AM'. To the right of this field is a dropdown menu set to 'PDT' and a calendar icon. Below the 'End time' field is a tooltip that reads 'Stack traces are retrieved at or shortly before the end date and time, or now if not specified'. At the bottom of the form is a blue 'Submit' button. Below the 'Submit' button is the text 'Worker stack traces'.

← nokill-tjs-dev-... Stop Create Snapshot Archive Send feedback

Job Graph Execution Details Job Metrics Cost Recommendations Autoscaling **Advanced**

Debug capture

You can use debug capture to retrieve stack traces from Dataflow workers. Worker harness IDs can be found in your job's worker logs.

Worker harness ID *
nokill-tjs-dev-rep-06151028-pngl-harness-04ms

The ID of the Dataflow worker harness from which to retrieve stack traces

End time
6/16/26, 8:40 AM PDT

Stack traces are retrieved at or shortly before the end date and time, or now if not specified

Submit

Worker stack traces

Observability: Advanced Debugging

The screenshot displays a 'Worker stack traces' window with a search filter and a table of thread information. The table has three columns: 'Thread count', 'Thread state', and 'Stack content'. There are four rows of data, each representing a different thread state. The first row shows 300 threads in a TIMED_WAITING state, with stack traces involving 'Unsafe.park(Native Method)', 'java.util.concurrent.locks.AbstractQueuedSynchronizer\$ConditionObject.awaitNanos()', and 'ThreadPoolExecutor\$Worker.run()'. The second row shows 7 threads in a TIMED_WAITING state, with stack traces involving 'ScheduledThreadPoolExecutor\$DelayedWorkQueue.take()' and 'ThreadPoolExecutor\$Worker.run()'. The third row shows 4 threads in a TIMED_WAITING state, with stack traces involving 'java.util.concurrent.SynchronousQueue\$Transferer.await()' and 'ThreadPoolExecutor\$Worker.run()'. The fourth row shows 4 threads in a TIMED_WAITING state, with stack traces involving 'java.util.concurrent.BlockingArrayQueue.poll()' and 'ThreadPoolExecutor\$Worker.run()'. At the bottom, there is a 'Logs' section with a 'Show' button and a notification icon. The footer indicates 'Rows per page: 50' and '1 - 21 of 21'.

Thread count	Thread state	Stack content
300	TIMED_WAITING	java.base@21.0.11/jdk.internal.misc.Unsafe.park(Native Method) java.base@21.0.11/java.util.concurrent.locks.AbstractQueuedSynchronizer\$ConditionObject.awaitNanos() java.base@21.0.11/java.util.concurrent.LinkedBlockingQueue.poll() java.base@21.0.11/java.util.concurrent.ThreadPoolExecutor\$Worker.run() java.base@21.0.11/java.lang.Thread.run(Thread.java:1583)
7	TIMED_WAITING	java.base@21.0.11/jdk.internal.misc.Unsafe.park(Native Method) java.base@21.0.11/java.util.concurrent.ScheduledThreadPoolExecutor\$DelayedWorkQueue.take() java.base@21.0.11/java.util.concurrent.ThreadPoolExecutor\$Worker.run() java.base@21.0.11/java.lang.Thread.run(Thread.java:1583)
4	TIMED_WAITING	java.base@21.0.11/jdk.internal.misc.Unsafe.park(Native Method) java.base@21.0.11/java.util.concurrent.SynchronousQueue\$Transferer.await() java.base@21.0.11/java.util.concurrent.ThreadPoolExecutor\$Worker.run() java.base@21.0.11/java.lang.Thread.run(Thread.java:1583)
4	TIMED_WAITING	java.base@21.0.11/jdk.internal.misc.Unsafe.park(Native Method) java.base@21.0.11/java.util.concurrent.BlockingArrayQueue.poll() java.base@21.0.11/java.util.concurrent.ThreadPoolExecutor\$Worker.run() java.base@21.0.11/java.lang.Thread.run(Thread.java:1583)

Hot Key Detection

New for this year: Opt-in logging of unencoded Hot Keys

- The Problem:
 - A Key is the fundamental unit of parallelism in Dataflow. As a consequence, processing for a single key is single threaded.
 - This means that if one key is overloaded compared to the others, the pipeline may fall behind since we are not able to parallelize efficiently.

Opt-in Logging of Hot Keys

- The way to fix hot keys is normally by either updating pipeline code or by updating upstream data sources but the challenge had been figuring out what the Hot Key is.
- By default, Dataflow shows a hashed version of the key.
- Now, opt a job into Hot-Key Logging with `--hotKeyLoggingEnabled=true`

Hot Key Logging

The screenshot displays the Databricks job execution interface. On the left, a job graph shows a sequence of steps: '100 elements', 'GroupByKey', 'Window 1', 'Simulate', and 'GroupByKey 1'. The 'Window 1' step is highlighted with a red arrow pointing to its details on the right. The 'Step info' panel for 'Window 1' shows the following information:

- Step name: Window 1
- System watermark: February 8, 2025 at 2:36:53 PM (UTC-8:00:00) [Log]
- Data watermark: February 8, 2025 at 2:12:19 PM (UTC-8:00:00) [Log]
- Wait time (% of Total): 0 sec (1%)
- Bottomlock status: ⚠ Processing is ongoing, but falling behind. This step is likely to be a bottleneck due to hot keys or insufficient key parallelism. A known issue with the Streaming Engine has been identified. [Bottomlock troubleshooting guide](#)
- Max operation latency: 1 sec
- Key parallelism: 1182 ⚠ 2 missing keys are bottlenecked with significant backlog. Recommended key parallelism is at least 180 evenly loaded keys.

Below the step info, the 'Input collections' section shows a line graph for 'Throughput (elements/sec)' over time, with a peak around 1:00:00. The graph is labeled 'GroupByKeyWindowKeys ReadStream (out) RTT 15/s'.

Hot-Key Logging

The screenshot displays a web-based interface for viewing project logs. At the top, there is a search bar and several filter buttons: "Definition step", "All log names", "All severities", "Correlate by", and "Filter". A "Run query" button is located in the top right corner. Below the filters, a code editor shows the following SQL query:

```
1 --resource_type='dataflow_step'  
2 --resource_name='job_run-2024-02-09_14_09_40-163476236217194440'  
3 --sqlPayload='logging'>org.apache.beam.runners.dataflow.worker.HotKeyLogger'
```

Below the code editor, there is a section for "Example queries" and "Query language guide". A "Timeline" view shows a horizontal axis with a blue bar representing the log duration and two yellow vertical markers. Below the timeline, a "Results" section displays two log entries:

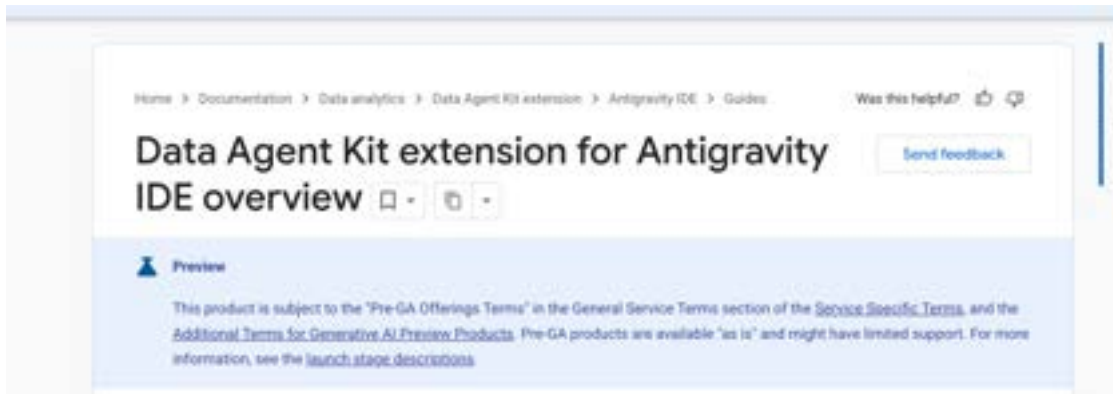
- 1 2024-02-09 14:09:46.404 [E] [H] A hot key 'T' was detected in step [GroupByKey@20240209/ResultWriter] with age of 318. This is a symptom of key distribution being skewed. To fix, please ensure...
- 2 2024-02-09 14:09:47.307 [E] [H] A hot key 'E' was detected in step [GroupByKey@20240209/ResultWriter] with age of 488. This is a symptom of key distribution being skewed. To fix, please ensure...

Each log entry includes a "Showing logs for last 1 hour from 2/9/24, 1:07PM to 2/9/24, 2:07PM" and "Expand time for 1 hour" button.

AI Pipeline Writing/Debugging

What is it?

- Data Agent Kit extension
- Brand new way to write and monitor your pipelines
- Get the wisdom of the whole team at your fingertips
- Programmed to intelligently traverse through documentation and Pantheon signals
- Built for Antigravity



Demo time

- Example use case of asking AntigraVity to help debug a poorly performing pipeline.

pubsubtoke..._E3111194 - 0

pantheon.corp.google.com/dataflow/jobs/...-west1-2026-05-11_13_25-413034484276277485,step=waitfab-JOB_GRAPH/buttonfab-JOB_LOGS/expand/buttonPan...

Google Cloud | Matthew | Search (Q) for resources, docs, products, and more

Action Required: One or more projects enabled with Google API (googleapis.com) have unattached API keys. To prevent unauthenticated usage and costs, restrict these keys or switch to authorization keys in APIs & Services - Credentials. This banner may persist for 24 hours after you address the issue. [Read more](#)

Dataflow / Jobs / Dataflow job details

Overview | pubsubtoke... | Stop | Create Snapshot | Import as pipeline | Share | Send feedback

Monitoring | Job Graph | Execution Details | Job Metrics | Cost | Recommendations | Autoccaling | Advanced

Jobs | Pipelines | Workbench | Snapshots

Job Graph | Graph view | Clear selection

Make KV1
Running
Data Lag: 34 ms (2.41 sec)
10% Top machines
Max Op Latency: 1.1 sec
1 Miss

GroupByKey_0000Keys
Running

Logs | Info | 1

Job Logs | Worker Logs | Diagnostics | Data Sampling

Severity: Default | Filter: Search of fields and values | 1:05 PM - 2:01 PM

Search: Type

2024-05-11 12:24:40 144 100 Worker starting Dataflow START and stop steps

2024-05-11 12:24:40 416 100 Occupying stage job.

2024-05-11 12:24:44 144 100 Starting worker pool setup

2024-05-11 12:24:44 487 100 Starting worker pool setup

2024-05-11 12:24:44 489 100 Starting 38 workers in us-west1...

2024-05-11 12:24:50 485 100 Your project already contains 196 Dataflow-created service descriptions. To see user service of the form `example.googleapis.com/`

2024-05-11 12:24:54 834 100 Available resources are up for region: `projects/1694646262/regions/us-west1-compute-`

2024-05-11 12:24:57 417 100 Autoccaling: Reduce the number of workers to 38 so that the pipeline can catch up with the backlog and keep up with the tra-

2024-05-11 12:24:58 434 100 Workers have started successfully.

2024-05-11 12:27:19 349 100 All workers have finished the startup processes and begin to receive work requests.

2024-05-11 12:40:34 271 100 Worker configuration: `xl-standard-4` in `us-west1-c`.

2024-05-11 12:40:36 494 100 Your project already contains 196 Dataflow-created service descriptions. To see user service of the form `example.googleapis.com/`

2024-05-11 12:40:36 583 100 Autoccaling: Reduce the number of workers to 38 so that the pipeline can catch up with the backlog and keep up with the tra-

Warning logs for this pipeline in query. To view more results open your query.

Stop sharing

Job info

Job name: pubsubtokekeys-1776211121194
Job ID: 2024-05-11_13_25-413034484276277485
Job type: Streaming
Job status: Running
SDK version: Apache Beam SDK for Java 2.12.0-SNAPSHOT
A newer version of the SDK family exists and updating is recommended. [Learn more](#) 12
Job region: us-west1
Service zones: us-west1-c
Worker location: us-west1
Current workers: 38
Latest worker status: Worker pool started
Debugger status: No active debugger
Start time: May 11, 2024, 1:05:27 PM GMT-7
Elapsed time: 36 min 20 sec
Encryption type: Google-managed
Job profile: [View in Thriller](#)
Dataflow Prime: Disabled
Knowledge Catalog: Enabled
Lineage: Enabled
Runner v2: Disabled
Streaming Engine: Enabled
Vertical Autoccaling: Disabled
Streaming Mode: Exactly once

Resource metrics

Current vCPUs: 60
Total vCPU time: 47 193 vCPU-hr
Current memory: 300 GB
Total memory time: 1 hr 9/3 GB-hr

Show activity panel

Thank you!

Questions? Feel free to reach out!

[linkedin.com/in/tomstepp](https://www.linkedin.com/in/tomstepp)

[linkedin.com/in/ryan-wigg](https://www.linkedin.com/in/ryan-wigg)

