BUSINESS
Using SQL to democratize streaming data

May 07, 2019

Partition Management in Hadoop

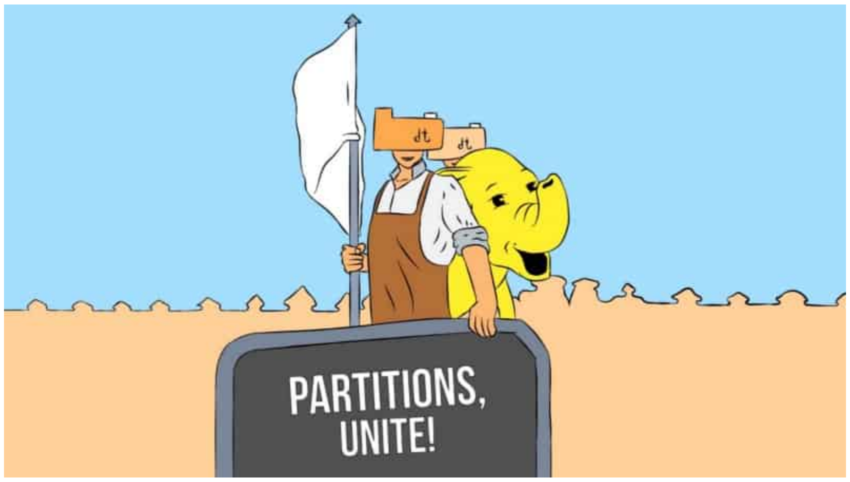
By Adir Mashiach

Apache Hive

BUSINESS
Change The Way You Do ML With Applied ML Prototypes

Guest blog post written by Adir Mashiach

In this post I'll talk about the problem of Hive tables with a lot of small partitions and files and describe my solution in details.



A little background

In my organization, we keep a lot of our data in HDFS. Most of it is the raw data but a significant amount is the final product of many data enrichment processes. In order to manage all the data pipelines conveniently, the default partitioning method of all the Hive tables is hourly DateTime partitioning (for example: dt='2019041316').

My personal opinion about the decision to save so many final-product tables in the HDFS is that **it's a bad practice**. But assuming those tables have to be stored in the HDFS—we need to face some issues regarding the subject of storage management, or what we call: "partition management".

The many-small-files problem

As I've written in a couple of my previous posts, one of the major problems of Hadoop is the "many-small-files" problem. When we have a data process that adds a new partition to a certain table every hour, and it's been running for more than 2 years, we need to start handling this table. There are 24 * 365 * 2 (17,520) hours in 2 years time, so we'll be having a table with almost 20k partitions. And I shall state that the volume we're talking about here is around 1MB per hour. Now imagine having 500 tables of that sort.

I don't know if any of you tried to scan 20,000 partitions (i.e. files) just to read 20GB of data, but the overhead is enormous. No matter the technology: Spark, Hive, MapReduce, Impala, Presto— they all suffer extremely bad performance when there are too many small partitions. Now imagine having thousands of queries every day, scanning thousands of partitions per table.

The problem of HDFS, is that it's simply a distributed filesystem—and nothing more. It's not a storage layer that lets you ingest data and handles everything in the background. It's a filesystem, plain and simple. That's why I personally suggest you to store your final-product tables in a decent store like Apache Kudu, or an RDBMS like MySQL or PostgreSQL. But if for some reason you keep your data in the HDFS, you need to write your own storage management layer.

Partition Management

Well then, what exactly this storage management layer should do — is up to your specific problems. For instance, in our case there are 3 goals:

1. Merge partitions on selected tables

I want the "Partition Manager" to merge hourly partitions to monthly ones on a regular basis. The reason I've chosen monthly resolution as a merging standard is because it generates optimal-sized partitions (100mb-1gb). I don't want one table to be partitioned monthly and the other yearly, for example, because I want to make it simple for our users (both analysts and data developers). The merging process will be described in detail later.

2. Archiving cold data

Sometimes, we want to keep certain tables' data for years, even though old data will probably be much less used. Therefore, I want my storage management layer to "archive" partitions that are older that 2 or 3 years (depends on your use-case of course). That is done by moving the data to another version of the table with a more aggressive compression algorithm like GZIP (compared to SNAPPY in the "hot" tables).

Editor's Choice

BUSINESS
Using SQL to democratize streaming data

BUSINESS
Change The Way You Do ML With Applied ML Prototypes

3. Delete partitions

And of course, we might want to choose a certain threshold (most probably a time threshold) for tables

that we want to delete their old data from the HDFS. That is a really basic (but necessary) feature we would want from our storage management layer.

All of those 3 features are important, but I think the first one is the trickiest, and the actual reason I started writing this post, is to explain how I think it should be done.

Partition Merging

Well first of all, I'll have to say that the complexity of this task really depends on your situation. In our situation, the task of merging partitions on a regular basis was not simple because of the following requirements:

1. **Production tables with zero tolerance to downtime**. Especially not if it's more than a few seconds.
2. **Not losing time resolution** — we found out some tables are partitioned by DT but there is no other matching time column. It means that if we are going to merge "2018041312" to "201804", the users lose the daily and hourly time resolution on those partitions.
3. **As seamless as possible** — the goal was to make the partition merging seamless to the users. We found out that in some cases it's simply impossible with the current partitioning method (flat string DT), but in a different partitioning method, it's quite possible. More on that later.

So now that we realize it may not be a simple problem to solve, let's see how we solved it.

How to merge partitions in Hive tables

The process is quite simple. All the queries described here are Impala queries, but the syntax is quite similar (and sometimes identical) in other technologies like Hive, SparkSQL, Presto, etc. Another thing to remember is the setting you may need to perform before the "merge queries", in order for them to generate optimal-sized files. For example in Impala you might want to execute:

```
set num_nodes=1;
```

1. **Perform a merge query**—the following example demonstrates merging all the small partitions of April 2019 to a single monthly partition:

   ```
   INSERT OVERWRITE tbl
   PARTITION(dt) AS SELECT t.col1,
   t.col2, ..., SUBSTR(t.dt, 1, 6)
   AS dt FROM tbl t WHERE t.dt LIKE
   '201904%';
   ```

2. **Drop the old partitions from the metastore** (if it's an external table, only the partition metadata will be deleted), for example:

   ```
   ALTER TABLE tbl DROP
   PARTITION(dt='2019040101');
   ```

3. **Delete the directory of each of the old partitions from the HDFS:**

```
curl -i -X DELETE
"http://:/webhdfs/v1/?
op=DELETE&recursive=true"
```

4. If you're using Impala, you should **invalidate the table's metadata cache**:

```
INVALIDATE METADATA tbl;
```

Pretty simple, isn't it? well, this way we solve the first problem—no downtime, but the trade-off is that there is a window of time in which the users will see a duplicated data (until all the DROP PARTITIONS finish). We decided it's fine.

Alright then, now instead of flat hourly DT partitions, we have flat monthly DT partitions. It's not seamless at all and we may lose time resolution in certain tables. Let's solve the first problem before we handle the resolution loss.

Merged partitions in a user view

Our users are divided to: "common" users, analysts, data scientists and data engineers. The data scientists and the data engineers are a small and technical group of people, so they can adapt the change quite easily. The problem is the analysts and the common users, and we have a lot of them.

Why is it hard to adapt to this simple change

Editor's Choice

BUSINESS
Using SQL to democratize streaming data

BUSINESS
Change The Way You Do ML With Applied ML Prototypes

anyway? It's simply 201801 instead of 2018010101. Well the reason is—if a user will query a table in the following way:

```
SELECT * FROM tbl WHERE dt >
'2018011220' AND < '2018022015';
```

It will be scanning only the partition of '201802'. Not only that, it will also get all the month instead of just the dates he wanted. In order to have the results correctly, they will have to change the DT filtering and add another time column ("insertion_time" in the following example) filtering:

```
SELECT * FROM tbl WHERE (dt BETWEEN
'201801' AND '201802') AND
(from_timestamp(insertion_time,
'yyyyMMddHH') BETWEEN '2018011220'
AND '2018022015');
```

But we don't handle merge partitions for **all** the tables, just for the problematic ones (those which consist of many small files and that are queried often). So I can't tell my users that from now on all of the tables are managed that way (last 2 months—hourly, and older than that—monthly). Because only some of the tables are handled like that, so they're required to check it before they query. **It's the opposite of seamless.**

For the common users, we solved the problem with a little effort: we changed the query templates in the BI systems they use to fit the new partitioning. The common user is querying our datalake through a certain BI tool. That way, the query itself is

Editor's Choice

BUSINESS

Using SQL to democratize streaming data

BUSINESS

Change The Way You Do ML With Applied ML Prototypes

template from:

```
... WHERE (dt BETWEEN
from_timestamp({from}, 'yyyyMMddHH')

AND from_timestamp({to},
'yyyyMMddHH')) AND (insertion_time
BETWEEN {from} and {to});
```

to:

```
... WHERE (dt BETWEEN
from_timestamp({from}, 'yyyyMM') AND
from_timestamp({to}, 'yyyyMMddHH'))
AND (insertion_time BETWEEN {from}
and {to});
```

Notice that I keep the {to} format hourly (yyyyMMddHH). Because if, for instance, a user want to query the last 6 months—I don't want him to miss the last month, like the following query:

```
SELECT * FROM tbl WHERE dt BETWEEN
'201810' AND '201904';
```

This query will miss all the partitions of April 2019, because they are still in an hourly format, instead I would like the query to look like this:

```
SELECT * FROM tbl WHERE dt BETWEEN
'201810' AND '2019041519';
```

The nice thing is, even if the users want to query only old months, say October-December 2018, it will still work and get all the relevant monthly partitions.

Editor's Choice

BUSINESS

Using SQL to democratize streaming data

BUSINESS

Change The Way You Do ML With Applied ML Prototypes

Well that was actually good enough for us, because the vast majority of our users use BI tools and don't write SQL themselves. As for the analysts who do write SQL, we decided they will have to check if a table is managed in the partition manager and to adjust their queries accordingly—of course we have to be supportive and help them adapt this methodology.

How we handle the resolution loss

That simply requires a change in the table itself: adding an "original_dt" column, and make sure the data process that populates that table is "aware" of it. Of course we need to apply the same process of changing the related query templates in the BI systems, and letting the analysts know about the change.

Nested DateTime Partitions

The easiest way to perform a seamless partition merging is when you have nested DT partitions instead of flat DT:

```
year=2019/month=04/day=16/hour=19
```

In that case, merging the small partitions while adding the omitted resolution as columns in the table, will be completely transparent to the users. For example, if I would like to merge hourly partitions to monthly ones, I'll operate according to the following steps:

1. **Create a merged version of the original**

Editor's Choice

BUSINESS

Using SQL to democratize streaming data

BUSINESS

Change The Way You Do ML With Applied ML Prototypes

table, like so:

```
CREATE EXTERNAL TABLE
tbl_merged_nested (
  col1 STRING,
  col2 STRING,
  ...,
  day STRING,
  hour STRING
) PARTITIONED BY (
  year STRING,
  month STRING
)
STORED AS PARQUET;
```

2. **Perform a merge query:**

```
INSERT OVERWRITE TABLE
tbl_merged_nested PARTITION(year,
month)
SELECT col1, col2, ..., day,
hour, year, month FROM
tbl_original_nested WHERE
year='2019' AND month='04';
```

3. **Drop the old partitions of the original table from the metastore** (and of course afterwards delete the directory from the HDFS):

```
ALTER TABLE tbl_original_nested
DROP PARTITION(year='2019',
month='04', day='17', hour='20');
```

4. Then, I can create a view that unions the 2 tables if I want to have a “hot” one and a “merged” one, and **it will be completely seamless to the user** who will not care if the year, month, day or hour columns are partitions or not:

Editor's Choice

BUSINESS
Using SQL to democratize streaming data

BUSINESS
Change The Way You Do ML With Applied ML Prototypes

```
CREATE VIEW union_view AS SELECT
* FROM tbl_original_nested UNION
ALL SELECT * FROM
tbl_merged_nested;
```

Therefore, the most important condition to really merge DT partitions seamlessly, is to have them nested and not flat.

Summary

- **Don't store small, frequently-queried tables in HDFS**, especially not if they consist of **thousands of files**. Store them in another place like an RDBMS (MySQL, PostgreSQL, etc.) or in Apache Kudu if you want to stay in the Hadoop ecosystem. Of course you'll have to provide a solution to perform join-queries between those tables and the Hive tables, I recommend Presto (if you're using Kudu, Impala can work too).
- If you have to store them in HDFS, **make sure to have a storage management layer** ("partition manager") that handles the partitions merging and prevent situations of tables with many small files.
- Partition merging can be difficult if you want it to be transparent to the users. But compared to flat DT partitions, nested ones make seamless merging much easier.

I hope the post was helpful to some of you and I invite you to comment and share you thoughts about partition management in Hadoop.

Adir is a Big data architect, specializes in the Hadoop ecosystem. Experienced at designing solutions in on-premise clusters with limited resources to efficiently

Editor's Choice

BUSINESS
Using SQL to democratize streaming data

BUSINESS
Change The Way You Do ML With Applied ML Prototypes

serve thousands of users and analysts.

Adir technical posts can also be found on https://medium.com/@adirmashiach

Adir Mashiach

Adir Mashiach

More from this author

1 Comments

by Yifei Hong on Sep 19, 2020 @ 8:14 pm PDT

We faced the same problem. The approach I devised to avoid downtime has 5 automated steps,

1) Create a _compact table to merge small files
2) Use coalesce() to merge to optimized size, while repartition() is slower due to reshuffling
3) Drop original partitions in source table
4) Exchange compacted partitions from _compact table to source table which takes almost no time to complete (https://cwiki.apache.org/confluence/display/Hive/Exchange+Partition)
5) Drop _compact table

Similar to your overall approach except for step #4, which is very neat operation to achieve seamless partition management. If the table is not partitioned, then drop the source table and rename the _compact table.

Reply

Leave a comment

Your email address will not be published. Links are not permitted in comments.

Editor's Choice

BUSINESS

Using SQL to democratize streaming data

BUSINESS

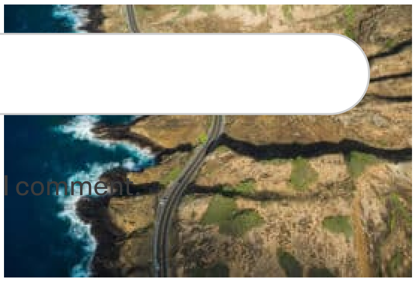
Change The Way You Do ML With Applied ML Prototypes

Your email address will not be published. Links are not permitted in comments.

NAME *

EMAIL *

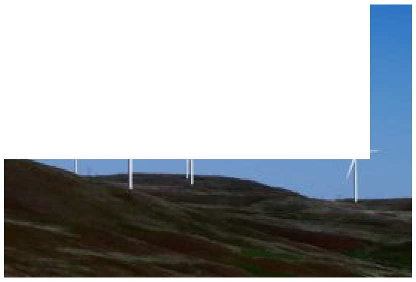
☐ Save my name, and email in this browser for the next time I comment.



BUSINESS

Using SQL to democratize

Please leave a comment here...



POST COMMENT

BUSINESS

Change The Way You Do ML With Applied ML Prototypes

About Products Solutions Services & Support

Contact Us

US: +1 888 789 1488

Outside the US: +1 650 362 0488