Scaling Iceberg Ingestion with Apache Beam

Tom Stepp

Contributors:
Jayaj Poudel
Ahmed Abualsaud

Iceberg

What

- **Open Table Format:** Brings structured, SQL-like table capabilities directly to data lakes.
- **Translation Layer:** Bridge between compute engines and physical cloud storage.
- **Metadata:** Tracks data via lightweight metadata files instead of slow scans.
- **Snapshots:** Every change creates a snapshot for consistent reads across data.

Why

- **Engine Agnostic:** Open standard prevents vendor lock-in (Spark, Flink, Dataflow, etc.).
- **High Performance:** File-level metadata drastically speeds up massive-scale queries.
- **ACID Compliance:** Safe, concurrent read/write operations without data corruption.
- **Seamless Schema Evolution:** Add or drop columns instantly without rewriting underlying data.
- **Time Travel:** Native support for querying and rolling back to historical data snapshots.

Benchmarking

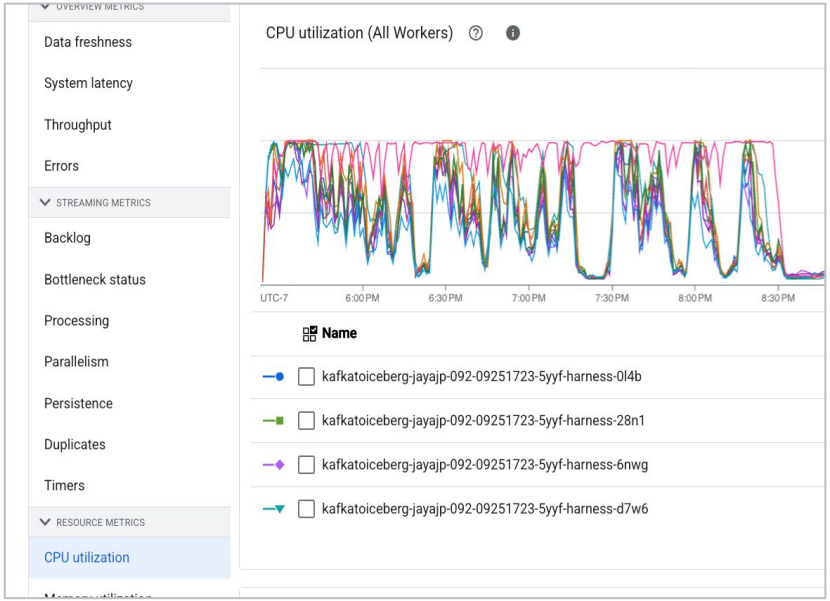
Setup

- **Pipelines:** Read from Kafka source and write to Iceberg sink.
    - **Flink**: Pipeline is written with Flink Table API.
    - **Beam**: Pipeline is written in Beam Java.
- **Input**: Start jobs with 1 TB of backlog data on a Kafka topic. Messages are 10 KB each. Schema is simple with two fields.
- **Catalog**: Hive metastore is used as the Iceberg catalog.

Benchmarking Results

- **Throughput**: Flink cleared 1 TB of Backlog almost 10 times faster (20 minutes vs 3 hours).
- **File Size**: Flink produced ~10x larger intermediate parquet files (400 MB vs 38 MB per file).
    - Note: Writing large files is desirable since they are more optimal for reads
- **Resource Utilization**: Flink task managers showed more stable and tighter CPU utilization than Beam+Dataflow VMs.

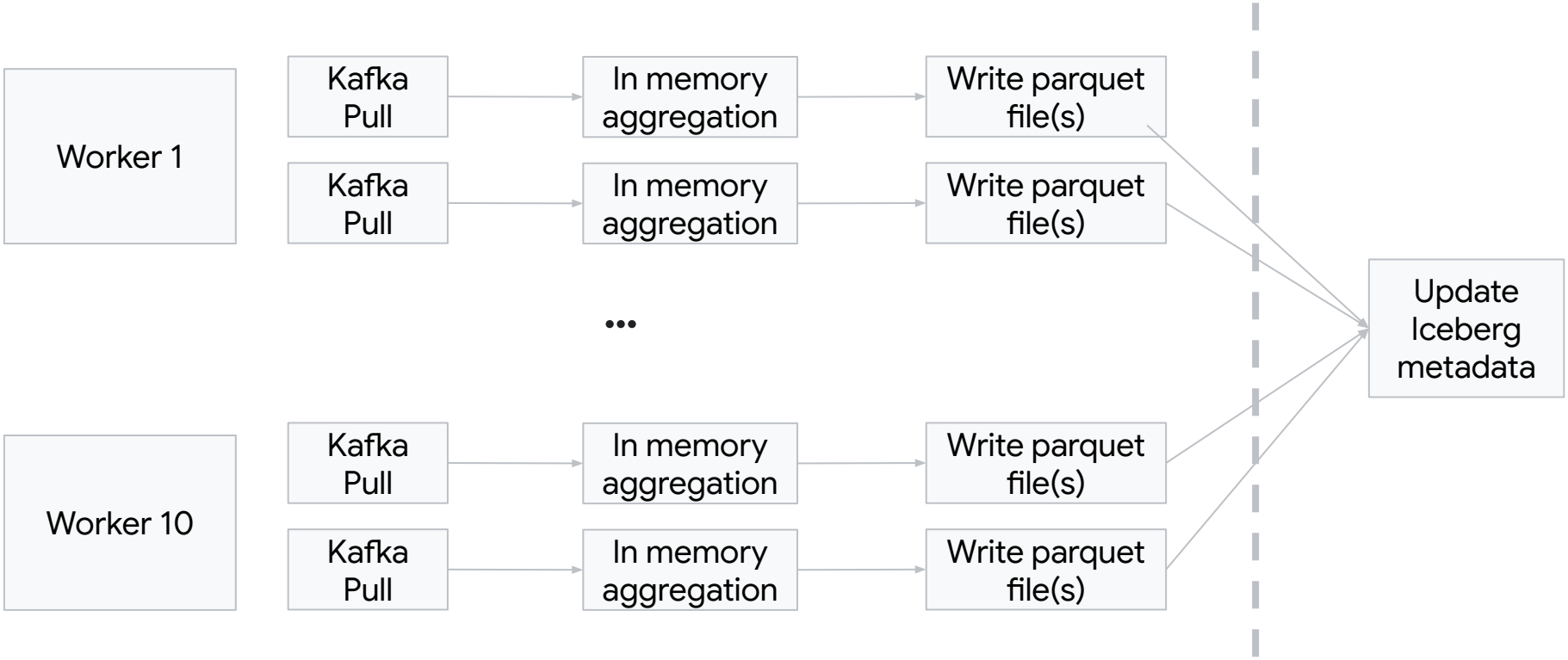
Beam vs Flink CPU Utilization



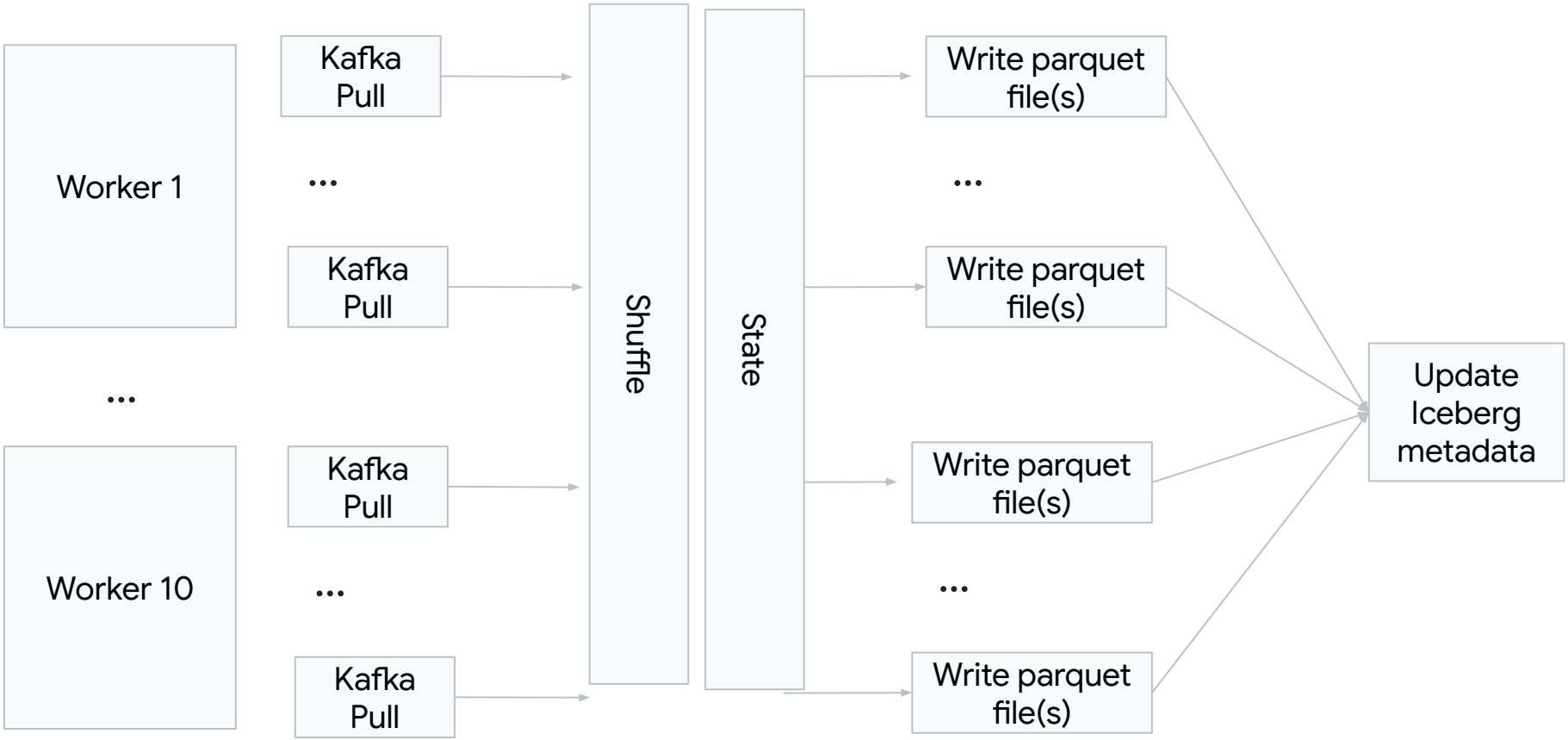
Beam + Dataflow

Flink

Flink Pipeline



Beam + Dataflow Pipeline



Improvements

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

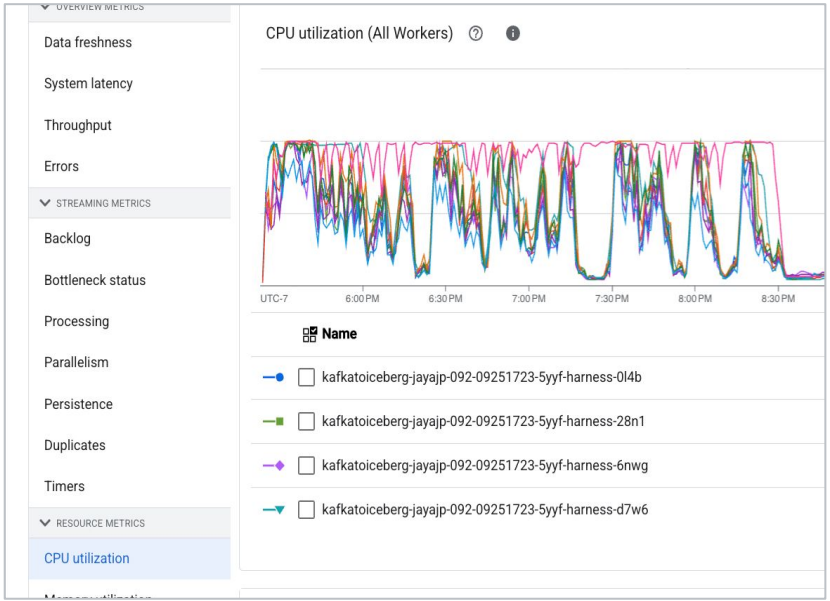
Compression: Overview

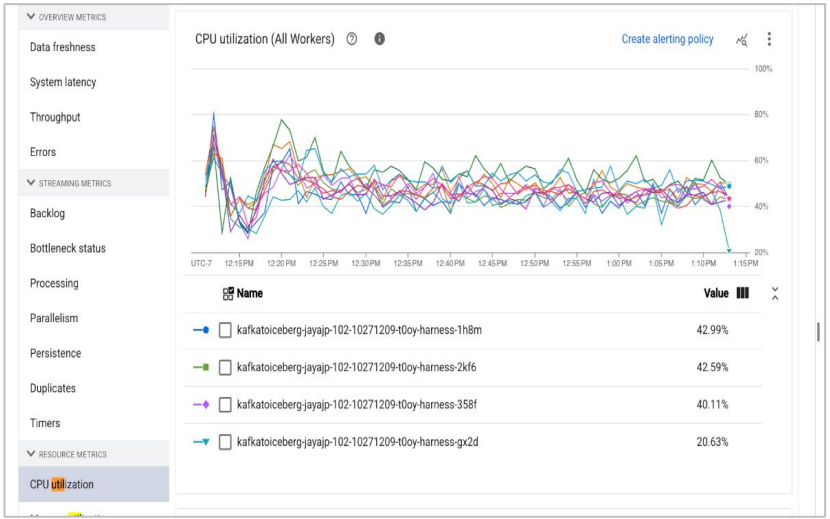
	Prior: gzip	Current: zstd
Year Released	1992	2016
Performance	Most reports show that zstd outperforms gzip on the following factors - compression ratio, CPU utilization, wall-time.	
Behavior with uncompressionable data	Still attempt to apply its compression algorithms leading to increased CPU usage and 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 Used on Flink for a while

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

Compression Load Tests

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

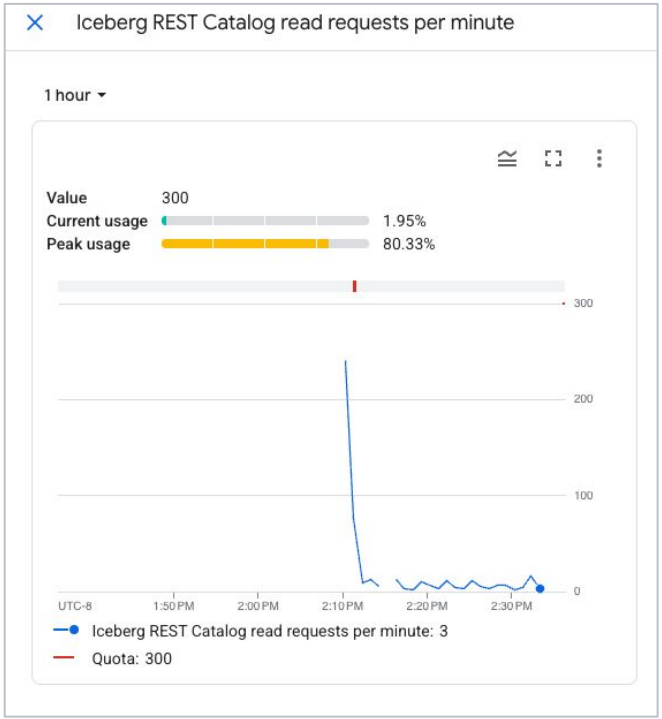




Metadata Caching

Metadata Caching

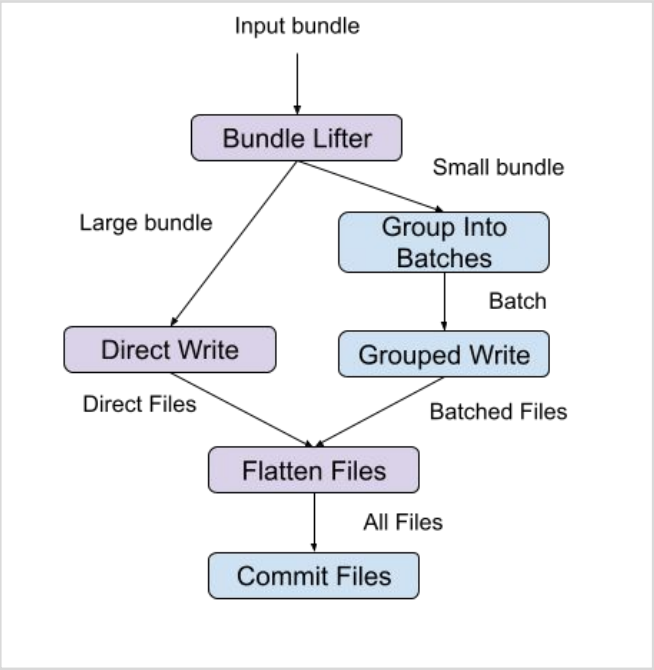
- Testing quickly exhausted BigLake Metastore Read Quota due to refreshing table metadata on each input.
- This blocked processing and reduced performance.
- Optimized by caching and refreshing every ~2m.

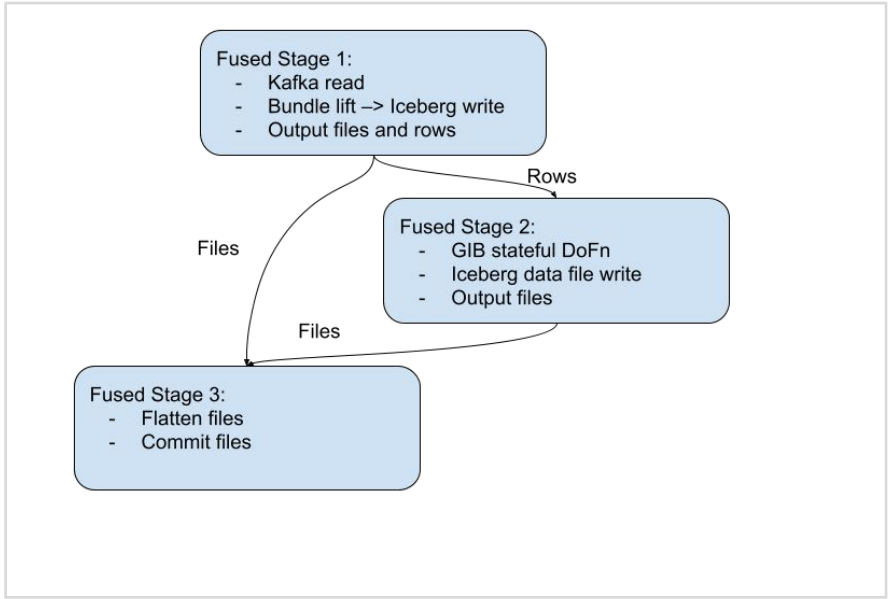


https://github.com/apache/beam/pull/37102

Direct Writes

Direct Writes: Overview





https://github.com/apache/beam/pull/36720

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

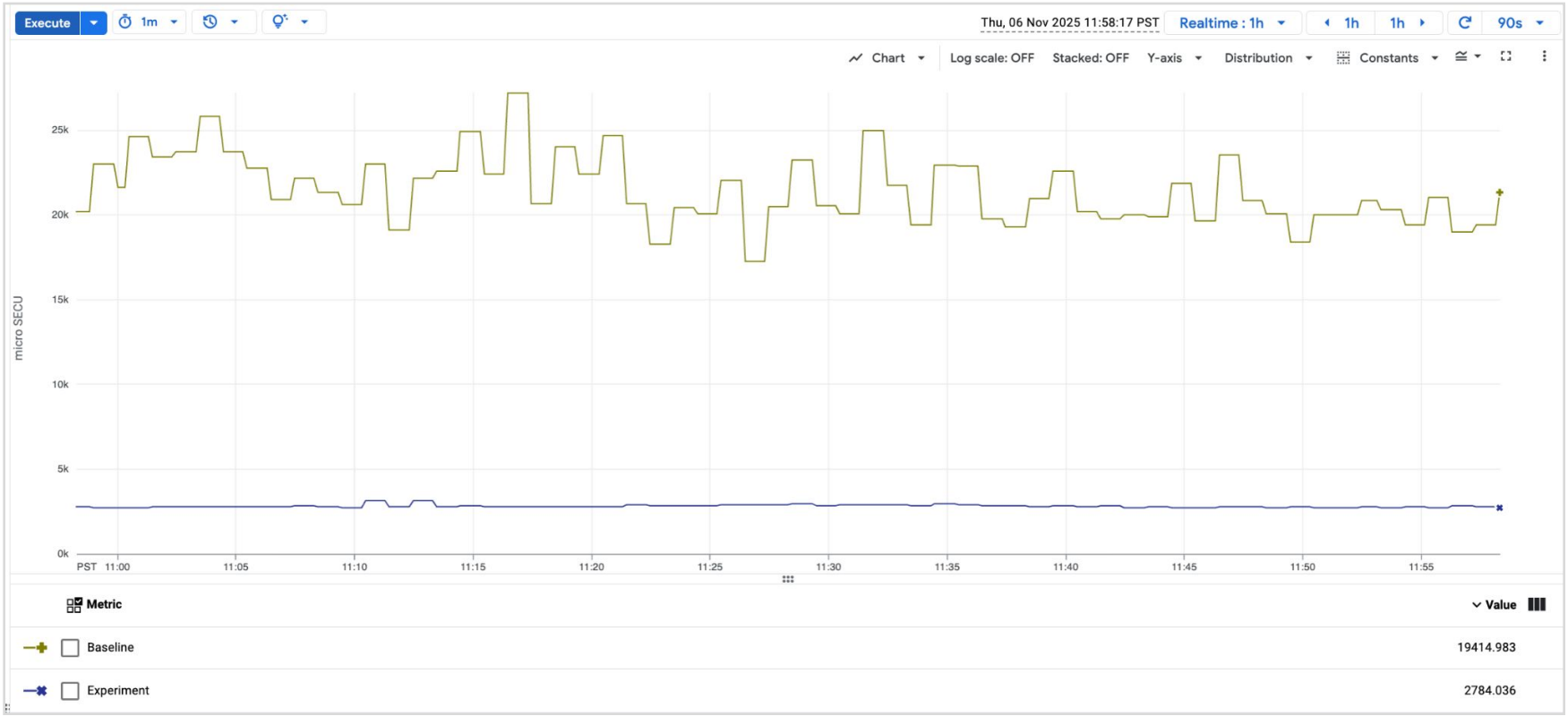
Custom Counters (Approximate)

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

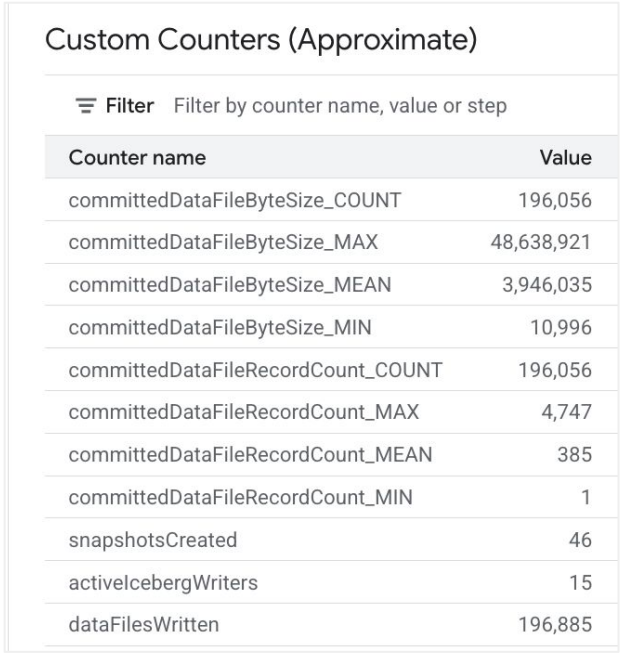


Autosharding

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.
- Manual tuning showed great promise
  - Configured workers to write 6 parallel files at once instead
  - Resulted in 8x larger files and 25% improved throughput
- Throughput-based autosharding is likely a better fit.
  - Allows Iceberg sink to dynamically adjust parallelism.
  - It is already used for BigQuery and GCS IO sinks.

Iceberg Autosharding: File Sizes

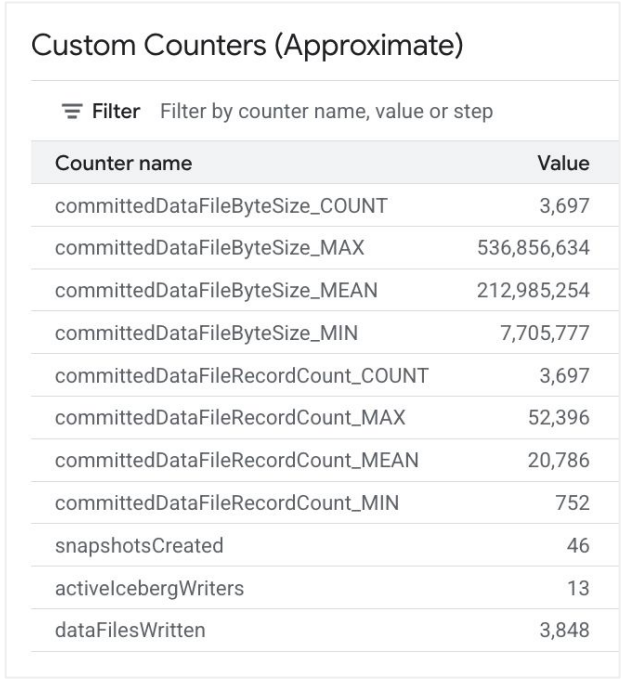


Custom Counters (Approximate)

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



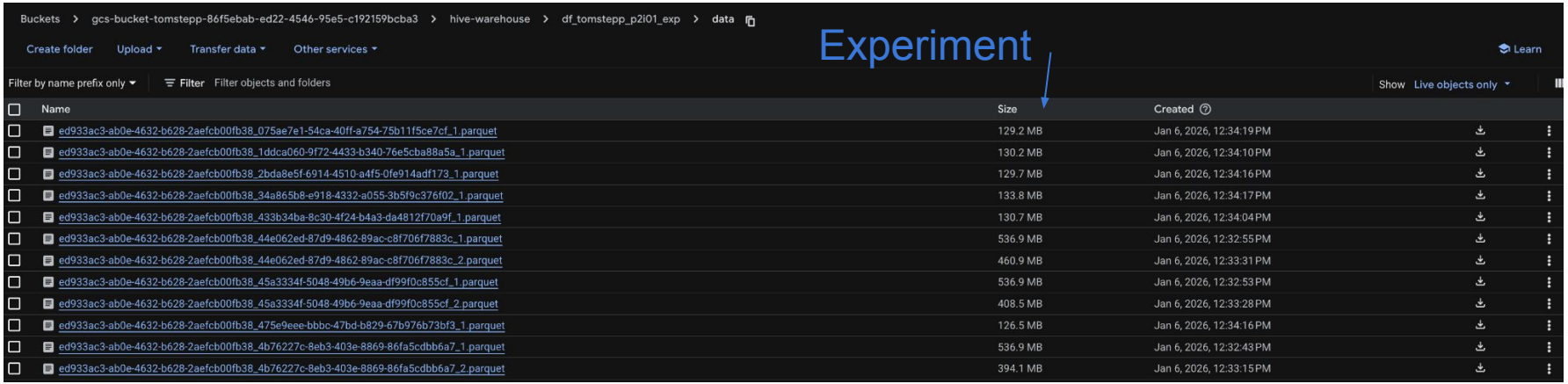
Custom Counters (Approximate)

Filter Filter by counter name, value or step

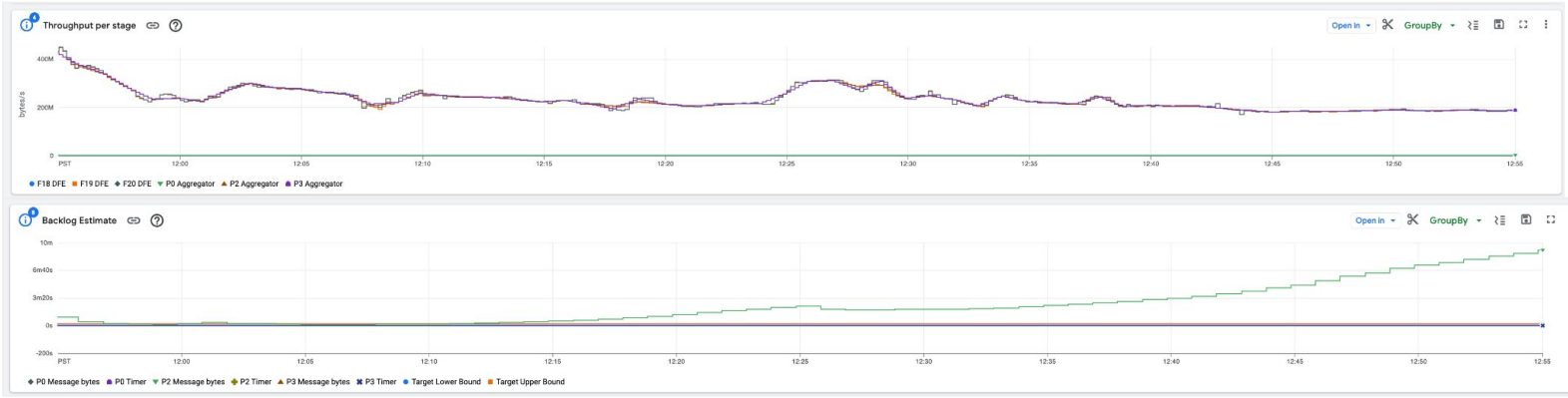
Counter name	Value
committedDataFileByteSize_COUNT	3,697
committedDataFileByteSize_MAX	536,856,634
committedDataFileByteSize_MEAN	212,985,254
committedDataFileByteSize_MIN	7,705,777
committedDataFileRecordCount_COUNT	3,697
committedDataFileRecordCount_MAX	52,396
committedDataFileRecordCount_MEAN	20,786
committedDataFileRecordCount_MIN	752
snapshotsCreated	46
activeIcebergWriters	13
dataFilesWritten	3,848

Experiment

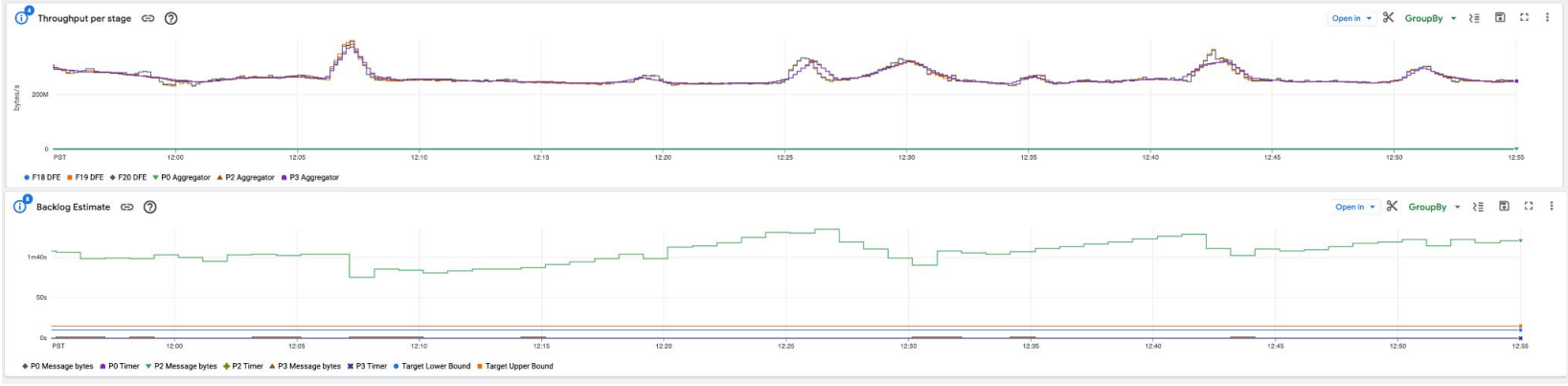
Iceberg Autosharding: File Size Sanity Check



Iceberg Autosharding: Performance



Baseline



Experiment

Growth

Growth

- Beam pipelines writing to IcebergIO sink on Dataflow has grown 20x since we made these improvements!

Thank you!