February 24, 2015

5 WAYS TO MAKE YOUR HIVE QUERIES RUN FASTER

by Ofer Mendelevitch

As a data scientist working with Hadoop, I often use Apache Hive to explore data, make ad-hoc queries or build data pipelines.

Until recently, optimizing Hive queries focused mostly on data layout techniques such as partitioning and bucketing or using custom file formats.

In the last couple of years, driven largely by the innovation of the Hive community around the Stinger initiative, Hive query time has improved dramatically, enabling Hive to support both batch and interactive workloads at speed and at scale.

However, many data scientists remain unfamiliar with basic techniques and best practices for running their Hive queries at maximum speed. In this blog post, I will highlight a few simple techniques I use most often to improve performance of my HIVE queries.

TECHNIQUE #1: USE TEZ

Hive can use the Apache Tez execution engine instead of the venerable Map-reduce engine. I won't go into details about the many benefits of using Tez which are mentioned here; instead, I want to make a simple recommendation: if

following in the beginning of your Hive query:

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

With the above setting, every HIVE query you execute will take advantage of Tez.

TECHNIQUE #2: USE ORCFILE

Hive supports ORCfile, a new table storage format that sports fantastic speed improvements through techniques like predicate push-down, compression and more.

Using ORCFile for every HIVE table should really be a no-brainer and extremely beneficial to get fast response times for your HIVE queries.

As an example, consider two large tables A and B (stored as text files, with some columns not all specified here), and a simple query like:

```
SELECT A.customerID, A.name, A.age, A.address join
B.role, B.department, B.salary
ON A.customerID=B.customerID;
```

This query may take a long time to execute since tables A and B are both stored as TEXT. Converting these tables to ORCFile format will usually reduce query time significantly:

```
CREATE TABLE A_ORC (

customerID int, name string, age int, address string

) STORED AS ORC tblproperties ("orc.compress" = "SNAPPY");

INSERT INTO TABLE A_ORC SELECT * FROM A;
```

```
customerID int, role string, salary float, department string

) STORED AS ORC tblproperties ("orc.compress" = "SNAPPY");

INSERT INTO TABLE B_ORC SELECT * FROM B;

SELECT A_ORC.customerID, A_ORC.name,

A_ORC.age, A_ORC.address join

B_ORC.role, B_ORC.department, B_ORC.salary

ON A_ORC.customerID=B_ORC.customerID;
```

ORC supports compressed storage (with ZLIB or as shown above with SNAPPY) but also uncompressed storage.

Converting base tables to ORC is often the responsibility of your ingest team, and it may take them some time to change the complete ingestion process due to other priorities. The benefits of ORCFile are so tangible that I often recommend a do-it-yourself approach as demonstrated above – convert A into A_ORC and B into B_ORC and do the join that way, so that you benefit from faster queries immediately, with no dependencies on other teams.

TECHNIQUE #3: USE VECTORIZATION

Vectorized query execution improves performance of operations like scans, aggregations, filters and joins, by performing them in batches of 1024 rows at once instead of single row each time.

Introduced in Hive 0.13, this feature significantly improves query execution time, and is easily enabled with two parameters settings:

```
set hive.vectorized.execution.reduce.enabled = true;
```

TECHNIQUE #4: COST BASED QUERY OPTIMIZATION

Hive optimizes each query's logical and physical execution plan before submitting for final execution. These optimizations are not based on the cost of the query – that is, until now.

A recent addition to Hive, Cost-based optimization, performs further optimizations based on query cost, resulting in potentially different decisions: how to order joins, which type of join to perform, degree of parallelism and others.

To use cost-based optimization (also known as CBO), set the following parameters at the beginning of your query:

```
set hive.cbo.enable=true;
set hive.compute.query.using.stats=true;
set hive.stats.fetch.column.stats=true;
set hive.stats.fetch.partition.stats=true;
```

Then, prepare the data for CBO by running Hive's "analyze" command to collect various statistics on the tables for which we want to use CBO.

For example, in a table tweets we want to collect statistics about the table and about 2 columns: "sender" and "topic":

```
analyze table tweets compute statistics;
analyze table tweets compute statistics for columns sender, topic;
```

With HIVE 0.14 (on HDP 2.2) the analyze command works much faster, and you don't need to specify each column, so you can just issue:

```
analyze table tweets compute statistics for columns;
```

That's it. Now executing a query using this table should result in a different execution plan that is faster because of the cost calculation and different execution plan created by Hive.

TECHNIQUE #5: WRITE GOOD SQL

SQL is a powerful declarative language. Like other declarative languages, there is more than one way to write a SQL statement. Although each statement's functionality is the same, it may have strikingly different performance characteristics.

Let's look at an example. Consider a click-stream event table:

```
CREATE TABLE clicks (
timestamp date, sessionID string, url string, source_ip string
) STORED as ORC tblproperties ("orc.compress" = "SNAPPY");
```

Each record represents a click event, and we would like to find the latest URL for each sessionID.

One might consider the following approach:

```
SELECT clicks.* FROM clicks inner join
(select sessionID, max(timestamp) as max_ts from clicks
group by sessionID) latest
ON clicks.sessionID = latest.sessionID and
clicks.timestamp = latest.max_ts;
```

In the above query, we build a sub-query to collect the timestamp of the latest event in each session, and then use an inner join to filter out the rest.

turns out there's a better way to re-write this query as follows:

```
SELECT * FROM
(SELECT *, RANK() over (partition by sessionID,
order by timestamp desc) as rank
FROM clicks) ranked_clicks
WHERE ranked_clicks.rank=1;
```

Here we use Hive's OLAP functionality (OVER and RANK) to achieve the same thing, but without a Join.

Clearly, removing an unnecessary join will almost always result in better performance, and when using big data this is more important than ever. I find many cases where queries are not optimal — so look carefully at every query and consider if a rewrite can make it better and faster.

SUMMARY

Apache Hive is a powerful tool in the hands of data analysts and data scientists, and supports a variety of batch and interactive workloads.

In this blog post, I've discussed some useful techniques—the ones I use most often and find most useful for my day-to-day work as a data scientist—to make Hive queries run faster.

Thankfully, the Hive community is not finished yet. Even between HIVE 0.13 and HIVE 0.14, there are dramatic improvements in ORCFiles, vectorization and CBO and how they positively impact query performance.

I'm really excited about Stinger.next, bringing performance improvements to the sub-second range.

I can't wait.

Categories:

SHARE

COMMENTS

- **Bhaskar** says:

March 15, 2015 at 2:14 pm

Your comment is awaiting moderation.

great know this. Good article

Reply

- **anand** says:

April 14, 2015 at 12:19 pm

precise & straight forward explanation, its helpful to setup & monitor the performance

Reply

- **Jack** says:

April 27, 2015 at 11:44 pm

Your comment is awaiting moderation.

Great post Ofer!

You're forgetting the 6th way though: Use http://www.atscale.com

AtScale lets regular users to get screaming performance and multi-dimensional analysis capabilities on Hadoop, straight from the tools they already know (Tableau, Qlik, Excel…). Sometimes 200X faster.

Business users love this and so does IT. With AtScale, they don't have to move data, install new drivers on end-user machines or deploy invasive bits on their clusters.

Reply

- **Naaraayanan** says:

June 5, 2015 at 4:32 am

Excellent. Explained to understand easily and can do practice.

Reply

October 17, 2015 at 2:15 am

Thanks For Your valuable posting,this is for wonderful sharing,i would like to see more information from your side.i am working in Erp Software Company In Dubai

Reply

- **rakesh** says:

January 12, 2016 at 4:05 am

extraordinary understanding

Reply

- **Ramesh Sreera** says:

May 7, 2016 at 8:15 pm

Excellent article to the point for performance tuning hive queries

Reply

- **Balachandar** says:

July 18, 2016 at 9:22 pm

Very good points mentioned in this article. It will help in the coming days. Thanks

Reply

- **Thiru** says:

August 3, 2016 at 4:48 pm

Very good information to optimize Hive queries

Reply

LEAVE A REPLY

Your email address will not be published. Required fields are marked *

Comment

SHARE

You may use these HTML tags and attributes: `<a href="" title=""> <abbr title=""> <acronym title=""> <b> <blockquote cite=""> <cite> <code> <del datetime=""> <em> <i> <q cite=""> <s> <strike> <strong>`

Name *

Email *

Website

POST COMMENT

RELATED POSTS

BLOG

8.30.16

Announcing the Availability of Hortonworks Data Platform 2.5

Hortonworks Empowers Organizations to Maximize the

SHARE

8.8.16

Top Articles on HCC -- Hortonworks Community

It has been another exciting week on Hortonworks Community Connection HCC. We continue to see great activity and recommend

8.1.16

Top Articles on HCC -- Hortonworks Community

It has been another exciting week on Hortonworks Community Connection HCC. We continue to see great activity and recommend

VIEW ALL EVENTS & WEBCASTS

SHARE