The Impala Cookbook

From Cloudera's Impala Team

Updated April 22, 2015

@RideImpala

cloudera

Topic Outline

- Part 1 – The Basics
  - Physical and Schema Design
  - Memory Usage in Impala
- Part 2 – The Practical Issues
  - Cluster Sizing and Hardware Recommendations
  - Benchmarking Impala
  - Multi-tenancy Best Practices
  - Query Tuning Basics
- Part 3 – Outside Impala
  - Interaction with Apache Hive, Apache Sentry, and Apache Parquet

Topic Outline

- Part 1 – The Basics
  - Physical and Schema Design
  - Memory Usage in Impala
- Part 2 – The Practical Issues
  - Cluster Sizing and Hardware Recommendations
  - Benchmarking Impala
  - Multi-tenancy Best Practices
  - Query Tuning Basics
- Part 3 – Outside Impala
  - Interaction with Apache Hive, Apache Sentry, and Apache Parquet

cloudera

Physical and Schema Design - Outline

- Schema design best practices
  - Datatypes
  - Partition design
  - Common questions
- Physical design
  - File format – when to use what?
  - Block size (option)

Physical and Schema Design - Datatypes

- Use Numeric Types (not strings)
  - Avoid string types when possible
  - Strings => higher memory consumption, more storage, slower to compute
- Decimal vs Float/Double
  - Decimal is easier to reason about
  - Currently can't use Decimal as partition key or in UDFs
- Use String only for:
  - HBase row key – string is the recommended type!
  - Timestamp – ok to use string, but consider using numeric as well!
- Prefer String over Char/Varchar (except for SAS)

cloudera

Partition Design: 3 Simple Rules

1. Identify access patterns from the use case or existing SQL.
2. Estimate the total number of partitions (better be <100k!).
3. (Optional) If needed, reduce the number of partitions.

Partition Design – Identify Access Patterns

- The table columns that are commonly used in the WHERE clause are candidates for partition keys.
  - Date is almost always a common access pattern and the most common partition key.
- Can have MULTIPLE PARTITION KEYS! Examples:
  - `Select col_1 from store_sales where sold_date between '2014-01-31' and '2016-02-23';`
  - `Select count(revenue)from store_sales where store_group_id in (1,5,10);`
  - `Partition keys => sold_date, store_group_id`

cloudera

Partition Design – Estimate the #partitions

- Estimate the number of distinct values (NDV) for each partition key (for the required storage duration). Example:
  - If partition by date and need to store for 1 year, then NDV for date partition key is 365.
  - num store_group will grow in time, but it will never exceed 52 (one for each state).
- Total number of partitions = NDV for part key 1 * NDV for part key 2 * … * NDV for part key N. Example:
  - Total number of part = 365 (for date part) * 52 (for store_group) ~= 19k
- Make sure #partition <=100k!

cloudera

Partition Design – Too Many Partitions?

- Remove some “unimportant” partition keys.
  - If a partition key isn’t as routinely used, or it doesn’t impact the SLA, remove it!
- Create partition “buckets.”
  - Use month rather than date.
  - Create artificial `store_group` to group individual stores.
  - Technique: use prefix or hash
    - Hash(store_id) % store_group size => hash it to store_group
    - Substring(store_id,0,2) => use the first 2 digits as artificial store_group

Schema Design – Common Issues

- Number of columns - 2k max
  - Not a hard limit; Impala and Parquet can handle even more, but…
  - It slows down Hive Metastore metadata update and retrieval
- Timestamp
  - No timestamp support in Hive's Parquet yet
    - Use either BIGINT (more efficient) or String (easier to read)
    - BLOB/CLOB – use string
- String size - no definitive upper bound but 1MB seems ok
  - Larger-sized string can crash Impala!
  - Use it sparingly - the whole 1MB string will be shipped everywhere

cloudera

Physical Design – File Format

- Parquet/Snappy
  - The long-term storage format
  - Always good for read!
  - Write is very slow (reportedly 10x slower than Avro).
- Snappy vs Gzip
  - Snappy is usually a better tradeoff between compression ration and CPU.
  - But, run your own benchmark to confirm!
- For write-once-read-once tmp ETL table, consider seq/snappy because:
  - The write is faster.
  - Impala can write.

Physical Design – Block Size

- Number of blocks defines the degree of parallelism:
  - True for both MapReduce and Impala
  - Each block is processed by a single CPU core
  - To leverage all CPU cores across the cluster**, #blocks >= #core**
- Larger block size:
  - Better IO throughput, but fewer blocks, could reduce parallelism
- Smaller block size:
  - More parallelism, but could reduce IO throughput

Physical Design – Block Size

- For Apache Parquet, ~256MB is good and no need to go above 1GB.
- Don’t go below 64MB!
- (Advanced) If you really want to confirm the block size, use the following equation:
  - Block Size <= p* t * c / s
    - p – disk scan rate at 100MB/sec
    - t – desired response time of the query in sec
    - c – concurrency
    - s - % of column selected

cloudera

Topic Outline

- Part 1 – The Basics
  - Physical and Schema Design
  - Memory Usage
- Part 2 – The Practical Issues
  - Cluster Sizing and Hardware Recommendations
  - Benchmarking Impala
  - Multi-tenancy Best Practices
  - Query Tuning Basics
- Part 3 – Outside Impala
  - Interaction with Apache Hive, Apache Sentry, and Apache Parquet

cloudera

Memory Usage – The Basics

- Memory is used by:
  - Hash join – RHS tables after decompression, filtering and projection
  - Group by – proportion to the #groups
  - Parquet writer buffer – 256MB per partition
  - IO buffer (shared across queries)
  - Metadata cache (no more than 1GB typically, except when incremental stats is used)
- Memory held and reused by later query
  - Impala releases memory from time to time in 1.4 and later

cloudera

Memory Usage – Estimating Memory Usage

- Use Explain Plan
  - Requires statistics! Mem estimate without stats is meaningless.
  - Reports per-host memory requirement for this cluster size.
    - Re-run if you've re-sized the cluster!

```
+----------------------------------------------------------+
| Explain String                                           |
+----------------------------------------------------------+
| Estimated Per-Host Requirements: Memory=26.51MB VCores=2 |
|                                                          |
| 07:AGGREGATE [MERGE FINALIZE]                            |
| |  output: sum(count(*))                                 |
| |                                                        |
```

cloudera

Memory Usage – Estimating Memory Usage (Cont'd)

- EXPLAIN's memory estimate issues
  - Can be way off – much higher or much lower.
  - group by's estimate can be particularly off – when there's a large number of group by columns.
    - Mem estimate = NDV of group by column 1 * NDV of group by column 2 * … NDV of group by column n
  - Ignore EXPLAIN's estimate if it's too high!
- Do your own estimate for group by
  - GROUP BY mem usage = (total number of groups * size of each row) + (total number of groups * size of each row) / num node

cloudera

Memory Usage – Finding Actual Memory Usage

- Search for “Per Node Peak Memory Usage” in the profile.
- This is accurate. Use it for production capacity planning.

```
Execution Profile b8414c34981f3ec9:a52048d52a00fb1:(Total: 9s754ms, non-child: 0ns, % non-child: 0.00%)
  Per Node Peak Memory Usage: alan-OptiPlex-790:22000(11.77 MB)
   - FinalizationTimer: 0ns
```

cloudera

Memory Usage – Finding Actual Memory Usage (Cont'd)

- For complex queries, how do I know which part of my query is using too much memory?
  - Use the ExecSummary from the query profile!

```
    ExecSummary:
Operator          #Hosts   Avg Time   Max Time    #Rows  Est. #Rows   Peak Mem  Est. Peak Mem  Detail
-------------------------------------------------------------------------------------------------------
07:AGGREGATE           1  77.817ms   77.817ms        1           1   48.00 KB       -1.00 B  MERGE FINALIZE
06:EXCHANGE            1  14.534us   14.534us        1           1          0       -1.00 B  UNPARTITIONED
03:AGGREGATE           1   1s025ms    1s025ms        1           1   84.56 KB      10.00 MB
02:HASH JOIN           1   8s300ms    8s300ms  366.55M      280.88M    7.48 MB     525.91 KB  INNER JOIN, PARTITIONED
|--05:EXCHANGE         1  22.500ms   22.500ms  183.59K     183.59K          0             0  HASH(s2.ss_sold_date_sk)
|  01:SCAN HDFS        1   1s791ms    1s791ms  183.59K     183.59K  288.00 KB      16.00 MB  tpcds_parquet.store_sales s2
04:EXCHANGE            1  19.189ms   19.189ms  183.59K     183.59K          0             0  HASH(s1.ss_sold_date_sk)
00:SCAN HDFS           1   1s782ms    1s782ms  183.59K     183.59K  408.00 KB      16.00 MB  tpcds_parquet.store_sales s1
```

cloudera

Memory Usage – Hitting Mem-limit

- Top causes (in order) of hitting mem-limit even when running a single query:

1. Lack of statistics
2. Lots of joins within a single query
3. Big-table joining big-table
4. Gigantic group by

Memory Usage – Hitting Mem-limit (Cont'd)

- Lack of stats
  - Wrong join order, wrong join strategy, wrong insert strategy
  - Explain Plan tells you that!

```
+----------------------------------------------------------------------------+
| Explain String                                                             |
+----------------------------------------------------------------------------+
| Estimated Per-Host Requirements: Memory=10.00MB VCores=2                   |
| WARNING: The following tables are missing relevant table and/or column statistics. |
| dmart.rate                                                                 |
```

  - Fix: Compute Stats <table>

Memory Usage – Hitting Mem-limit (Cont'd)

- Lots of joins within a single query
  - `select…from fact, dim1, dim2,dim3,…dimN where …`
  - Each dim tbl can fit in memory, but not all of them together
  - As of Impala 2.2, Impala might choose the wrong plan – BROADCAST
  - As of Impala 2.2, Impala sometimes require 256MB as the minimal requirement per join!
  - FIX 1: use shuffle hint
    - `Select … from fact join [shuffle] dim1 on … join dim2 [shuffle]…`
  - FIX 2: pre-join the dim tables (if possible)
    - few join=>better perf!

cloudera

Memory Usage - Hitting Mem-limit (Cont'd)

- Big-table joining big-table
  - Big-table (after decompression, filtering, and projection) is a table that is bigger than total cluster memory size.
  - Impala 2.0 will do this (via disk-based join). Consider using Hive for now.
  - (Advanced) For a simple query, you can try this advanced workaround – per-partition join
    - Requires the partition key be part of the join key

```
Select … from BigTbl_A a join BigTbl_B b where a.part_key =
b.part_key and a.part_key in (1,2,3)
union all
Select … from BigTbl_A a join BigTbl_B b where a.part_key =
b.part_key and a.part_key in (4,5,6)
```

Memory Usage – Disk-based Join/Agg

- Disk-based join/agg should be your last resort to deal with hitting mem-limit.
- Rely on disk-base join/agg if there is only one join/agg operator in the query. For example:
  - Good: select a.*, b.* from a, b where a.id=b.id
  - Good: select a.id, a.timestamp, count(*) from a group by a.id, a.timestamp
  - OK: select large_tbl.id, count(*) from large_tbl join tiny_tbl on (id) group by id
  - Bad: select t1.id, count(*) from large_tbl_1 t1, large_tbl_2 t2 where t1.id=t2.id group by t1.id
  - Bad: select a.*, b.*, c.* from a, b, c where a.id=b.id and b.col1=c.col2;
- Always set the per-query mem-limit (2GB min) when using disk-based join/agg!

cloudera

Memory Usage - Additional Notes

- Use explain plan for estimate; use profile for accurate measure
- Data skew can use uneven memory usage
- Review previous common issues on out-of-memory
- Even with disk-based joins in Impala 2.0 and later, you'll want to review these steps to speed up queries and use memory more efficiently.

Topic Outline

- Part 1 – The Basics
  - Physical and Schema Design
  - Memory Usage
- Part 2 – The Practical Issues
  - Cluster Sizing and Hardware Recommendations
  - Benchmarking Impala
  - Multi-tenancy Best Practices
  - Query Tuning Basics
- Part 3 – Outside Impala
  - Interaction with Apache Hive, Apache Sentry, and Apache Parquet

cloudera

Hardware Recommendations

- 128GB (assigned to Impala) or more for best price/performance
- Spindles vs SSD
  - Spindles are more cost effective
  - Most workload is CPU bounded; SSD won't make a difference at all
- 10GB network

Cluster Sizing – Objective and Keys

- Objective:
  - The recommended cluster size should run the desired workload within a given SLA throughout the projected life span of the cluster.
- Keys:
  - Workload - defines the functional requirement
  - SLA – defines the performance requirement
  - Projected life span – how things will change over time?

cloudera

Cluster Sizing - SLA

- Query Throughput
  - How many queries should this cluster process per second?
  - This is the more meaningful measurement of “computing power.”
- Query Response Time
  - How fast do you need the query to run?
  - Typically, single query response time isn’t too meaningful because there are always multiple queries running concurrently!
  - Some use cases require very fast response time, such as powering web UI.
- Will more people be running queries over time? This means higher query throughput!

cloudera

Cluster Sizing - Workload

- From the workload, you'll want to know:
  - How much memory do you need?
  - How much processing power do you need? (i.e. how complex is the workload?)
  - How much IO bandwidth do you need?
- The bigger the cluster, the more total memory, CPU, and disk-IO bandwidth you have.
- But usually, the network bandwidth is fixed.

Cluster Sizing – Memory Requirements

- How much memory do you need?
  - Any huge group by?
    - Per Node Mem >= number of distinct group * row size +(number of distinct group * row size) / num node
    - Number of distinct group: hard to guess; just get a rough ballpark.
    - Row size: # columns involved in the query *column width
    - Column width for int 4 byte, bigint 8 byte, etc. For string columns, take some rough guess.
    - Increasing the cluster size won't help much to reduce the per-node mem requirement.

cloudera

Cluster Sizing – Workload Complexity (Cont'd)

- (Advanced) If you're ready to dive deep into workload analysis…
  - Typically, you can assume the following processing rate:
    - scan node ~40m rows per sec
    - join node ~10m rows per sec per core
    - agg node ~5m rows per sec per core
- From the sample query, you know the #join/agg. Know #input rows and estimate the effect of partition pruning.
- Using the above processing rate, you can derive a tighter bound.

Cluster Sizing – Summary

- Cluster sizing depends on SLA and workload. You need to know both!
- Memory requirement for doing big join/agg in memory:
  - Total Cluster mem >= the 2nd largest big table in the join after decompression, filtering, and projection
  - Per Node Mem >= number of distinct group * row size +(number of distinct group * row size) / num node

cloudera

Topic Outline

- Part 1 – The Basics
  - Physical and Schema Design
  - Memory Usage
- Part 2 – The Practical Issues
  - Cluster Sizing and Hardware Recommendations
  - Benchmarking Impala
  - Multi-tenancy Best Practices
  - Query Tuning Basics
- Part 3 – Outside Impala
  - Interaction with Apache Hive, Apache Sentry, and Apache Parquet

cloudera

Benchmarking – Why Run One?

- Understand how Impala performs, how it scales, how it compares to the current system
  - Measures query response time as well as query throughput
- Understand how Impala utilizes resources
  - Measure CPU, disk, network and memory

Benchmarking Impala – Preparing the Workload

- Should be relevant to (or satisfy) the business requirement.
  - Don't run `select * from big_tbl limit 10` – it's meaningless.
- Should not be dictated on the query form.
  - You should be prepared to change the query/schema to deliver a meaning benchmark.
  - Tune the schema/query!
- Stay close to the query that you're going to run in production.
  - If the result has to be written to disk, then write to disk and DO NOT send result back to the client.
  - Don't stream all the data to the client (i.e. data extraction).
- Use a fast client: You're benchmarking the server, not the client, so don't make a slow client the bottleneck.

Benchmarking – Avoiding Traps

- It's easier to start with a smaller data set and simpler query. Trying to run complex query on a huge data set on a small cluster is not effective.
- A data set that's too small can't utilize the whole cluster. Have at least one block per disk.
- Disable Admission Control when you're doing benchmark!

Benchmarking – Preparing the Hardware

- Should be as similar to go-live hardware as possible
- Recommended: at least 10 nodes with 128GB each
- CAUTION: If the cluster is too small (i.e. 3 nodes), it's very hard to see the effect of scalability and identify potential bottlenecks

Benchmarking – Measuring Single-Query Response Time

- Use Impala-shell (simple, easy to use) with the -B option. This disables pretty formatting so client won't be the bottleneck.
  - `Impala-shell –B –q "<your query>"`
- To simulate the effect of buffer cache, run it a few times to warm the buffer cache before measuring the result.
- To simulate the effect without buffer cache, clear the buffer cache by running: `echo 1 > /proc/sys/vm/drop_caches`

cloudera

Benchmarking – Measuring Query Throughput

- Benchmark running using JDBC (or use Jmeter!)
- Input parameter: list of hosts to connect to, workload queries, duration of the benchmark, and number of concurrent connections.
- Each connection will pick a host to connect to and keep running a query for the specified duration.
- Report QPS.
- Just keep increasing the number of connections until QPS doesn't increase – that will be the QPS of the system.

Benchmarking – General Performance Notes

- Performance vs Hive - Impala will ALWAYS be faster. If not, something is wrong with the benchmark.
- Impala vs Hive-on-Tez/Spark SQL benchmark: http://blog.cloudera.com/blog/2014/09/new-benchmarks-for-sql-on-hadoop-impala-1-4-widens-the-performance-gap/
- See the open TPC-DS toolkit to run your own! https://github.com/cloudera/impala-tpcds-kit

Topic Outline

- Part 1 – The Basics
  - Physical and Schema Design
  - Memory Usage
- Part 2 – The Practical Issues
  - Cluster Sizing and Hardware Recommendations
  - Benchmarking Impala
  - Multi-tenancy Best Practices
  - Query Tuning Basics
- Part 3 – Outside Impala
  - Interaction with Apache Hive, Apache Sentry, and Apache Parquet

cloudera

Multi-tenancy Best Practices – Admission Control vs LLAMA

- With current versions, Admission Control is typically your best option
- With low concurrency (i.e. not oversubscribing the cluster), LLAMA seems to work better, but if you're not oversubscribing the cluster, you probably don't need it.

cloudera

Multi-tenancy Best Practices – Preventing a “Runaway” Query

- “Runaway" query = user submitted a "wrong" query accidentally that consumes a significant amount of memory
  - Limit the amount of memory used by an individual query using `per query mem-limit`:
    - Set it from Impala shell (on a per query basis):
      `set mem_limit=<per query limit>`
    - Set a default per-query mem limit:
      `-default_query_options='mem_limit=<per query limit>'`
  - Works with or without LLAMA

cloudera

Multi-tenancy Best Practices – Admission Control

- How to approach Admission Control config:
  - Workload dependent – memory bounded, or not?
    - “Memory bounded” means you’ve used up the memory allocated to Impala before hitting limit on cpu, disk, network.
  - Memory bounded – use mem-limit
  - Non-mem bounded – use num of concurrent queries

cloudera

Multi-tenancy Best Practices – Admission Control (Cont'd)

- Memory bounded goals:
  - Prevents each group of users from overcommitting system memory
  - Prevents query from hitting mem-limit
  - (Secondary) Simulates priority by allocating more memory to important group

cloudera

Multi-tenancy Best Practices – Admission Control (Cont'd)

- Step 1: Identify sample workload from each user "group", such as HR, Analyst, C-level exec.
- Step 2*: For each query in the workload, identify the memory requirement from Explain Plan AND the accurate memory by running the query. Take the max between Explain Plan and the actual. This is the memory requirement for the query.
- Step 3: Minimal memory requirement for each group = max (mem requirement from the query set).
- Step 4: You can divide the memory based on % too, but each group should have at least the min mem derived from Step 3.
- NOTE: sum(mem assigned to all groups) can be greater than total mem available. This is OK.

cloudera

Multi-tenancy Best Practices – Admission Control (Cont'd)

- More on Step 2
  - If the memory estimate from the explain is inaccurate:
    - FIX: Use per-query limit to override it, but that will require you to submit query through the shell.
    - FIX: Adjust the pool mem-limit accordingly; if it's over the estimate, give it a higher mem-limit and vice versa.

Multi-tenancy Best Practices – Admission Control (Cont'd)

- Limiting the number of concurrent queries
  - Goal:
    - Avoid over subscription to CPU, disk, network because this can lead to longer response time (without improving throughput).
  - Probably not too useful in general
  - Works best with homogeneous workload
  - With heterogeneous workload, you can still apply the same approach, but the result won't be as optimal.

Topic Outline

- Part 1 – The Basics
  - Physical and Schema Design
  - Memory Usage
- Part 2 – The Practical Issues
  - Cluster Sizing and Hardware Recommendations
  - Benchmarking Impala
  - Multi-tenancy Best Practices
  - Query Tuning Basics
- Part 3 – Outside Impala
  - Interaction with Apache Hive, Apache Sentry, and Apache Parquet

cloudera

Query Tuning Basics - Overview

- Given that your query runs, know how to make it faster and consume fewer resources.
- Always `Compute Stats`.
- Examine the logic of the query.
- Validate it with `Explain Plan`.
- Use `Query Profile` to identify bottlenecks and skew

cloudera

Query Tuning Basics – More on `Compute Stats`

- `Compute Stats` is very CPU-intensive, but on Impala 1.4 and later is much faster than previous versions.
- Speed Reference: ~50M cells per sec per node + HMS update time (1 sec per 100 partitions)
- Total number of cells of a table = num rows * num cols
- Only need to recomputed stats with significant changes of data (30% or more)
- `Compute Stats` on tables, not view

Query Tuning Basics – Incremental Stats Maintenance

- `Compute Stats` is slow and through 2.0, Impala does not support Incremental Stats
- Column Stats (number of distinct value, min, max) can be updated by `Compute Stats` infrequently (when 30% or more data has changed)
- When adding a new partition, run a count(*) query on the partition and update the partition row count stats manually via `ALTER TABLE`.

cloudera

Query Tuning Basics – Incremental Stats Maintenance

- Impala 2.1 or later supports “Compute Incremental Stats”, but use it only if the following conditions are met:
  - For all the tables that are using incremental stats, Σ(num columns * num partitions) < 650000.
  - The size of the cluster is less than 50 nodes.

Query Tuning Basics – Examing Query Logic

- Sometimes, the query can have redundant joins, distinct, group by, order by (very common during migration). Remove them!
- For common join patterns, consider pre-joining the tables. Example:

```
select fact.col, max(dim2.col) from fact, dim1, dim2
where fact.key = dim1.key and fact.key2 = dim2.key
```

  - The join on dim1 should be a semi-join!

Query Tuning Basics – Validating `Explain Plan`

- Key points:
  - Validate join order and join strategy.
  - Validate partition pruning or HBase row key lookup.
- Even with stats, at times the join order/strategy might go wrong (particularly with view).

Query Tuning Basics – Validating Join Order & Strategy

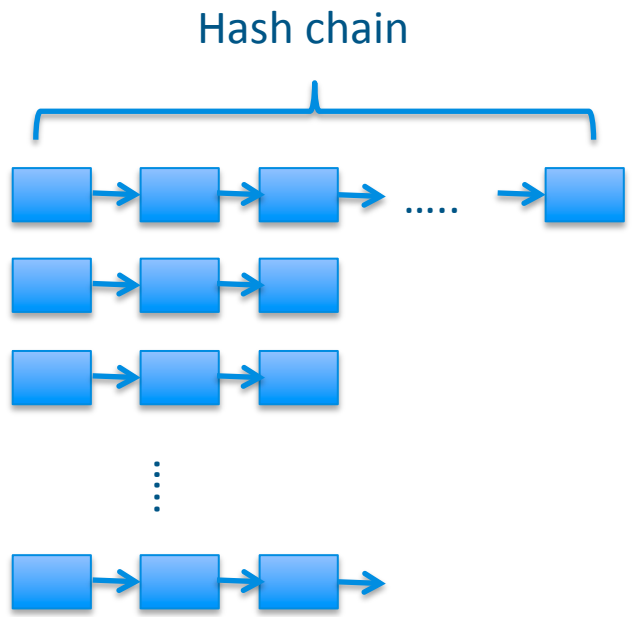
- Join Order
  - RHS is smaller than LHS
- Join Strategy - BROADCAST
  - RHS must fit in memory!

```
06:TOP-N [LIMIT=100]
11:AGGREGATE [MERGE FINALIZE]
10:EXCHANGE [PARTITION=HASH(b.int_col)]
05:AGGREGATE
04:HASH JOIN [INNER JOIN, PARTITIONED]
|--09:EXCHANGE [PARTITION=HASH(c.id)]
|  02:SCAN HDFS [functional.alltypes c]
03:HASH JOIN [INNER JOIN, BROADCAST]
|--08:EXCHANGE [PARTITION=HASH(a.id)]
|  00:SCAN HDFS [functional.smallTbl a]
07:EXCHANGE [PARTITION=HASH(b.id)]
01:SCAN HDFS [functional.BigTbl b]
```

cloudera

Query Tuning Basics – Common Join Performance Issues

- Hash Table has long hash chain
  - Ideally, the join key should be evenly distributed; only a few rows share the same join key from the RHS.
  - Is it a true foreign key join or more like a range join?
- Wrong join order – RHS is bigger than LHS table from the plan
- Too many LHS rows



cloudera

Query Tuning Basics - Check for HBase Row Key Filter Propagation!

- Row key must be string type:

```
Select … from … where row_key = 5
```

```
00:SCAN HBASE [functional_hbase.stringids]
   start key: 5
   stop key: 5\0
```


Query Tuning Basics – Finding Bottlenecks

- Use `ExecSummary` from Query Profile to identify bottlenecks

```
    ExecSummary:
Operator              #Hosts   Avg Time   Max Time  #Rows  Est. #Rows   Peak Mem  Est. Peak Mem  Detail
------------------------------------------------------------------------------------------------------------------
09:MERGING-EXCHANGE        1    4.394ms    4.394ms  7.30K       8.16K          0       -1.00 B  UNPARTITIONED
04:SORT                    1   38.492ms   38.492ms  7.30K       8.16K   32.02 MB       8.00 MB
08:AGGREGATE               1    8.397ms    8.397ms  7.30K       8.16K  458.25 KB      10.00 MB  MERGE FINALIZE
07:EXCHANGE                1  779.810us  779.810us  7.30K       8.16K          0             0  HASH(a.id)
03:AGGREGATE               1  161.736ms  161.736ms  7.30K       8.16K  466.25 KB      10.00 MB
02:HASH JOIN               1  289.552ms  289.552ms  5.33M       5.33M  318.25 KB      20.91 KB  INNER JOIN, PARTITIONED
|--06:EXCHANGE             1    1.93ms     1.93ms  7.30K       7.30K          0             0  HASH(b.float_col)
|  01:SCAN HDFS            1  227.978ms  227.978ms  7.30K       7.30K  193.00 KB     160.00 MB  functional.alltypes b
05:EXCHANGE                1  816.252us  816.252us  7.30K       7.30K          0             0  HASH(a.float_col)
00:SCAN HDFS               1  228.362ms  228.362ms  7.30K       7.30K  193.00 KB     160.00 MB  functional.alltypes a
```

cloudera

Query Tuning Basics – Finding Skew

- Use `ExecSummary` from Query Profile to identify skew
  - Max Time is significantly more than Avg Time => Skew!

```
    ExecSummary:
Operator              #Hosts   Avg Time   Max Time  #Rows  Est. #Rows   Peak Mem  Est. Peak Mem  Detail
------------------------------------------------------------------------------------------------------------
09:MERGING-EXCHANGE        1    4.394ms    4.394ms  7.30K       8.16K          0       -1.00 B  UNPARTITIONED
04:SORT                    1   38.492ms   38.492ms  7.30K       8.16K   32.02 MB       8.00 MB
08:AGGREGATE               1    8.397ms    8.397ms  7.30K       8.16K  458.25 KB      10.00 MB  MERGE FINALIZE
07:EXCHANGE                1  779.810us  779.810us  7.30K       8.16K          0             0  HASH(a.id)
03:AGGREGATE               1  161.736ms  161.736ms  7.30K       8.16K  466.25 KB      10.00 MB
02:HASH JOIN               1  289.552ms  289.552ms  5.33M       5.33M  318.25 KB      20.91 KB  INNER JOIN, PARTITIONED
|--06:EXCHANGE             1    1.93ms     1.93ms  7.30K       7.30K          0             0  HASH(b.float_col)
|  01:SCAN HDFS            1  227.978ms  227.978ms  7.30K       7.30K  193.00 KB     160.00 MB  functional.alltypes b
05:EXCHANGE                1  816.252us  816.252us  7.30K       7.30K          0             0  HASH(a.float_col)
00:SCAN HDFS               1  228.362ms  228.362ms  7.30K       7.30K  193.00 KB     160.00 MB  functional.alltypes a
```

cloudera

Query Tuning Basics – Finding Skew (cont')

- In addition to profile, always measure CPU, memory, disk IO and network IO across the cluster.
  - An uneven distribution of the load means skew!
- Cloudera Manager's chat can do that, or use Colmux if your workload is short.

Query Tuning Basics – Improving Scan Node Performance

- HDFS Scan time – check out how much data is read; always do as little disk read as possible; review partition strategy.
- Column materialization time – only select necessary columns! Materializing 1k col is a lot of work.
- Complex predicate – string, regex are costly; avoid them.

Query Tuning Basics – Aggregate Performance Tuning

- Needed when many rows going into aggregate
- Complex UDA
- (Usually, not a big issue)

cloudera

Query Tuning Basics – Exchange Performance Issues

- Too much data across network:
  - Check the query on data size reduction.
  - Check join order and join strategy; wrong order/strategy can have a serious effect on network!
  - For agg, check the number of groups – affect memory too!
  - Remove unused columns.
- Keep in mind that network is at most 10GB.
- Cross-rack network slowness
- Query profile is usually not useful. Use CM or other system monitoring tools.

cloudera

Query Tuning Basics – Storage Skew

- HDFS Block Placement Skew
  - HDFS balances disk usage evenly across the whole cluster only. An individual table (or partition)'s data could be clustered in a handful of nodes.
  - If this happens, you'll see that some nodes are busier (disk read and usually cpu as well) than the others.
  - This is inherent to HDFS: more pronounced if query data volume is tiny when compared to the total storage capacity.
  - By running a mixed workload that access data of a bigger set of tables, this type of hdfs block placement skew usually even out.

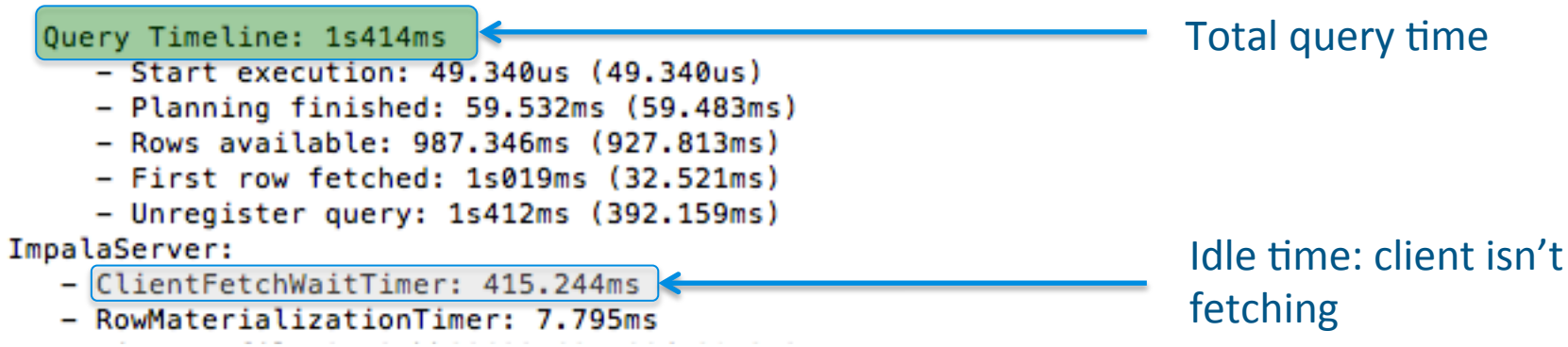
Query Tuning Basics – Data Skew

- Partitioned Join Node Performance Skew
  - Join key data skew.
  - Each join key is re-shuffled and processed by one node.
  - If a single join key value account for a huge chuck of the data, then the processing node that process that join key will become the bottleneck!

- Possible workaround: use broadcast join but it uses more memory

Query Tuning Basics – Clientside Performance Issues

- Avoid large data extract.
  - It's usually not a good idea to dump tons of data out using JDBC/ODBC.
- For Impala-shell, use the –b option to fetch lots of data.



Topic Outline

- Part 1 – The Basics
  - Physical and Schema Design
  - Memory Usage
- Part 2 – The Practical Issues
  - Cluster Sizing and Hardware Recommendations
  - Benchmarking Impala
  - Multi-tenancy Best Practices
  - Query Tuning Basics
- Part 3 – Outside Impala
  - Interaction with Apache Hive, Apache Sentry, and Apache Parquet

cloudera

Interaction with Sentry, Hive, and Parquet

- Setup: Cloudera Manager 5.x does a good job; verify the dependency parents, such as Hive Metastore, HDFS.
  - Stability in HMS might affect Impala; check HMS health.
- File-based and Apache Sentry security
  - Even with Sentry, Impala needs to read/write all dir/files. No impersonation.

Creating Parquet Files

- How to create proper Parquet file from Hive:
  - `mapred.min.split.size = 1GB`
  - `parquet.block.size = INFINITY`
  - `dfs.block.size = 1GB`
  - You can only create a file from Hive reliably by running:
    `INSERT INTO parquet_tbl select * from src_tbl`
  - DO NOT running a complex ETL query and then write it to Parquet.
- Impact of "incorrect" Parquet file: lots of remote read
- Doesn't have to be 1GB per file; 300MB or so is ok.

Summary

- Approach cluster sizing systematically - SLA and workload
- Benchmark running technique and measurement – QPS
- Use Admission Control for multi-tenancy
- Tune your queries - identify and attack bottlenecks
- Cloudera Manager 5.0+ provides a tool for verifying whether many best practices have been followed

Other Resources

- Impala User Guide:
  http://www.cloudera.com/content/cloudera-content/cloudera-docs/CDH5/latest/Impala/impala.html
- Impala website (roadmap, repository, books, more) & Twitter:
  http://impala.io, @RideImpala
- Community resources:
  - Mailing list:
    impala-user@cloudera.org
  - Forum:
    http://community.cloudera.com/t5/Interactive-Short-cycle-SQL/bd-p/Impala

cloudera

cloudera

Thank you

impala.io | @RideImpala