Scalability Considerations for Impala

More is not always better

- The size of the cluster and the volume of data influence the performance of your SQL queries.
- Typically, adding more resources reduces problems due to memory or disk.
    - The more machines, the more data you need to divide the work evenly.
- Furthermore, larger clusters tend to have other issues.
- `catalogd` daemon might encounter an out-of-memory error

Too many small tables

- HDFS is optimized for reading and writing large files, and so Impala is optimized for tables containing relatively few, large data files.
- Schemas containing thousands of tables, or tables containing thousands of partitions, can encounter performance issues during startup or during DDL operations such as `ALTER TABLE` statements.
- Easy to miss: If you do hourly partitions, after 2 years of data you have *24*365*2 = 17520 partitions.

Too many nodes

- Impala daemons need to exchange info with other daemons across the cluster, for instance, during a partitioned join.
- The number of connection per hosts is then proportional to the number of nodes.
- For instance, in a 100 nodes cluster:
  - If you have 32 concurrent queries each of 50 partition fragments, you get $100 \times 32 \times 50$ connections per host, or 16 million connections across the cluster.
- This is an issue in Impala versions previous to 6.1

Everyone is a coordinator

- It is possible to configure in Cloudera Manager which nodes are coordinators and which are executors (since Impala 2.9).
- Having daemon being a coordinator means that you need a copy of the metadata everywhere.
    - 20 GB metadata and 50 nodes = 1 TB of wasted RAM.
- Test with two coordinators per cluster and add more if needed.

Spill to disk

- Some memory-intensive queries will write to disk whenever a host is about to run out of memory.
- The result is that the query finishes, but there is a loss in performance due to the extra I/O operation.

Spill to disk (cont.)

- Queries that might spill to disk:
  - `GROUP BY` queries with millions of different values. Impala needs to keep millions of different temporary results in memory to accummulate the aggregates per group.
  - Broadcast joins where the broadcasted table is large.
  - `ORDER BY`, `DISTINCT`, `UNION`
- You can detect queries that spill to disk using `PROFILE`, under `WriteIoBytes` (also called `ScratchBytesWritten` or `BytesWritten` in older versions).

Preventing spill to disk

- Increase memory if practical, for example, by increasing more than the amount you read from the previous profile.
- `SET MEM_LIMIT` allows you to fine-tune memory requirements from JDBC/ODBC applications.
- Use Resource Manager to allocate more memory to Impala.
- Follow best practices for your query.
- Nuclear option: `DISABLE_UNSAFE_SPILLS` will prevent you from running queries that exceed the Impala memory limit.

Limits on query size and complexity

- Hardcoded maximum number of expressions in a query (2000).
- BI tools or other query-generating tools can make you exceed this limit.
- If you can customized the queries, it is better to replace statements of the form
  `WHERE val=1 OR val=2 OR val=3...`
  by
  `WHERE val IN (1,2,3)`.

Runtime filtering

Some preliminaries

- **Plan fragment:** Small units of work distributed across the cluster.
- **Build phase of a query:** When rows containing the join key columns travel across the network.
- **Probe phase of a query:** Data is read locally, and the join key columns are compared to the values in the hash table.
- The **runtime filter** is then applied when Impala takes data from the origin node to the destination (where the scan is happening), and filters out useless information before arrival.
- They are sometimes implemented as *probabilistic* algorithms.

Wait intervals

- It takes time to produce runtime filters, especially if they come from partitioned nodes and have to be combined into the coordinator.
- Sometimes, it is more efficient to just drag all the data into the scan node (where other data is sitting, waiting for the join).
- `RUNTIME_FILTER_WAIT_TIME_MS` paramenter can control this.

Wait intervals (cont.)

- Whether to use runtime filters or not, depends on the workload of the cluster
  - Busy cluster + resource-intensive workload: increase waiting time, to give opportunity to optimize the query.
  - Light load + many small queries: decrease waiting time.
- Runtime filters are visible on the `EXPLAIN` plan (`RFXX`), setting the `EXPLAIN_LEVEL` to 2 gives additional info.
- `PROFILE` output also shows info about the filter routing table.

When is runtime filtering useful?

```
SELECT col
FROM table_part_by_year
WHERE year IN
(SELECT DISTINCT year from years_lookup_tbl);
```

Some caveats

- Runtime filters only apply to Parquet tables!
- Stats are required
- If joining tables have unique columns as join keys (for example, a primary and a foreign key), then the runtime filter will not remove too much data. Hence there would be no benefit on trying, just overhead.
    - Might be better to `SET RUNTIME_FILTER_MODE=OFF;`

Partitioned Tables

When to use partitioned tables

- Very large tables that would be costly to scan.
- Tables that are often queried with conditions on partitioning columns (e.g. querying by year).
- Tables with a moderate amount of distinct values:
  - Male/Female partition may not help much (50%)
  - Partition by hour might hurt you

SQL statements for partitioned columns

- You can create a partition with `CREATE TABLE... PARTITIONED BY` and modify them with `ALTER TABLE`
- `INSERT` statements require to specify the partition
- `SELECT` can have huge impact on performance when applied to partitioned tables. This is called *partition pruning*
- `SHOW PARTITIONS`

Static Partition Pruning

- When one specifies all the columns for the partition is called **static** partitioning. The output of the query will affect a single, predictable partition.

```
INSERT INTO t1
PARTITION(x=10, y='a')
SELECT c1 FROM some_table;
```

- Similarly, a `WHERE` clause on a partitioned column determines partitions that can be skipped.

Dynamic Partition Pruning

- Sometimes, the information on which partitions to skip is only available at runtime. In this case, dynamic partition pruning happens.

```
SELECT AVG(sales)
FROM sales_table
WHERE year in
(
 SELECT year
 FROM years_lookup_table
);
```

Dynamic Partition Pruning (cont.)

- Dynamic partition pruning is especially effective for queries involving joins of several large partitioned tables.
- Evaluating the `ON` clauses of the join predicates might normally require reading data from all partitions of certain tables.
- If the `WHERE` clauses of the query refer to the partition key columns, Impala skips reading partitions while evaluating the `ON` clauses.
- This reduces the amount of I/O and the amount of intermediate data stored and transmitted across the network during the query.