



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

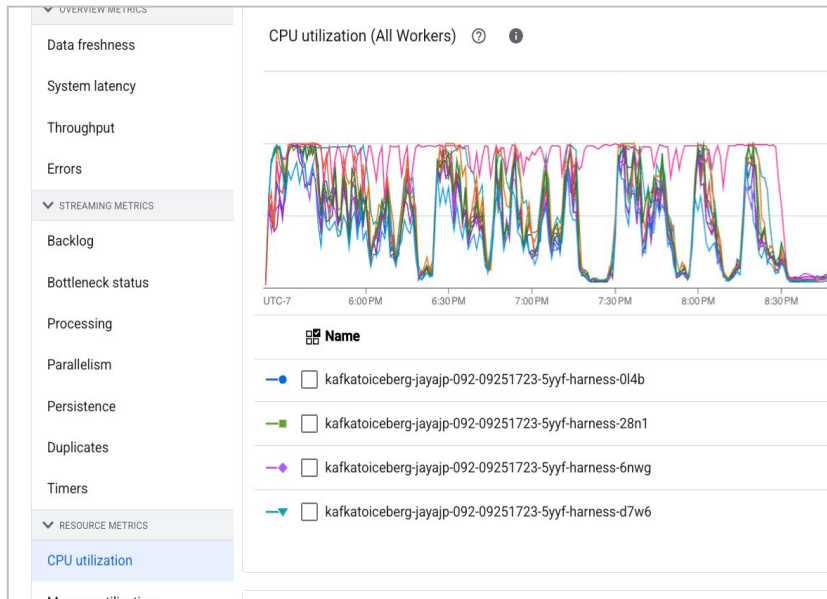
Setup

- **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

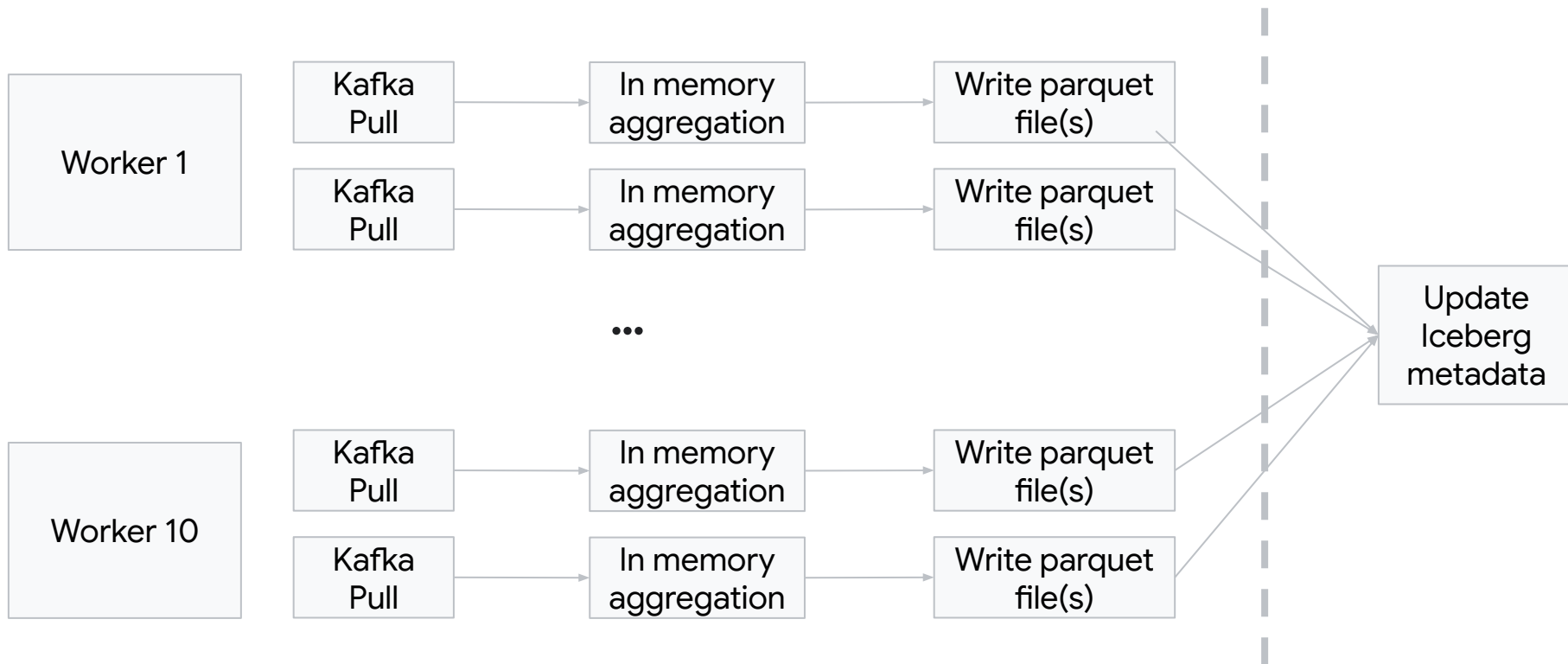


Beam + Dataflow

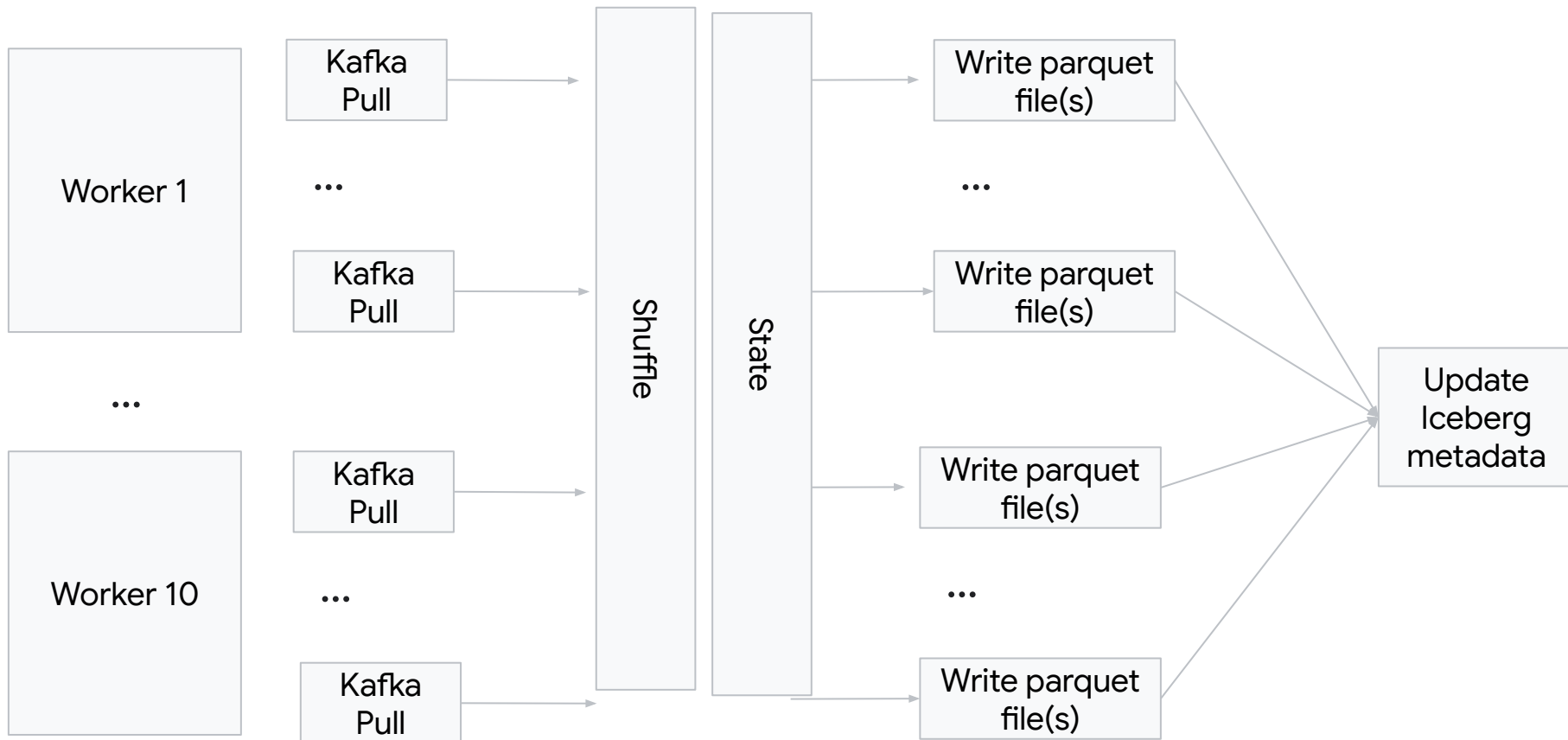


Flink

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

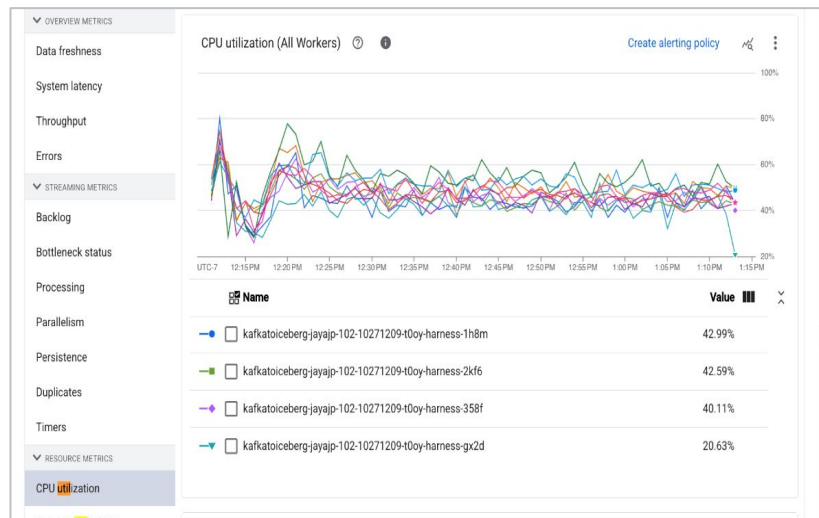
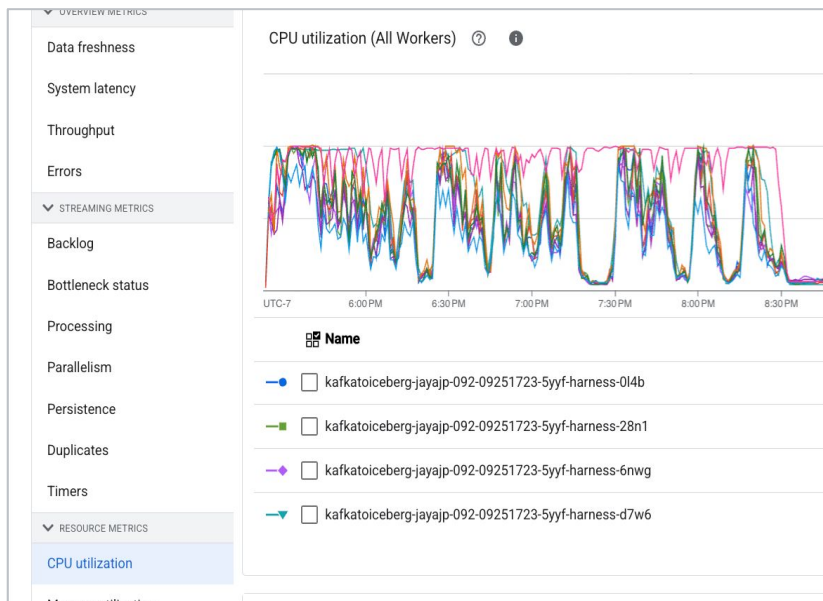
Compression: Overview

	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 uncompressible 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

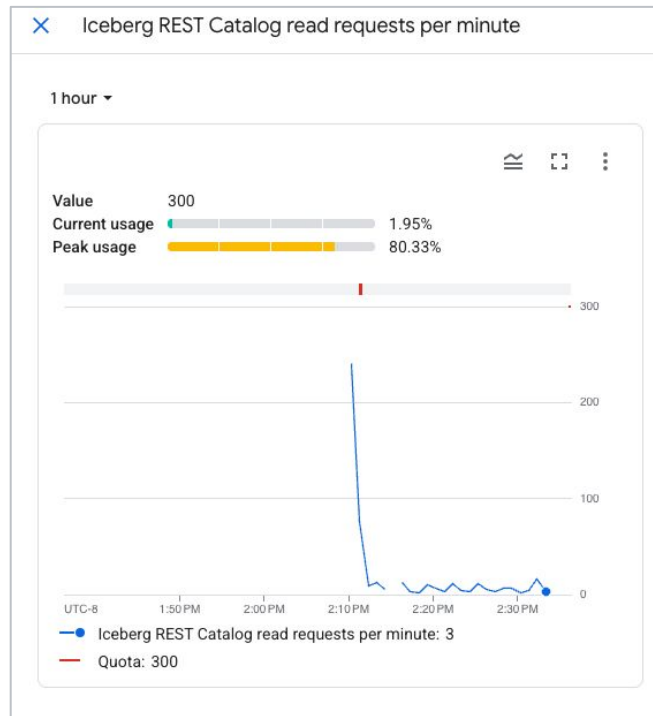
Time to clear backlog 1.2tb Kafka backlog: 3 hours -> 1 hour 40 minutes



Metadata Caching

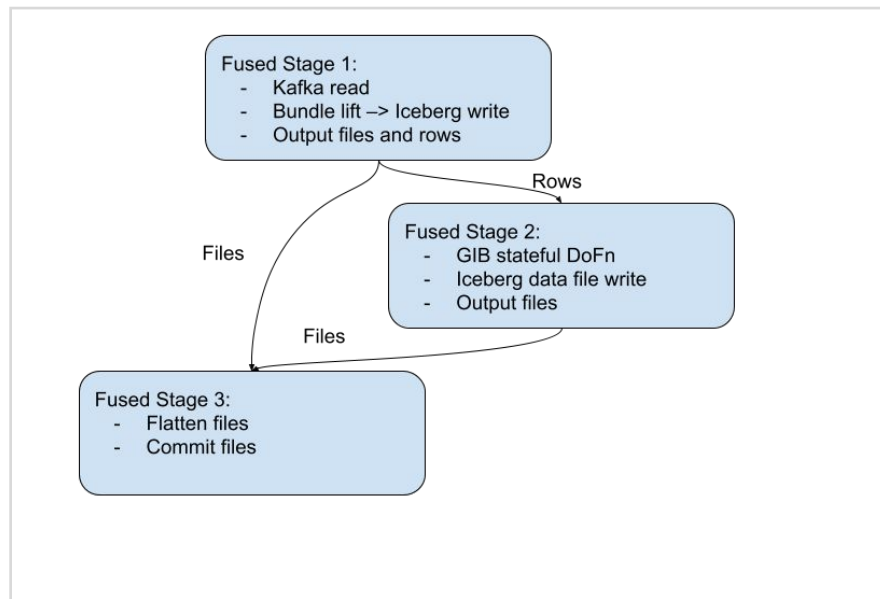
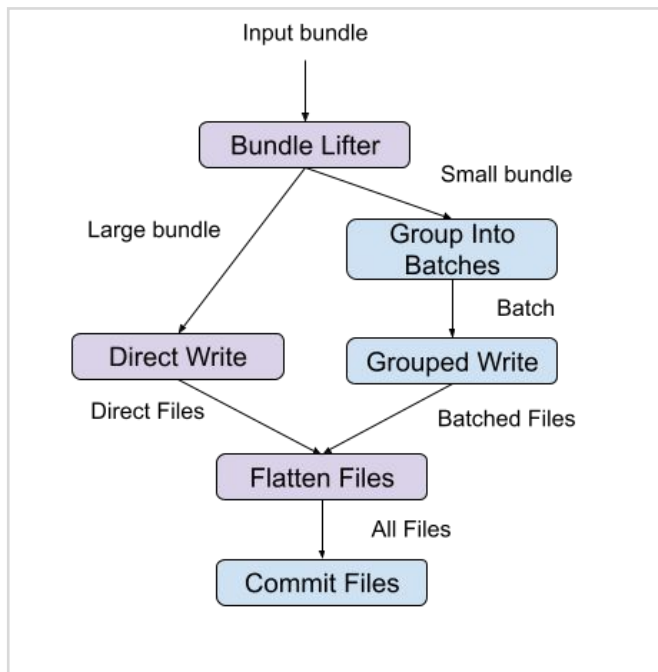
Metadata Caching

- 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.



Direct Writes

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

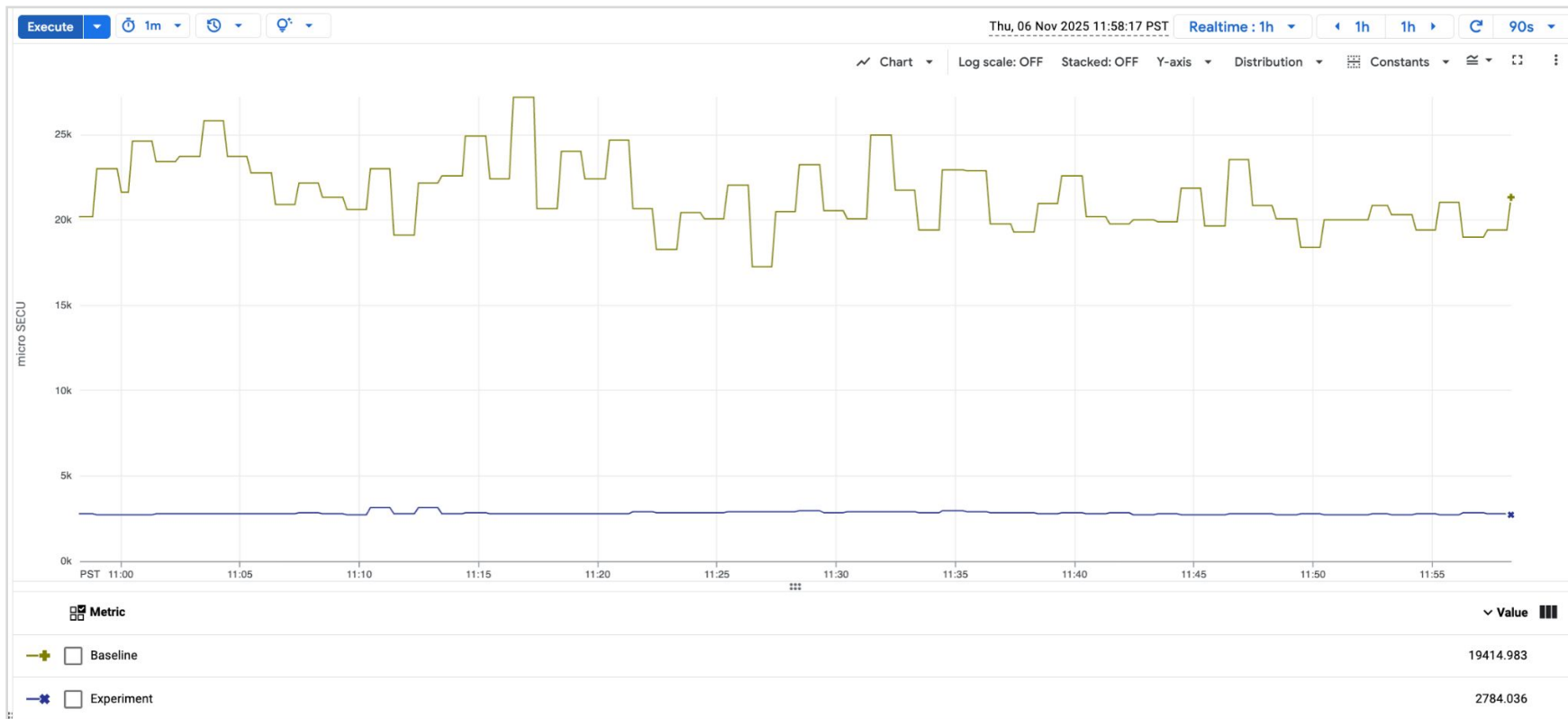
Custom Counters (Approximate)

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




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

Buckets > gcs-bucket-tomstepp-86f5ebab-ed22-4546-95e5-c192159bcba3 > hive-warehouse > df_tomstepp_p2i01_base > data

Create folder Upload Transfer data Other services

Baseline

Filter by name prefix only Filter Filter objects and folders Show Live objects only

Name	Size	Created
c93c0fe9-61fb-4e58-ac8d-00bc7776cd01_0010a360-6c4f-472c-8b28-2d39030f40d8_1.parquet	8.4 MB	Jan 6, 2026, 12:32:28 PM
c93c0fe9-61fb-4e58-ac8d-00bc7776cd01_0021c8f2-5d52-45a2-99e0-460c7df27681_1.parquet	15.5 MB	Jan 6, 2026, 12:33:12 PM
c93c0fe9-61fb-4e58-ac8d-00bc7776cd01_00442c79-46d8-41ca-8ba2-144cfbf403e7_1.parquet	21.1 MB	Jan 6, 2026, 12:34:54 PM
c93c0fe9-61fb-4e58-ac8d-00bc7776cd01_00625ba0-ea82-4306-a530-3d56d904ddc7_1.parquet	28.5 MB	Jan 6, 2026, 12:33:22 PM
c93c0fe9-61fb-4e58-ac8d-00bc7776cd01_006e2a34-fc1c-4fd5-b70c-d3051657cb1e_1.parquet	25.3 MB	Jan 6, 2026, 12:33:32 PM
c93c0fe9-61fb-4e58-ac8d-00bc7776cd01_006e7a16-96d0-4db3-a559-2f890358ff05_1.parquet	32 MB	Jan 6, 2026, 12:34:06 PM
c93c0fe9-61fb-4e58-ac8d-00bc7776cd01_006f8f30-2214-4de3-ad2c-44be6480e76c_1.parquet	19 MB	Jan 6, 2026, 12:32:56 PM
c93c0fe9-61fb-4e58-ac8d-00bc7776cd01_00752a78-99fe-43b5-9c35-584b05b94147_1.parquet	20.2 MB	Jan 6, 2026, 12:33:54 PM
c93c0fe9-61fb-4e58-ac8d-00bc7776cd01_007e53d0-23ab-4577-8fa2-5fe438d130f2_1.parquet	21.2 MB	Jan 6, 2026, 12:33:08 PM
c93c0fe9-61fb-4e58-ac8d-00bc7776cd01_00c3ebad-914d-4f0d-8212-c2ceda749e59_1.parquet	15.4 MB	Jan 6, 2026, 12:33:30 PM
c93c0fe9-61fb-4e58-ac8d-00bc7776cd01_01134e97-d27c-4478-9088-4dcd3f47383c_1.parquet	7.8 MB	Jan 6, 2026, 12:32:13 PM
c93c0fe9-61fb-4e58-ac8d-00bc7776cd01_011aca52-39ab-4985-9d5d-885e3a01d845_1.parquet	6.8 MB	Jan 6, 2026, 12:32:17 PM

Buckets > gcs-bucket-tomstepp-86f5ebab-ed22-4546-95e5-c192159bcba3 > hive-warehouse > df_tomstepp_p2i01_exp > data

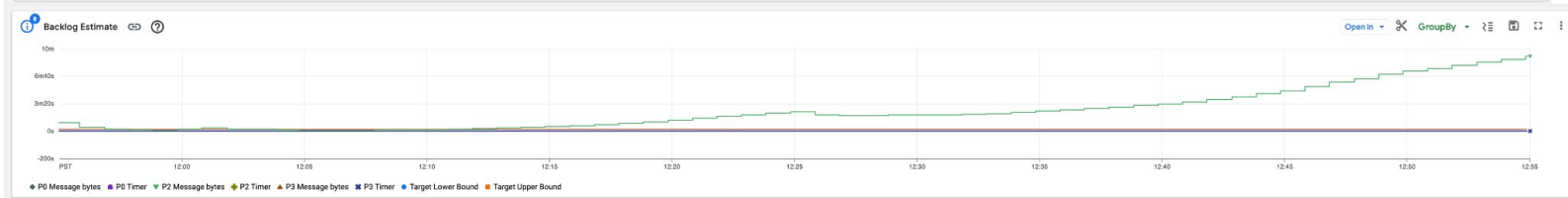
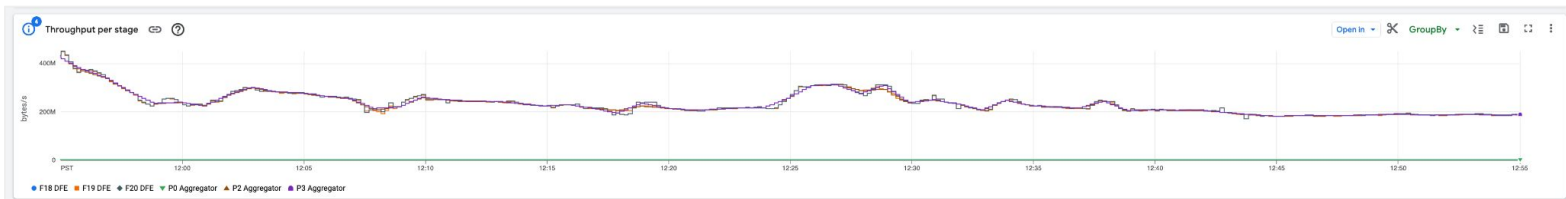
Create folder Upload Transfer data Other services

Experiment

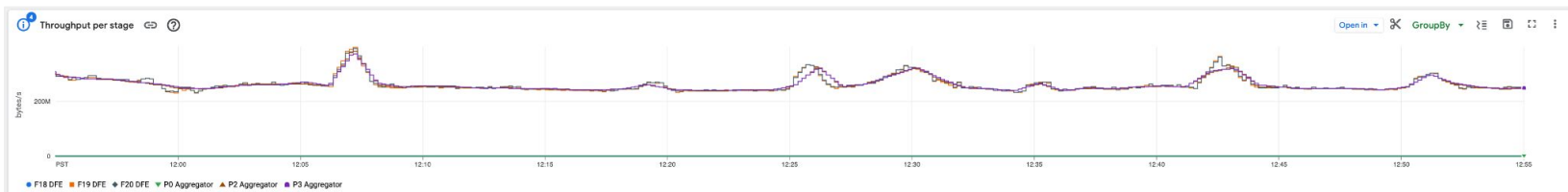
Filter by name prefix only Filter Filter objects and folders Show Live objects only

Name	Size	Created
ed933ac3-ab0e-4632-b628-2aefcb00fb38_075ae7e1-54ca-40ff-a754-75b11f5ce7cf_1.parquet	129.2 MB	Jan 6, 2026, 12:34:19 PM
ed933ac3-ab0e-4632-b628-2aefcb00fb38_1ddca060-9f72-4433-b340-76e5cba88a5a_1.parquet	130.2 MB	Jan 6, 2026, 12:34:10 PM
ed933ac3-ab0e-4632-b628-2aefcb00fb38_2bda8e5f-6914-4510-a4f5-0fe914adf173_1.parquet	129.7 MB	Jan 6, 2026, 12:34:16 PM
ed933ac3-ab0e-4632-b628-2aefcb00fb38_34a865b8-e918-4332-a055-3b59c376f02_1.parquet	133.8 MB	Jan 6, 2026, 12:34:17 PM
ed933ac3-ab0e-4632-b628-2aefcb00fb38_433b34ba-8c30-4f24-b4a3-da4812f70a9f_1.parquet	130.7 MB	Jan 6, 2026, 12:34:04 PM
ed933ac3-ab0e-4632-b628-2aefcb00fb38_44e062ed-87d9-4862-89ac-c8f706f7883c_1.parquet	536.9 MB	Jan 6, 2026, 12:32:55 PM
ed933ac3-ab0e-4632-b628-2aefcb00fb38_44e062ed-87d9-4862-89ac-c8f706f7883c_2.parquet	460.9 MB	Jan 6, 2026, 12:33:31 PM
ed933ac3-ab0e-4632-b628-2aefcb00fb38_45a3334f-5048-49b6-9eaa-df99f0c855cf_1.parquet	536.9 MB	Jan 6, 2026, 12:32:53 PM
ed933ac3-ab0e-4632-b628-2aefcb00fb38_45a3334f-5048-49b6-9eaa-df99f0c855cf_2.parquet	408.5 MB	Jan 6, 2026, 12:33:28 PM
ed933ac3-ab0e-4632-b628-2aefcb00fb38_475e9eee-bbbc-47bd-b829-67b976b73bf3_1.parquet	126.5 MB	Jan 6, 2026, 12:34:16 PM
ed933ac3-ab0e-4632-b628-2aefcb00fb38_4b76227c-8eb3-403e-8869-86fa5cddb6a7_1.parquet	536.9 MB	Jan 6, 2026, 12:32:43 PM
ed933ac3-ab0e-4632-b628-2aefcb00fb38_4b76227c-8eb3-403e-8869-86fa5cddb6a7_2.parquet	394.1 MB	Jan 6, 2026, 12:33:15 PM

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!

