10 ways to Optimize Hive Queries

To get best out of Hadoop, you need to know How to use it efficiently. Initially Apache Hadoop's Hive was not best choice for interactive and ad-hoc queries, but now with new enhancements, It has gained popularity for both ad-hoc query and regular ETL workload. While It is one of the mature and popular solutions in Hadoop ece-system, There are lots of scope for Data Scientist and Engineers to learn to use it efficiently.
Quiet some time I was involved in migrating ETL workload from Enterprise Data Warehouse like Teradata and Netezza etc. to Hive.
Here is how I started :

- Extracted EDW Table to Hive using Sqoop
- ETL script which are simply SQL statements have been made compatible and executable on Hive.
- Execute migrated ETL scripts on Hive.

ETL migration to HIVE was not smooth and Queries were taking time more than queries in ETW. It was not optimized and was performing worst. To optimize queries on HIVE , You need to think about :

- What are query pattern?
- How should Table schema be ? Table should be denormalized and flatten to avoid costly joins.
- Enable parallel execution.
- Hive tuning etc.

While ETL migration to Hive, Following are 10 hard rule that I had followed

. It helped me reducing query time drastically.

1. Demoralise table and use Filtering and projection as early as possible to reduce data before join.

With Hive, a good practice is to denormalize your data. Join is costly affair and requires extra map-reduce phase to accomplish query job. With De-normalisation, the data is present in the same table so there is no need for any joins, hence the selects are very fast.

As join requires data to be shuffled across nodes, Use Filtering and projection as early as possible to reduce data before join.

Hive automatically does Filtering and Projection ahead of join through *Projection Pruning* and *Predicate Push down. We will talk about CBO (Cast Based Optimization) later.*

2. Optimize Joins

- Sort data ahead of time to simplify joins
- Use map (broadcast) joins whenever possible

A MAPJOIN can be invoked either through an optimizer hint:

```
      select /*+ MAPJOIN(time_dim) */ count(*) from store_sales
```

With Hive 0.14.0, Hint are no longer needed and Map joins are automatically picked up by the optimizer.

* OR auto join conversion

```
      SET hive.auto.convert.join=true;
      SET hive.mapjoin.smalltable.filesize=1000000000;  (1GB)
```

Optimize Chains of Map Joins

```
      SET hive.auto.convert.join=true;
      select count(*) from store_sales join time_dim on (ss_sol
```

The option can be enabled with two configuration parameters:

```
        SET hive.auto.convert.join.noconditionaltask = true;
        SET hive.auto.convert.join.noconditionaltask.size = 10000
```

Auto Conversion to SMB Map Join

- Sort and Bucket on common join keys

Sort-Merge-Bucket (SMB) joins can be converted to SMB map joins as well.SMB joins are used wherever the tables are sorted and bucketed.

```
        SET hive.auto.convert.sortmerge.join=true;
        SET hive.optimize.bucketmapjoin = true;
        SET hive.optimize.bucketmapjoin.sortedmerge = true;
        SET hive.auto.convert.sortmerge.join.noconditionaltask=tr
```

There is an option to set the big table selection policy using the following configuration:

```
        SET hive.auto.convert.sortmerge.join.bigtable.selection.p
```

By default, the selection policy is average partition size. The big table selection policy helps determine which table to choose for only streaming, as compared to hashing and streaming.

The available selection policies are:

```
        org.apache.hadoop.hive.ql.optimizer.AvgPartitionSizeBased
        org.apache.hadoop.hive.ql.optimizer.LeftmostBigTableSelec
        org.apache.hadoop.hive.ql.optimizer.TableSizeBasedBigTabl
```

For More Refer : Join Optimization [https://cwiki.apache.org/confluence/display/Hive/LanguageManual+JoinOptimization]

3. Leveraging Cost Based Optimization

Hive 0.13 is support for Cost Based Optimization (CBO) that leverages statistics collected on Hive tables.

Currently logical query optimizations in Hive can be broadly categorized as follows:

- Projection Pruning
- Deducing Transitive Predicates
- Predicate Push down
- Merging of Select-Select, Filter-Filter in to single operator
- Multi-way Join
- Query Rewrite to accommodate for Join skew on some column values.

We are not discussing CBO here in detail, but will only talk about enabling CBO in Hive.

To collect statistics for the Hive query optimizer, set two parameters:

```
        SET hive.compute.query.using.stats=true;
        SET hive.stats.dbclass=fs;
        SET hive.stats.fetch.column.stats=true;
        SET hive.stats.fetch.partition.stats=true;
```

and then within Hive, enter the command:

```
ANALYZE TABLE orc_table COMPUTE STATISTICS;
```

This computes statistics for the table in question, which can then be utilized by Hive's CBO engine. Together with Tez, CBO can produce significant performance improvements for queries.

Predicate Push down can be enabled by setting :

```
SET hive.optimize.ppd=true;
```

To enable Cost based optimization in Hive :

```
SET hive.cbo.enable=true;
```

For more refer : Cost based optimization in Hive [https://cwiki.apache.org/confluence/display/Hive/Cost-based+optimization+in+Hive]

4. Partition Tables ,bucketing Tables and Skewed Tables

Divide data amongst different files which can be pruned out by using partitions, buckets and skews.

- **Partitioning**

Hive is frequently used to query from larger tables and run queries through full table scan. Since Hive tables are linked to physical directory in HDFS, Hive scans through data stored in that directory and applies filtering as it goes through data. This process claims unnecessary time and resources as full table scan is required at each time you run query.

In order to improve query performance, You can use Hive table partitioning to help Hive to focus on only data that is important.Partitioning a table stores data in sub-directories categorized by table location, which allows Hive to exclude unnecessary data from queries without reading all the data every time a new query is made.
For example, if the you wanted to partition data based on department, it would define the department column as a partition as it was creating the Hive table. Then, as data is written to the table, it will be written to sub-directories named by the department.

```
SELECT * FROM employee WHERE department = ‘ADMIN’;
```

When you query for a department as given for 'ADMIN' department , will skip all sub-directories that doesn't have data for department 'ADMIN'.

When partitioning table , You need to pay attention in designing partition key. If you select the partition key which have very low cardinality like Gender column, then there will be only few partition. in case for gender partition key , will be only two. It will hardly give major performance benefit.

And if you select the partition key which have very high cardinality like last name/first name, you will end-up reading lots of partitions and increased overhead maintaining partitions.
While partitioning is a great technique to improve performance by distributing data, it also requires a careful review of the number of partitions and frequency of partition additions/deletions. Creating partitions in the scale of 10's of thousands should be avoided unless there is a very strong reason.

- **Dynamic Partitioning**

Hive does support Dynamic Partitioning (DP) where columns valuea are only known at EXECUTION TIME. We have a parameter hive.exec.dynamic.partition=true/false to control whether to allow dynamic partition at all. To enable Dynamic Partitioning :

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

Dynamic partition insert could potentially resource hog in that it could generate a large number of partitions in a short time. To get yourself buckled, There are three parameters :

- **hive.exec.max.dynamic.partitions.pernode** (default value being 100) is the maximum dynamic partitions that can be created by each mapper or reducer.
- **hive.exec.max.dynamic.partitions** (default value being 1000) is the total number of dynamic partitions could be created by one DML.
- **hive.exec.max.created.files** (default value being 100000) is the maximum total number of files created by all mappers and reducers.

Another situation we want to protect against dynamic partition insert is that the user may accidentally specify all partitions to be dynamic partitions without specifying one static partition, while the original intention is to just overwrite the sub-partitions of one root partition. We define another parameter hive.exec.dynamic.partition.mode=strict to prevent the all-dynamic partition case. In the strict mode, you have to specify at least one static partition. The default mode is strict.

```
SET hive.exec.dynamic.partition.mode=strict;
```

I have observed instances where large number of partition say 10K or more have resulted meta-store overhead and slowed down operation gradually. Some times insert operation keep hanging for infinite as number of partition increases on table.

For more refer : Dynamic Partitioing [https://cwiki.apache.org/confluence/display/Hive/Tutorial#Tutorial-Dynamic-PartitionInsert]

- **Bucketing**

  every time before writing data to the bucketed table

```
SET hive.enforce.bucketing=true;
SET hive.optimize.bucketmapjoin=true;
```

- **Skewed Table**

5. Parallel execution

Applies to MapReduce jobs that can run in parallel, for example jobs processing different source tables before a join.

```
SET hive.exec.parallel=true;
```

[http://4.bp.blogspot.com/--_WY73MCY2U/VYVXrpnAyAI/AAAAAAAAA_o/osJzLYqX_zc/s1600/parallal.jpg]

6. Use the "ORC" file format

The *Optimized Row Columnar* (ORC) file format provides a highly efficient way to store Hive data. It was designed to overcome limitations of the other Hive file formats. Using ORC files improves performance when Hive is reading, writing, and processing data. Remember that the ORC file format is new as of Hive 0.11.

File formats are specified at the table (or partition) level. You can specify the ORC file format with HiveQL statements such as these:

```
        CREATE TABLE ... STORED AS ORC
        ALTER TABLE ... [PARTITION partition_spec] SET FILEFORMA
        SET hive.default.fileformat=Orc
```

For More refer : **LanguageManual+ORC [https://cwiki.apache.org/confluence/display/Hive/LanguageManual+ORC]**

7. Enabling vectorization

Vectorization is a parallel processing technique in which operation are applied on multiple rows od data rather than single row. In Hive, vectorization works with a column of data (the vector) and applies a single operation to the entire column. Having access to data in columnar format is crucial for vectorization, so it only works with ORC formatted tables.

Vectorization along with ORC format does miracle sometimes and improved performance a lot.

To enable Vectorization :

```
        SET hive.vectorized.execution.enabled=true;
```

8. Hive Indexing

The goal of Hive indexing is to improve the speed of query lookup on certain columns of a table.
Hive doesn’t provide automatic index maintenance, so you need to rebuild the index if you overwrite or append data to the table. Also, Hive indexes support table partitions, so a rebuild can be limited to a partition.

Hive indexing was added in version 0.7.0, and bitmap indexing was added in version 0.8.0.

For more refer : Indexing Manual [https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Indexing]

9. Tez as an Execution Engine

By Default, Hive uses Map-Reduce(mr) as execution engine. You can often use Tez as execution engine to improve performance.

```
        SET hive.execution.engine=tez;
```

Enable Hive to use Tez DAG APIs. On the Hive client machine, add the following to your Hive script or execute the following in the hive shell

```
        SET hive.use.tez.natively=true;
```

Allow Hive to combine multipe mapreduce (MR) jobs into a *single MRR job* where possible:

```
        SET hive.enable.mrr=true;
```

10. Tune configuration

- **Adjust Configuration for Uneven Distribution**

```
             SET hive.groupby.skewindata=true;
```

- **Increase the replication factor on dimension tables.**
- **Use the distributed cache to hold dimension tables.**
- **Don't forget the hive.exec.mode.local.auto configuration variable.**

```
        SET hive.exec.mode.local.auto=true;
```

- **Compress map/reduce output**

```
                    SET mapred.compress.map.output=true;
                    SET mapred.output.compress=true;
```

- **Increase Parallelism**

By default, the parallelism is dictated by the size of the data set and the default block size of 64 MB.
To increase number of mapper , reduce split size :

```
             SET mapred.max.split.size=1000000; (1 MB)
```

And adjust following parameters :

```
             SET mapreduce.input.fileinputformat.split.maxsize
             SET mapreduce.input.fileinputformat.split.minsize
```

If the min is too large, you will have too few mappers. If the max is too small, you will have too many mappers.

```
              SET mapreduce.tasks.io.sort.mb
```

Hope that helped in improving hive performance. Will try to add thing that can help optimizing things.

Posted 27th May 2015 by sanjiv singh

Labels: Apache Hadoop, apache hive, Configure hive for optimization, Hadoop, Hive, Hive tuning, improve hive queries, optimization, Optimize hive queries

5 View comments

5 comments

Add a comment as Vamsi Krishna

Top comments

sanjiv singh shared this via Google+ 1 year ago - Shared publicly

1 · Reply

sanjiv singh via Google+ 1 year ago - Shared publicly

10 ways to Optimizing Hive Queries

To get best out of Hadoop, you need to know How to use it efficiently. Initially Apache Hadoop's Hive was not best choice for interactive and ad-hoc queries, but now with new enhancements, It has gained popularity for both ad-hoc query and regular ETL work...

+1 1 · Reply

Bigdata Spark Online Training 1 year ago - Shared publicly

How to optimize **#hive** http://sanjivblogs.blogspot.com/2015/05/10-ways-to-optimizing-hive-queries.html

1 · Reply

Venkatesh M 1 week ago - Shared publicly

Thank you.. Very helpful post for hive queries... The following link also give some posts for hive and hadoop ecosystems with examples http://www.geoinsyssoft.com/blog/

1 · Reply

Rakesh Kam 6 months ago - Shared publicly

high and extrordinary

nice

1 · Reply