COMMUNITY
THE ASF CONFERENCE
CODE

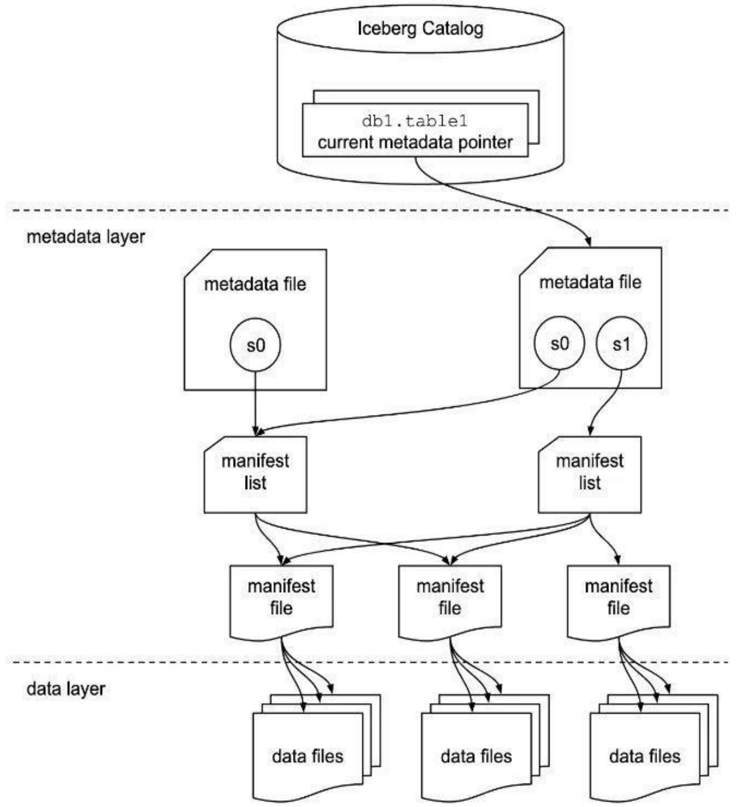
Impalas living on Iceberg

Gabor Kaszab, Impala PMC member

Contents

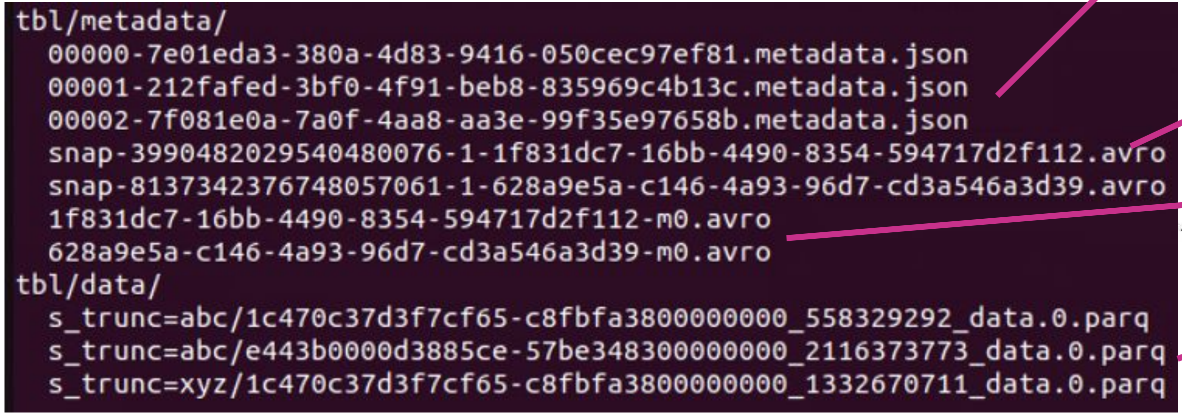
1. Introduction - Iceberg

- Popular **table format**
- Defines how to:
    - Organize table data and metadata
    - Interact with meta/data -> Spec
- Table metadata on storage
- Famous features:
    - Flexible partitioning (transforms)
    - Partition/schema evolution
    - Time travel
    - Branching and tagging
    - Row-level modifications
- **Library/API**
    - Clients can interact with tables
- **Catalogs**
    - HMS, Glue, JDBC Rest (Polaris)

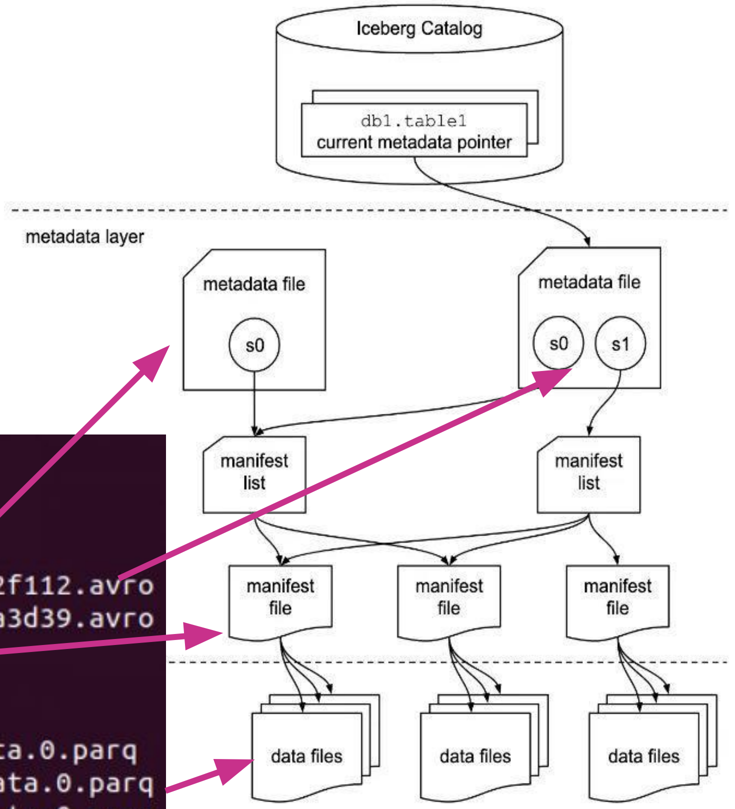


1. Introduction - Iceberg

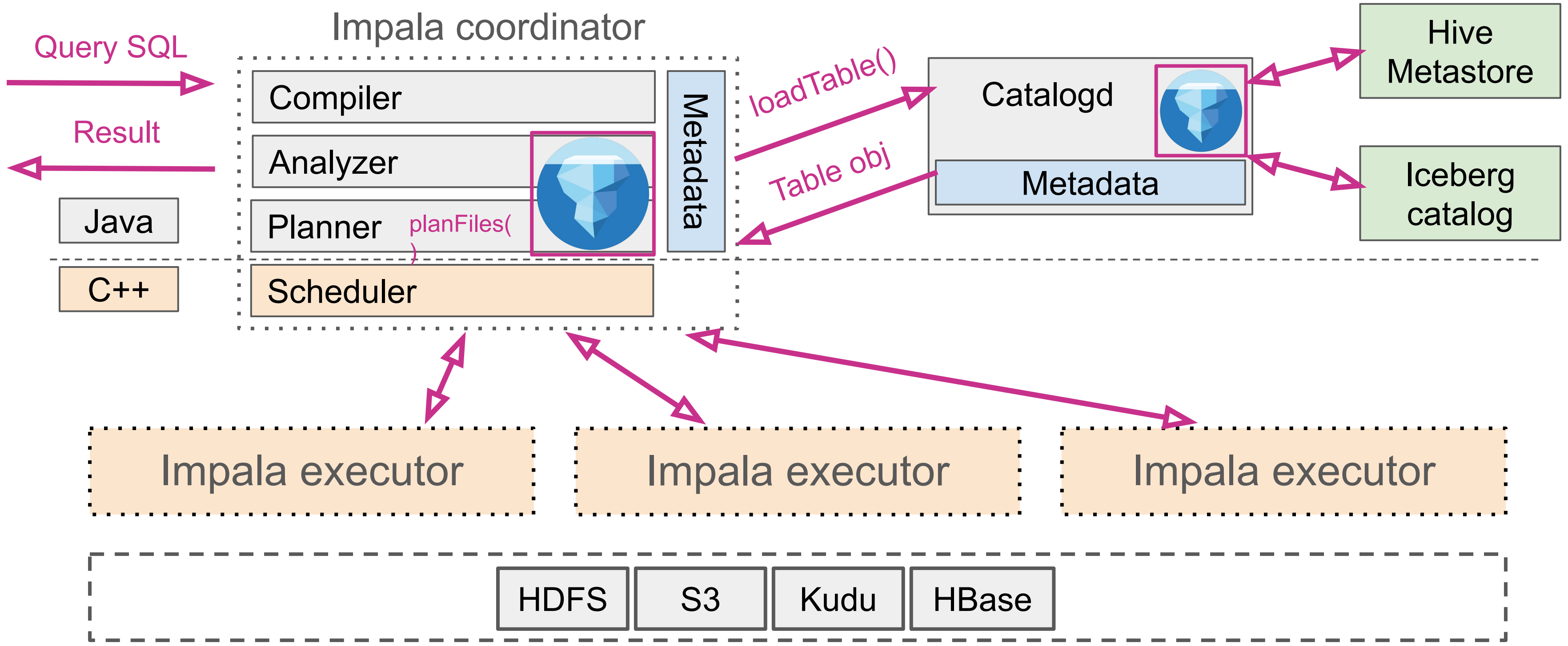
- CREATE TABLE tbl (i int, s string)
  PARTITIONED BY SPEC (truncate(3, s))
  STORED AS ICEBERG
  TBLPROPERTIES ('format-version'='2');
- INSERT INTO tbl VALUES (1, "**abc**d"), (2, "**xyz**1");
- INSERT INTO tbl VALUES (3, "**abc**xyz");



```
tbl/metadata/
  00000-7e01eda3-380a-4d83-9416-050cec97ef81.metadata.json
  00001-212fafed-3bf0-4f91-beb8-835969c4b13c.metadata.json
  00002-7f081e0a-7a0f-4aa8-aa3e-99f35e97658b.metadata.json
  snap-3990482029540480076-1-1f831dc7-16bb-4490-8354-594717d2f112.avro
  snap-8137342376748057061-1-628a9e5a-c146-4a93-96d7-cd3a546a3d39.avro
  1f831dc7-16bb-4490-8354-594717d2f112-m0.avro
  628a9e5a-c146-4a93-96d7-cd3a546a3d39-m0.avro
tbl/data/
  s_trunc=abc/1c470c37d3f7cf65-c8fbfa3800000000_558329292_data.0.parq
  s_trunc=abc/e443b0000d3885ce-57be348300000000_2116373773_data.0.parq
  s_trunc=xyz/1c470c37d3f7cf65-c8fbfa3800000000_1332670711_data.0.parq
```



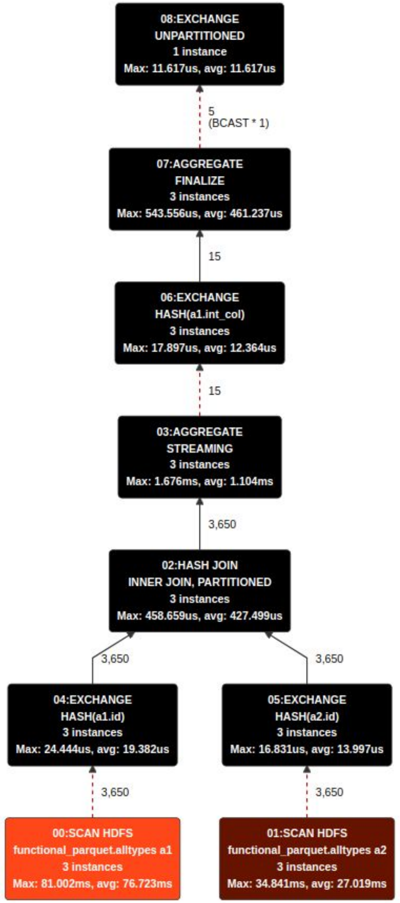
1. Introduction - Impala



1. Introduction - Impala

- Query plan example

```
SELECT count(1), avg(a1.int_col)
FROM
    functional_parquet.alltypes a1,
    functional_parquet.alltypes a2
WHERE
    a1.id = a2.id AND
    a1.id % 2 = 0
GROUP BY a1.int_col;
```



Contents

2. Row-level deletes - Concepts

```
DELETE FROM tbl WHERE id = 15;
```

Merge-on-read

- Tracking deleted rows in a separate "delete file"
- Good for frequent, small modifications
- Low write amplification
- High read amplification
- Table maintenance is a MUST

Copy-on-write

- Replaces old data files with rewritten data files
- Useful for infrequent, large modifications
- High write amplification
- No read amplification

2. Row-level deletes - Concepts

DELETE FROM tbl WHERE id = 15;

Positional deletes

- File_path + position
- Slower writes
- Better perf. to read

‘path1/abc.parquet’	13
‘path1/abc.parquet’	1234
‘path2/xyz.parquet’	1

Equality deletes

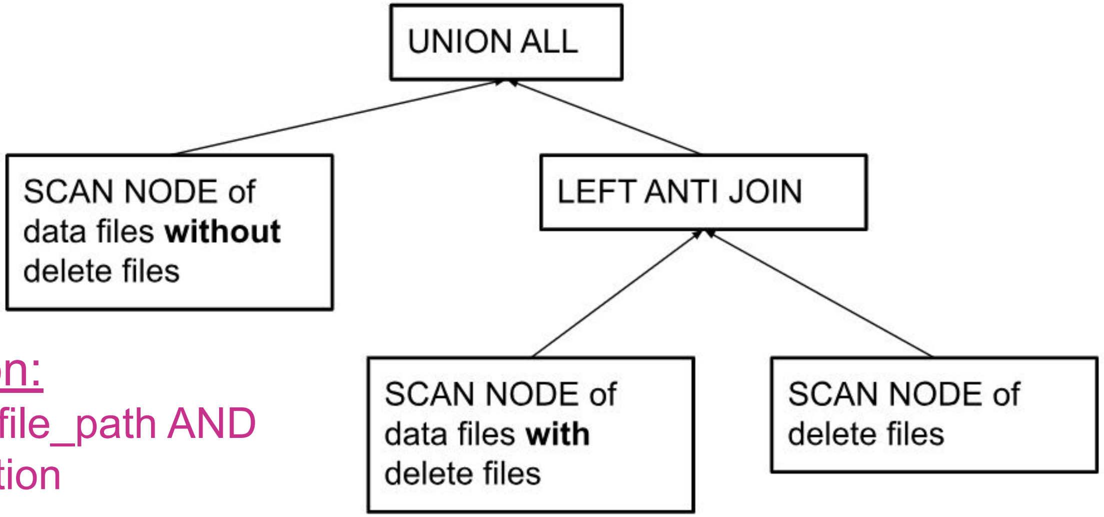
- Schema depends on ‘identifier-field-ids’
- Cheap to write
- Inefficient to read

ID
15

ID_col1	ID_col2	ID_col3
42	“string”	07.10.2024

2. Row-level deletes - Implementation

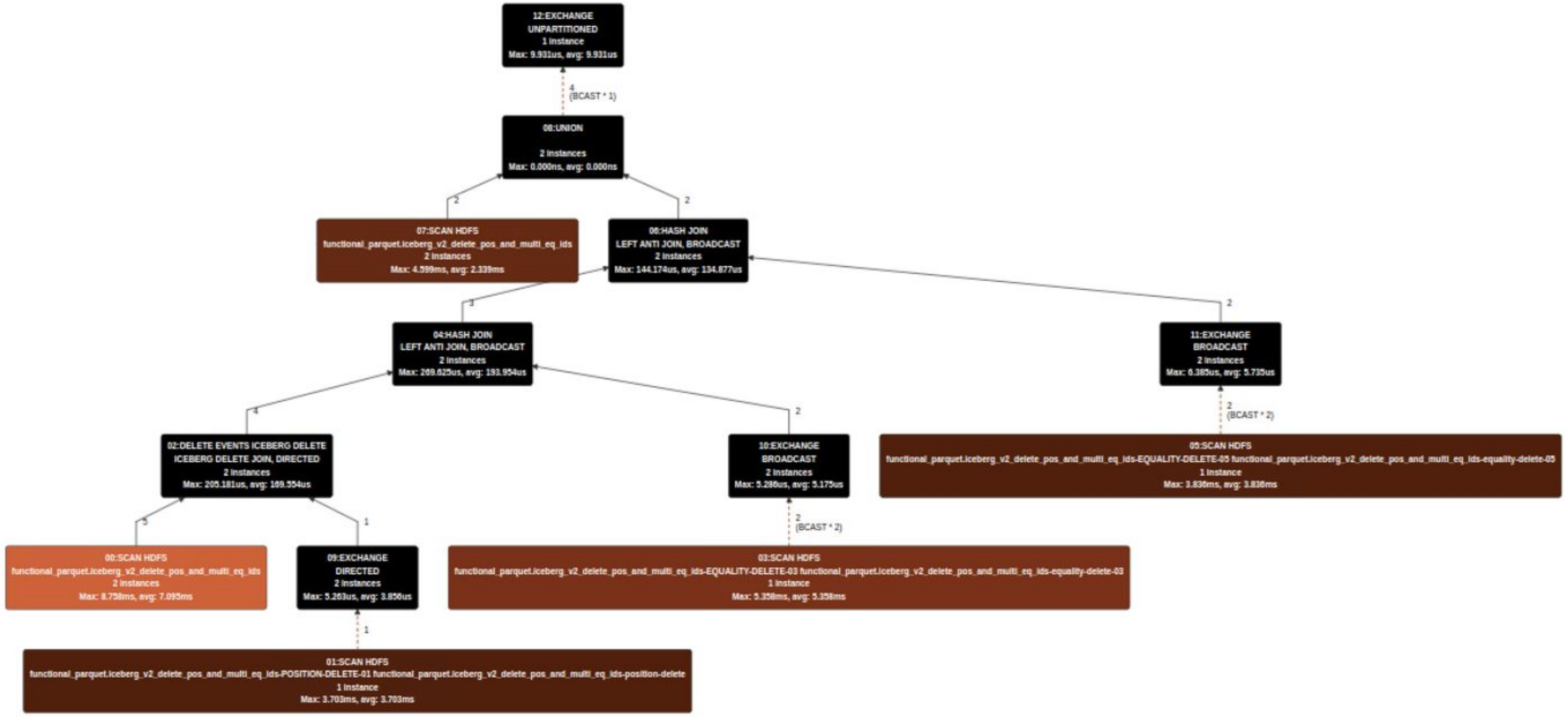
- Read data files w/o deletes
- Read delete files
- Read data file w/ deletes but add **virtual columns**


UNION ALL
SCAN NODE of data files **without** delete files
LEFT ANTI JOIN
SCAN NODE of data files **with** delete files
SCAN NODE of delete files

Position delete Anti-Join condition:
tbl.**INPUT__FILE__PATH** = DEL-tbl.file_path AND
tbl.**FILE__POSITION** = DEL-tbl.position

Equality delete Anti-Join condition:
tbl.ID_col1 IS NOT DISTINCT FROM DEL-tbl.ID_col1 AND … AND
tbl.DATA__SEQUENCE__NUMBER < DEL-tbl.DATA__SEQUENCE__NUMBER

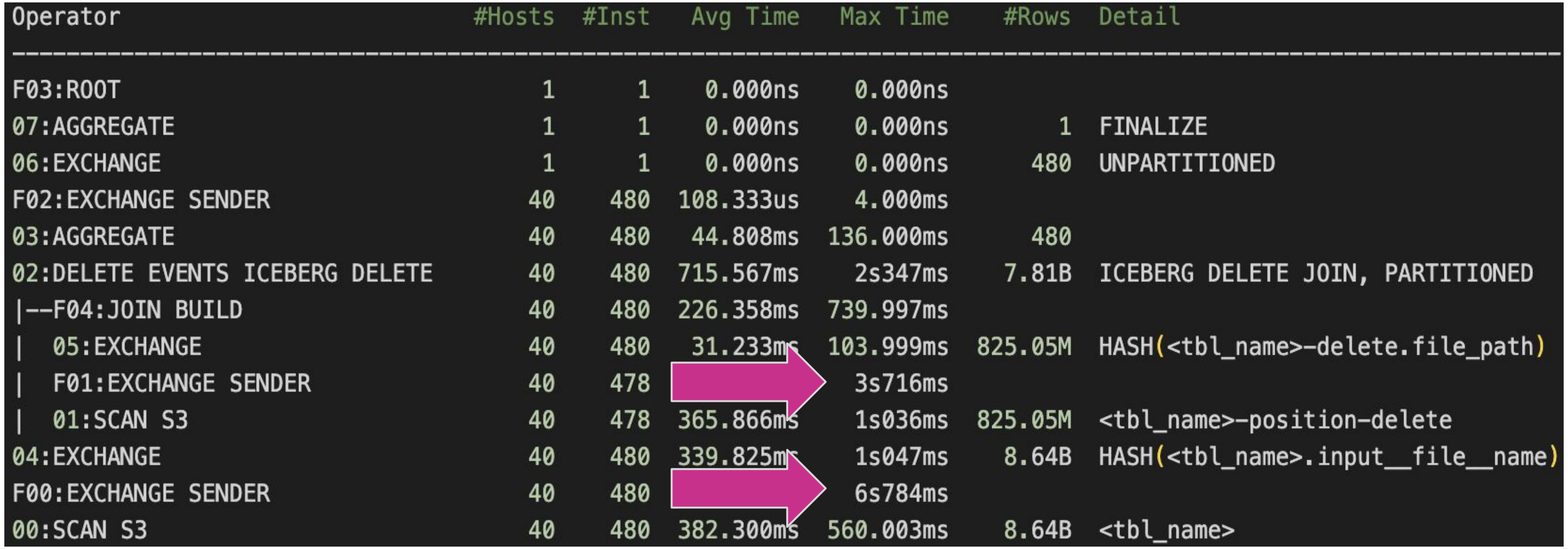
2. Row-level deletes - Implementation


12:EXCHANGE
UNPARTITIONED
1 instance
Max: 9.931us, avg: 9.931us
4
(BCAST * 1)
08:UNION
2 instances
Max: 0.000ns, avg: 0.000ns
2
2
07:SCAN HDFS
functional_parquet.iceberg_v2_delete_pos_and_multi_eq_ids
2 instances
Max: 4.599ms, avg: 2.339ms
06:HASH JOIN
LEFT ANTI JOIN, BROADCAST
2 instances
Max: 144.174us, avg: 134.877us
3
2
04:HASH JOIN
LEFT ANTI JOIN, BROADCAST
2 instances
Max: 209.625us, avg: 193.954us
11:EXCHANGE
BROADCAST
2 instances
Max: 6.385us, avg: 5.735us
4
2
2
(BCAST * 2)
02:DELETE EVENTS ICEBERG DELETE
ICEBERG DELETE JOIN, DIRECTED
2 instances
Max: 205.181us, avg: 169.554us
10:EXCHANGE
BROADCAST
2 instances
Max: 5.286us, avg: 5.175us
05:SCAN HDFS
functional_parquet.iceberg_v2_delete_pos_and_multi_eq_ids-EQUALITY-DELETE-05 functional_parquet.iceberg_v2_delete_pos_and_multi_eq_ids-equality-delete-05
1 instance
Max: 3.836ms, avg: 3.836ms
5
1
2
(BCAST * 2)
00:SCAN HDFS
functional_parquet.iceberg_v2_delete_pos_and_multi_eq_ids
2 instances
Max: 8.758ms, avg: 7.095ms
09:EXCHANGE
DIRECTED
2 instances
Max: 5.263us, avg: 3.856us
03:SCAN HDFS
functional_parquet.iceberg_v2_delete_pos_and_multi_eq_ids-EQUALITY-DELETE-03 functional_parquet.iceberg_v2_delete_pos_and_multi_eq_ids-equality-delete-03
1 instance
Max: 5.358ms, avg: 5.358ms
1
01:SCAN HDFS
functional_parquet.iceberg_v2_delete_pos_and_multi_eq_ids-POSITION-DELETE-01 functional_parquet.iceberg_v2_delete_pos_and_multi_eq_ids-position-delete
1 instance
Max: 3.703ms, avg: 3.703ms

2. Row-level deletes - Pos delete read performance - 1

Details for performance measurement:

- Table with 8.64bn rows. ~10% deleted by position delete files.
- Query: SELECT count(1) ran approx **21 sec**

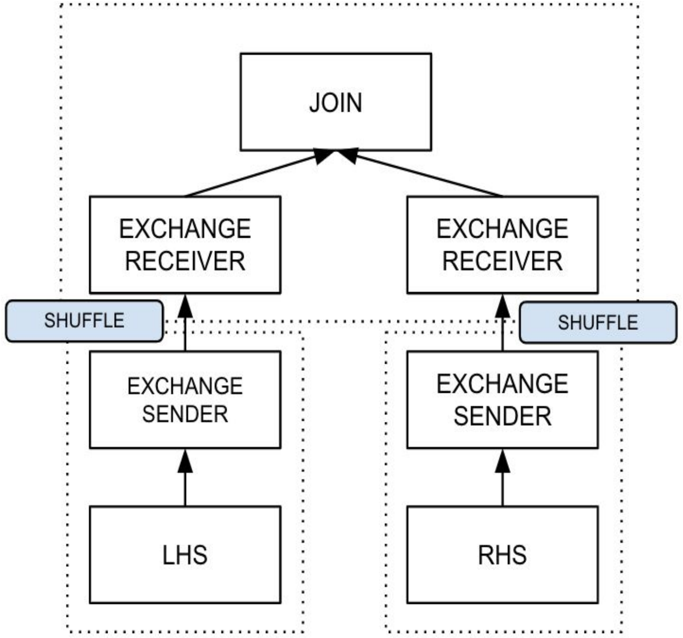


```
Operator                          #Hosts  #Inst   Avg Time   Max Time     #Rows  Detail
-----------------------------------------------------------------------------------------------------------------
F03:ROOT                               1      1    0.000ns    0.000ns
07:AGGREGATE                           1      1    0.000ns    0.000ns         1  FINALIZE
06:EXCHANGE                            1      1    0.000ns    0.000ns       480  UNPARTITIONED
F02:EXCHANGE SENDER                   40    480  108.333us    4.000ms
03:AGGREGATE                          40    480   44.808ms  136.000ms       480
02:DELETE EVENTS ICEBERG DELETE       40    480  715.567ms    2s347ms     7.81B  ICEBERG DELETE JOIN, PARTITIONED
|--F04:JOIN BUILD                     40    480  226.358ms  739.997ms
|  05:EXCHANGE                        40    480   31.233ms  103.999ms   825.05M  HASH(<tbl_name>-delete.file_path)
|  F01:EXCHANGE SENDER                40    478               3s716ms
|  01:SCAN S3                         40    478  365.866ms    1s036ms   825.05M  <tbl_name>-position-delete
04:EXCHANGE                           40    480  339.825ms    1s047ms     8.64B  HASH(<tbl_name>.input__file__name)
F00:EXCHANGE SENDER                   40    480               6s784ms
00:SCAN S3                            40    480  382.300ms  560.003ms     8.64B  <tbl_name>
```

2. Row-level deletes - Pos delete read performance - 1

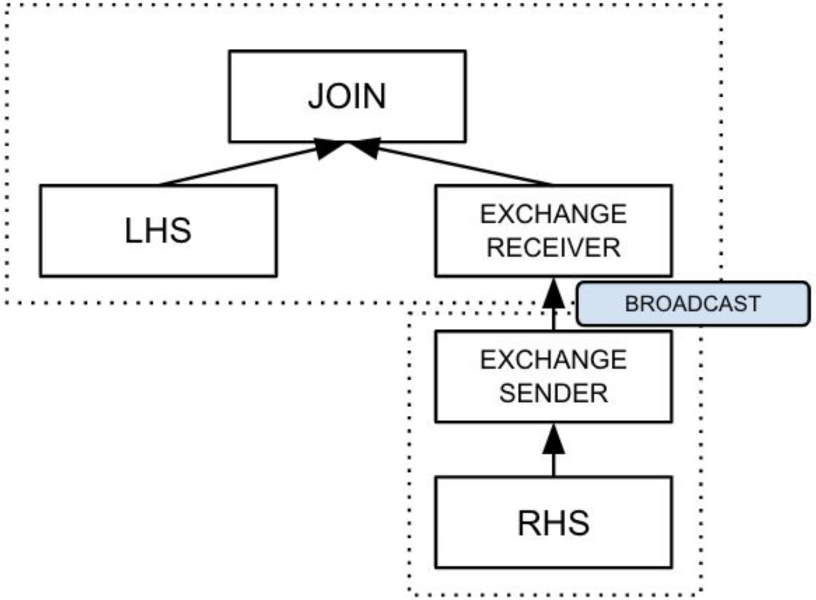
Partitioned

Cost:
#rows_left_scan + rows_right_scan



Broadcast

Cost: #rows_right_scan * #JOIN_nodes

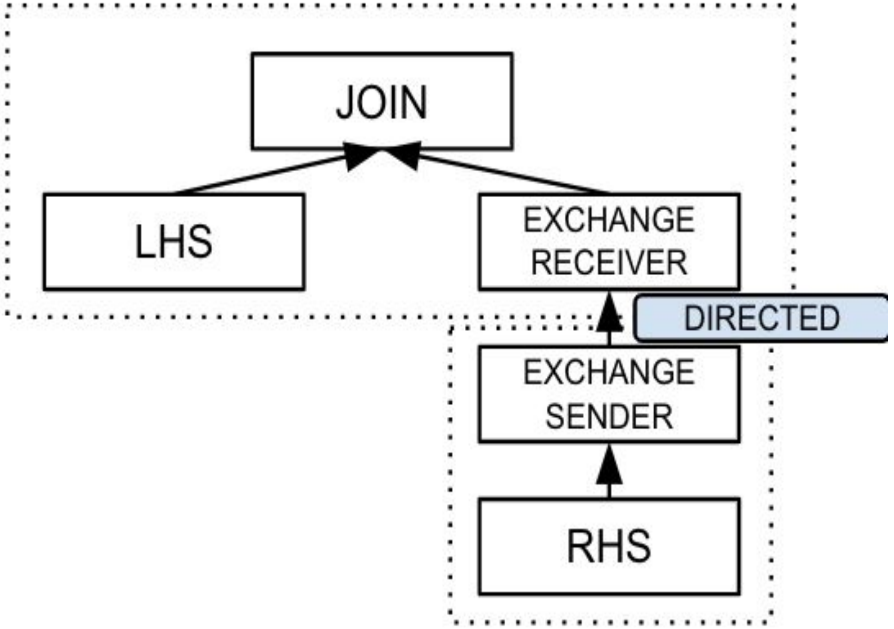


2. Row-level deletes - Pos delete read performance - 1

- **DIRECTED** distribution mode:
  Use 'file_path' in delete file to route rows
- No need to send 'left' rows
- No need to broadcast 'right' rows
- Cost: #rows_right_scan
- **Reduced** query runtime by **42%**

Directed

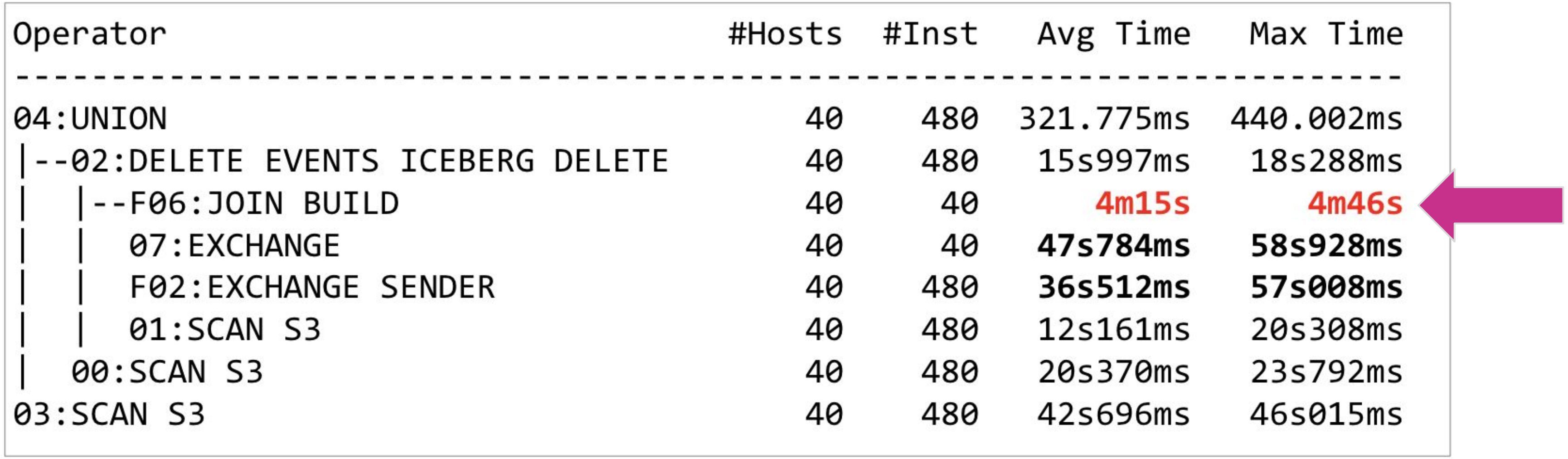
Cost: #rows_right_scan



2. Row-level deletes - Pos delete read performance - 2

Another measurement:

- Table with 1 trillion rows. ~68.5bn rows deleted by position delete files.
- Query: SELECT count(1) ran approx **7m15s**



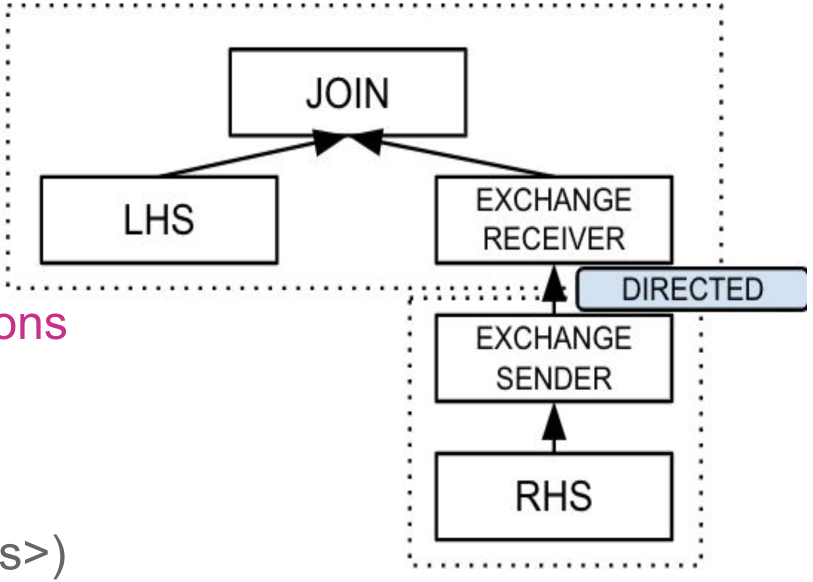
```
Operator                                #Hosts  #Inst    Avg Time    Max Time
-----------------------------------------------------------------------------
04:UNION                                    40    480   321.775ms   440.002ms
|--02:DELETE EVENTS ICEBERG DELETE          40    480    15s997ms    18s288ms
|  |--F06:JOIN BUILD                        40     40       4m15s       4m46s
|  |  07:EXCHANGE                           40     40    47s784ms    58s928ms
|  |  F02:EXCHANGE SENDER                   40    480    36s512ms    57s008ms
|  |  01:SCAN S3                            40    480    12s161ms    20s308ms
|  00:SCAN S3                               40    480    20s370ms    23s792ms
03:SCAN S3                                  40    480    42s696ms    46s015ms
```

2. Row-level deletes - Pos delete read performance - 2

JOIN BUILD: 4m46s

- We build hashmap:
  ("file path" -> vector<positions>)
- Many re-allocation when adding positions to the vector

- Instead build hashmap:
  ("file_path" -> RoaringBitmap<positions>)
- Join build from 4m46s -> 1m49s
- Query runtime: 7m15s -> 4m59s

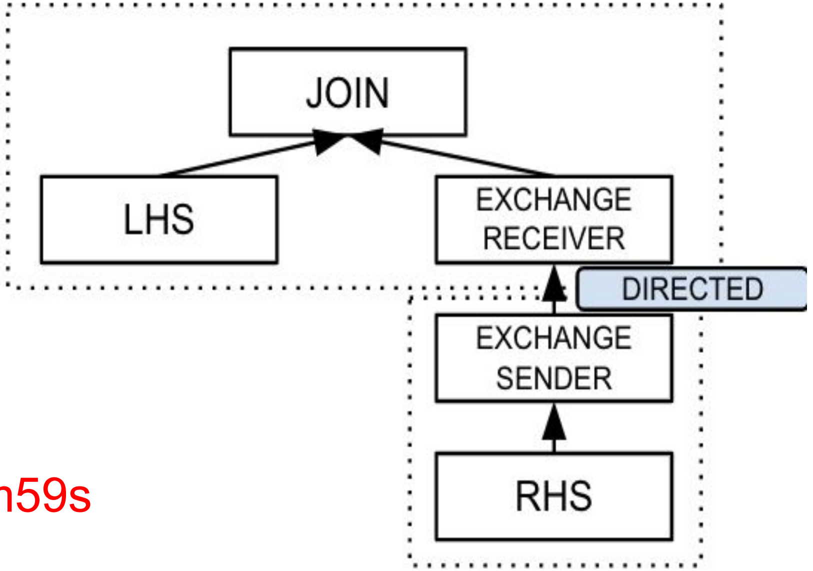


2. Row-level deletes - Pos delete read performance - 3

EXCHANGE SENDER: ~1m

- File paths being sent redundantly
- Pos dels ordered by file path.
- Remove redundancy

- Exchange sender from ~1m -> ~21.5s
- Query runtime: 7m15s -> 4m59s -> 3m59s



StringVal	int64	“long_file_path.parq”	StringVal	int64	“long_file_path.parq”	StringVal	int64	“long_file_path.parq”

“long_file_path”	StringVal	int64	StringVal	int64	StringVal	int64

Contents

3. Metadata table queries - Implementation

Iceberg API to query metadata tables:

data_files	**all_data_files**
delete_files	**all_delete_files**
entries	**all_entries**
files	**all_fi**
manifests	**all_m**

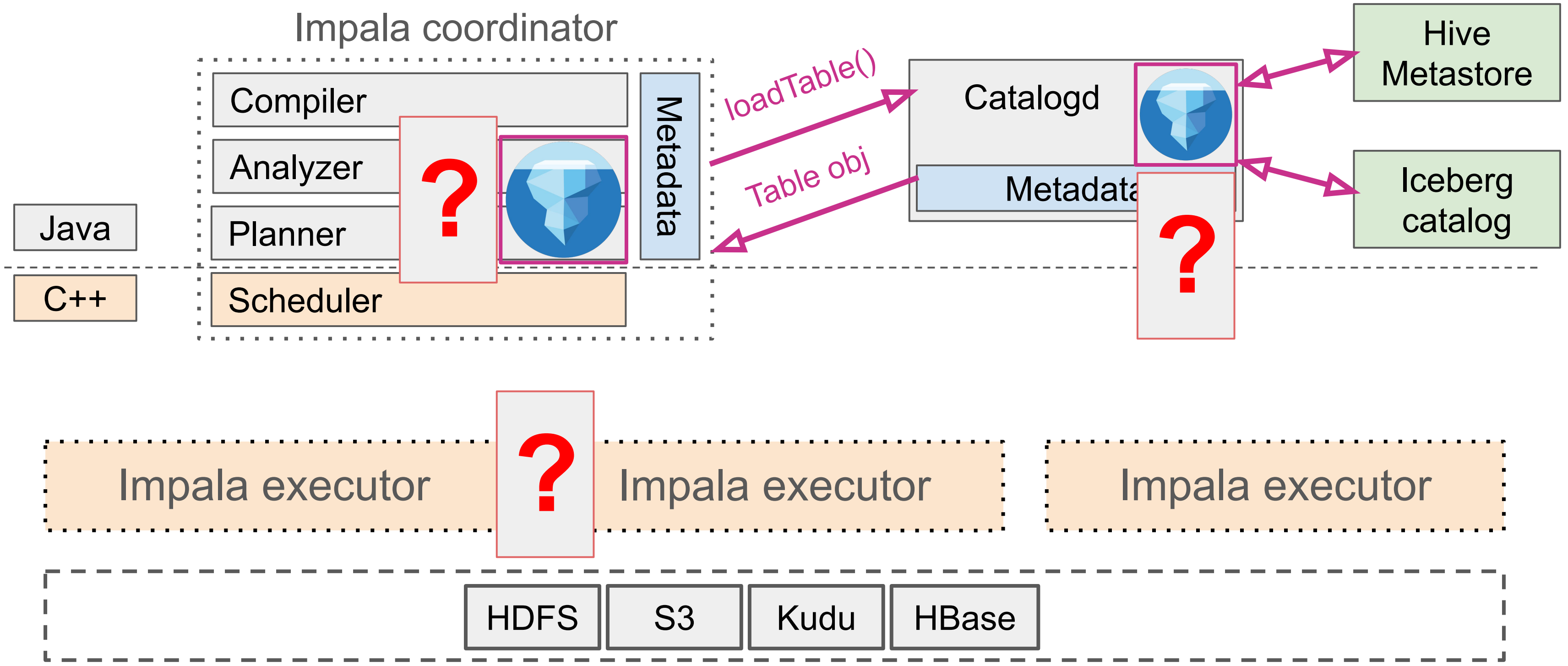
history
metadata_log_entrie
partitions
position_deletes
refs
snapshots

```
SELECT
    s.operation,
```

```
Table metaTbl = MetadataTableUtils.createMetadataTableInstance(
    tbl, /* An Iceberg table object */
    MetadataTableType.PARTITIONS);
for (FileScanTask task : metaTbl.newScan().planFiles()) {
  for (StructLike row : ((DataScan)task).rows()) {
    // Get fields from 'row'
  }
}
```

```
ry h
hots s
```

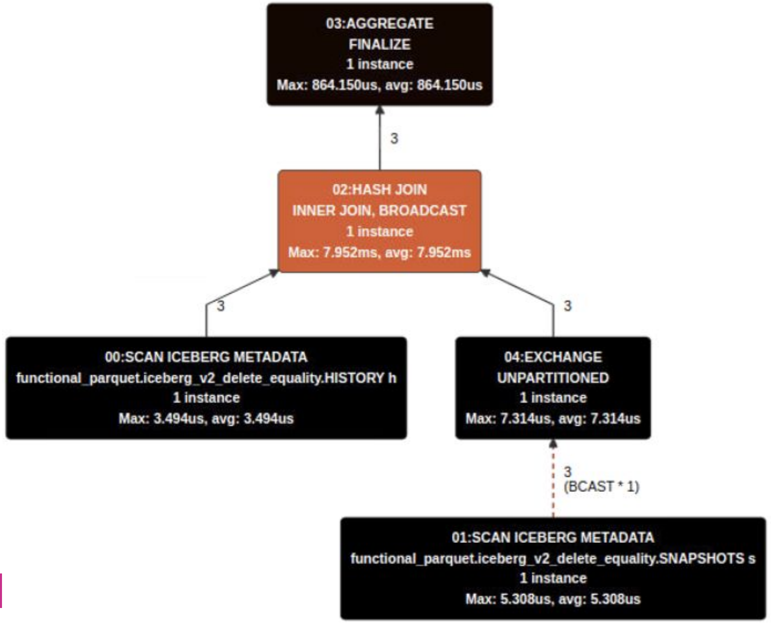
3. Metadata table queries - Implementation



3. Metadata table queries - Implementation

Thought process:

- Implement metadata SCAN in Executor?
  - SCAN would fit into Impala’s architecture (plan tree).
  - Would need C++ Iceberg API or implement reads for ourselves
- Answer metadata SCAN in coordinator?
  - It’s Java, simple to implement
  - Can’t do ‘regular query’ functionality like joining, aggregating, etc.
- Still, should do the SCAN as part of the pl tree


03:AGGREGATE
FINALIZE
1 instance
Max: 864.150us, avg: 864.150us
3
02:HASH JOIN
INNER JOIN, BROADCAST
1 instance
Max: 7.952ms, avg: 7.952ms
3
3
00:SCAN ICEBERG METADATA
functional_parquet.iceberg_v2_delete_equality.HISTORY h
1 instance
Max: 3.494us, avg: 3.494us
04:EXCHANGE
UNPARTITIONED
1 instance
Max: 7.314us, avg: 7.314us
3
(BCAST * 1)
01:SCAN ICEBERG METADATA
functional_parquet.iceberg_v2_delete_equality.SNAPSHOTS s
1 instance
Max: 5.308us, avg: 5.308us

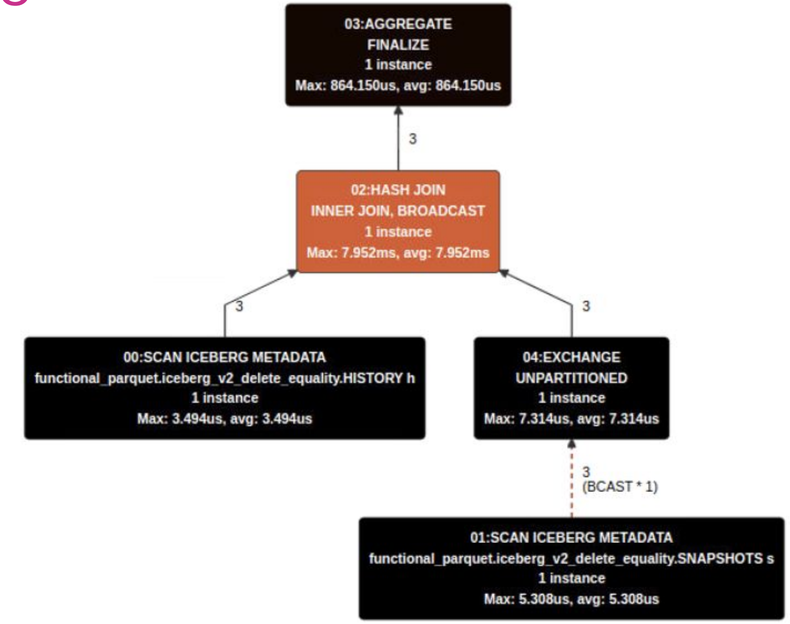
3. Metadata table queries - Implementation

Solution:

- SCAN ICEBERG METADATA node on C++ side
- Metadata Scanner on Java side
- JNI call from C++ to Java to get rows

Trade-offs and difficulties:

- Metadata SCANs are coordinator only
  - There is C++ and Java too
- Beware! GC vs access from C++
- Type conversion from Java to C++
- Extra steps to populate 'RowBatch'
- Code readability
- Performance?


03:AGGREGATE
FINALIZE
1 instance
Max: 864.150us, avg: 864.150us
3
02:HASH JOIN
INNER JOIN, BROADCAST
1 instance
Max: 7.952ms, avg: 7.952ms
3
3
00:SCAN ICEBERG METADATA
functional_parquet.iceberg_v2_delete_equality.HISTORY h
1 instance
Max: 3.494us, avg: 3.494us
04:EXCHANGE
UNPARTITIONED
1 instance
Max: 7.314us, avg: 7.314us
3
(BCAST * 1)
01:SCAN ICEBERG METADATA
functional_parquet.iceberg_v2_delete_equality.SNAPSHOTS s
1 instance
Max: 5.308us, avg: 5.308us

Contents

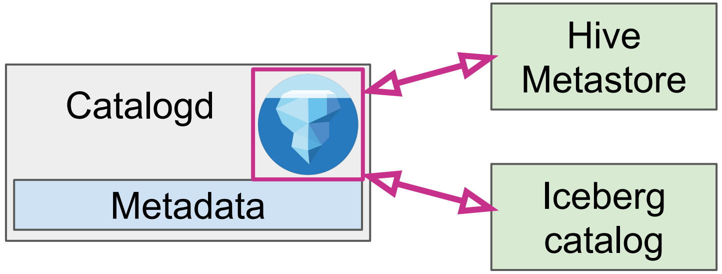
4. Catalogs - Currently

Iceberg catalogs:

- HiveCatalog, HadoopCatalog, JdbcCatalog, NessieCatalog, RestCatalog, GlueCatalog, SnowflakeCatalog, etc.

Catalogs supported by Impala:

- HiveCatalog and HadoopCatalog (non-prod)

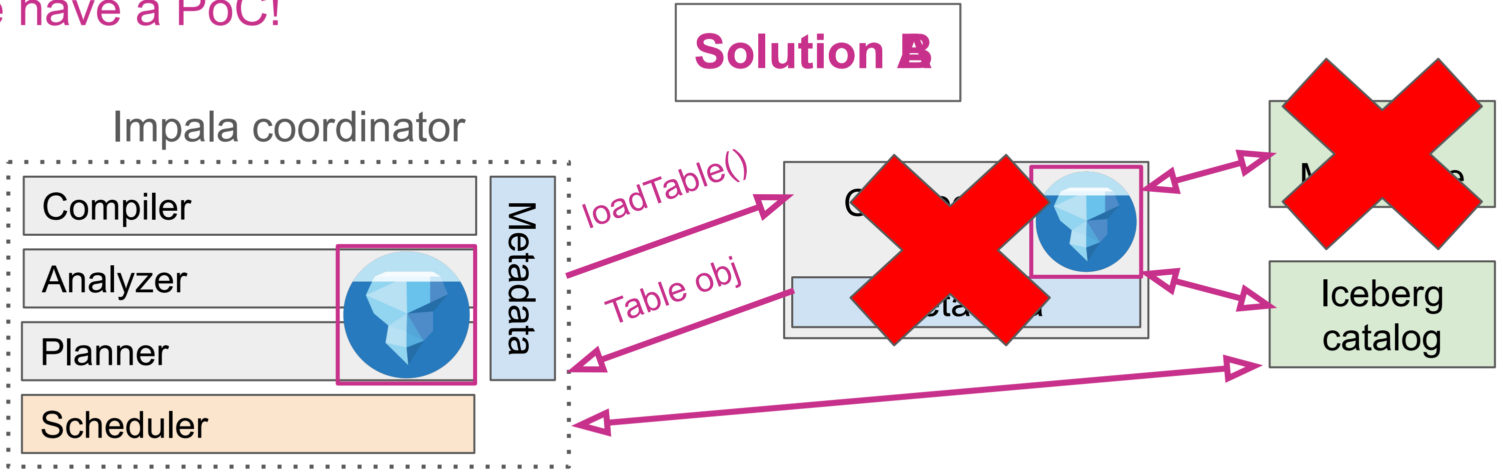


Other limitations:

- Heavy HMS dependency
- Full table name: DB.TBL instead of CATALOG.DB.TBL
- No flexibility for configuration

4. Catalogs - Future plans

- More catalog types: RESTCatalog!
- Catalog abstraction on top of DB.TBL
- More flexible creation + configuration
- Reduce HMS dependency
- We have a PoC!



Contents

+1 Iceberg V3 positional deletes

Proposal for new Positional delete design

V2 Positional deletes

- File_path + position
- Single delete file for multiple data files
- New deletes for writes

'path1/abc.parquet'	13
'path1/abc.parquet'	1234
'path2/xyz.parquet'	1

V3 Positional deletes

- Delete vector as a RoaringBitmap
- One delete vector for one data file
- Multiple bitmaps in a Puffin file
- File path + offset + length stored in Iceberg metadata
- Merge bitmaps for writes

[Puffin header][bitmap1]...[bitmapN][Puffin footer]

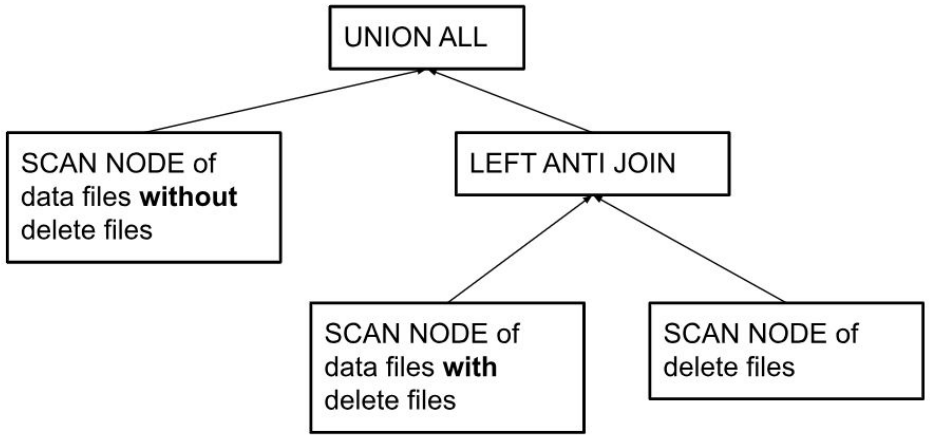
+1 Iceberg V3 positional deletes

Exploring opportunities

1) Delete SCAN node to read bitmaps and return as if V2
2) Delete SCAN to read bitmaps and return bitmaps
3) No Delete SCAN node, JOIN BUILD to read bitmaps

Difficulties

- Need a C++ Puffin reader and writer
- Merge bitmaps before writing
- Cross language compatibility?


UNION ALL
SCAN NODE of data files **without** delete files
LEFT ANTI JOIN
SCAN NODE of data files **with** delete files
SCAN NODE of delete files

Questions?