Throttling Detection and Reactive Worker Downscaling

Yi Hu
Google
GitHub: @Abacn

BEAM SUMMIT

September 4-5, 2024

Sunnyvale, CA. USA

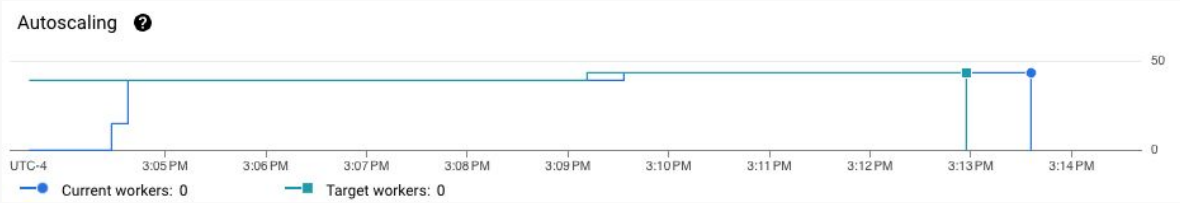
Agenda

- Problem statement
- Revisit: How does parallelism get determined?
- Throttling metrics
- SDK implementations
- Runner implementations

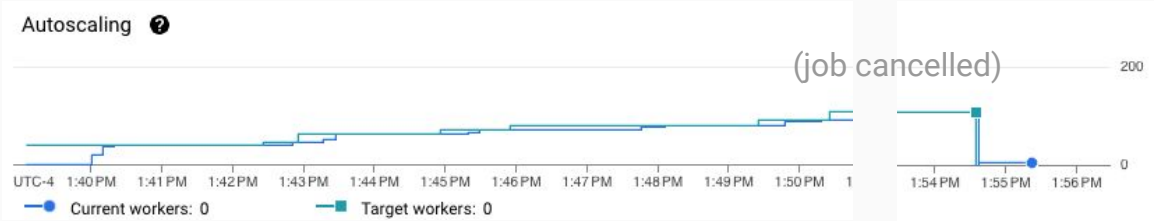
BEAM SUMMIT

Problem statement

*Executing Beam pipelines on Dataflow exposes some rough edges. Dataflow autoscaling can spin up a large number of workers that can easily **overload** services that IO connector involves...*



sufficient quota



limited quota

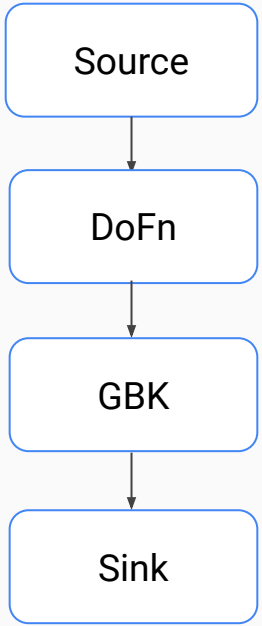
Beam 2.56, BigQuery DIRECT_READ TPC-DS_1T (tpc.org/tpcds)

How does parallelism get determined?

- Available parallelism
  - Parallelism of source / Number of keys
  - Whether source supports liquid sharding, extent of granularity

Type of sources by ability to parallelize

  - *No parallelism* (e.g. JdbcIO.read)
  - *Static parallelism* (e.g. JdbcIO.readWithPartitions)
  - Bounded Dynamic parallelism (e.g. KafkaUoundedSource)
  - Liquid sharding (e.g. FileIO readRange, BigQueryIO DIRECT READ)



BEAM SUMMIT

How does parallelism get determined?

- Job signal (up/back/two-way pressure)
  - CPU usage
    - [back pressure] Batch: no upscaling if CPU usage < 5% [1]
    - [two-way pressure] Streaming: utilization hint [2]
  - User supplied metrics
    - [up pressure] Streaming: sources' BacklogBytes
    - [back pressure] Client side **throttling metrics**

[1] https://cloud.google.com/dataflow/docs/horizontal-autoscaling#batch

[2] cloud.google.com/dataflow/docs/guides/tune-horizontal-autoscaling#update-hint

BEAM SUMMIT

Throttling metrics

- A Beam counter metrics under dedicated name

  (*THROTTLE_TIME_COUNTER_NAME = throttling-msecs*)

To setup such metrics

- Java SDK:

```
Counter throttlingMsecs = Metrics.counter(
      Metrics.THROTTLE_TIME_NAMESPACE, Metrics.THROTTLE_TIME_COUNTER_NAME);
```

- Python SDK:

```
THROTTLE_COUNTER = Metrics.counter(__name__, 'cumulativeThrottlingSeconds')
```

BEAM SUMMIT

Throttling metrics

Notes:

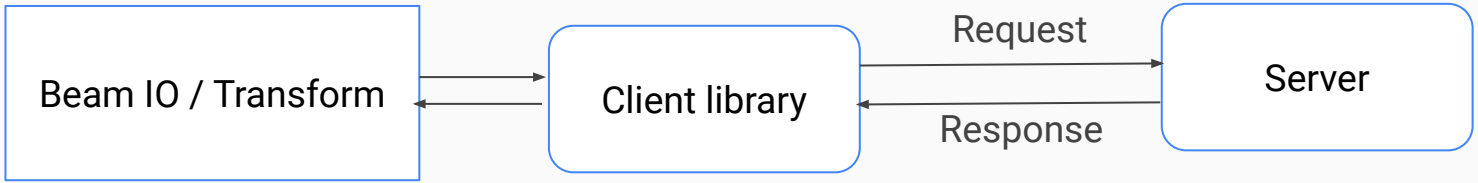
- Under standard Beam metrics API, **NO** dedicated Beam model (proto) level API
- *throttling-msecs* counter long existing in a few Beam IOs (Datastore, GcsIO, BigQuery *STREAM_INSERT*) and works for Dataflow legacy worker.
- Recently generalized (#31924, 2.59.0) and extended into more IOs (BigQuery Storage API, Bigtable write), works with Dataflow runner v2.
- Up to runner to react on the throttling metrics as of scaling decision.

BEAM SUMMIT

Throttling metrics: SDK implementations

- Goal: extract the throttled time spent on interacting with external resources
  - wait time, latency
  - retry backoff, etc

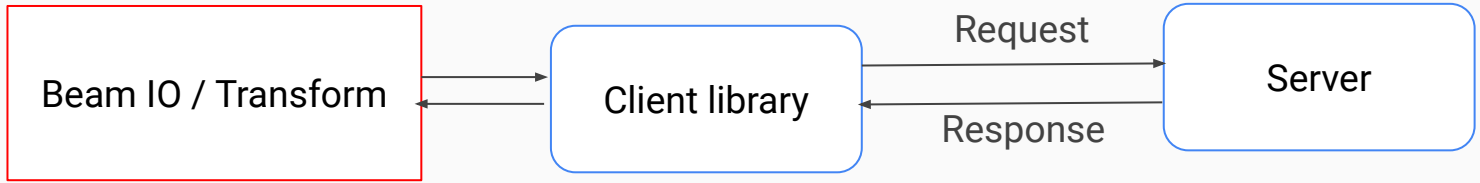
  Depends on where the throttled time incurred



Throttling metrics: SDK implementations

- Case 1: Server returns HTTP 429 immediately, and **Beam** retry with backoff (e.g. BigQueryIO Storage Write API)

  Approach: Set FluentBackoff.withThrottledTimeCounter (#29098, 2.53.0)



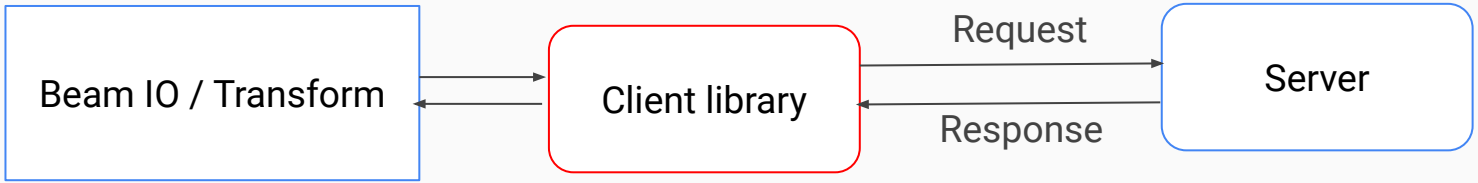
Throttling metrics: SDK implementations

- Case 2: Client library involves retry, and backoff time tracked in retry callback

  (e.g. GcsUtil, BigQueryIO Storage Read API)

  Caveat: callback thread may not have metrics container initialized*

  Approach: incrementing pending throttling time and report back in DoFn (#31096)



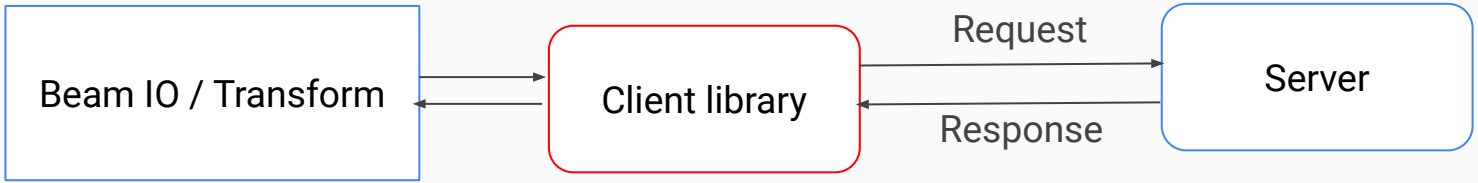
*Warning: Reporting metrics are not supported in the current execution environment

Throttling metrics: SDK implementations

- Case 3: Client library involves retry, and backoff time not tracked

  (e.g. BigtableIO write)

  Bigtable client has built-in rate limiting mechanism, and associated configs, e.g.
  - setThrottlingTargetMs
  - setFlowControl



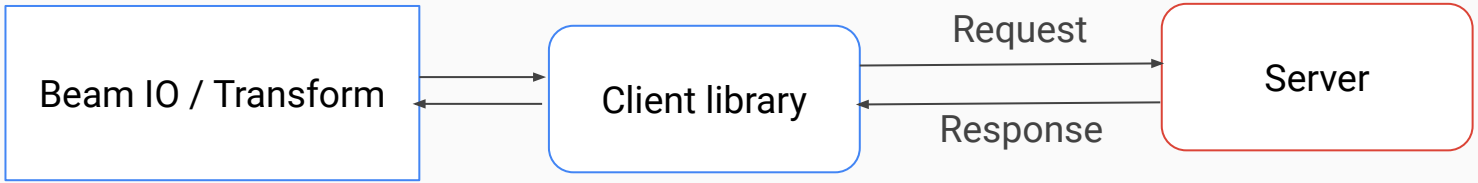
Throttling metrics: SDK implementations

- Case 4: Server latency spikes, time out

  Exact throttled time not known, same as Case 3.

  Approach (for both Case 3&4):

  - a configurable targeted latency time on API call
  - if latency>set time, excessive amount is reported as throttling time



Throttling metrics: Runner implementations

- Dataflow runner scaling decision

  Fraction of throttled time:

  $$r = \text{recent throttle time} / \text{recent done time}$$

  The targeted parallelism overwritten as

  $$p = (1-r)p'$$

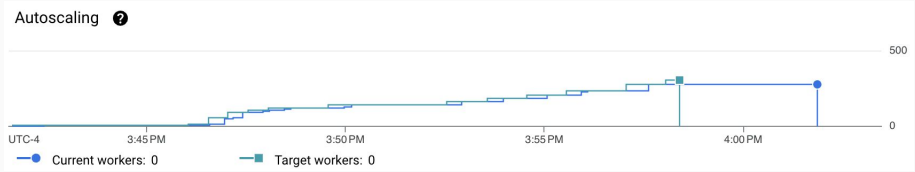
  (p': targeted parallelism without considering the throttled time)

  Downscale decision issued after p<current parallelism for 3 minutes.

BEAM SUMMIT

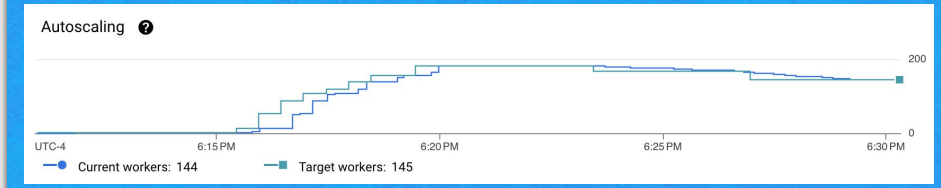
Example: BigQuery Storage Read API (#31404)

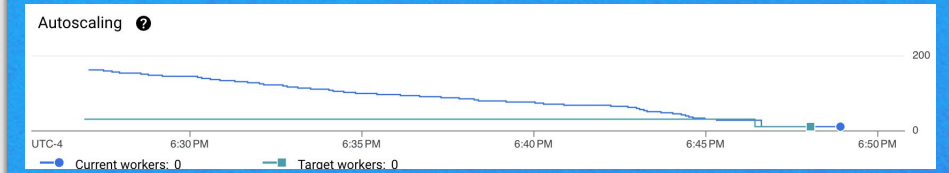
Before:



upscale to 270 worker aggressively, before job cancel

After:





scales down until finish (autoscaler won't kill active worker until current work item finish)

Throttling metrics: Runner implementations

- Metrics defined in SDK side, and it is IO connector's responsibility to report
- Runner can react on the throttling metrics, and
- Decision on how runner reacts is runner dependent
  - Currently supported by Dataflow
  - Interface for other runners' support is open

BEAM SUMMIT

Thank you!

Questions?

Resources:

Dataflow: Tune Horizontal autoscaling
https://cloud.google.com/dataflow/docs/guides/tune-horizontal-autoscaling

BEAM SUMMIT