Performance & Query Tuning

Query Lifecycle

Profile, EXPLAIN plans and friends…

- Common questions:
  - What is the bottleneck for this query?
  - Why this runs fast but this runs slow?
  - How can I tune to improve this query‘s performance?

Trivia: Which query will run faster?

```
SELECT AVG(ss_list_price)
FROM store_sales;
```

```
SELECT AVG(ss_list_price)
FROM store_sales
WHERE ss_sold_date
BETWEEN date1 AND date2;
```

Query Planning: Goales

- Reduce execution cost and get better performance
- Reduce unnecessary work by steering the Impala optimizer into optimal solutions
- Maximize scan locality / Minimize data movement across the network
- Parallel tasks → run small tasks on many nodes to shorten execution time.

What issues can query plan and profile solve?

Query Plan

- Missing stats
- Partition pruning
- Predicate pushdown
- Join order
- Join stratégy
- Parallelism

Query Profile

- Identify bottlenecks
- Runtime filter effectiveness
- Memory usage
- Network slowness
- Skew

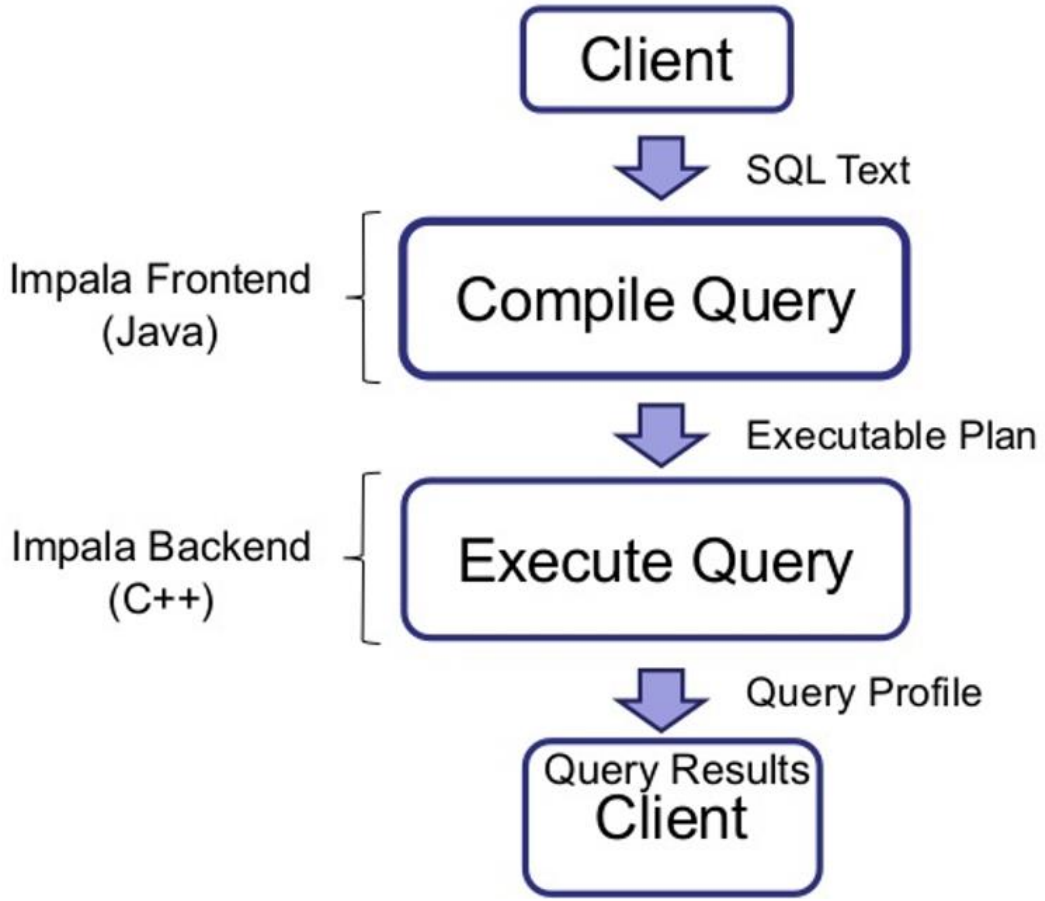
- Client side issues
- Metadata loading

Where to get Query Plan and Profile

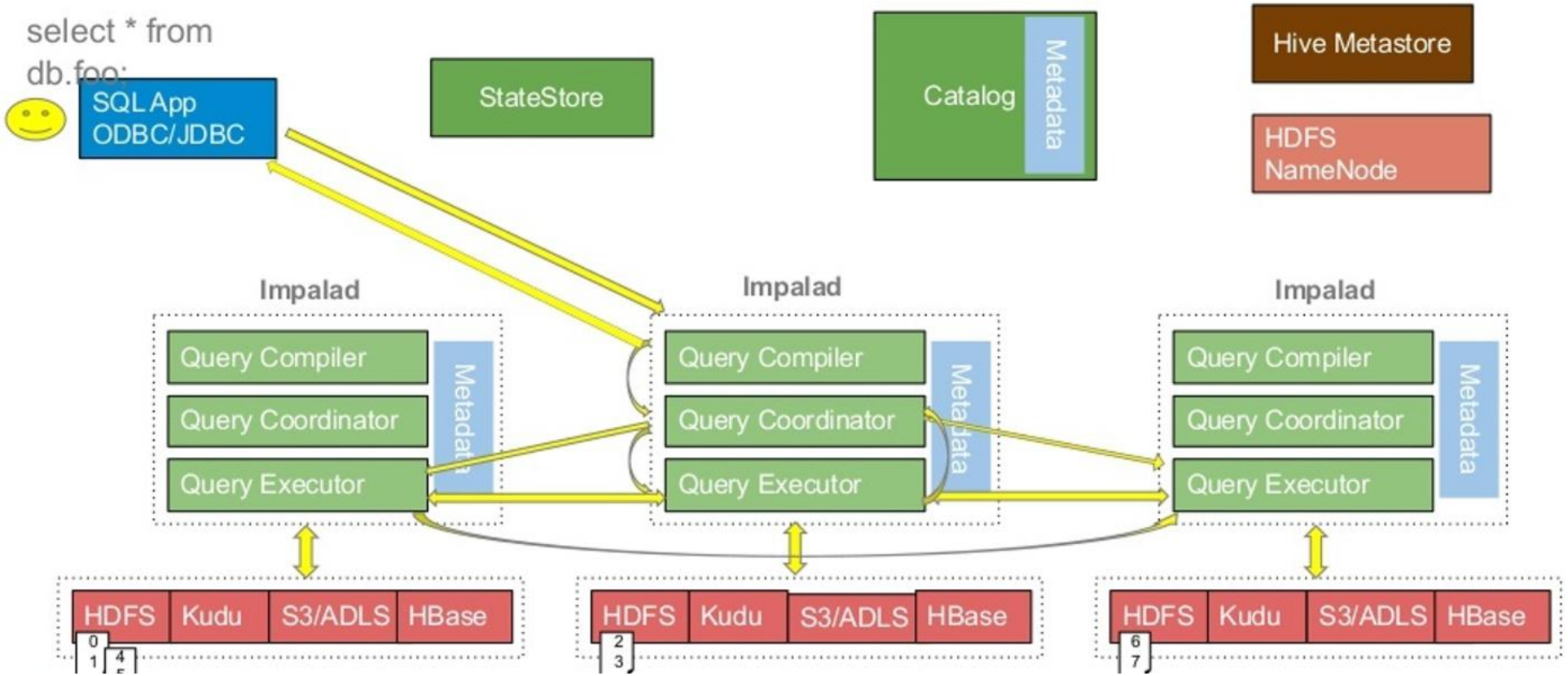
- Impala-shell
- Impala webUI queries page
- Cloudera Manager Query History page

➔Restore desktop from fresh start

Flow of an SQL query



Impala in Action: SQL query

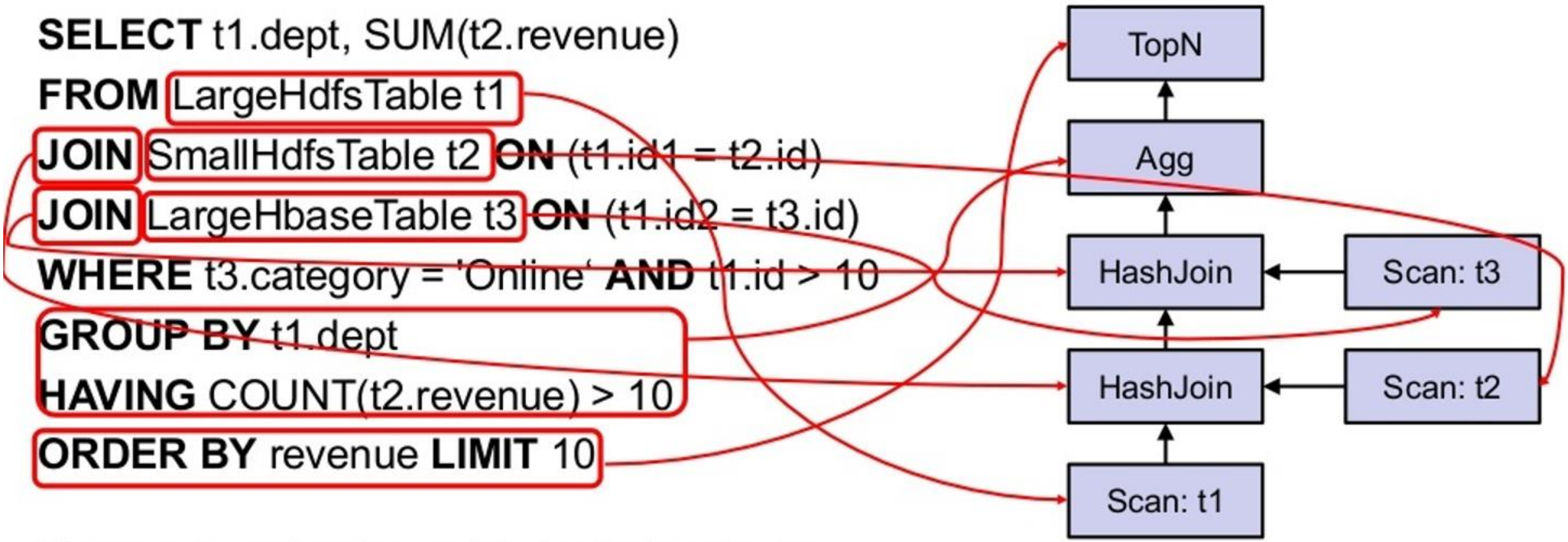


Single vs Distributed Plan

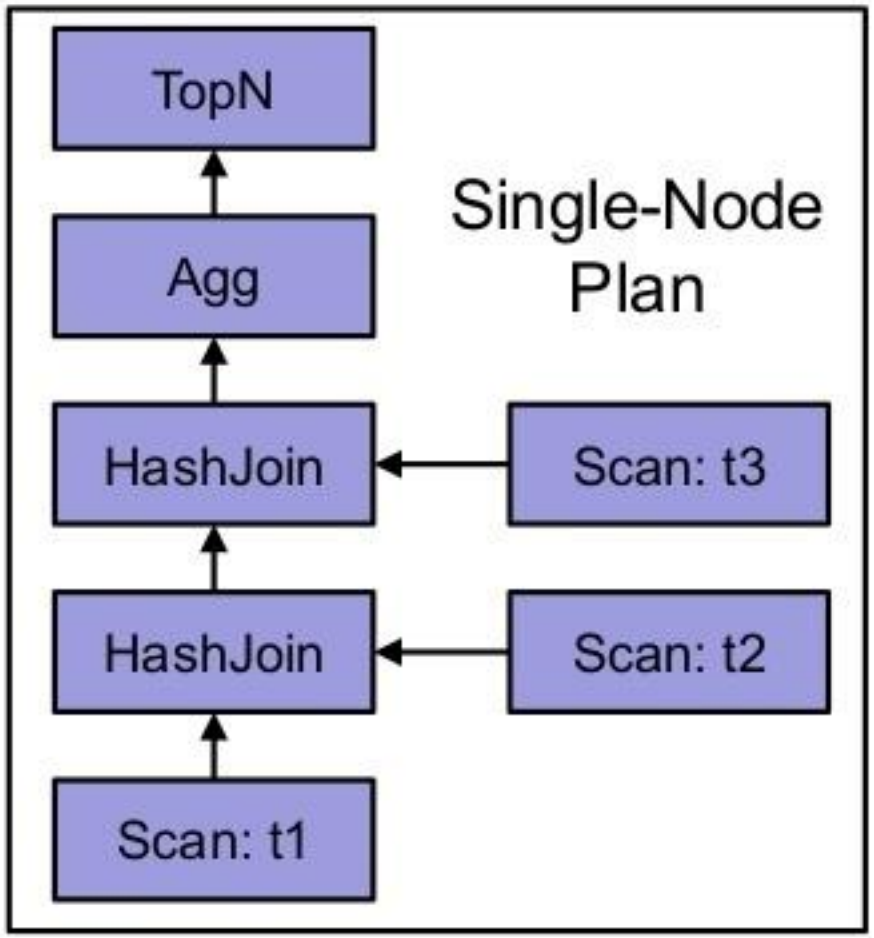
- It is reccommended to run the explain plan in single-node form first (SET NUM_NODES=1) to help us identify easily what columns and partitions should be prunned.

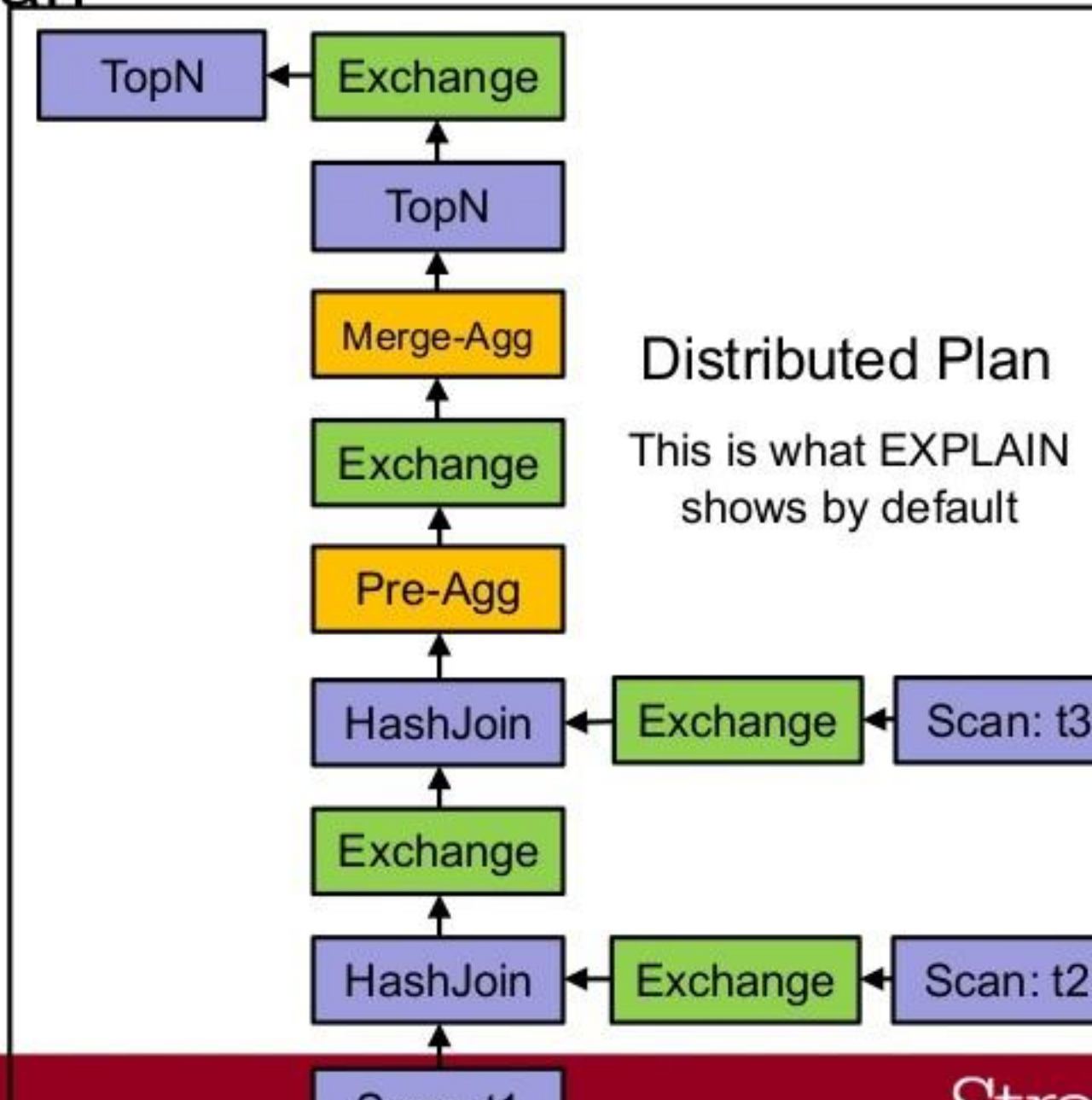
- Distributed plan (SET NUM_NODES=0) : Once we tweaked our query, the distributed plan will identify the best impalad‘s to do the scan work, pick execution strategy for the JOIN (broadcast vs partitioned) and introduce Exchange operators.

Compile Query

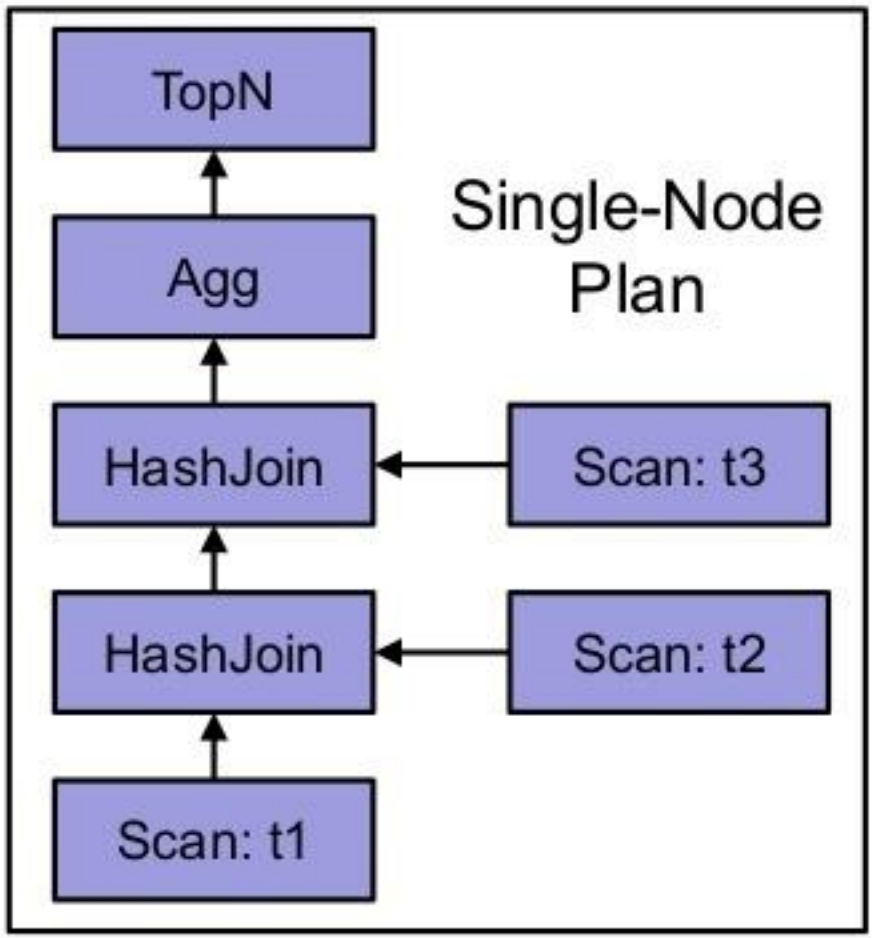


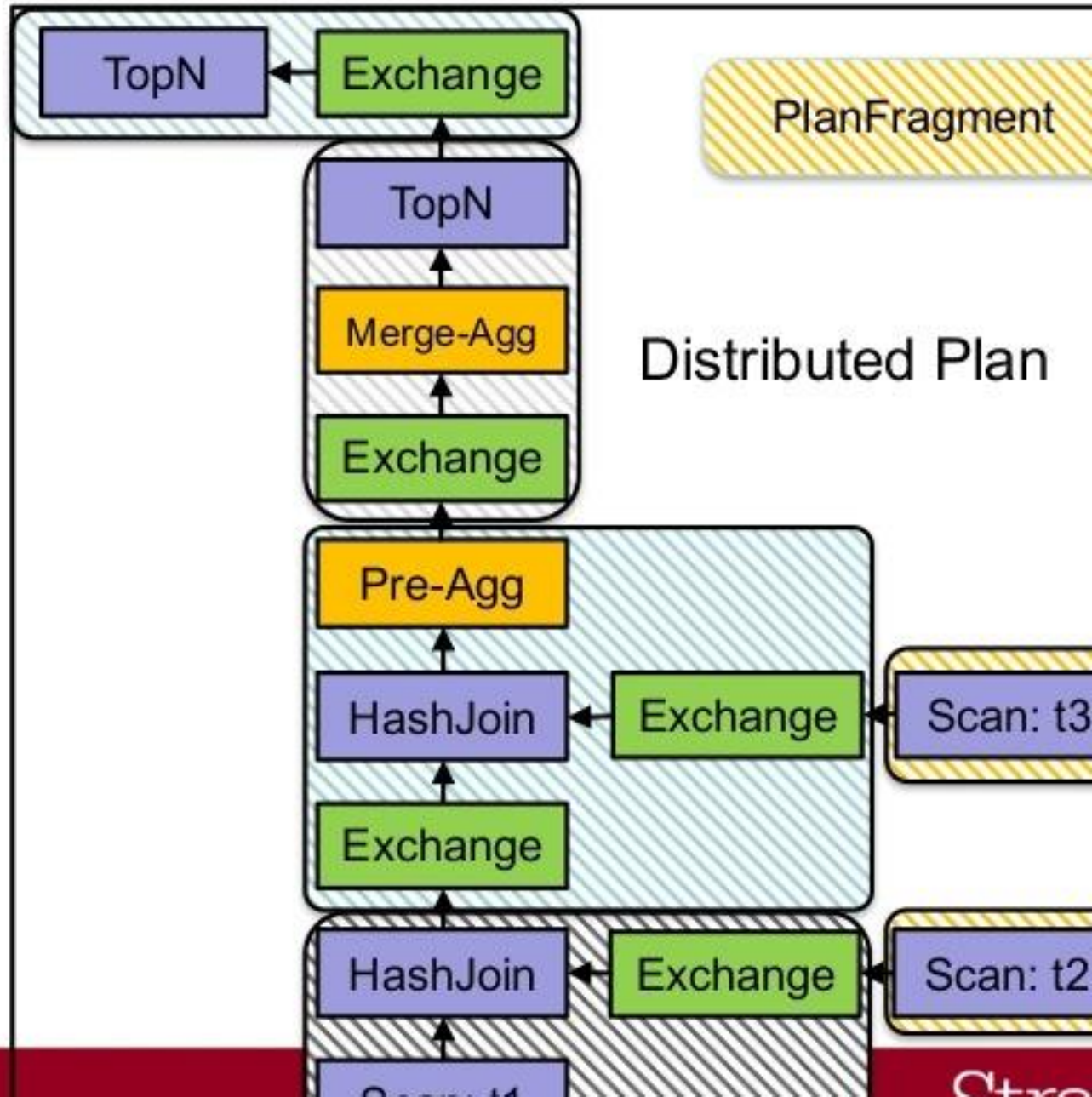
Single to Distributed Node Plan





Plan Fragmentation

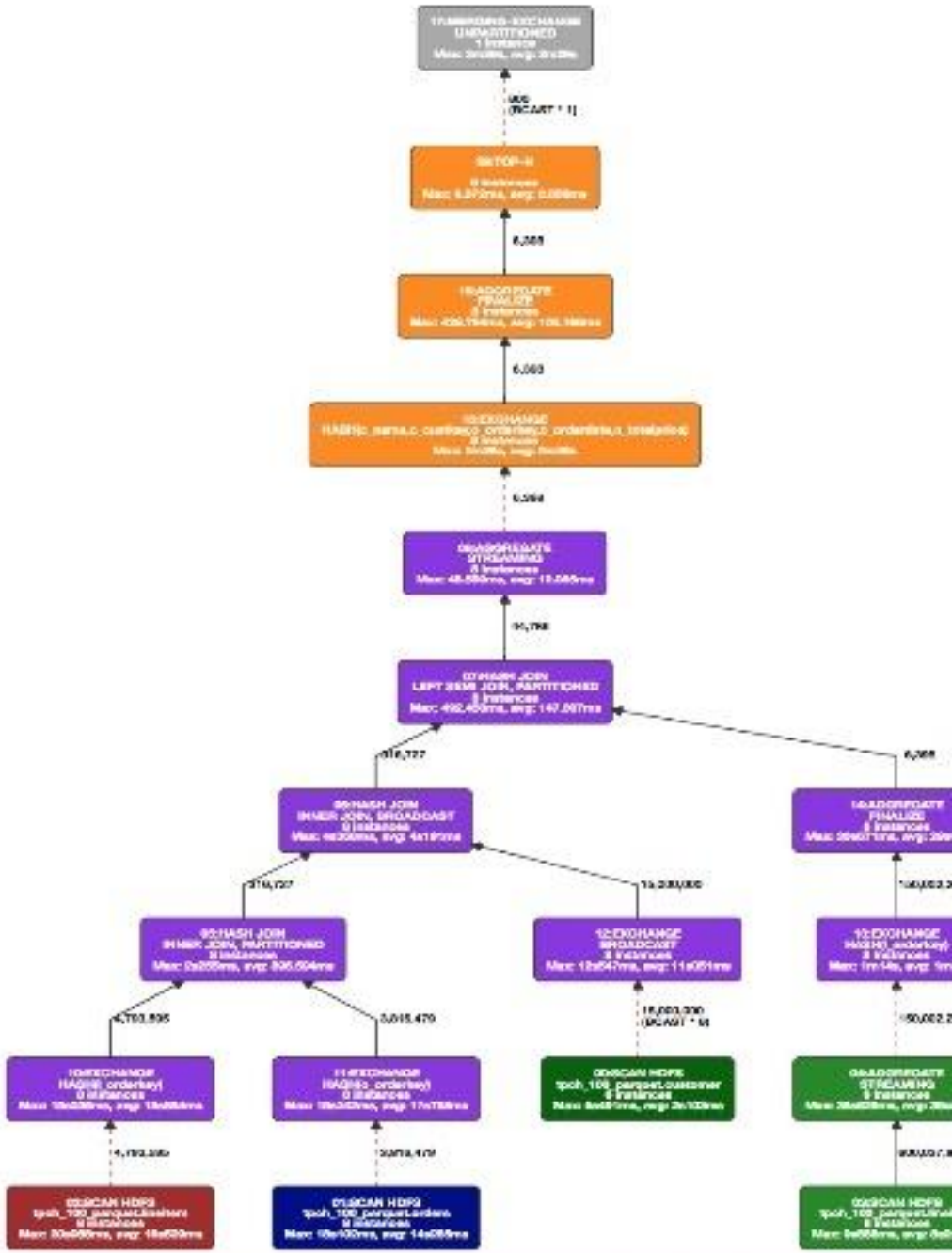




execution model subject to change

Query Plan Visualization

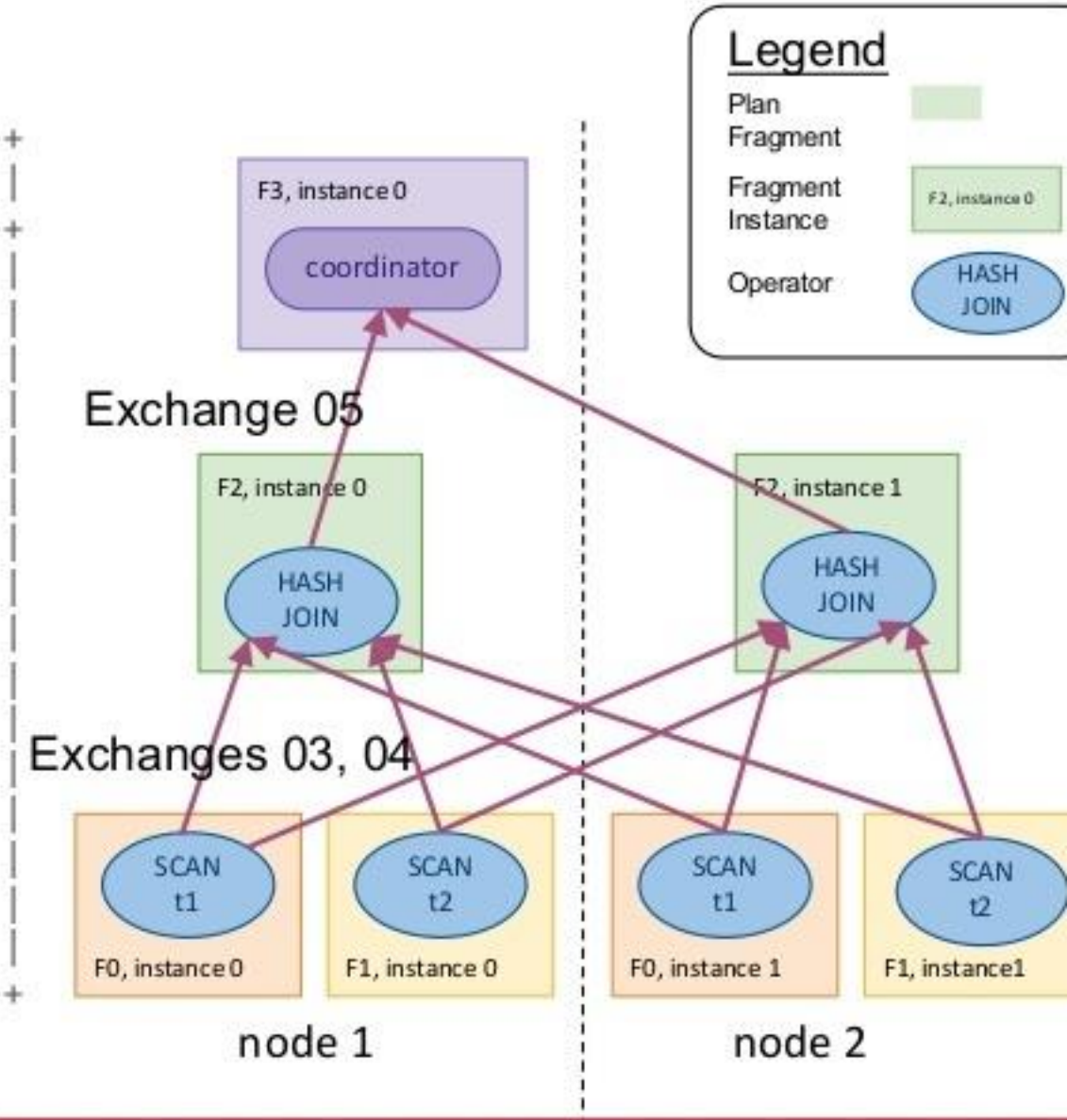
```
select c_name, c_custkey, o_orderkey, o_orderdate,
        o_totalprice, sum(l_quantity)
from customer, orders, lineitem
where o_orderkey in (
        select l_orderkey
        from lineitem
        group by l_orderkey
        having sum(l_quantity) > 300 )
and c_custkey = o_custkey
and o_orderkey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

```
+------------------------------------------------------------------------------+
| Explain String                                                               |
+------------------------------------------------------------------------------+
| Max Per-Host Resource Reservation: Memory=0B                                 |
| Per-Host Resource Estimates: Memory=52.00MB                                  |
| Codegen disabled by planner                                                  |
|                                                                              |
| F01:PLAN FRAGMENT [UNPARTITIONED] hosts=1 instances=1                        |
| |  Per-Host Resources: mem-estimate=10.00MB mem-reservation=0B               |
| PLAN-ROOT SINK                                                               |
| |  mem-estimate=0B mem-reservation=0B                                        |
| |                                                                            |
| 03:AGGREGATE [FINALIZE]                                                      |
| |  output: avg:merge(salary)                                                 |
| |  mem-estimate=10.00MB mem-reservation=0B spill-buffer=2.00MB               |
| |  tuple-ids=2 row-size=8B cardinality=1                                     |
| |                                                                            |
| 02:EXCHANGE [UNPARTITIONED]                                                  |
| |  mem-estimate=0B mem-reservation=0B                                        |
| |  tuple-ids=1 row-size=8B cardinality=1                                     |
| |                                                                            |
| F00:PLAN FRAGMENT [RANDOM] hosts=1 instances=1                               |
| Per-Host Resources: mem-estimate=42.00MB mem-reservation=0B                  |
| 01:AGGREGATE                                                                 |
| |  output: avg(salary)                                                       |
| |  mem-estimate=10.00MB mem-reservation=0B spill-buffer=2.00MB               |
| |  tuple-ids=1 row-size=8B cardinality=1                                     |
| |                                                                            |
```

explain_level = 3

Query Summary
- |- Basic info: state, type, user, statement, coordinat
- |- 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
    Start Time: 2018-02-09 13:17:31.162274000
    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

```
  Query Timeline: 1s414ms
     - Start execution: 49.340us (49.340us)
     - Planning finished: 59.532ms (59.483ms)
     - Rows available: 987.346ms (927.813ms)
     - First row fetched: 1s019ms (32.521ms)
     - Unregister query: 1s412ms (392.159ms)
ImpalaServer:
     - ClientFetchWaitTimer: 415.244ms
     - RowMaterializationTimer: 7.795ms
```

Total query time

Idle time: client isn't fetching

- 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.

Query Tuning Basics - Overview

- How to make our query faster & consume less resources.

1. Keep **up-to-date statistics** with COMPUTE STATS

2. Examine the **logic of the query** with the EXPLAIN PLAN
   - Identify the runtime filtering algorithm used and try to influence it through changes in our query

3. Query Profile to identify further bottlenecks and skews

Statistics

More on COMPUTE STATS

- COMPUTE STATS is very CPU-intensive, but it has improved since Impala 1.4 (as of 2021, Impala‘s most recent version is 3.4)
- ~40M cells per second per node + HMS update time
- Total number of cells of a table = num_rows * num_columns
- Need to recompute when data changes on 30% or more
- Needs to be computed directly on tables, not on views

Incremental Stats Maintenance

- Since COMPUTE STATS is expensive, you might want to do some of the work yourself sometimes
- Column and table stats can be set manually via an ALTER TABLE statement
- COMPUTE INCREMENTAL STATS can also help (available from Impala 2.1)

Incremental Stats Maintenance

- COMPUTE INCREMENTAL STATS will check for partitions where there are data changes, and only recalculate stats on those
- If you switch to COMPUTE STATS after doing COMPUTE INCREMENTAL STATS in a table, all the incremental stats are discarded (similar on the other direction)
- However, it is only reccommended to use COMPUTE INCREMENTAL STATS if the following conditions are met:
  - Cluster has less than 50 nodes
  - On each table, num_columns*num_partitions < 500,000

Performance for JOIN queries

Query Logic

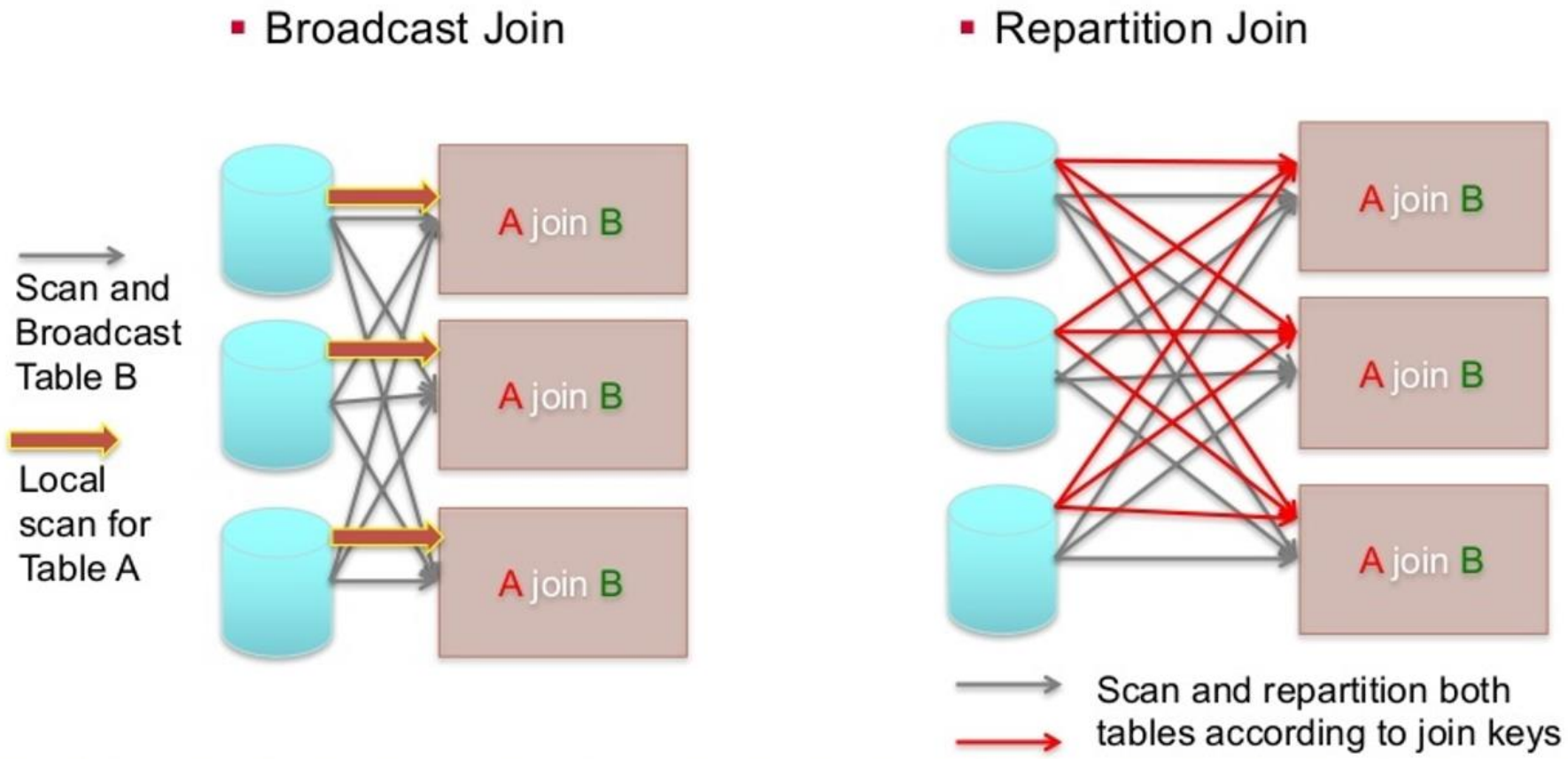
- Sometimes the query can have elements that are better to remove
  - redundant JOIN clauses, DISTINCT, GROUP BY, ORDER BY
- It is better to join the tables in advance, if the JOIN appears often
- Without statistics, the JOIN strategy can go wrong, especially when dealing with layers of views.

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]
```

JOIN types


Broadcast Join
Scan and Broadcast Table B
Local scan for Table A
A join B
A join B
A join B
Repartition Join
A join B
A join B
A join B
Scan and repartition both tables according to join keys

JOIN strategy 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]

Join strategy cost	Network Traffic	Memory Usage (HashTable)
Broadcast Join	RHS table size[1] x number of node	RHS table size[1] x number of node
Partitioned Join	LHS + RHS table size[1]	RHS table size[1]

[1] table size refers to the data flowed from the child node (i.e. only required column data after filtering counts).

Sure, but what if stats are not available?

- Impala allows you to override the automatic JOIN order optimization by specifying the STRAIGHT_JOIN keyworkd immediately after SELECT

- This comes at a cost: you must order the tables by yourself instead of relying in the optimizer.

- Heuristic:
  - Largest table first (since it is read from disk)
  - Join with the smallest table

How joins are processed without stats

- If table or column statistics are not available, Impala will reorder the tables using the available information.

- Tables with statistics will be on the left side of the join order, and then in descending order of cost based on total size and cardinality.

- **Tables without statistics are treated as zero-size** which means that they are placed always at the right.

Overriding Join reordering

```
SELECT STRAIGHT_JOIN cols
FROM table
JOIN another_table
WHERE
table.id=another_table.id
```

```
SELECT DISTINCT STRAIGHT_JOIN cols
FROM table
JOIN another_table
WHERE table.id=another_table.id
```

Optimizer hints

- Helps fine-tuning of inner working of queries, where missing stats cause inefficient performance

- They are most often used for resource-intensive queries such as
  - JOIN
  - Inserting into partitioned Parquet tables

Hints for JOIN queries

- /* +BROADCAST */ and /* +SHUFFLE */ control the execution of the join queries

- /* +SHUFFLE */ forces a partitioned join. When there are no statistics, the default join mechanism is broadcast. Partitioned joins are helpful when joining tables of similar size without (updated) statistics

- /* +BROADCAST */ sends the right table across the nodes involved in the JOIN. This is the default mode, it is typically only needed if Impala has stale metadata.

Examples

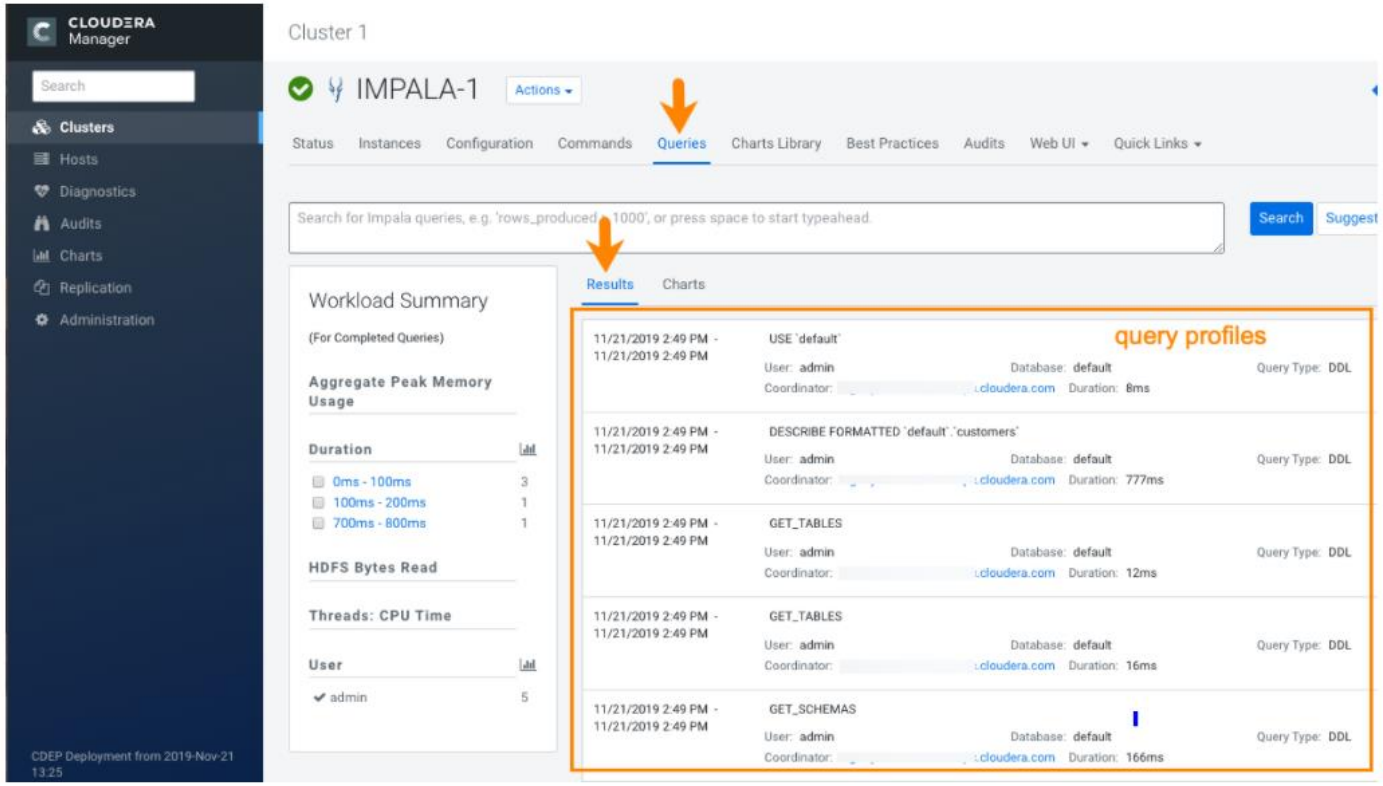
```
SELECT STRAIGHT_JOIN customer.address, state_lookup.state_name
FROM customer JOIN /* +BROADCAST */ state_lookup
ON customer.state_id = state_lookup.state_id;
```

```
SELECT STRAIGHT_JOIN t1.name, t2.id, t3.price
FROM t1.join /* +SHUFFLE */ t2
JOIN /* +BROADCAST */ t3
ON t1.id = t2.id AND t2.id=t3.id;
```

Query Profiles

Runtime info

- Runtime details are captured by the Query Profile
- It is available in impala-shell with PROFILE; statement



Execution Summary

- This shows all the operators in the query, and the resources they used

Operator	#Hosts	Avg Time	Max Time	#Rows	Est. #Rows	Peak Mem	Est. Peak Mem	Detail
10:SORT	23	8m15s	1h	290.01M	44.89M	58.29 GB	436.00 MB	
05:HASH JOIN	23	1s733ms	9s700ms	290.01M	44.89M	20.81 MB	1.94 MB	LEFT SEMI JOIN, BROADCAST
|--09:EXCHANGE	23	151.541us	664.160us	13	97	112.00 KB	0	BROADCAST
| 08:AGGREGATE	23	2.731ms	11.367ms	13	97	1.97 MB	10.00 MB	FINALIZE
| 07:EXCHANGE	23	319.697us	1.677ms	4.04K	968	192.00 KB	0	HASH(vpmn)
| 03:AGGREGATE	23	8s782ms	1m11s	4.04K	968	14.63 MB	10.00 MB	STREAMING
| 02:SCAN HDFS	23	10s745ms	3m41s	438.93M	447.94M	1.59 GB	4.81 GB	test.ext_call_event_fact cef
04:HASH JOIN	23	15s968ms	3m20s	438.93M	447.94M	13.18 MB	1.94 MB	INNER JOIN, BROADCAST
|--06:EXCHANGE	23	339.291us	3.671ms	8.04K	8.04K	544.00 KB	0	BROADCAST
| 01:SCAN HDFS	1	4.194ms	4.194ms	8.04K	8.04K	1.25 MB	32.00 MB	test.date_dim dd
00:SCAN HDFS	23	541.093ms	3s665ms	438.93M	447.94M	2.08 GB	4.81 GB	test.ext_call_event_fact cef

Query Timeline

```
  Query Compilation: 1m
     - Metadata load started: 351.060us (351.060us)
     - Metadata load finished. loaded-tables=1/1 load-requests=2 catalog-updates=37: 1m (1m)
     - Analysis finished: 1m (1.596ms)
     - Value transfer graph computed: 1m (18.009us)
     - Single node plan created: 1m (190.919us)
     - Distributed plan created: 1m (70.456us)
     - Lineage info computed: 1m (38.695us)
     - Planning finished: 1m (329.095us)
  Query Timeline: 1m2s
     - Query submitted: 120.423us (120.423us)
     - Planning finished: 1m (1m)
     - Submit for admission: 1m (1.758ms)
     - Completed admission: 1m (137.653us)
     - Ready to start on 1 backends: 1m (438.325us)
     - All 1 execution backends (1 fragment instances) started: 1m (2.886ms)
     - Rows available: 1m (5.431ms)
     - First row fetched: 1m1s (913.202ms)
     - Last row fetched: 1m1s (347.012ms)
     - Released admission control resources: 1m1s (99.751us)
     - Unregister query: 1m2s (560.234ms)
   - ComputeScanRangeAssignmentTimer: 1.063ms
ImpalaServer:
   - ClientFetchWaitTimer: 1s819ms
   - RowMaterializationTimer: 1.133ms
```

Common Use Cases

Memory Limit Exceeded

- Typically this occurs because the memory limit is not set for the pool or table statistics are missing, or both.

- **Solution:**
  - Run COMPUTE STATS for each table involved in the query.
  - Rerun the query with a memory limit set using the SET MEM_LIMIT query option. For example:
  - SET MEM_LIMIT=3gb;

Query runs slowly

- Missing Load
  - Impala may not have the metadata of the table cached in catalogd
- Missing statistics
  - Calculate statistics or do optimization by hand as in the previous section

Admission Control

- Impala Admission Control controls the number of concurrent queries.
- If the difference between the query being queued and when the admission is completed is high, you might want to revisit the resources allocated to that resource pool

```
Queued: 127ms (127000586)
Completed admission: 3.50s (3498016148)
```

Issues with JOINs

- 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

Time and data skews

- Data Skews can be treated by the HDFS balancer, and then running INVALIDATE METADATA
  - HDFS Balancer is accessible through Cloudera Manager.
  - It is recommended that there is no more than 10% difference in disk usage across the cluster.
  - For instance, if average disk usage is 40% then disk usage should be between 30% and 50% in the data nodes.
- Workload Skews (time) can be treated by ensuring similar CPU and memory configurations apply to all Impala daemons

Time and data skews (cont.)

- 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 M
08:EXCHANGE	1	113.775us	113.775us	31	227	0	-1.00
07:AGGREGATE	81	359.985ms	436.922ms	31	227	3.44 MB	10.00
06:EXCHANGE	81	57.25us	515.364us	397	227	0	
03:AGGREGATE	81	561.26ms	1s344ms	397	227	3.66 MB	10.00
02:HASH JOIN	81	3s730ms	18s695ms	184.02M	2.08M	3.04 MB	13.64
|--05:EXCHANGE	81	27.471us	42.597us	26.74K	25.40K	0	
| 01:SCAN HDFS	1	359.799ms	359.799ms	26.74K	25.40K	4.47 MB	16.00
04:EXCHANGE	81	130.853ms	1s608ms	184.03M	2.08M	0	
00:SCAN HDFS	81	154.864ms	553.824ms	184.03M	2.08M	11.46 MB	88.00

Improving Scan Node performance

- Check how much data is read, always do as little disk read as possible
- Only SELECT necessary columns
- Avoid, whenever possible, costly predicates like STRING or REGEX

Expression Evaluation

- Expressions are evaluated lazily (only when the value is needed)

- Subqueries/inline views are dealt with by substituting the output of the inline view in the parent query

```
SELECT CONCAT(col1, col1)
FROM (SELECT REGEXP_EXTRACT(col, `..`) as col1 ) FROM TBL x
```

This will evaluate the regex predicate 3 times. You can avoid it by materializing the value (for example, adding an ORDER BY col1) in the end.

References

- Impala Cookbook
- Juan Yu‘s talk on Performance Issues at Strata Conference
  - Part 1
  - Part 2