BigQuery Streaming Insert Benchmark

heejong@google.com

12/11/2018

Background

This document describes a quick benchmark for 4 different streaming insert implementations for BigQuery in Apache Beam. The benchmark is a part of research for finding a method to reduce overheads on BigQuery backend because of uncontrolled API requests regarding BEAM-5514.

Implementations

Current Status BigQueryIO.write() uses streaming insert API to write data bundles to BigQuery table. There can be hundreds of concurrent tasks running for streaming insert and it causes multiple problems such as excessive rate limit exceeded error logs or unnecessary overheads on BigQuery API backend. BigQueryIO.write() first splits a PCollection into 50 shards and creates dozens of inserting futures for each processed bundle per shard. The number of generated futures depends on the size of the bundle. If each bundle generates 20 futures, the maximum number of concurrency will be 1000.

Dynamic Throttling RateLimiter class in Guava library offers uniformly distributed rate controlling. It controls the rate by issuing the given number of tickets each second and workers are competing for getting those tickets before doing their job. Dynamic throttling implementation utilizes RateLimiter class in Guava and Meter class in Codehale. RateLimiter issues 2000 tickets each second (default rows per second quota 100000 divided by the number of shards 50). On every minute, if there was no rate limit exceeded error during the previous one minute, RateLimiter increases the number of issuing tickets by 1 percent. Otherwise, the number of tickets decreases 5 percent.

Shared Backoff By simply moving backoff instance to the outer future scope, all futures can share the same backoff. Since default backoff implementation is not thread-safe, it's also required to make nextBackoffMillis() synchronized. In this case, futures will receive significantly large backoff intervals as compared to using multiple backoffs.

Any IOException Handler BigQuery API sends two different responses for the same rate limit quota exceeded error. Rate limit exceeded error is properly handled with backoff but quota exceeded error is not. When unhandled exception is thrown in any future, it crashes a whole bundle and delays the process at least 10 seconds (in case of Dataflow Runner). By handling all types of IOException in the same way as rate limit exception, we can remove the worker restarting delay and retrying the failed job right away.

Configuration

The benchmark program is a simple streaming pipeline that generates 4 bytes random string of integer numbers and inserts them into BigQuery table. The number of workers is set to 4 and running time is 20 minutes on production DataflowRunner. At the time of benchmark, master branch is pointing to 29a7917.

Benchmark Result

	Writer Wall Time	Bytes Written
Master	11 hr 0 min 56 sec	1,281,740,388
Master + dynamic throttling	15 hr 50 min 35 sec	1,119,466,932
Master + shared backoff	8 hr 43 min 59 sec	536,049,960
Master + any IOException handler	13 hr 34 min 5 sec	1,374,636,336

Observations

- “Master + any IOException handler” shows better performance than raw master as expected since there's no total failure of worker which delays a task for a whole bundle
- As expected, the best performance can be achieved by pushing the BigQuery backend to its limit (with no rate controlling)
- Dynamic throttling underperforms no throttling by 20 percent but generates near-zero log messages about exceeded rate limit error (which also implies minimal overheads to the backend)
- If achieving the best possible performance is the goal, we should stick to the current implementation. If reducing the backend overheads is considered beneficial, we can try dynamic throttling by sacrificing some performance

UPDATED 1/15/2019

Additional benchmarks are performed for comparing single thread pool and unlimited thread pool scenarios (BEAM-6443). Benchmark configurations are the same as the previous settings except that running time has changed to 35 minutes (9 bytes per row) and 15 minutes (1MB per row). Master branch is now pointing to f8ef83b, the version after any IOException handler fix.

	Writer Wall Time	Bytes Written
Unlimited thread pool 1	22 hr 0 min 26 sec	5,330,000,000
Single thread pool 1	1 day 3 hr 51 min 21 sec	5,331,000,000
Unlimited thread pool 2	22 hr 6 min 4 sec	5,317,000,000
Single thread pool 2	1 day 3 hr 40 min 27 sec	5,466,750,000

Benchmark result for very small size of data (9 bytes per row)

	# of quota exceeded error messages
Unlimited thread pool	106,099
Single thread pool	26,018

Number of quota exceeded error messages in worker log (9 bytes per row)

	Writer Wall Time	Bytes Written
Unlimited thread pool	9 hr 2 min 2 sec	27,944,763,600
Single thread pool	8 hr 42 min 59 sec	28,490,027,280

Benchmark result for maximum size of data (1MB per row, same as streaming insert row size limit)

The result shows that the change from unlimited thread pool to single thread pool execution does not undermine the performance of streaming insert and reduces the number of error messages to 1/4 of its original size.

UPDATED 2/4/2019

20 minutes / BoundedExecutorService / Semaphore(1)
2,779,680,000 bytes written / 19441 ‘retrying:’ messages / writer wall time 16:22:13

20 minutes / BoundedExecutorService / Semaphore(3)
2,779,920,000 bytes written / 65897 ‘retrying:’ messages / writer wall time 14:39:56

20 minutes / Unlimited Pool from GcsOption
2,777,280,000 bytes written / 103773 ‘retrying:’ messages / writer wall time 12:51:49