Apache Impala: My Insights and Best Practices

How did we make our Impala run faster?

Adir Mashiach
Mar 21, 2018 · 7 min read

So you have your Hadoop, terabytes of data are getting into it per day, ETLs are done 24/7 with Spark, Hive or god forbid — Pig. And then after the data is in the exact shape you want it to be (or even before that) and everything is just perfect — analysts want to query it. If you chose Impala for that mission, this article is for you.

Impala In Our Data Lake

We use Impala for a few purposes:

- Let analysts query new types of data that the data engineers haven't created any ETLs on yet.
- Let analysts query data that its final destination (after the ETL process) is HDFS.
- Report generation (with our own tools).

- Monitoring & alert systems (on new data that's getting in every hour).

We have tens of thousands of queries per day, each query scans on average a few gigabytes of data and takes 10 seconds. Our cluster is not huge in terms of hardware and number of nodes we have. Not to mention we have hundreds of other workflows (Spark, Pig, Hive) running every day on the same cluster.

In this article I will share the knowledge and conclusions we learned from our Impala optimization project.

What is Impala?

If you don't know what it is — read about it in the Cloudera Impala Guide, and then come back here for the interesting stuff.

Impala Vs. Other SQL-on-Hadoop Solutions

Impala Vs. Hive

There's nothing to compare here. These days, Hive is only for ETLs and batch-processing. Your analysts will get their answer way faster using Impala, although unlike Hive, Impala is not fault-tolerance. But that's ok for an MPP (Massive Parallel Processing) engine.

Impala is faster than Hive because it's a whole different engine and Hive is over MapReduce (which is very slow due to its too many disk I/O operations).

Impala Vs. SparkSQL

Yes, SparkSQL is much faster than Hive, especially if it performs only in-memory computations, but Impala is still faster than SparkSQL.

It's faster because Impala is an engine designed especially for the mission of interactive SQL over HDFS, and it has architecture concepts that helps it achieve that. For example the Impala 'always-on' daemons are up and waiting for queries 24/7 — something that is not part of SparkSQL. And some more reasons like Impala's codegen mechanism, the Parquet format optimization, statistics, metadata cache, etc.

The JDBC/ODBC Thrift Server of SparkSQL may be a comparable competitor to Impala, but since I couldn't find much about it on the web—I'll just state it here and if you have some knowledge and experience on this subject please write a medium post about it. I might write one in the near future.

Impala Vs. Presto

Presto is a very similar technology with similar architecture. According to almost every benchmark on the web — Impala is faster than Presto, but Presto is much more pluggable than Impala. Impala suppose to be faster when you need SQL over Hadoop, but if you need to query multiple datasources with the same query engine — Presto is better than Impala. Just see this list of Presto Connectors. Impala can also query Amazon S3, Kudu, HBase and that's basically it.

For further reading about Presto— this is a PrestoDB full review I made.

Impala Best Practices

Use The Parquet Format

Impala performs best when it queries files stored as Parquet format. The whole technology that Cloudera Impala is based on comes from the Google Dremel Whitepaper and in that paper you can find the concept that Parquet is based on. So just remember not to query json, csv or sequence files — parquet your files before you let analysts query them.

Work With Partitions

Partition your data according to your analysts queries. Impala doesn't have any indexes so that's the only way to reduce the amount of data you process in each query. We use DT (date time) as the main partitioning method for most of our tables. Over partitioning can be dangerous (keep reading for more details).

The REFRESH Statement

The REFRESH statement can be an expensive operation, especially if you have thousands of tables with data adding to them every hour. We run more than 10,000 of refreshes per day. Those are my best practices for refreshes:

- Since Impala 2.7 you can perform a refresh on a specific partition, use that to make the REFRESH statement much lighter.
- **Hot & Archived tables architecture** — each table will have a hot version and an archived version. The hot version will hold the last 24 hours and a **refresh on that**

table will occur every hour and will be much lighter. Every day the hot table will be merged to the archived table and a heavier refresh over that table will occur. And of course a VIEW above those 2 that unions them so the users will not even be aware of this.

- Don't have too much metadata (files and partitions) per table (see the 'Optimal File Size' section).

Compute Stats

Statistics will make your queries much more efficient, especially the ones that involve more than one table (joins). Therefore you should compute stats for all of your tables and maintain a workflow that keeps them up-to-date with incremental stats. For more technical details read about Cloudera Impala Table and Column Statistics.

Optimal File Size — 256MB/File

TL;DR: *Make sure you don't have too many small files — it will hurt your catalog server, refreshes and query performance* ***really badly****.*

We noticed our Impala catalog server keeps crashing 4 times a week and that our queries took too much time. Then we realized we have way too much files and partitions. We were working with DT partitions on an hourly basis, no matter the size of the partitions.

That way after 2 years we had tables with 17GB of data and 17,000 partitions — which means each partition is about 1mb. We had tables with partitions at the size of 50KB. And if thats not enough, some of those small partitions had multiple files in them.

At the beginning we thought that this unreasonable amount of files is only causing our metadata to be too big and that's why the refreshes are so heavy and the catalog server is crashing. But then we realized that the **amount of files also affects our query performance very badly** (because the number of scanner threads required to read so many parquet files).

How badly? We performed a test on an actual 500MB table we have, with 17,280(!) partitions and files. We created a new merged version of this table with only 2 files, 1 per year. The SCAN HDFS part in the query execution summary was **2,000 times faster**. That was the part we understood just about how much our small files are hurting us.

So we started to manage our partitions and merge them into the optimal file size which is about 256mb.

Tables with an hourly partitioning became daily, monthly or yearly partitioned. Each partition has only 1 file (unless its size is > 256mb)

Configure Coordinators & Executors Per Daemon

Since Impala 2.9 you can determine which impala daemons are coordinators and which are executors. That's an enormous change because before that — all the daemons were coordinators and executors and the overhead of being a coordinator is very resource consuming for big clusters.

It means for example that every node keeps all the metadata cache in his RAM. And if you have for instance, 20GB metadata and 50 nodes — that means a waste of 1TB of RAM just because all of the nodes are also coordinators!

Test what is the best coordinators/executors rate for you and use it. Tip: start with 2 coordinators per cluster.

Column Data Types

At first we used STRING for all of our columns in all of the tables. It's a really bad practice that hurt performance very much. You should try to choose the most fit type to the column out of all the data types Impala supports.

Impala Query Limits

You should use the Impala Admission Control to set different pools to different groups of users in order to limit the use of some users to X concurrent queries or Y memory. But even more than that — we've built our own system to handle problematic users and queries.

Our system samples the currently running queries every 10 seconds and if a query is running for more than 30 minutes—it's getting killed. If the same query template was killed 90% of the times last month because it took more than 30 minutes—our system will kill it immediately when recognizing the template, in order to protect our infrastructure.

Killing queries is not a new concept — Amazon Athena has done that before us, but kill queries only by problematic pattern — that's something no one else does and that's the only way to handle thousands of hungry analysts.

Summary

If you need to let your analysts perform ad-hoc, interactive SQL queries over your Hadoop — Impala is a great solution. You should read the Cloudera Impala Guide to become a really good Impala administrator and remember to work by the best practices I explained here.

Big Data Cloudera Impala Hadoop Best Practices