Partitioning Best Practices

Date published: 2019-12-5
Date modified:

https://docs.cloudera.com/

Legal Notice

Contents

Overview

This document provides best practice recommendations for handling small files and partitioning with Impala tables. The guidelines included here are for HDFS-backed tables only. When you are using a cloud service, such as Amazon S3, different guidelines apply because different conditions exist. Although the focus of this document is partitioning recommendations for Impala, these guidelines can also be applied to partitioning Hive tables as well.

File counts and file size

The number of files and their size are crucial to effective table partitioning.

By: Balazs Jeszenszky, Field Engineer, Cloudera, Inc.

Impala uses as many nodes to scan an HDFS table as the number of files being scanned. For example, to scan a 150-file HDFS table, Impala can use up to 150 nodes. Due to this, Cloudera recommends having at least as many files in a table or set of partitions, depending on what is queried more frequently, as the number of nodes in the cluster.

Limiting concurrency by having fewer files than the number of nodes affects both table scans and all other operations running on the same query fragment. For heavily used tables, this makes “hotspotting” more likely to occur. In this situation, *hotspotting* occurs when a cluster’s overall performance is limited due to a few nodes performing most of the operations.

On the other hand, having too many small files also hurts overall performance because Impala must make costly remote procedure calls (RPCs) to the HDFS Namenode to open each file for reading. The exact level of degradation depends on several factors, such as overall Namenode load, file count, network latency, and the query SQL. Small files also add excess pressure to both the Namenode and Impala’s Catalog Service (catalogd). Also see Scalability Considerations for File Handle Caching, which addresses file handle caching as a means of addressing performance issues. If you are using CDH 5.14 or earlier versions, try adjusting the file handle caching before you try adjusting your partitioning strategy.

The goal is to strike a balance between file size and count. Generally, optimal file size is 256 MBs. For tables where this file size might limit parallelism because you have fewer files than the number of nodes on the cluster, reduce the file size to take full advantage of the cluster. File sizes of 64 MBs or even 32 MBs can be useful in these cases.

When a table reaches 10-15,000 files with 256 MBs average file size, either compact further or split the table while retaining the 256 MBs file size if the table access patterns permit it. The latter strategy is more desirable. For example, if most operations against the table access only the more recent partitions, it makes sense to create a separate history table with a larger file size.

Partitioning considerations

It is useful to incorporate partition pruning into your partitioning strategy.

By: Balazs Jeszenszky, Field Engineer, Cloudera, Inc.

Organizing data sets with partitioning enables you to perform partition pruning. Partition pruning enables queries to scan only a subset of the table data. Generally, this is beneficial, but creating partitions that are too small can create a small files problem, which can negatively impact performance. As a general rule, partitions that are under 1 GB or that contain only 4 files are usually not worth creating.

For what size of table should you consider partitioning? As an example, if you have an unpartitioned table with 15 GBs of data that resides in 60 files, which are 256 MBs each and spread evenly across a 60-node HDFS cluster, a full table scan reads 256 MBs per node. Partitioning this table into 15 equal partitions and filtering to a single partition leads to less data nodes participating in the scan, but each of them still must read 256 MBs of data. This means less workload is spread across the cluster, but involves roughly the same run time. This example represents an optimal

case, indicating that it might be beneficial to partition even smaller tables. However, unless you have a specific request for partitioning, one file per node is a good threshold for when to consider partitioning a table.

Partition pruning only works if the queries that are run against a table have a predicate on the partition column. Because of this, partitioning even a large table is only helpful if the queries against that table are understood. For example, if a partitioning strategy is based on the date column, but most of the queries that are run against the table do not include predicates for date, a full table scan will still be necessary and the partitioning will not provide any benefit. Also note that dynamic partition pruning can be helpful. Dynamic partition pruning enables partition pruning at runtime if the partitions are part of a join key. For more information about dynamic partition pruning, see Runtime Filtering for Impala Queries (CDH 5.7 or higher only).

To refresh data in partitioned tables, use the per-partition REFRESH statement to pick up new data in an already existing partition. Use ALTER TABLE [...] RECOVER PARTITIONS to pick up new partitions and their data. These are the most lightweight statements for picking up new data.

It is a good idea to review your partitioning and table strategy when a table reaches 50,000 partitions. Make sure you implement your new strategies before a table reaches 100,000 partitions. These are not hard-and-fast rules on what the system can handle, rather a best practice that helps you avoid major issues.

Small table considerations

Small tables present a challenge when partitioning.

By: Balazs Jeszenszky, Field Engineer, Cloudera, Inc.

Partitioning is often suggested as a way to speed up any query. As seen in earlier sections, this may not be true for small tables or ineffective partitioning. Instead, consider using HDFS caching for heavily queried small tables. Storing the data in the DataNode's cache greatly speeds up scans at the cost of memory allocation on the DataNodes:

- HDFS caching reference
- HDFS caching usage notes

Compacting partitions

How to change your partition strategy with the least amount of disruption.

By: Balazs Jeszenszky, Field Engineer, Cloudera, Inc.

When reducing the number of partitions it is best to retain as much benefit from the original partitioning strategy as possible. This might not be a trivial task if you cannot influence how end-user queries are written, which might use the original partitioning scheme. For example, a table that used to have daily partitions with a key of the string data type, for example, ‘1970-01-01,’ might have to be compressed into yearly partitions, which are stored as integer data types, such as 1970, to arrive at an acceptable file and partition size:

```
CREATE TABLE _OLD (c1 int) PARTITIONED BY (_day string);
CREATE TABLE _NEW (c1 int, _day string) PARTITIONED BY (year int);
```

The non-partition _day column is added to avoid losing data. To retain partition pruning for queries that filter on _day, you can define a view:

```
CREATE VIEW _NEW_VIEW AS SELECT * FROM _NEW WHERE year = year(_day);
```

When queries run against this view their _day predicate is translated to a partition filter. The year built-in UDF is useful in this case, but others, such as substr also work.

Summary guidelines

Summary of best practice guidelines for partitioning.

By: Balazs Jeszenszky, Field Engineer, Cloudera, Inc.

- When Impala runs against HDFS-backed tables, scan parallelism is driven by the number of files.
- Small files seriously impact performance and put excess pressure on clusters, especially Namenode hosts and Impala Catalog Service hosts. Too few files can lead to hotspotting when too many tasks touch the same data, potentially overloading the node.
- Generally, keep files at around 256 MBs and avoid partitions that are less than 256 MBs in size.
- For small tables that are heavily queried, consider HDFS caching.

Partitioning is best approached as an iterative process, and not a scientific one. Following these simple general guidelines can yield a good experience. It is possible for you to make further improvements based on your unique usage patterns and experience.