Bulk loading best practices

This page provides guidelines for efficiently bulk loading large amounts of data into Cloud Spanner.

You have several options for loading data in bulk into Cloud Spanner:

- Insert rows using Data Manipulation Language (DML) (https://cloud.google.com/spanner/docs/dml-tasks).
- Insert rows using mutations (https://cloud.google.com/spanner/docs/modify-mutation-api) through the client library.
- Use the Dataflow connector (https://cloud.google.com/spanner/docs/dataflow-connector). If you are importing data using the Dataflow connector, the best practices for the Dataflow connector (#dataflow) apply.
- Import a database (https://cloud.google.com/spanner/docs/import) using Avro files. If you are importing data from Avro files, the best practices for importing Avro files (#importing-avro) apply.
- Insert rows using the `gcloud` command-line tool (https://cloud.google.com/spanner/docs/modify-gcloud#modify-data). However, we don't recommend that you use the `gcloud` tool to bulk load data.

Performance guidelines for bulk loading

You can achieve optimal bulk loading performance by following a few guidelines:

- **Minimize the number of splits** (https://cloud.google.com/spanner/docs/schema-and-data-model#database-splits) **that are involved in each write transaction.** Performance is maximized because write throughput is maximized when fewer splits are involved in a transaction.
- **Maximize the use of partitioning to distribute writing the partitions across worker tasks.**

Cloud Spanner uses load-based splitting (https://cloud.google.com/spanner/docs/schema-and-data-model#load-based_splitting) to evenly distribute your data load across nodes. After a few minutes of high load, Cloud Spanner introduces split boundaries between rows of non-interleaved tables. In general, if your data load is well-distributed and you follow best practices for schema design and bulk loading, your write throughput should double every few minutes until you saturate the available CPU resources in your instance.

Partition your data by primary key

To get optimal write throughput for bulk loads, partition your data by primary key with this pattern:

- Each partition contains a range of consecutive rows, as determined by the key columns.
- Each commit contains data for only a single partition.

We recommend that the number of partitions be 10 times the number of nodes in your Cloud Spanner instance. To assign rows to partitions:

- Sort your data by primary key.
- Divide the data into 10 * (number of nodes) separate, equally sized partitions.
- Create and assign a separate worker task to each partition. Creating the worker tasks happens in your application. It is not a Cloud Spanner feature.

Following this pattern, you should see a maximum overall bulk write throughput of 10-20 MB per second per node for large loads.

As you load data, Cloud Spanner creates and updates splits to balance the load on the nodes in your instance. During this process, you may experience temporary drops in throughput.

Example

You have a regional configuration with 3 nodes. You have 90,000 rows in a non-interleaved table. The primary keys in the table range from 1 to 90000.

- Rows: 90,000 rows
- Nodes: 3
- Partitions: 10 * 3 = 30

- Rows per partition: 90000 / 30 = 3000.

The first partition includes the key range 1 to 3000. The second partition includes the key range 3001 to 6000. The 30th partition includes the key range 87001 to 90000. (You should not use sequential keys in a large table. This example is only for demonstration.)

Each worker task sends the writes for a single partition. Within each partition, you should write the rows sequentially by primary key. Writing rows randomly, with respect to the primary key, should also provide reasonably high throughput. Measuring test runs will give you insight into which approach provides the best performance for your dataset.

If you decide not to use partitions

Writing random rows in a commit where each mutation inserts a single row may be slower than writing one row at a time. Multiple splits are likely involved, because each random row could belong to a different split. In a worst case scenario, each write involves every split in your Cloud Spanner instance. As mentioned above, write throughput is lowered when more splits are involved for a write.

Avoid pushback

It's possible to send more write requests than Cloud Spanner can handle. Cloud Spanner handles the overload by aborting transactions, which is called pushback. For write-only transactions, Cloud Spanner automatically retries the transaction. In those cases, the pushback shows up as high latency. During heavy loads, pushback can last for up to a minute. During severely heavy loads, pushback can last for several minutes. To avoid pushback, you should throttle write requests to keep CPU utilization within reasonable limits (https://cloud.google.com/spanner/docs/cpu-utilization#recommended-max).

Commit between 1 MB to 5 MB of mutations at a time

Each write to Cloud Spanner contains some overhead, whether the write is big or small. To maximize throughput, maximize the amount of data stored per write. Larger writes lower the ratio of overhead per write. A good technique is for each commit to mutate hundreds of rows. When writing relatively large rows, a commit size of 1 MB to 5 MB usually provides the best performance. When writing small values, or values that are indexed, it is generally best to write at most a few hundred rows in a single commit. Independently from the commit size and number of rows, be aware that there is a limitation of 20,000 mutations per commit (https://cloud.google.com/spanner/quotas#limits_for_creating_reading_updating_and_deleting_data). To determine optimal performance, you should test and measure (#test-measure) the throughput.

Commits larger than 5 MB or more than a few hundred rows don't provide extra benefit, and they risk exceeding the Cloud Spanner limits (https://cloud.google.com/spanner/docs/limits) on commit size and mutations per commit.

Guidelines for secondary indexes

If your database has secondary indexes (https://cloud.google.com/spanner/docs/secondary-indexes), you must choose between adding the indexes to the database schema before or after loading the table data.

In general, if you plan to create secondary indexes for your table, you should create the indexes at the same time that you create the table. Following this rule of thumb usually makes the index-creation process much more efficient. To create a table and its indexes at the same time, send the DDL statements for the new table and the new indexes in a single request to Cloud Spanner.

If you prefer, you can load the data first, then create the indexes later. However, this approach is usually slower.

Test and measure the throughput

Predicting throughput can be difficult. We recommend that you test your bulk loading strategy before running the final load. For a detailed example using partitioning and monitoring performance, see Maximizing data load throughput (https://medium.com/google-cloud/cloud-spanner-maximizing-data-load-throughput-23a0fc064b6d).

Best practices for periodic bulk loading to an existing database

If you are updating an existing database that contains data but does not have any secondary indexes (https://cloud.google.com/spanner/docs/secondary-indexes), then the recommendations in this topic still apply.

If you do have secondary indexes, the instructions might yield reasonable performance. Performance depends on how many splits (https://cloud.google.com/spanner/docs/schema-and-data-model#database-splits), on average, are involved in your transactions. If throughput drops too low, you can try the following:

- Include a smaller number of mutations in each commit, which might increase throughput.
- If your upload is larger than the total current size of the table being updated, delete your secondary indexes and then add them again after you upload the data. This step is usually not necessary, but it might improve the throughput.

Best practices for importing Avro files

These pages provide information on improving import performance of Avro files:

- Importing Cloud Spanner Avro files (https://cloud.google.com/spanner/docs/import#performance-factors)
- Importing data from other databases (https://cloud.google.com/spanner/docs/import-non-spanner)

Best practices for using the Dataflow connector

For performance tips on using the Dataflow connector, see Writing to Cloud Spanner and transforming data (https://cloud.google.com/spanner/docs/dataflow-connector#writing-transforming).

Last updated December 4, 2019.