Hive Optimization Techniques

IMPORTANT

Copyright Infringement and Illegal Content Sharing Notice

1. Partitioning - Works by dividing the data into smaller segments. These segments are created using logical groupings.
For example state can be the column on which partitioning can be done.
We finally scan only one partition and avoid scanning the other partitions.
This gives lot of performance gains.
We can do partitioning on columns where we have less number of distinct values.

2. Bucketing - Works also by dividing the data into smaller segments. These segments are created based on system defined hash functions.
For example productid , customerid can be the columns on which we can do bucketing.
We finally scan only one bucket and avoid scanning the other buckets. This gives lot of performance gains.
We can do bucketing on columns where we have large number of distinct values.

3. Join optimizations techniques- Map side join , Bucket Map Join, Sort Merge Bucket Join also called as SMB join.

4. Vectorization -
Vectorization improves the performance by fetching 1,024 rows in a single operation instead of fetching single row each time. It improves the performance for operations like filter, join, aggregation, etc.

5. Changing the execution engine to Tez or Spark as mapreduce is quite slow.

6. Use Orc file format with a compression like snappy.

Orc provides highly efficient ways of storing the hive data by reducing the data storage format by 75%
of the original. It uses techniques like predicate push-down, compression, and more to improve the performance of the query.
(predicate push-down means filtered are performed earlier at storage level)

Snappy provides a fast compression.

7. UDF's are not very optimized.

filters operations are evaluated left-to-right, so for best performance, put UDFs on the right in an ANDed list of expressions in the WHERE clause. E.g., use

column1 = 10 and myUDF(column2) = "x"

instead of

myUDF(column2) = "x" and column1 = 10

8. Cost-based optimization
CBO in Hive is powered by Apache Calcite (http://calcite.apache.org/), which is an open source, enterprise-grade cost-based logical optimizer and query execution framework.

Hive CBO generates efficient execution plans by examining the query cost, which is collected by ANALYZE statements, ultimately cutting down on query execution time and reducing resource utilization.

To use CBO, set the following properties:

SET hive.cbo.enable=true; -- default true after v0.14.0
SET hive.compute.query.using.stats=true; -- default false
SET hive.stats.fetch.column.stats=true; -- default false
SET hive.stats.fetch.partition.stats=true; -- default true

However we do not have to worry. This will be configured at cluster level and as a developer we do not have to do anything.

The below link can help DBA's to understand internals of it

https://docs.cloudera.com/HDPDocuments/HDP2/HDP-2.6.5/bk_hive-performance-tuning/content/ch_cost-based-optimizer.html

We have learnt Optimization Techniques in hive

Happy Learning!!!

Follow US

Trainer	Mr. Sumit Mittal
Phone	9108179578
Email	trendytech.sumit@gmail.com
Website	https://trendytech.in/courses/big-data-online-training/
LinkedIn	https://www.linkedin.com/in/bigdatabysumit/
Twitter	@BigdataBySumit
Instagram	bigdatabysumit
Facebook	https://www.facebook.com/trendytech.in/
Youtube	https://www.youtube.com/channel/UCbTggJVf0NDTfWX-C_gUGSg