Cloudera Runtime 7.1.0

Apache Hive Performance Tuning

Date published: 2019-08-21
Date modified:

https://docs.cloudera.com/

Legal Notice

Contents

Low-latency analytical processing

CDP Public Cloud supports low-latency analytical processing (LLAP) of Hive queries. Using LLAP available in the CDP Data Warehouse service, you can tune your data warehouse infrastructure, components, and client connection parameters to improve the performance and relevance of business intelligence and other applications.

Increasingly, enterprises want to run SQL workloads that return faster results than batch processing can provide. These enterprises often want data analytics applications to support interactive queries. Low-latency analytical processing (LLAP) can improve the performance of interactive queries. A Hive interactive query that runs on the CDP Public Cloud meets low-latency, variably guaged benchmarks to which Hive LLAP responds in 15 seconds or less. LLAP enables application development and IT infrastructure to run queries that return real-time or near-real-time results.

CDP Private Cloud Base does not support LLAP.

Query results cache

Hive filters and caches similar or identical queries in the query result cache. Caching repetitive queries can reduce the load substantially when hundreds or thousands of users of BI tools and web services query Hive.

Some operations support hundreds of thousands of users who connect to Hive using BI systems such as Tableau. In these situations, repetitious queries are inevitable. The query result cache, which is on by default, filters and stores common queries in a cache. When you issue the query again, Hive retrieves the query result from a cache instead of recomputing the result, which takes a load off the backend system.

Every query that runs in Hive 3 stores its result in a cache. Hive evicts invalid data from the cache if the input table changes. For example, if you preform aggregation and the base table changes, queries you run most frequently stay in cache, but stale queries are evicted. The query result cache works with managed tables only because Hive cannot track changes to an external table. If you join external and managed tables, Hive falls back to executing the full query. The query result cache works with ACID tables. If you update an ACID table, Hive reruns the query automatically.

Configuring the query results cache

You need to understand the query results cache to enable or disable it for debugging and to configure the memory allocated for the results.

Configuring the query results cache

Some operations support hundreds of thousands of users who connect to Hive using BI systems such as Tableau. In these situations, repetitious queries are inevitable. The query results cache, which is on by default, filters and stores common queries in a cache. When you issue the query again, Hive retrieves the query results from a cache instead of recomputing the result, which takes a load off the backend system.

Every query that runs in Hive 3 stores its result in a cache. Hive evicts invalid data from the cache if the input table changes. For example, if you preform aggregation and the base table changes, queries you run most frequently stay in cache, but stale queries are evicted. The query results cache works with managed tables only because Hive cannot track changes to an external table. If you join external and managed tables, Hive falls back to executing the full query. The query results cache works with ACID tables. If you update an ACID table, Hive reruns the query automatically.

You can enable and disable the query results cache from command line. You might want to do so to debug a query. You disable the query results cache in HiveServer by setting the following parameter to false: SET hive.query.resul ts.cache.enabled=false;

By default, Hive allocates 2GB for the query results cache. You can change this setting by configuring the following parameter in bytes: hive.query.results.cache.max.size

Related Information

Custom Configuration (about Cloudera Manager Safety Valve)
Example of using the Cloudera Manager Safety Valve

Best practices for performance tuning

Review certain performance tuning guidelines related to configuring the cluster, storing data, and writing queries so that you can protect your cluster and dependent services, disable user impersonation, automatically scale resources to handle queries, and so on.

Best practices

- Adjust autoscaling in CDP Public Cloud to scale up when you need resources to handle queries.
- Accept the default to use Tez as the execution engine. In CDP, the MapReduce execution engine is replaced by Tez.
- Accept the default to disable user impersonation. If enabled, disable hive.server2.enable.doAs in hive-site.xml using the Cloudera Manager Safety Valve feature (see link below).

  LLAP caches data for multiple queries and this capability does not support user impersonation.
- Use Ranger security service to protect your cluster and dependent services.
- Store data using the ORC File format. Others, such as Parquet are supported, but not as fast for Hive queries.
- Ensure that queries are fully vectorized by examining explain plans.

Related Information

Custom Configuration (about Cloudera Manager Safety Valve)
Example of using the Cloudera Manager Safety Valve

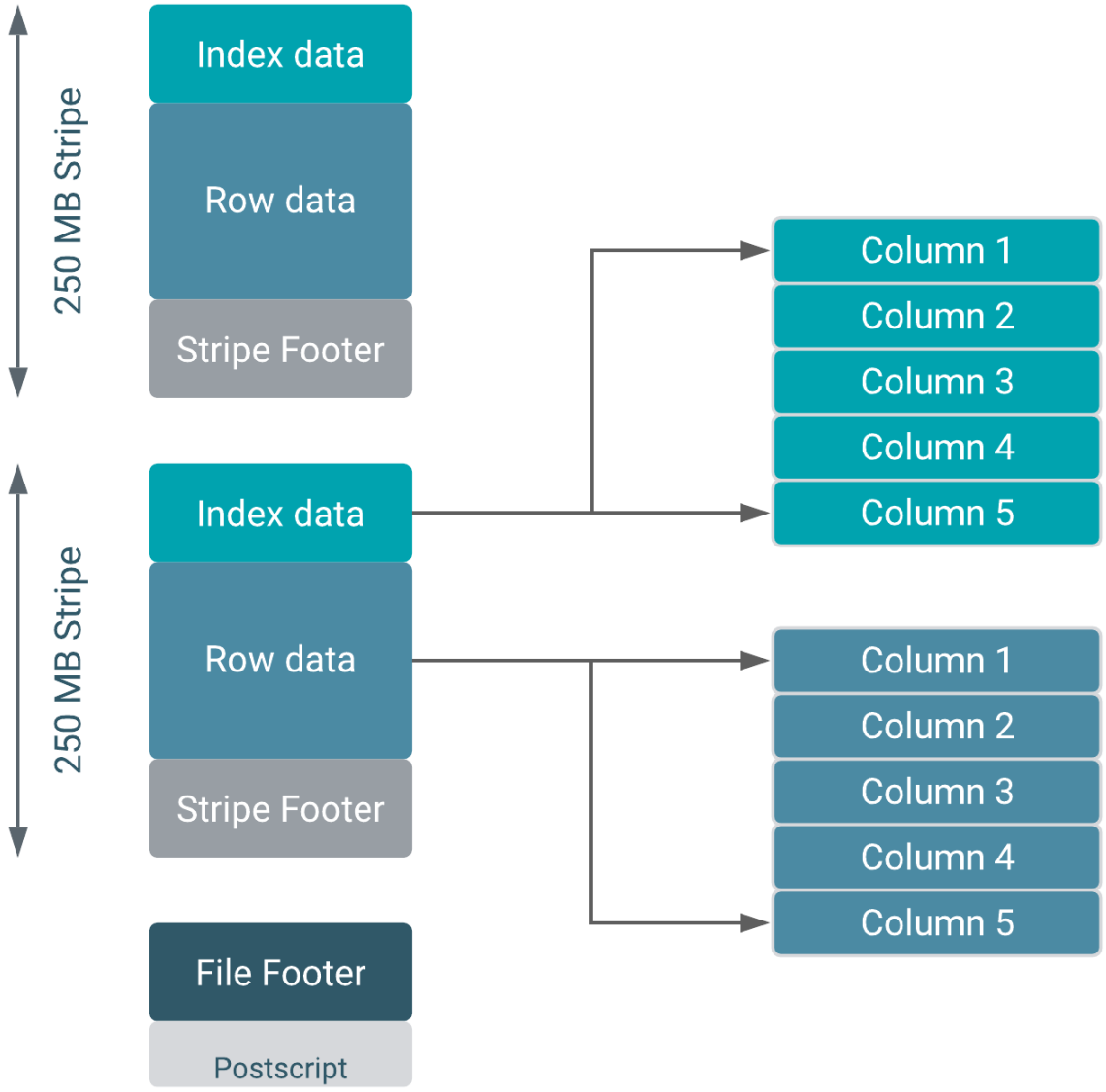
Maximizing storage resources using ORC

You can conserve storage in a number of ways, but using the Optimized Row Columnar (ORC) file format for storing Apache Hive data is most effective. ORC is the default storage for Hive data.

The ORC file format for Hive data storage is recommended for the following reasons:

- Efficient compression: Stored as columns and compressed, which leads to smaller disk reads. The columnar format is also ideal for vectorization optimizations in Tez.
- Fast reads: ORC has a built-in index, min/max values, and other aggregates that cause entire stripes to be skipped during reads. In addition, predicate pushdown pushes filters into reads so that minimal rows are read. And Bloom filters further reduce the number of rows that are returned.

- Proven in large-scale deployments: Facebook uses the ORC file format for a 300+ PB deployment.



ORC provides the best Hive performance overall. In addition, to specifying the storage format, you can also specify a compression algorithm for the table, as shown in the following example:

```
CREATE TABLE addresses (
 name string,
 street string,
 city string,
 state string,
 zip int
 ) STORED AS orc TBLPROPERTIES ("orc.compress"="Zlib");
```

Setting the compression algorithm is usually not required because your Hive settings include a default algorithm. Using ORC advanced properties, you can create bloom filters for columns frequently used in point lookups.

Hive supports Parquet and other formats for insert-only ACID tables and external tables. You can also write your own SerDes (Serializers, Deserializers) interface to support custom file formats.

Advanced ORC properties

Usually, you do not need to modify Optimized Row Columnar (ORC) properties, but occasionally, Cloudera Support advises making such changes. Review the property keys, default values, and descriptions you can configure ORC to suit your needs.

Key	Default Setting	Notes
orc.compress	ZLIB	Compression type (NONE, ZLIB, SNAPPY).
orc.compress.size	262,144	Number of bytes in each compression block.
orc.stripe.size	268,435,456	Number of bytes in each stripe.
orc.row.index.stride	10,000	Number of rows between index entries (>= 1,000).
orc.create.index	true	Sets whether to create row indexes.
orc.bloom.filter.columns	--	Comma-separated list of column names for which a Bloom filter must be created.
orc.bloom.filter.fpp	0.05	False positive probability for a Bloom filter. Must be greater than 0.0 and less than 1.0.

Related Information

Custom Configuration (about Cloudera Manager Safety Valve)
Example of using the Cloudera Manager Safety Valve

Improving performance using partitions

You must be aware of what partition pruning is, how to enable dynamic partitioning, and the configuration required for bulk-loading of data to ensure significant improvements in performance."?>You can use partitions to significantly improve performance. You can design Hive table and materialized views partitions to map to physical directories on the file system/object store. For example, a table partitioned by date-time can organize data loaded into Hive each day.

Large deployments can have tens of thousands of partitions. Partition pruning occurs indirectly when Hive discovers the partition key during query processing. For example, after joining with a dimension table, the partition key might come from the dimension table. A query filters columns by partition, limiting scanning that occurs to one or a few matching partitions. Partition pruning occurs directly when a partition key is present in the WHERE clause. Partitioned columns are virtual, not written into the main table because these columns are the same for the entire partition.

You do not need to specify dynamic partition columns. Hive generates a partition specification if you enable dynamic partitions.

Configuration for loading 1 to 9 partitions:

```
SET hive.exec.dynamic.partition.mode=nonstrict;
SET hive.exec.dynamic.partition=true;
```

For bulk-loading data into partitioned ORC tables, you use the following property, which optimizes the performance of data loading into 10 or more partitions.

Configuration for loading 10 or more partitions:

```
hive.optimize.sort.dynamic.partition=true
```

Handling bucketed tables

If you migrated data from earlier Apache Hive versions to Hive 3, you might need to handle bucketed tables that impact performance. Review how CDP simplifies handling buckets. You learn about best practices for handling dynamic capabilities.

You can divide tables or partitions into buckets, which are stored in the following ways:

- As files in the directory for the table.
- As directories of partitions if the table is partitioned.

Specifying buckets in Hive 3 tables is not necessary. In CDP, Hive 3 buckets data implicitly, and does not require a user key or user-provided bucket number as earlier versions (ACID V1) did. For example:

V1:

```
CREATE TABLE hello_acid (load_date date, key int, value int)
CLUSTERED BY(key) INTO 3 BUCKETS
STORED AS ORC TBLPROPERTIES ('transactional'='true');
```

V2:

```
CREATE TABLE hello_acid_v2 (load_date date, key int, value int);
```

Performance of ACID V2 tables is on a par with non-ACID tables using buckets. ACID V2 tables are compatible with native cloud storage.

A common challenge related to using buckets in tables migrated from earlier versions is maintaining query performance while the workload or data scales up or down. For example, you could have an environment that operates smoothly using 16 buckets to support 1000 users, but a spike in the number of users to 100,000 for a day or two creates problems if you do not promptly tune the buckets and partitions. Tuning the buckets is complicated by the fact that after you have constructed a table with buckets, the entire table containing the bucketed data must be reloaded to reduce, add, or eliminate buckets.

In CDP, which uses Tez, you only need to deal with the buckets of the biggest table. If workload demands change rapidly, the buckets of the smaller tables dynamically change to complete table JOINs.

You perform the following tasks related to buckets:

- Setting hive-site.xml to enable buckets

  SET hive.tez.bucket.pruning=true
- Bulk-loading tables that are both partitioned and bucketed:

  When you load data into tables that are both partitioned and bucketed, set the following property to optimize the process:

  SET hive.optimize.sort.dynamic.partition=true

If you have 20 buckets on user_id data, the following query returns only the data associated with user_id = 1: SELE CT * FROM tab WHERE user_id = 1;

To best leverage the dynamic capability of table buckets on Tez, adopt the following practices:

- Use a single key for the buckets of the largest table.
- Usually, you need to bucket the main table by the biggest dimension table. For example, the sales table might be bucketed by customer and not by merchandise item or store. However, in this scenario, the sales table is sorted by item and store.
- Normally, do not bucket and sort on the same column.

A table that has more bucket files than the number of rows is an indication that you should reconsider how the table is bucketed.