Cloudera Runtime 7.3.1

Apache Hive Performance Tuning

Date published: 2019-08-21
Date modified: 2024-12-10

CLOUDERA

https://docs.cloudera.com/

Legal Notice

The documentation is and contains Cloudera proprietary information protected by copyright and other intellectual property rights. No license under copyright or any other intellectual property right is granted herein.

Unless otherwise noted, scripts and sample code are licensed under the Apache License, Version 2.0.

Copyright information for Cloudera software may be found within the documentation accompanying each component in a particular release.

Cloudera software includes software from various open source or other third party projects, and may be released under the Apache Software License 2.0 (“ASLv2”), the Affero General Public License version 3 (AGPLv3), or other license terms. Other software included may be released under the terms of alternative open source licenses. Please review the license and notice files accompanying the software for additional licensing information.

Please visit the Cloudera software product page for more information on Cloudera software. For more information on Cloudera support services, please visit either the Support or Sales page. Feel free to contact us directly to discuss your specific needs.

Cloudera reserves the right to change any products at any time, and without notice. Cloudera assumes no responsibility nor liability arising from the use of products, except as expressly agreed to in writing by Cloudera.

Cloudera, Cloudera Altus, HUE, Impala, Cloudera Impala, and other Cloudera marks are registered or unregistered trademarks in the United States and other countries. All other trademarks are the property of their respective owners.

Disclaimer: EXCEPT AS EXPRESSLY PROVIDED IN A WRITTEN AGREEMENT WITH CLOUDERA, CLOUDERA DOES NOT MAKE NOR GIVE ANY REPRESENTATION, WARRANTY, NOR COVENANT OF ANY KIND, WHETHER EXPRESS OR IMPLIED, IN CONNECTION WITH CLOUDERA TECHNOLOGY OR RELATED SUPPORT PROVIDED IN CONNECTION THEREWITH. CLOUDERA DOES NOT WARRANT THAT CLOUDERA PRODUCTS NOR SOFTWARE WILL OPERATE UNINTERRUPTED NOR THAT IT WILL BE FREE FROM DEFECTS NOR ERRORS, THAT IT WILL PROTECT YOUR DATA FROM LOSS, CORRUPTION NOR UNAVAILABILITY, NOR THAT IT WILL MEET ALL OF CUSTOMER’S BUSINESS REQUIREMENTS. WITHOUT LIMITING THE FOREGOING, AND TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, CLOUDERA EXPRESSLY DISCLAIMS ANY AND ALL IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, QUALITY, NON-INFRINGEMENT, TITLE, AND FITNESS FOR A PARTICULAR PURPOSE AND ANY REPRESENTATION, WARRANTY, OR COVENANT BASED ON COURSE OF DEALING OR USAGE IN TRADE.

Contents

Query results cache

Hive filters and caches similar or identical queries in the query results 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 vast user populations, 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 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 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.

Configuring the query results cache

You can enable and disable the query results cache from the 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.results.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

Managing high partition workloads

You learn how to identify an error related to high partition workloads that require configuration of your Hive Virtual Warehouse to run successfully. You might need to configure your Virtual Warehouse to prevent these errors when inserting data into workloads with a large number of partitions and columns.

About this task

To prevent an error when inserting data into a high partition workload, such as a table having 5,000 partitions and 100 columns, configure the Hive Virtual Warehouse as described in the steps below.

The error message you might see when inserting data into a high partition workload that is not configured properly looks something like this:

```
ERROR : FAILED: Execution Error, return code 40000 from org.apache.hadoop.hi
ve.ql.exec.MoveTask. MetaException(message:One or more instances could not b
e made persistent)
```

Procedure

1. Try changing the size of your Hive Virtual Warehouse.
   For example, if you experience the high partition error in a small Virtual Warehouse, try using a medium Virtual Warehouse and set min/max nodes to 40.
2. In the CDW UI, in Overview, go to the Virtual Warehouses tab, click ⋮ , and select Edit corresponding to your Virtual Warehouse.
3. Click Configurations HiveServer2 .

4. Tune HS2 parameters in the Virtual Warehouse.
   For example, search for and set the following properties, or if the property is not found, click Add Custom Configuration to add a custom configuration, and set it as follows:

   ```
   set hive.optimize.sort.dynamic.partition.threshold=0;
   set hive.thrift.client.max.message.size=2147483647;
   set hive.metastore.client.skip.columns.for.partitions=true;
   set hive.stats.autogather=false;
   set hive.stats.column.autogather=false;
   set hive.msck.repair.batch.size=[***TABLE SCHEMA SIZE***];
   ```

5. Go to the Database Catalogs tab, and select your Database Catalog.
6. Click Options ⋮ , and select Edit.
7. Tune Metastore parameters in the Database Catalog.
   For example, click Configurations Metastore , search for and set the following properties, or if the property is not found, click + to add a custom configuration, and set it as follows.

   ```
   hive.metastore.direct.sql.batch.size=[***TABLE SCHEMA SIZE***]
   hive.txn.timeout=3600
   hive.metastore.try.direct.sql=true
   hive.metastore.try.direct.sql.ddl=true
   ```

8. Click Apply Changes.

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, automatically scale resources to handle queries, and so on.

Best practices

- 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

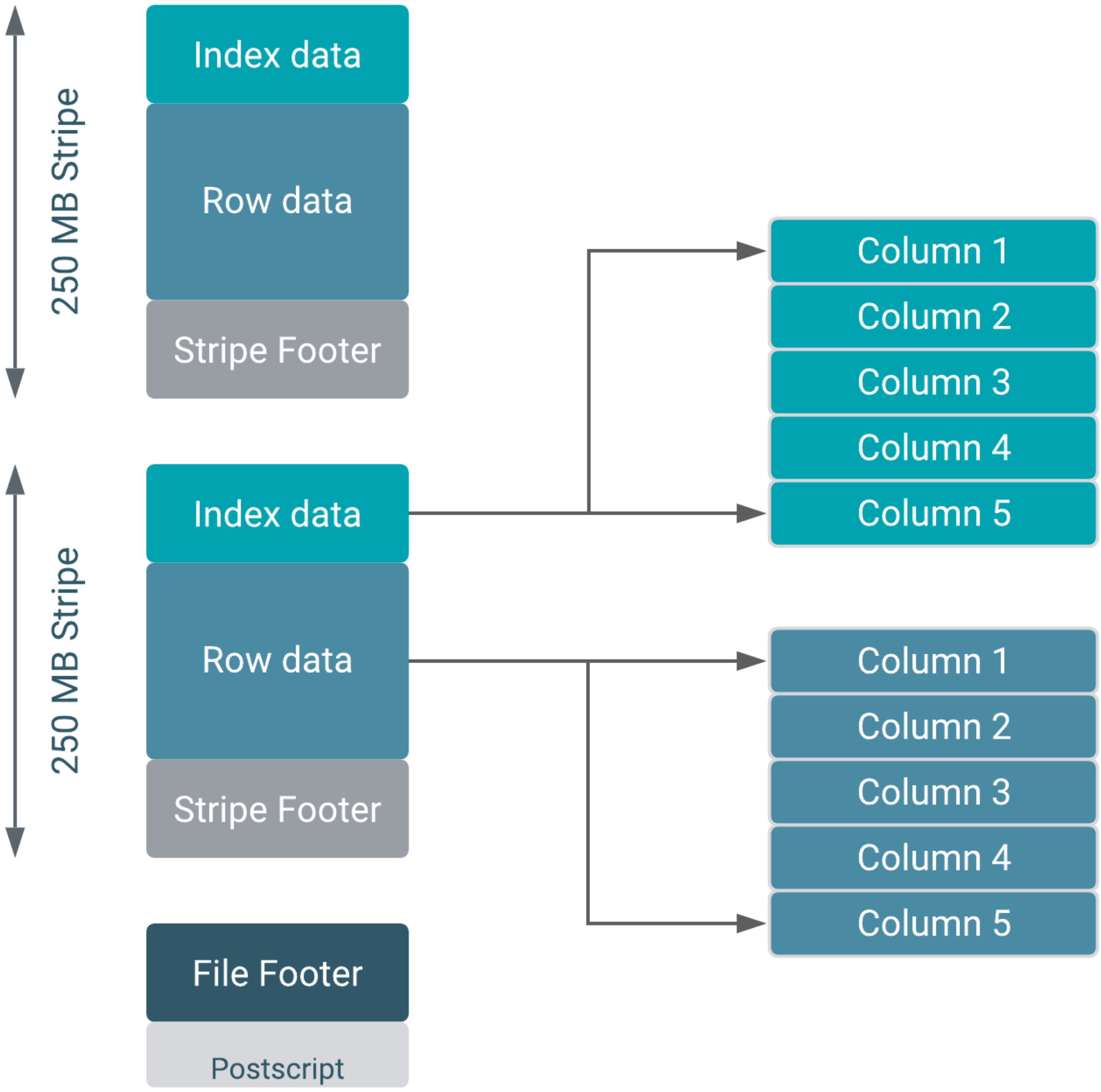
ORC file format

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.
- 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: Meta (aka 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.

Property keys and defaults

You use the Safety Valve feature in Cloudera Manager to change ORC properties.

Key	Default Setting	Description
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

Performance improvement using partitions

You must be aware of partition pruning benefits, 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
```

Apache Tez and Hive LLAP

Cloudera Data Warehouse supports low-latency analytical processing (LLAP) of Hive queries.

Hive uses Apache Tez to execute queries internally. Apache Tez provides the following execution modes:

- Container mode

  Every time you run a Hive query, Tez requests a container from YARN.
- LLAP mode

  Every time you run a Hive query, Tez asks the LLAP daemon for a free thread, and starts running a fragment.

Apache Tez provides the framework to run a job that creates a graph with vertexes and tasks. SQL semantics for deciding the query physical plan, which identifies how to execute the query in a distributed fashion, is based on Apache Tez. The entire execution plan is created under this framework. SQL syntax in Hive is the same irrespective of execution engine (mode) used in Hive.

Apache Tez does not have to start from the ground up, requesting a container and waiting for a JVM, to run a fragment in LLAP mode. LLAP mode provides dedicated capacity. Caching is automated. The primary difference between LLAP mode and container mode, is that in LLAP mode the LLAP executors are used to run query fragments.

In Cloudera Data Warehouse (CDW), the Hive execution mode is LLAP. In Cloudera Data Hub on CDP Public Cloud and CDP Private Cloud Base, the Hive execution mode is container, and LLAP mode is not supported. When Apache Tez runs Hive in container mode, it has traditionally been called Hive on Tez.

Bucketed tables in Hive

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, 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.

Bucket configurations

You can enable buckets as follows:

```
SET hive.tez.bucket.pruning=true
```

When you load data into tables that are both partitioned and bucketed, set the hive.optimize.sort.dynamic.partition 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:

```
SELECT * FROM tab WHERE user_id = 1;
```

To best leverage the dynamic capability of table buckets, 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.