How to use Impala query plan and profile to fix performance issues

Juan Yu
Impala Field Engineer, Cloudera
Profiles?! Explain plans!? Arggghhh…

- For the end user, **understanding Impala performance is like**…

  - Lots of commonality between requests, *e.g.*
  - *What’s the bottleneck for this query?*
  - *Why this run is fast but that run is slow?*
  - *How can I tune to improve this query’s performance.*
Agenda

- What are query plan and profile
- What kind of issues query plan and profile can help you solve
- Structure of query plan and profile
- Basic troubleshooting
- Advanced query tuning and troubleshooting
Why did the following queries take different time?

```
SELECT AVG(ss_list_price) FROM store_sales;
Fetched 1 row(s) in 3.60s
```

```
SELECT AVG(ss_list_price) FROM store_sales WHERE ss_sold_date_sk BETWEEN 2451959 AND 2451989;
Fetched 1 row(s) in 0.28s
```

Partition Pruning reduces scan work and intermediate result.
Where to get Query plan and profile

- Cloudera Manager Query history page
- Impala webUI queries page
- Impala-shell

- Profile examples: https://github.com/yjwater/strata-sj
Query Planning: Goals

- Lower execution cost and get better performance
- Reduce unnecessary work as much as possible by partition pruning, predicate pushdown, runtime filter, etc.
- Maximize scan locality using DN block metadata.
- Minimize data movement (optimize join order, choose better join strategy)
- Parallel tasks and run them on many nodes to shorten execution time.
Query profile - Execution Results

- Provide metrics and counters to show
  - how execution is ongoing.
  - how much resources are used.
  - how long each piece takes.
  - Is there any bottleneck and/or abnormal behavior.
What issues query plan and profile can help you solve?

**Plan:**
- Missing stats
- Partition pruning
- Predicate pushdown
- Join order
- Join strategy
- Parallelism

**Profile:**
- Identify Bottleneck
- Runtime filter effectiveness
- Memory usage/Spill to disk
- Network slowness
- Skew
- Client side issues
- Metadata loading
Flow of a SQL Query

Client

SQL Text

Impala Frontend (Java)

Compile Query

Executable Plan

Impala Backend (C++)

Execute Query

Query Profile

Query Results

Client
Impala in action - Select query

select * from db.foo;
Query Planning

- Single-Node Plan (SET NUM_NODES=1):
  - Assigns predicates to lowest plan node.
  - Prunes irrelevant columns and partitions (if applicable).
  - Optimizes join order.
  - Determine effective runtime filters

- Distributed Plan:
  - Provides list of best Impalads to do the scan work.
  - Picks an execution strategy for each join (broadcast vs hash-partitioned)
  - Introduces exchange operators and groups nodes in plan fragments (unit of work).
SELECT t1.dept, SUM(t2.revenue) 
FROM LargeHdfsTable t1 
JOIN SmallHdfsTable t2 ON (t1.id1 = t2.id) 
JOIN LargeHbaseTable t3 ON (t1.id2 = t3.id) 
WHERE t3.category = 'Online' AND t1.id > 10 
GROUP BY t1.dept 
HAVING COUNT(t2.revenue) > 10 
ORDER BY revenue LIMIT 10

Hint: set num_nodes=1; to explain the single node plan
Single to Distributed Node Plan

Single-Node Plan

1. TopN
2. Agg
   - HashJoin
     - Scan: t3
3. HashJoin
   - Scan: t2
   - Scan: t1

Distributed Plan

1. TopN
2. Exchange
3. TopN
   - Merge-Agg
   - Exchange
   - Pre-Agg
   - HashJoin
     - Exchange
     - Scan: t3
3. Exchange
   - Scan: t2
   - HashJoin
   - Exchange
   - Scan: t1

This is what EXPLAIN shows by default

#StrataData
Plan Fragmentation

Single-Node Plan

- TopN
- Agg
- HashJoin
- Scan: t1
- HashJoin
- Scan: t2
- Scan: t3

Distributed Plan

- TopN
- Exchange
- Merge-Agg
- Exchange
- Pre-Agg
- HashJoin
- Exchange
- HashJoin
- Exchange
- Scan: t1
- Scan: t2
- Scan: t3

execution model subject to change

#StrataData
Query Plan Visualization

```
select c_name, c_custkey, o_orderkey, o_orderdate, o_totalprice, sum(l_quantity)
from customer, orders, lineitem
where o_orderkey in (select l_orderkey
                     from lineitem
                     group by l_orderkey
                     having sum(l_quantity) > 300 )
  and c_custkey = o_custkey
  and o_orderkey = l_orderkey
group by c_name, c_custkey, o_orderkey, o_orderdate, o_totalprice
order by o_totalprice desc, o_orderdate
limit 100
```
Query Execution

```
explain SELECT * FROM t1 JOIN [shuffle] t2 ON t1.id = t2.id;
```

Explain String

Estimated Per-Host Requirements: Memory=240.00MB VCores=2

05:EXCHANGE [UNPARTITIONED]

02:HASH JOIN [INNER JOIN, PARTITIONED]
hash predicates: t1.id = t2.id

--04:EXCHANGE [HASH(t2.id)]

01:SCAN HDFS [functional.alltypestiny t2]
partitions=4/4 files=4 size=460B

03:EXCHANGE [HASH(t1.id)]

00:SCAN HDFS [functional.alltypes t1]
partitions=24/24 files=24 size=478.45KB
Plan and Profile structure

Query Summary
- Basic info: state, type, user, statement, coordinator
- Query plan
- Execution summary
- Timeline

Client side info
Execution details
- Runtime Filter table
- Coordinator Fragment
  - Instance
  - Operator node A
  ...
- Average Fragment 3
  - Fragment instance 0
  - Operator node B
  ...
- Fragment instance 1
  ...
- Average Fragment 2
- Average Fragment 0
Basic Query information

Query (id=5349daec57a4d786:9536a60900000000):
  Summary:
    Session ID: 5245f03b5b3c17ec:1dbd50ff83471f95
    Session Type: HIVESERVER2
    HiveServer2 Protocol Version: V6
    End Time: 2018-02-09 13:20:05.281900000
    Query Type: QUERY
    Query State: FINISHED
    Query Status: OK
    Impala Version: impalad version 2.11.0-cdh5.14.0
    User: REDACTED
    Connected User: REDACTED
    Delegated User:
    Network Address: REDACTED:54129
    Default Db: tpch_100_parquet
    Sql Statement: select * from lineitems limit 100
    Coordinator: REDACTED:22000
    Query Options (set by configuration): ABORT_ON_ERROR=1,MEM_LIMIT=5658116096
    Query Options (set by configuration and planner): ABORT_ON_ERROR=1,MEM_LIMIT=5658116096,MT_DOP=0
Look into the Timeline

Planner Timeline: 218.105ms
- Analysis finished: 159.486ms (159.486ms)
- Value transfer graph computed: 159.522ms (35.958us)
- Single node plan created: 162.606ms (3.083ms)
- Runtime filters computed: 162.869ms (262.984us)
- Distributed plan created: 162.908ms (39.628us)
- Lineage info computed: 163.006ms (97.845us)
- Planning finished: 218.105ms (55.098ms)

Query Timeline: 2m34s
- Query submitted: 12.693ms (12.693ms)
- Planning finished: 350.339ms (337.646ms)
- Submit for admission: 422.437ms (72.097ms)
- Completed admission: 433.900ms (11.462ms)
- Ready to start on 8 backends: 648.182ms (214.282ms)
- All 8 execution backends (47 fragment instances) started: 5s683ms (5s035ms)
- First dynamic filter received: 1m50s (1m44s)
- Rows available: 2m32s (41s835ms)
- First row fetched: 2m32s (447.280ms)
- Unregister query: 2m34s (1s232ms)
Client side

- Avoid large data extract.
- It’s usually not a good idea to dump lots of data out using JDBC/ODBC.
- For Impala-shell, use the –B option to fetch lots of data.
<table>
<thead>
<tr>
<th>Operator</th>
<th>#Hosts</th>
<th>Avg Time</th>
<th>Max Time</th>
<th>#Rows</th>
<th>Est. #Rows</th>
<th>Peak Mem</th>
<th>Est. Peak Mem</th>
<th>Detail</th>
</tr>
</thead>
<tbody>
<tr>
<td>30: MERGING-EXCHANGE</td>
<td>1</td>
<td>211.877us</td>
<td>211.877us</td>
<td>52</td>
<td>5</td>
<td>0</td>
<td>-1.00 B</td>
<td>UNPARTITIONED</td>
</tr>
<tr>
<td>16: TOP-N</td>
<td>15</td>
<td>148.816us</td>
<td>204.292us</td>
<td>52</td>
<td>5</td>
<td>12.00 KB</td>
<td>130.00 B</td>
<td>FINALIZE</td>
</tr>
<tr>
<td>29: AGGREGATE</td>
<td>15</td>
<td>2.462ms</td>
<td>2.795ms</td>
<td>52</td>
<td>5</td>
<td>2.34 MB</td>
<td>10.00 MB</td>
<td>HASH(a.ca_state)</td>
</tr>
<tr>
<td>25: EXCHANGE</td>
<td>15</td>
<td>259.949us</td>
<td>650.720us</td>
<td>728</td>
<td>51</td>
<td>17.52 MB</td>
<td>10.00 MB</td>
<td>STREAMING</td>
</tr>
<tr>
<td>15: AGGREGATE</td>
<td>15</td>
<td>427.784ms</td>
<td>575.825ms</td>
<td>728</td>
<td>51</td>
<td>14.00 MB</td>
<td>314.00 B</td>
<td>FINALIZE</td>
</tr>
<tr>
<td>14: HASH JOIN</td>
<td>15</td>
<td>352.894ms</td>
<td>584.100ms</td>
<td>3.13M</td>
<td>1.13M</td>
<td>10.00 MB</td>
<td>10.00 MB</td>
<td>LEFT SEMI JOIN, BROADCAST</td>
</tr>
<tr>
<td>12: EXCHANGE</td>
<td>15</td>
<td>62.910us</td>
<td>87.360us</td>
<td>11</td>
<td>11</td>
<td>1.68 MB</td>
<td>10.00 MB</td>
<td>STREAMING</td>
</tr>
<tr>
<td>11: HASH JOIN</td>
<td>15</td>
<td>64.072us</td>
<td>64.072us</td>
<td>11</td>
<td>11</td>
<td>5.75 MB</td>
<td>80.00 MB</td>
<td>tpcds_100_parquet.item j</td>
</tr>
<tr>
<td>10: SCAN HDFS</td>
<td>1</td>
<td>31.863ms</td>
<td>31.863ms</td>
<td>11</td>
<td>11</td>
<td>8.22 MB</td>
<td>5.00 MB</td>
<td>LEFT SEMI JOIN, BROADCAST</td>
</tr>
<tr>
<td>9: SCAN HDFS</td>
<td>1</td>
<td>7.636ms</td>
<td>7.636ms</td>
<td>300.00K</td>
<td>300.00K</td>
<td>2.27 MB</td>
<td>10.00 MB</td>
<td>FINALIZE</td>
</tr>
<tr>
<td>8: AGGREGATE</td>
<td>1</td>
<td>2531ms</td>
<td>25419ms</td>
<td>1</td>
<td>1</td>
<td>2.27 MB</td>
<td>10.00 MB</td>
<td>HASH(d_month_seq)</td>
</tr>
<tr>
<td>7: SCAN HDFS</td>
<td>1</td>
<td>2.462ms</td>
<td>2.795ms</td>
<td>108</td>
<td>108</td>
<td>6.41 MB</td>
<td>627.77 KB</td>
<td>tpcds_100_parquet.date_dim</td>
</tr>
<tr>
<td>6: AGGREGATE</td>
<td>1</td>
<td>8.183us</td>
<td>8.183us</td>
<td>108</td>
<td>108</td>
<td>2.20 MB</td>
<td>10.00 MB</td>
<td>STREAMING</td>
</tr>
<tr>
<td>5: SCAN HDFS</td>
<td>1</td>
<td>8.183us</td>
<td>8.183us</td>
<td>108</td>
<td>108</td>
<td>1.04 MB</td>
<td>48.00 MB</td>
<td>tpcds_100_parquet.date_dim</td>
</tr>
<tr>
<td>4: SCAN HDFS</td>
<td>1</td>
<td>7.680us</td>
<td>7.680us</td>
<td>108</td>
<td>108</td>
<td>396.38 MB</td>
<td>138.10 MB</td>
<td>INNER JOIN, BROADCAST</td>
</tr>
<tr>
<td>3: SCAN HDFS</td>
<td>1</td>
<td>1.316ms</td>
<td>1.316ms</td>
<td>2.69B</td>
<td>2.73B</td>
<td>148.37 MB</td>
<td>10.67 MB</td>
<td>tpcds_100_parquet.date_dim</td>
</tr>
<tr>
<td>2: SCAN HDFS</td>
<td>1</td>
<td>8.183us</td>
<td>8.183us</td>
<td>300.00K</td>
<td>300.00K</td>
<td>10.37 MB</td>
<td>120.00 MB</td>
<td>tpcds_100_parquet.customer. i</td>
</tr>
<tr>
<td>1: SCAN HDFS</td>
<td>1</td>
<td>208971ms</td>
<td>26141ms</td>
<td>2.75B</td>
<td>2.73B</td>
<td>650.67 MB</td>
<td>100.71 MB</td>
<td>INNER JOIN, BROADCAST</td>
</tr>
<tr>
<td>0: SCAN HDFS</td>
<td>1</td>
<td>7.430ms</td>
<td>8.579ms</td>
<td>300.00K</td>
<td>300.00K</td>
<td>20.80 MB</td>
<td>112.00 MB</td>
<td>tpcds_100_parquet.customer. i</td>
</tr>
</tbody>
</table>
| 0: SCAN HDFS       | 1      | 14.279ms   | 14.279ms   | 300.00K | 300.00K    | 528.36 MB | 176.00 MB     | tpcds_100_parquet.store_s.
ExecSummary – Find Bottlenecks

- Use ExecSummary from Query Profile to identify bottlenecks

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

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

- Predicate pushdown
- Remote read
- Codegen
- Planning time
- DDLs
Advanced Query Tuning and Troubleshooting
Advanced Query Tuning

- Common issues
  - Query succeeds but very slow
  - Query fails with OOM error

- How to address them
  - Examine the logic of the query and validate the Explain Plan
  - Use Query Profile to identify bottlenecks.
**Performance analysis – Example 1**

<table>
<thead>
<tr>
<th>Operator</th>
<th>#Hosts</th>
<th>Avg Time</th>
<th>Max Time</th>
<th>#Rows</th>
<th>Est. #Rows</th>
</tr>
</thead>
<tbody>
<tr>
<td>02:AGGREGATE</td>
<td>1</td>
<td>15s129ms</td>
<td>15s129ms</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>01:AGGREGATE</td>
<td>1</td>
<td>46s515ms</td>
<td>46s515ms</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>00:SCAN HDFS</td>
<td>1</td>
<td>2s962ms</td>
<td>2s962ms</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

- **Fix:** Compute stats <table>

<table>
<thead>
<tr>
<th>Operator</th>
<th>#Hosts</th>
<th>Avg Time</th>
<th>Max Time</th>
<th>#Rows</th>
<th>Est. #Rows</th>
</tr>
</thead>
<tbody>
<tr>
<td>02:AGGREGATE</td>
<td>17</td>
<td>196.914ms</td>
<td>196.914ms</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>01:AGGREGATE</td>
<td>1</td>
<td>145.541ms</td>
<td>145.541ms</td>
<td>17</td>
<td>1</td>
</tr>
<tr>
<td>02:AGGREGATE</td>
<td>17</td>
<td>285.524ms</td>
<td>308.758ms</td>
<td>17</td>
<td>1</td>
</tr>
<tr>
<td>04:AGGREGATE</td>
<td>17</td>
<td>1s106ms</td>
<td>1s185ms</td>
<td>51.63M</td>
<td>51.63M</td>
</tr>
<tr>
<td>03:EXCHANGE</td>
<td>17</td>
<td>79.289ms</td>
<td>87.953ms</td>
<td>51.63M</td>
<td>51.63M</td>
</tr>
<tr>
<td>01:AGGREGATE</td>
<td>17</td>
<td>1s487ms</td>
<td>2s318ms</td>
<td>51.63M</td>
<td>51.63M</td>
</tr>
<tr>
<td>00:SCAN HDFS</td>
<td>17</td>
<td>190.709ms</td>
<td>264.350ms</td>
<td>51.63M</td>
<td>51.63M</td>
</tr>
</tbody>
</table>
Estimated Per-Host Requirements: Memory=84.76GB Vcores=3

WARNING: The following tables are missing relevant table and/or column statistics.

tpcds500gb_parquet.date_dim, tpcds500gb_parquet.store_sales, tpcds500gb_parquet.item

04: HASH JOIN [INNER JOIN, BROADCAST]
| hash predicates: store_sales.ss_item_sk = item.i_item_sk

--08: EXCHANGE [BROADCAST]

| 02: SCAN HDFS [tpcds500gb_parquet.item]
| partitions=1/1 size=3.14MB
| predicates: item.i_manager_id = 1

03: HASH JOIN [INNER JOIN, BROADCAST]
| hash predicates: dt.d_date_sk = store_sales.ss_sold_date_sk

--07: EXCHANGE [BROADCAST]

| 01: SCAN HDFS [tpcds500gb_parquet.store_sales]
| partitions=1823/1823 size=180.21GB

00: SCAN HDFS [tpcds500gb_parquet.date_dim dt]
| partitions=1/1 size=2.41MB

Performance analysis – Example 2

Missing stats
Performance analysis – Example 2 (good plan)

set explain_level=3;
set num_nodes=0;

|--07:EXCHANGE [BROADCAST]
|  | hosts=3 per host mem=0B
|  | tuple-ids=0 row-size=16B cardinality=29

00:SCAN HDFS [tpcds500gb_parquet.date_dim dt, RANDOM]
  partitions=1/1 size=2.41MB
  table stats: 73049 rows total
  column stats: all
  hosts=3 per-host-mem=48.00MB
  tuple-ids=0 row-size=16B cardinality=29

01:SCAN HDFS [tpcds500gb_parquet.store_sales, RANDOM]
  partitions=1823/1823 size=180.21GB
  table stats: 4125561494 rows total
  column stats: all
  hosts=10 per-host-mem=176.00MB
  tuple-ids=1 row-size=20B cardinality=4125561494
Query Tuning Basics - Join

- Validate join order and join strategy
  - Optimal Join Order
    - RHS should be smaller than LHS
    - Minimize intermediate results
  - Strategy – Broadcast vs. Partitioned
    - Network costs (partition and send lhs+rhs or broadcast rhs)
    - Memory costs
    - RHS must fit in memory!
Join-Order Optimization

```
SELECT ... FROM T1, T2, T3, T4
WHERE T1.id = T2.id AND T2.id = T3.id AND T3.id = T4.id;
```
Use of Statistics during Plan Generation

- **Table statistics**: Number of rows per partition/table.
- **Column statistics**: Number of distinct values per column.

- **Use of table and column statistics**
  - Estimate selectivity of predicates (esp. scan predicates like “month=10”).
  - Estimate selectivity of joins $\rightarrow$ join cardinality (#rows)
    - Heuristic FK/PK detection.
  - Pick distributed join strategy: broadcast vs. partitioned
    - Join inputs have very different size $\rightarrow$ broadcast small side.
    - Join inputs have roughly equal size $\rightarrow$ partitioned.
Join Execution Strategy

- Broadcast Join
  - Scan and broadcast Table B
  - Local scan for Table A

- Repartition Join
  - Scan and repartition both tables according to join keys

- $A \text{ join } B$

Scan and broadcast Table B
Local scan for Table A
Join Strategy – what’s the cost

- Impala choses the right strategy based on stats (**collect stats!**)  
- Use the join strategy that minimizes data transfer  
- Use explain plan to see the data size  
- Join Hint: [shuffle] or [broadcast]

<table>
<thead>
<tr>
<th>Join strategy</th>
<th>Network Traffic</th>
<th>Memory Usage (HashTable)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Broadcast Join</td>
<td>RHS table size(^1) x number of node</td>
<td>RHS table size(^1) x number of node</td>
</tr>
<tr>
<td>Partitioned Join</td>
<td>LHS + RHS table size(^1)</td>
<td>RHS table size(^1)</td>
</tr>
</tbody>
</table>

\(^1\) table size refers to the data flowed from the child node (i.e. only required column data after filtering counts).
Join - Exercise

- TPCH 100GB data on 8-node cluster
  - Table1: lineitem. cardinality=600M, row size ~ 260B
  - Table2: orders. cardinality=150M, row size ~ 200B
  - orderkey is bigint.

- For the following query, what’s the optimal join order and join strategy.
  - `SELECT count(*) FROM lineitem JOIN orders ON l_orderkey = o_orderkey;`
  - `SELECT lineitem.* FROM lineitem JOIN orders ON l_orderkey = o_orderkey;`
  - `SELECT orders.* FROM lineitem JOIN orders ON l_orderkey = o_orderkey;`

- How about on a 100-node cluster?
Runtime Filter - Check how selective the join is

Large hash join output (~2.8B), the output of upstream one is much smaller (32M, reduced to only ~1%). This indicates runtime filter can be helpful.
RF002 <- d_d_date_sk
Filter coming from Exch Node 20, Applied to Scan Node 02, reduce scan output from 2.8B to 32M and all the upstream hash join as well.

RF001 <- (d_month_seq)
Filter coming from Scan Node 05, Applied to Scan Node 03, Reduce scan output from 73049 to 31
Why Runtime Filter doesn’t work sometimes?

Filter doesn’t take effect because scan was short and finished before filter arrived.

HDFS_SCAN_NODE (id=3):

- Rows processed: 16.38K (16383)
- Rows rejected: 0 (0)
- Rows total: 16.38K (16384)
How to tune it?

- Increase RUNTIME_FILTER_WAIT_TIME_MS to 5000ms to let Scan Node 03 wait longer time for the filter.
  
  HDFS_SCAN_NODE (id=3)
  
  Filter 1 (1.00 MB)
  - InactiveTotalTime: 0
  - Rows processed: 73049
  - Rows rejected: 73018
  - Rows total: 73049
  - TotalTime: 0

- If the cluster is relatively busy, consider increasing the wait time too so that complicated queries do not miss opportunities for optimization.
Runtime filter profile examples

- Profile walkthrough
  - Effective vs Non-Effective
  - Local vs Global
  - Filter Wait Time
  - Filter Memory Usage
Memory

- Planner estimation

  Estimated Per-Host Requirements: Memory=68.01MB VCores=2

  06:SORT
  | order by: i_class ASC NULLS FIRST
  | hosts=1 per-host-mem=16.00MB
  | tuple-ids=7 row-size=214B cardinality=48

- Actual usage from profile

  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

- Metrics per node

  - PeakMemoryUsage: 70048
  - PerHostPeakMemUsage: 1779374788
  - MemoryLimit: 85899345920
  - PeakMemoryUsage: 1219248384

#StrataData
Memory - OOM

Query Status: Memory limit exceeded: FunctionContext::Allocate's allocations exceeded memory limits. Exprs could not allocate 384.00 B without exceeding limit.
Error occurred on backend vd1337.halxg.clooudera.com:22000 by fragment c74ce10ea42773c1:8f12f6e900000080
Memory left in process limit: -853607.00 B
Process: memory limit exceeded. Limit=201.73 GB Total=201.73 GB Peak=201.73 GB
  RequestPool=root.default: Total=189.29 GB Peak=189.29 GB
    Query(c74ce10ea42773c1:8f12f6e900000000): Total=189.22 GB Peak=189.22 GB
      Fragment c74ce10ea42773c1:8f12f6e9000000101: Total=10.30 MB Peak=11.02 MB
        AGGREGATION_NODE (id=2): Total=8.00 KB Peak=8.00 KB
          Exprs: Total=4.00 KB Peak=4.00 KB
        AGGREGATION_NODE (id=4): Total=10.27 MB Peak=10.27 MB
          Exprs: Total=4.00 KB Peak=4.00 KB
        EXCHANGE_NODE (id=3): Total=0 Peak=0
      DataStreamRecvr: Total=0 Peak=0
      DataStreamSender (dst_id=5): Total=7.52 KB Peak=7.52 KB
    CodeGen: Total=6.79 KB Peak=750.50 KB
  Block Manager: Limit=161.39 GB Total=13.63 GB Peak=13.63 GB
    Fragment c74ce10ea42773c1:8f12f6e900000080: Total=189.21 GB Peak=189.21 GB
      AGGREGATION_NODE (id=1): Total=188.83 GB Peak=188.83 GB
        Exprs: Total=175.20 GB Peak=175.20 GB
      HDFS_SCAN_NODE (id=0): Total=385.53 MB Peak=601.23 MB
      DataStreamSender (dst_id=3): Total=660.12 KB Peak=660.12 KB
      CodeGen: Total=4.48 KB Peak=610.00 KB
    Query(8743be49f34a3cb9:8bb6ace100000000): Total=34.30 MB Peak=49.49 MB
      Fragment 8743be49f34a3cb9:8bb6ace100000000f7: Total=34.30 MB Peak=35.22 MB
Memory – Insert strategy

- Non-shuffle
  - Node A
    Partition data 1, 3, 5, 6
    4 writers
  - Node B
    Partition data 2, 3, 6, 7
    4 writers
  - Node C
    Partition data 3, 4, 8
    3 writers
  - Node D
    Partition data 1, 4, 7, 8
    4 writers

- Shuffle
  - Node A
    Partition data 1, 5
    2 writers
  - Node B
    Partition data 2, 6
    2 writers
  - Node C
    Partition data 3, 7
    2 writers
  - Node D
    Partition data 4, 8
    2 writers
**Memory – Group By**

- `SELECT product, count(1), sold_date FROM sales_history GROUP BY product, sold_date;`
- We have one million different products, table contains data in the past 5 years.

**Total groups = 1M * 5 * 365 = ~1.8B**

Query Status: Memory limit exceeded

<table>
<thead>
<tr>
<th>Operator</th>
<th>#Hosts</th>
<th>#Rows</th>
<th>Est. #Rows</th>
<th>Peak Mem</th>
<th>Est. Peak Mem</th>
</tr>
</thead>
<tbody>
<tr>
<td>04:EXCHANGE</td>
<td>1</td>
<td>0</td>
<td>10</td>
<td>0</td>
<td>-1.00 B</td>
</tr>
<tr>
<td>03:AGGREGATE</td>
<td>68</td>
<td>0</td>
<td>3.24B</td>
<td>260.27 MB</td>
<td>84.34 MB</td>
</tr>
<tr>
<td>02:EXCHANGE</td>
<td>68</td>
<td>6.87M</td>
<td>18.50B</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>01:AGGREGATE</td>
<td>68</td>
<td>6.38M</td>
<td>18.50B</td>
<td>20.89 GB</td>
<td>984.90 GB</td>
</tr>
<tr>
<td>00:SCAN HDFS</td>
<td>68</td>
<td>12.95B</td>
<td>18.50B</td>
<td>414.27 MB</td>
<td>48.00 MB</td>
</tr>
</tbody>
</table>

---

#StrataData

**AGGREGATION_NODE (id=1)**
- ExecOption: Codegen Enabled, Spilled
  - AsyncTotalTime: 0
  - BuildTime: 165828722183
  - GetNewBlockTime: 191148346
  - GetResultsTime: 1035063643
  - HashBuckets: 16777216
  - InactiveTotalTime: 0
  - LargestPartitionPercent: 6
  - MaxPartitionLevel: 0
  - NumRepartitions: 0
  - PartitionsCreated: 16
  - PeakMemoryUsage: 22434481544
  - PinTime: 0
  - RowsRepartitioned: 0
  - RowsReturned: 6375424
  - RowsReturnedRate: 37756
  - SpilledPartitions: 1
  - TotalTime: 168854653348
Memory Usage – Estimation

- EXPLAIN’s memory estimation issues
  - Can be way off – much higher or much lower.
  - Group-by/distinct estimate can be particularly off – when there’s a large number of group by columns (independence assumption)
    - Memory estimate = NDV of group by column 1 * NDV of group by column 2 * … NDV of group by column n
  - Ignore EXPLAIN’s estimation if it’s too high!

- Do your own estimate for group by
  - GROUP BY memory usage = (total number of groups * size of each row) + (total number of groups * size of each row) / number node
Memory Usage – Hitting Mem-limit

- Gigantic group by
  - The total number of distinct groups is huge, such as group by userid, phone number.
  - For a simple query, you can try this advanced workaround – per-partition agg
    - Requires the partition key to be part of the group by

```
SELECT part_key, col1, col2, ...agg(..) FROM tbl WHERE part_key in (1,2,3)
UNION ALL
SELECT part_key, col1, col2, ...agg(..) FROM tbl WHERE part_key in (4,5,6)
```
Memory Usage – Hitting Mem-limit

- Big-table joining big-table
  - Big-table (after decompression, filtering, and projection) is a table that is bigger than total cluster memory size.
  - For a simple query, you can try this advanced workaround – per-partition join
    - Requires the partition key to 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)
```
Query Execution – Typical Speed

- In a typical query, we observed following processing rate:
  - Scan node 8~10M rows per core
  - Join node ~10M rows per sec per core
  - Agg node ~5M rows per sec per core
  - Sort node ~17MB per sec per core
  - Row materialization in coordinator should be tiny
  - Parquet writer 1~5MB per sec per core

- If your processing rate is much lower than that, it’s worth a deeper look
Performance analysis – Example 3

<table>
<thead>
<tr>
<th>Operator</th>
<th>#Hosts</th>
<th>Avg Time</th>
<th>Max Time</th>
<th>#Rows</th>
<th>Est. #Rows</th>
<th>Peak Mem</th>
<th>Est. Peak Mem</th>
</tr>
</thead>
<tbody>
<tr>
<td>04:EXCHANGE</td>
<td>1</td>
<td>64.320us</td>
<td>64.320us</td>
<td>18</td>
<td>18</td>
<td>0</td>
<td>-1.00 B</td>
</tr>
<tr>
<td>03:AGGREGATE</td>
<td>3</td>
<td>222.379ms</td>
<td>248.857ms</td>
<td>18</td>
<td>18</td>
<td>6.27 MB</td>
<td>10.00 MB</td>
</tr>
<tr>
<td>02:EXCHANGE</td>
<td>3</td>
<td>59.650us</td>
<td>60.607us</td>
<td>54</td>
<td>18</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>01:AGGREGATE</td>
<td>3</td>
<td>6s453ms</td>
<td>6s815ms</td>
<td>54</td>
<td>18</td>
<td>6.67 MB</td>
<td>10.00 MB</td>
</tr>
<tr>
<td>00:SCAN HDFS</td>
<td>3</td>
<td>203.150ms</td>
<td>216.808ms</td>
<td>144.62M</td>
<td>144.62M</td>
<td>105.24 MB</td>
<td>176.00 MB</td>
</tr>
</tbody>
</table>

~8M rows per second per core, this is normal aggregation speed.
### Performance analysis – Example 4

#### Data Distribution

Skew causes execution skew

<table>
<thead>
<tr>
<th>Operator</th>
<th>#Hosts</th>
<th>Avg Time</th>
<th>Max Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>04:EXCHANGE</td>
<td>1</td>
<td>999.998us</td>
<td>999.998us</td>
</tr>
<tr>
<td>02:HASH JOIN</td>
<td>7</td>
<td>2s636ms</td>
<td>4s879ms</td>
</tr>
<tr>
<td>00:EXCHANGE</td>
<td>7</td>
<td>142.860us</td>
<td>1.000ms</td>
</tr>
<tr>
<td>01:SCAN HDFS</td>
<td>7</td>
<td>461.717ms</td>
<td>486.001ms</td>
</tr>
<tr>
<td>00:SCAN HDFS</td>
<td>7</td>
<td>2s862ms</td>
<td>3s515ms</td>
</tr>
</tbody>
</table>

Averaged Fragment F00

- Split sizes: min: 982.25 MB, max: 2.96 GB
- Completion times: min: 4s474ms, max: 3s515ms
- Execution rates: min: 219.50 MB/s, max: 159.50 MB/s
- Num instances: 7

```plaintext
HDFS_SCAN_NODE (id=0)
  Hdfs split stats (<volume id>:<# splits>/<split lengths>): 0:15/2.96 GB
    - BytesRead: 1710183933

HDFS_SCAN_NODE (id=0)
  Hdfs split stats (<volume id>:<# splits>/<split lengths>): 0:5/982.25 MB
    - BytesRead: 556141382
```
Performance analysis – Example 5

Possible root cause:
1. The host has a slow/bad disk
2. The host has trouble to communicate to NN
3. Hotspotting, the host does more IO (not just this single query, but overall) than others
4. ...

<table>
<thead>
<tr>
<th>Operator</th>
<th>#Hosts</th>
<th>Avg Time</th>
<th>Max Time</th>
<th>#Rows</th>
<th>ES</th>
</tr>
</thead>
<tbody>
<tr>
<td>00:SCAN HDFS</td>
<td>7</td>
<td>6s577ms</td>
<td>44s614ms</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Averaged Fragment F00:
split sizes: min: 1.62 GB, max: 2.06 GB, avg: 1.84 GB
completion times: min: 9s835ms max: 47s249ms
execution rates: min: 35.02 MB/sec max: 190.00 MB/sec
num instances: 7

- TotalStorageWaitTime: 5m19s
  - BytesRead: 120.19 MB (126032776)
  - RowBatchQueueGetWaitTime: 44s585ms
- TotalStorageWaitTime: 594.017ms
  - BytesRead: 152.74 MB (160156619)
  - RowBatchQueueGetWaitTime: 223.006ms
- TotalStorageWaitTime: 678.025ms
  - BytesRead: 137.05 MB (143707548)
  - RowBatchQueueGetWaitTime: 151.005ms
Performance analysis – Example 6

<table>
<thead>
<tr>
<th>Operator</th>
<th>#Hosts</th>
<th>Avg Time</th>
<th>Max Time</th>
<th>#Rows</th>
<th>Est. #Rows</th>
</tr>
</thead>
<tbody>
<tr>
<td>05:AGGREGATE</td>
<td>2</td>
<td>10s520s</td>
<td>10s534s</td>
<td>3.14M</td>
<td>2.95M</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>03:SCAN HDFS</td>
<td>22</td>
<td>608.374ms</td>
<td>610.741ms</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>00:SCAN HDFS</td>
<td>2</td>
<td>3.181s</td>
<td>3.649s</td>
<td>100.95M</td>
<td>100.95M</td>
</tr>
</tbody>
</table>

00:SCAN HDFS [store sales, RANDOM]

- partitions=2/138 files=2 size=229.00MB
- table stats: 12705876000 rows total

Only two parquet files to scan, not enough parallelism. Try using smaller parquet block size for this table to increase parallelism.
SELECT colE, colU, colR, ... FROM temp WHERE year = 2017 AND month = 03
GROUP BY colE, colU, colR;
Exchange and DataStreamSender

- Impala uses Thrift to transmit compressed data between plan fragments
- Exchange operator is the receiver of the data
- DataStreamSender transmits the output rows of a plan fragment

**EXCHANGE_NODE (id=3)**
- ConvertRowBatchTime: 0ns (0)
- InactiveTotalTime: 0ns (0)
- RowsReturned: 24 (24)
- RowsReturnedRate: 7 per second (7)
- TotalTime: 3.02s (3016602849)

DataStreamReceiver
- BytesReceived: 402 B (402)
- DeserializeRowBatchTime: 0ns (0)
- FirstBatchArrivalWaitTime: 1.83s (1830275553)
- InactiveTotalTime: 0ns (0)
- SendersBlockedTimer: 0ns (0)
- SendersBlockedTotalTimer(*): 0ns (0)

**DataStreamSender (dst_id=3)**
- BytesSent: 402 B (402)
- InactiveTotalTime: 0ns (0)
- NetworkThroughput(*): 87.3 KiB/s (89436)
- OverallThroughput: 283.6 KiB/s (290417)
- PeakMemoryUsage: 120.6 KiB (123488)
- RowsReturned: 24 (24)
- SerializeBatchTime: 0ns (0)
- TotalTime: 888.97us (888969)
- TransmitDataRPCTime: 222.24us (222241)
- UncompressedRowBatchSize: 504 B (504)
Network slowness

- Exchange performance issues
  - Too much data across network:
    - Check the query on data size reduction.
    - Check join order and join strategy; Wrong join 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 typically at most 10Gbit
- Cross-rack network slowness
More examples/Exercises

- Parquet min/max filter, dictionary filter
- Spill to disk
- CPU saturated
- Scanner thread
- Detect small files
- Sink (DML queries)
Recap: Query Tuning

- Performance: Examine the logic of the query and validate the Explain Plan
  Validate join order and join strategy.
  Validate partition pruning and/or predicate pushdown.
  Validate plan parallelism.
- Memory usage: Top causes (in order) of hitting mem-limit:
  Lack of statistics
  Lots of joins within a single query
  Big-table joining big-table
  Gigantic group by (e.g., distinct)
  Parquet Writer
More resources

- Impala cookbook  https://blog.cloudera.com/blog/2017/02/latest-impala-cookbook/
- Impala perf guideline  https://www.cloudera.com/documentation/enterprise/latest/topics/impala_perf_cookbook.html
Thank you

Q & A