Journey to Iceberg with Trino

SK telecom

Jaechang Song - Data Engineer

Jennifer Oh - Data Engineer

Agenda

About us

Challenges with Trino and Hive

Trino meets Iceberg

Implementation work through

Performance improvement

What’s next

About us

About us

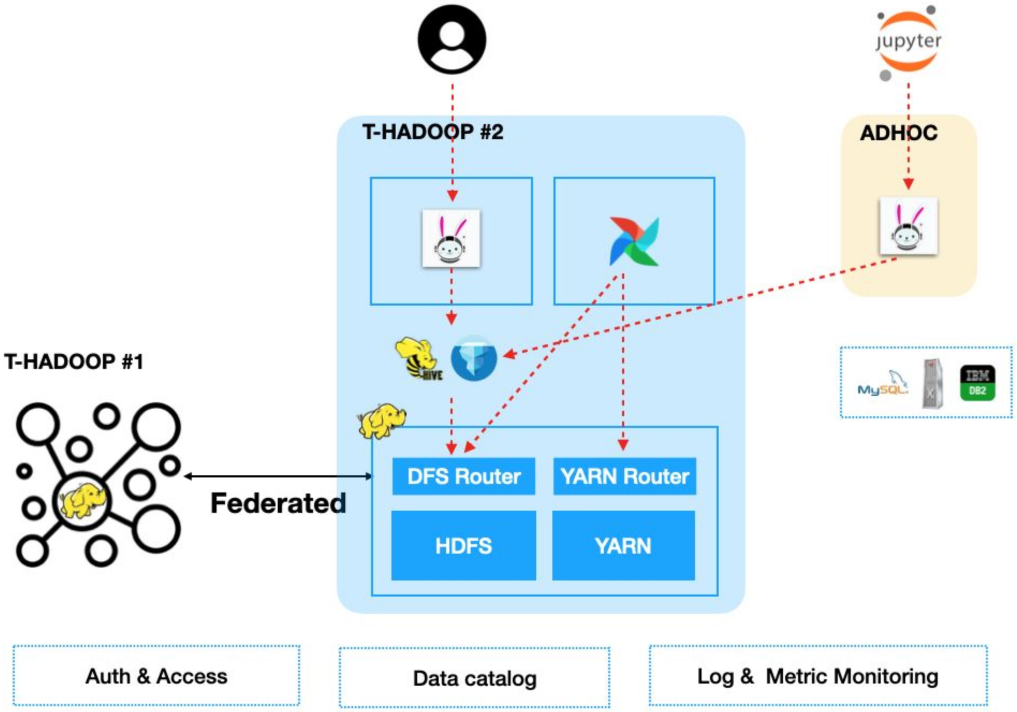
Data engineers in SKT's Emerging Data Platform team

Build open source based on-premise hadoop data platforms

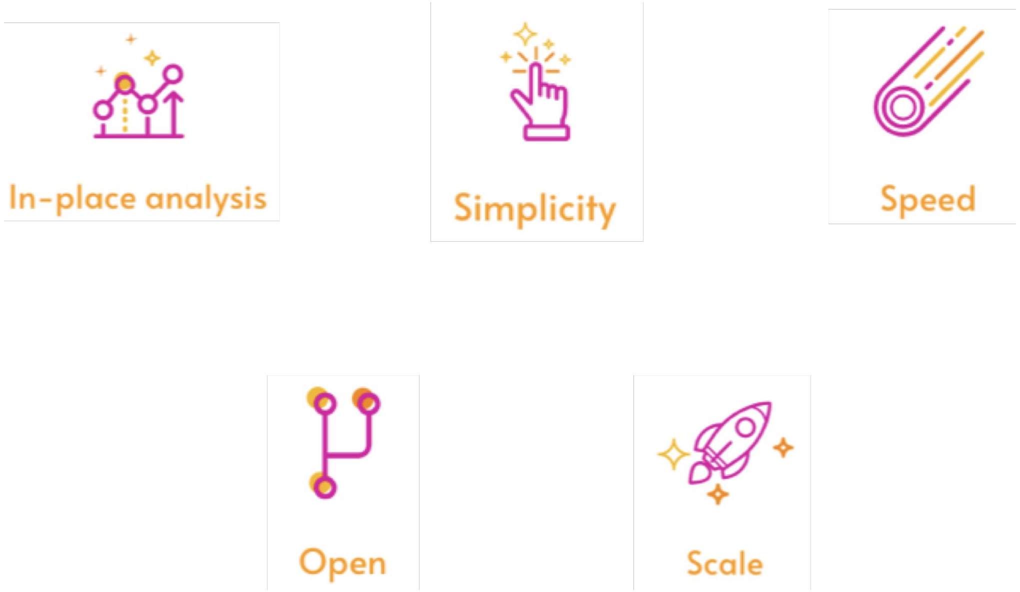
Aim for an observable, scalable, federate data platform

Believe in the power of open source

Data Ecosystem



Why Trino?



Challenges with Trino and Hive

SK telecom

Challenges

Growing services & users

300 requests per minute

TB size query input

Hundreds of nodes

PB size data

...

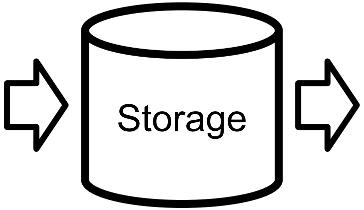
Challenges - Finding the root cause

Request result JSON

Coordinator/Worker JMX

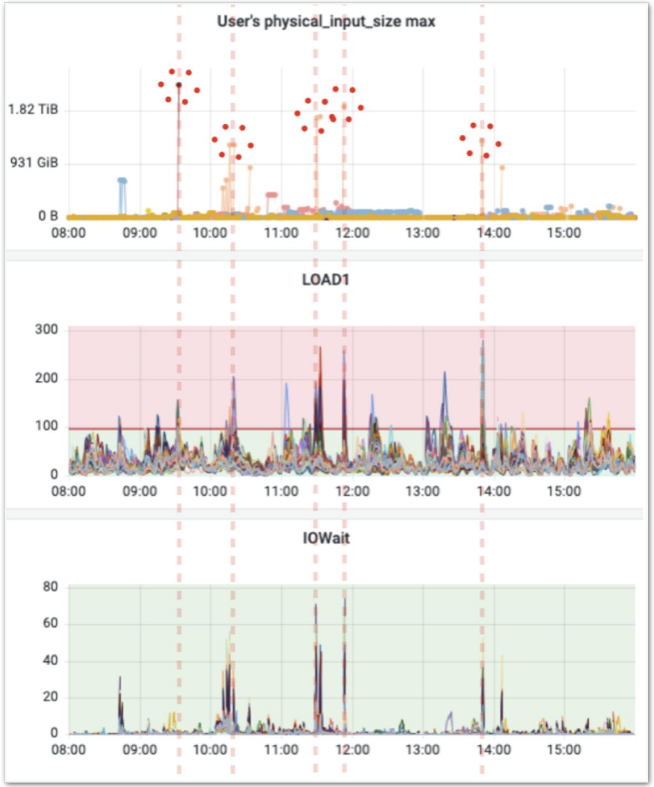
System metric

Trino logs



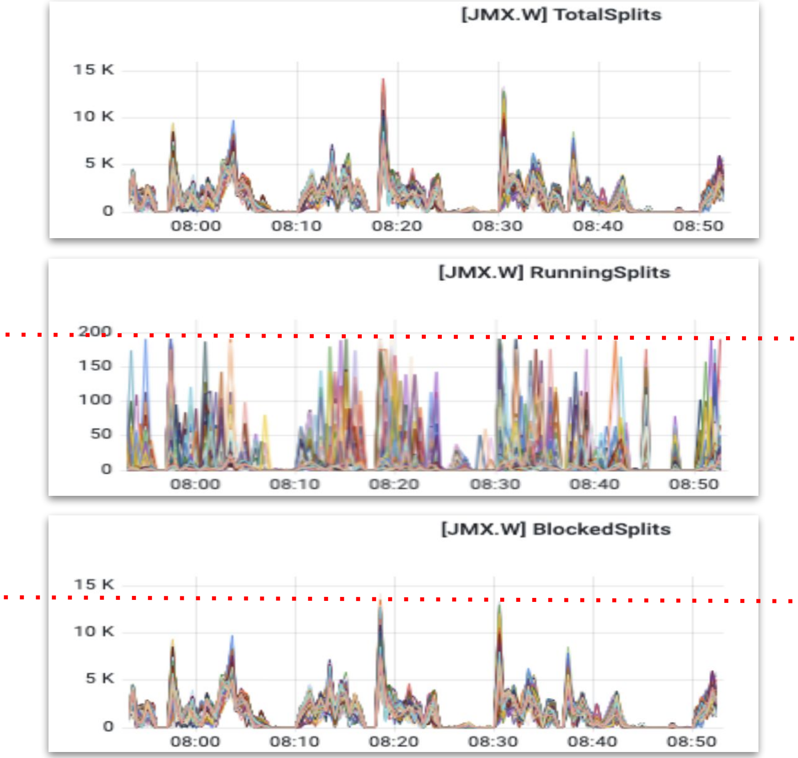
Challenges - Finding the root cause 1

1. Huge input data



Challenges - Finding the root cause 2

2\. Too many BlockedSplits



Trino worker JMX metric per node

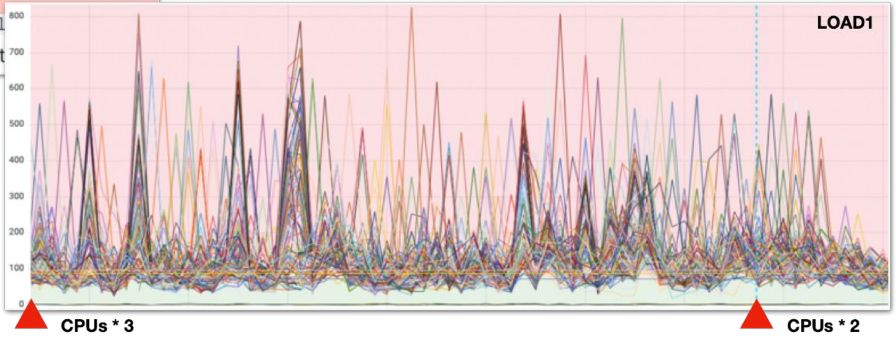
Challenges - Trial 1

Adjusting task.max-worker-threads

`task.max-worker-threads`

- **Type:** integer
- **Default value:** (Node CPUs * 2)

Sets the number of threads used by workers to process splits. Increasing this number can improve throughput, if worker CPU utilization is low and all the threads are in use, but it causes increased heap space usage. Setting the value too high may cause a drop in performance due to a context switching. The number of active threads is available via the `RunningSpl` the `trino.execution.executor:name=TaskExecutor.RunningSplits` JMX object



Challenges - Trial 2

Setting resource limit on specific user using Database Resource Group Manager

```
MariaDB [trino]> select resource_group_id, name, soft_concurrency_limit, hard_concurrency_limit from resource_groups where resource_group_id = 4;
+-------------------+---------------+------------------------+------------------------+
| resource_group_id | name          | soft_concurrency_limit | hard_concurrency_limit |
+-------------------+---------------+------------------------+------------------------+
|                 4 | mining_${USER} |                      2 |                      2 |
+-------------------+---------------+------------------------+------------------------+
1 row in set (0.000 sec)

MariaDB [trino]> select * from selectors where resource_group_id=4;
+-------------------+----------+------------+--------------+------------+-------------+---------------------------+
| resource_group_id | priority | user_regex | source_regex | query_type | client_tags | selector_resource_estimate |
+-------------------+----------+------------+--------------+------------+-------------+---------------------------+
|                 4 |        2 | tasapp     | NULL         | NULL       | NULL        | NULL                      |
+-------------------+----------+------------+--------------+------------+-------------+---------------------------+
1 row in set (0.019 sec)
```

Challenges - Trial 3

Query Tuning

Redesign partition
for partition pruning

Column sorting
for predicate pushdown

Challenges - Limits

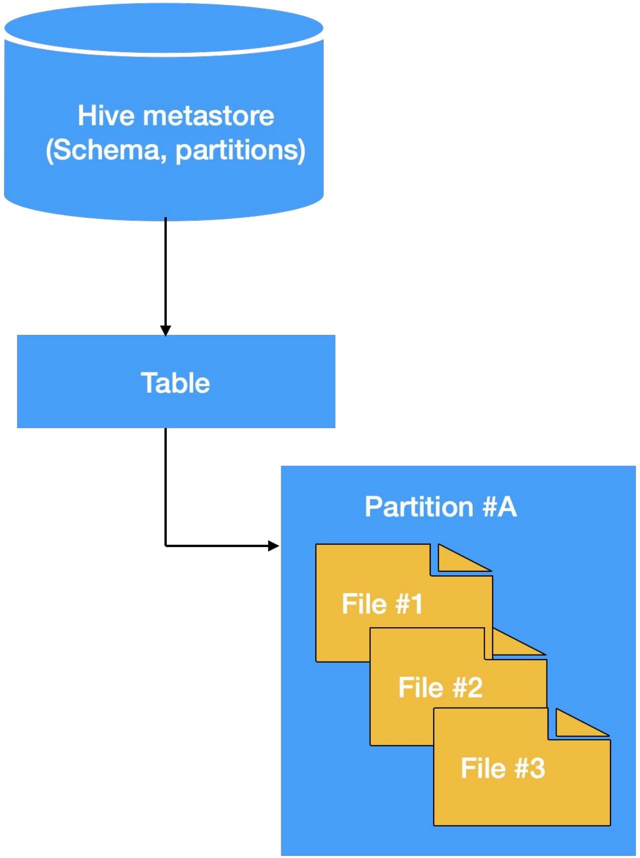
Restriction on Hive table structure

Single metastore

Operation bottleneck handling many partitions
(big metadata, small file issues…)

List operation

List operation occurs for all files in the partition to check if they match query



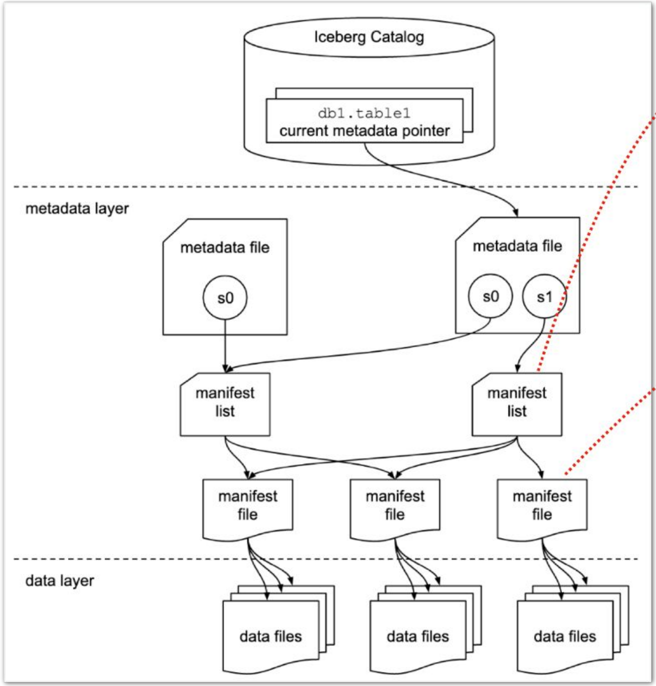
Our needs

1.	Indexing strategy	Cherry-picking required files only
2.	Flexible partitioning	Avoid creating inefficient partitions
3.	Separated metadata	Avoid bottleneck for sharing single metadata
4.	Compatibility	Compatibility across many engines and implementations (Trino, Spark, Hive)

Trino meets Iceberg

SK telecom

Why Apache Iceberg?



- **Manifest file pruning by partition range**


```
{
  "manifest_path":
  "manifest_length": 26316,
  "partition_spec_id": 0,
  "added_snapshot_id":
  {
    "long": S1
  },
  "added_data_files_count":
  {
    "int": 42
  },
  "existing_data_files_count":
  {
    "int": 0
  },
  "deleted_data_files_count":
  {
    "int": 0
  },
  "partitions":
  {
    "array":
    [
      {
        "contains_null": false,
        "contains_nan":
        {
          "boolean": false
        },
        "lower_bound":
        {
          "bytes": 2022-05-22
        },
        "upper_bound":
        {
          "bytes": 2022-05-22
        }
      },
      {
        "contains_null": false,
        "contains_nan":
        {
          "boolean": false
        },
        "lower_bound":
        {
          "bytes": AAA
        },
        "upper_bound":
        {
          "bytes": AAD
        }
      },
```


- **data file pruning by partition & col stats**


```
{
  "status": 1,
  "snapshot_id":
  {
    "long": S1
  },
  "data_file":
  {
    "file_path": "hdfs://iceberg_sample_table/data/dt=2
    "file_format": "ORC",
    "partition":
    {
      "dt": { "string": "2022-05-22" },
      "depth1": { "string": "AAA" },
      "depth2": { "string": "BBB" },
      "depth3_trunc": { "string": "CCC" },
      "depth4_trunc": { "string": "DDD" },
      "depth5_trunc": { "string": "EEE" },
      "depth6_trunc": { "string": "FFF" }
    },
    "record_count": 2,
    "file_size_in_bytes": 6892,
    "block_size_in_bytes": 67108864,
    "column_sizes":
    { ...
    },
    "value_counts":
    { ...
    },
    "null_value_counts":
    { ...
    },
    "nan_value_counts":
    { ...
    },
    "lower_bounds":
    { ...
    },
    "upper_bounds":
    { ...
    },
```

PoC - How Iceberg cherry picks data files

	Task id	Read file path	Start	Length(MB)
Hive	20221023_011445_00163_yyn5y.1.5-1	.../00163-16-9f4bd6fa-...-00001.orc	0	**32**
	20221023_011445_00163_yyn5y.1.2-0	.../00163-16-9f4bd6fa-...-00001.orc	33554432	32
	20221023_011445_00163_yyn5y.1.5-0	.../00163-16-9f4bd6fa-...-00001.orc	67108864	32
	20221023_011445_00163_yyn5y.1.5-2	.../00163-16-9f4bd6fa-...-00001.orc	100663296	32
	20221023_011445_00163_yyn5y.1.4-0	.../00163-16-9f4bd6fa-...-00001.orc	134217728	32
	20221023_011445_00163_yyn5y.1.0-0	.../00163-16-9f4bd6fa-...-00001.orc	167772160	32
	20221023_011445_00163_yyn5y.1.0-1	.../00163-16-9f4bd6fa-...-00001.orc	201326592	32
	20221023_011445_00163_yyn5y.1.1-1	.../00163-16-9f4bd6fa-...-00001.orc	234881024	32
	20221023_011445_00163_yyn5y.1.4-1	.../00163-16-9f4bd6fa-...-00001.orc	268435456	25
	20221023_011445_00163_yyn5y.1.6-1	.../00163-218-797d8829-...00001.orc	0	32
	20221023_011445_00163_yyn5y.1.3-0	.../00163-218-797d8829-...00001.orc	33554432	32
	20221023_011445_00163_yyn5y.1.6-0	.../00163-218-797d8829-...00001.orc	67108864	32
	20221023_011445_00163_yyn5y.1.3-1	.../00163-218-797d8829-...00001.orc	100663296	32
	20221023_011445_00163_yyn5y.1.2-1	.../00163-218-797d8829-...00001.orc	134217728	16
	20221023_011445_00163_yyn5y.1.1-0	.../00163-218-797d8829-...00001.orc	151456965	16
Iceberg	20221023_011505_00165_yyn5y.1.1-0	.../00163-218-797d8829-...00001.orc	0	**128**
	20221023_011505_00165_yyn5y.1.0-0	.../00163-218-797d8829-...00001.orc	134217728	33

[trino] hive.max-initial-split-size

Resource Utilization Summary

CPU Time	1.52s
Scheduled Time	1.86s
Input Rows	20.0K
Input Data	149KB
Physical Input Rows	20.0K
Physical Input Data	90.0KB

[iceberg] read.split.target-size

PoC - How to design partitions

Good partition design is important in Iceberg table

- Too subdivided partitions -> metadata increase + small file issue
- More metadata -> longer planning / commit time

Better to go with little larger read input for appropriate partition

	Table type	Partition depth	Partition count	Total drivers	Physical input data	Planning time	Elapsed time
Table1	Hive	3	35,127	1,148,570	209GB	1.70s	1.69m
Table2	Iceberg	7	5,907,732	52,256	1.24GB	46.58s	1.23m
Table3	Iceberg	4	109,765	124,696	6.11GB	69.59ms	7.24s

PoC - Parallelism

Read split size had gap between Hive and Iceberg tables in Trino

- Iceberg table uses 128MB read.split-target-size, so parallelism may get low

You can modify Iceberg table property to increase parallelism

	read.split-target-size	Parallelism	Elapsed time
Hive	32MB -> 64MB	36.6	1.06m
Iceberg	128MB	64.1	7.69s
Iceberg	64MB	69.8	7.55s
Iceberg	32MB	75.4	7.35s

Case1

	read.split-target-size	Parallelism	Elapsed time
Hive	32MB -> 64MB	40.7	1.62m
Iceberg	128MB	62.9	3.09s
Iceberg	64MB	73.8	3.39s
Iceberg	32MB	80.8	5.52s

Case2

PoC - How simple can it be?

Before

- Save intermediate files in temporary directory to handle job failure
- Move to target directory after job finishes

After

- Iceberg snapshot isolation enables writing to target directory directly
- Can easily track table commits using Iceberg metadata

Snapshot makes ETL so simple!

Implementation work through

SK telecom

Implementation - Data writes

We stack data frequently so we used APPEND

When using MERGE INTO with hidden partition for reprocessing

- table structure: days()_partition/sub_partitions/data_files

```
CREATE TABLE iceberg.product
AS … PARTITIONED BY days(event_time)
```

```
MERGE INTO iceberg.product t USING (SELECT * FROM source) s
ON t.event_time = s.event_time
 and t.sub_partiotions = s.sub_partiotions
 and t.processing_time = s.processing_time
WHEN MATCHED THEN UPDATE SET *
WHEN NOT MATCHED THEN INSERT *
```

Implementation - Sort Strategy

Sort strategy is very important since it's directly related to filter predicate

We set table sort strategy with DDL

-> automatically write sorted records in every new writes to this table

```
ALTER TABLE ... WRITE DISTRIBUTED BY
PARTITION LOCALLY
ORDERED BY
```

We run spark rewrite_data_files procedure daily using sort option -> compact separated data files from frequent batch jobs

Implementation - Sort Strategy

How to run heavy sorting efficiently?

Try setting threshold for the size you can process in one job

-> If file < min-file-size-bytes || file > max-file-size-bytes

-> If min-input-files

Try splitting the compaction of partitions into multiple jobs to run in parallel

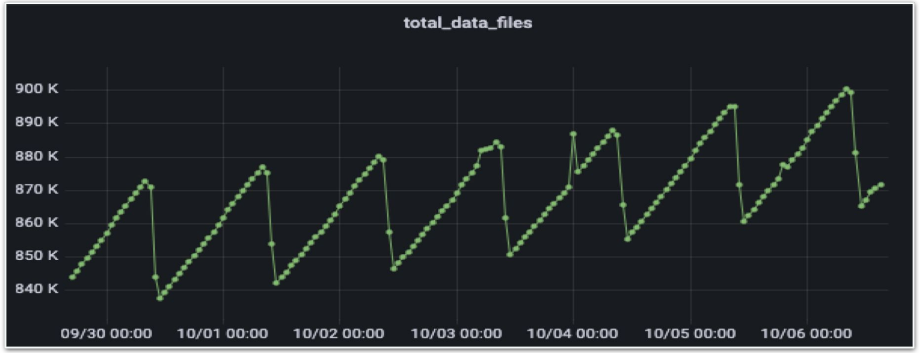
```
CALL catalog_name.system.rewrite_data_files(
        table => 'db.sample',
        strategy => 'sort',
        sort_order => ‘c1,c2’,
        options => map(‘max-concurrent-file-group-rewrites’,’100’, ‘min-input-files’,’20'...)
)
```

Implementation - Table Optimization

Expire snapshots

Rewrite data files/manifests

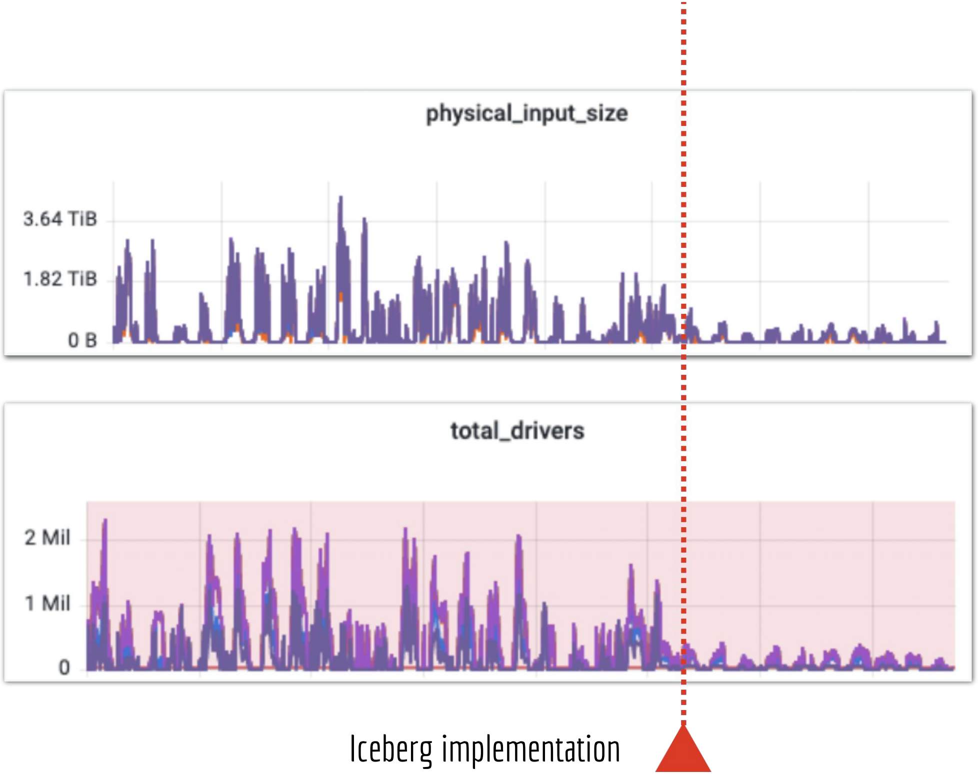
Remove orphan files



Performance improvement

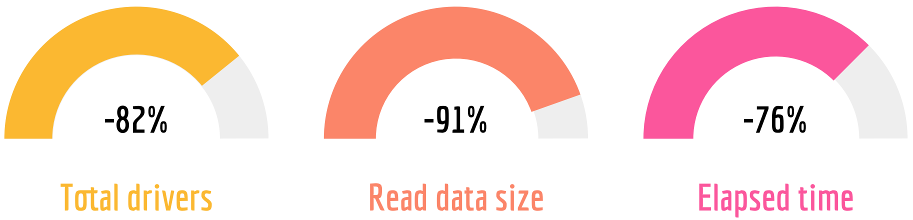
Input overheads reduction

1. Reduced driver count & physical input size



Query performance improvement

2. Reduced query elapsed time by 80%

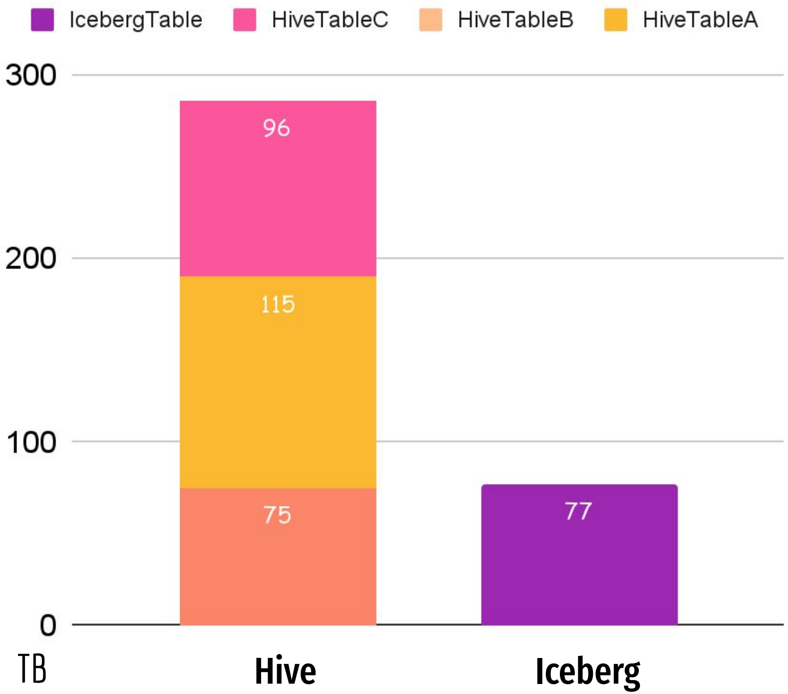


	Total drivers	Read data size	Planning time	Elapsed time
Case 1	1148570 -> 124696 (-89.1%)	209GB -> 6.11GB (-97%)	1.52s -> 27.99ms	97.2s -> 3.39s (-96.5%)
Case 2	39502 -> 2555 (-93.5%)	70.32GB -> 6.59GB (-90.6%)	1.88s -> 501.40ms	63.6s -> 7.55s (-88.1%)
Case 3	77386 -> 27095 (-64.9%)	90.70GB -> 11.3GB (-87.5%)	195.08ms -> 170.45ms	54.34s -> 29.56s (-45.6%)

Hive table logical size: 52.1TB partition count: 33,942 -> Iceberg table logical size: 52.7TB partition count: 109,765

Storage cost reduction

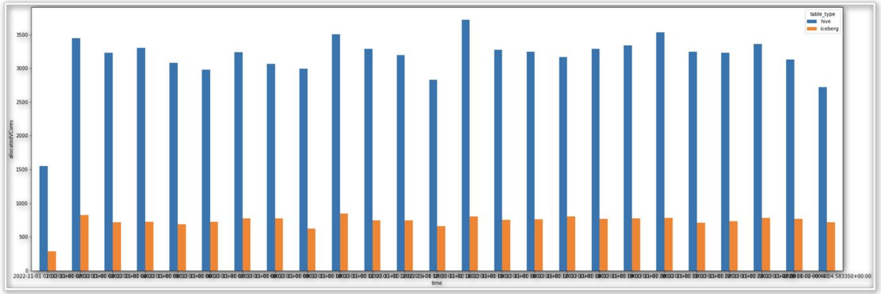
3. Reduced storage by 75%



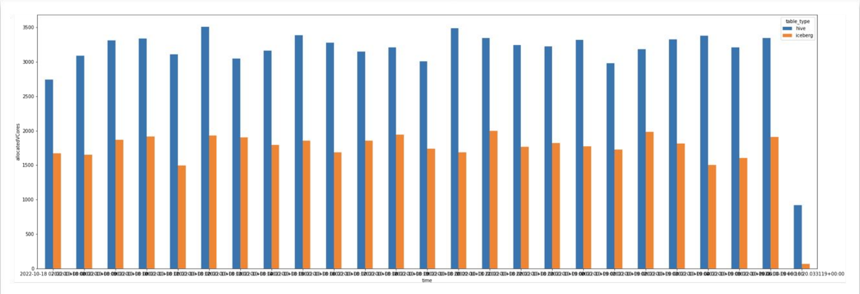
Processing cost reduction

4. Reduced cores & memories by 60%

cores



memories



Hadoop Yarn resource usage daily

What's next

What’s next

Separate Trino clusters

Extend Iceberg tables

The END