LakeSoul Introduction

A Cloud-native Realtime LakeHouse Framework

https://github.com/meta-soul/LakeSoul

chenxu@dmetasoul.com

5/4/2023

Why donate to LF AI & Data

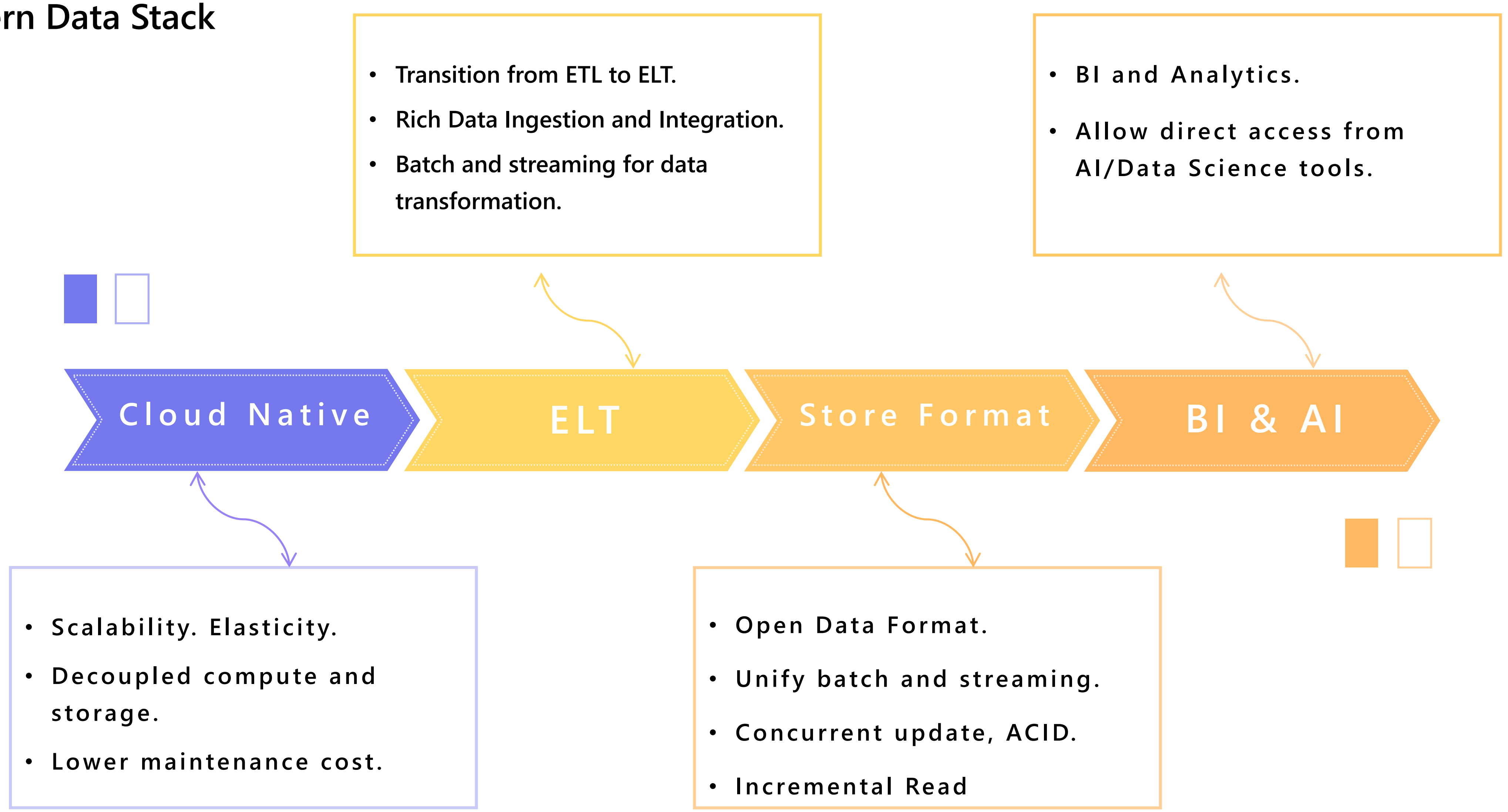
- Neutral Holding Ground
  - Vendor-neutral, Not for profit
- Growing Community
  - Increase users by outreach and involving through the foundation
  - Increase contributors from developer users
  - Collaboration with other projects in the foundation
- Open Governance Model
  - Open governance + open source license
  - Neutral management of project by the foundation
  - Instill trust in contributors and adopters in the management of the project

Company Profile

- We are DMetaSoul
  - A startup based in Beijing
  - Building better data & intelligence infrastructure
  - We believe opensource is the key!

Background

Modern Data Stack



- Scalability. Elasticity.
- Decoupled compute and storage.
- Lower maintenance cost.

- Transition from ETL to ELT.
- Rich Data Ingestion and Integration.
- Batch and streaming for data transformation.

- Open Data Format.
- Unify batch and streaming.
- Concurrent update, ACID.
- Incremental Read

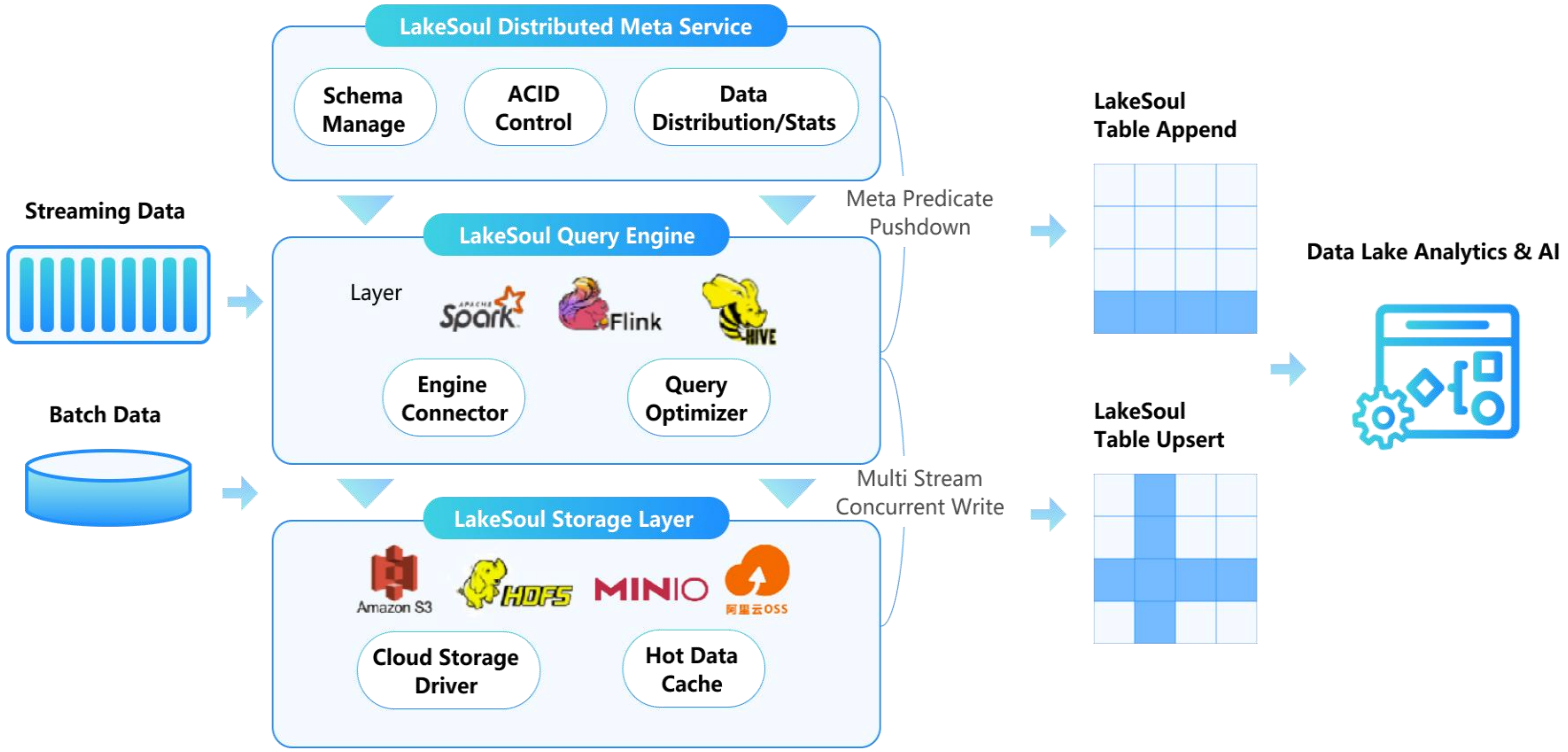
- BI and Analytics.
- Allow direct access from AI/Data Science tools.

LakeSoul’s Positioning

<u>Cloud Native Lakehouse Framework with Unified Batch and Streaming Support</u>

- Goals
  - Cloud first, without dedicated storage, and optimizations for object store
  - Centralized metadata management, ACID, concurrent upsert and snapshot read
  - Streaming data ingestion and incremental pipeline
  - Make data analytics and AI on data lake more efficient and easy
- Non Goals
  - Create a new compute engine
  - Create a new file format
  - Optimize for point update or query

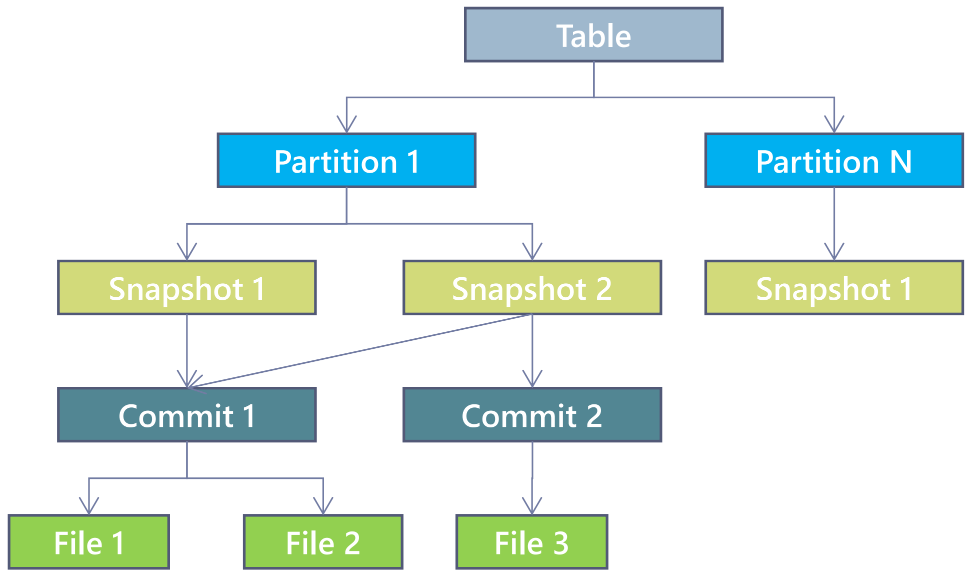
LakeSoul Architectural Overview



Data Model and Metadata

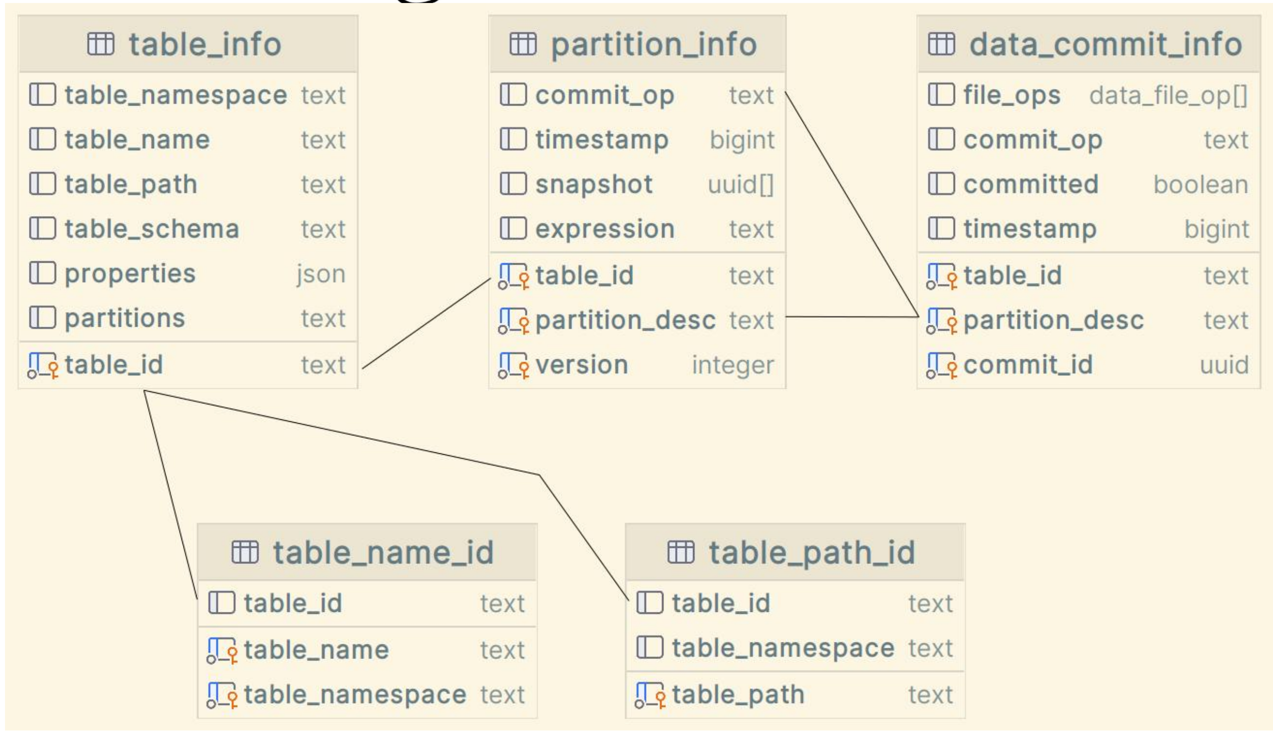
Data Modeling

- Physical Data
  - Files are stored physically with Parquet format
  - Table could optionally have primary key constraint
    - Files are hash bucketed (with a predefined bucket number), and each upserted file is sorted by PKs
  - Table could have multi-level range partitions
- Meta Data
  - Commit: Files sequence with add/delete ops
  - Snapshot: Commits sequence with commit types (Append, Merge, Compaction, Update)
  - Version: Monotonic increasing number that identifies a snapshot and its timestamp



Centralized Metadata Management

- Centralized metadata management through PostgreSQL
  - Concurrent ACID via PG's transaction
  - Two-phase commit protocol
  - Fine-grained write conflicts resolving
  - Trigger function in PLSQL for event publish
- PG is generally available on most of the cloud vendors
- Java wrapper and Spark/Flink's Catalog interface implementations



table_info

table_namespace	text
table_name	text
table_path	text
table_schema	text
properties	json
partitions	text
table_id	text

partition_info

commit_op	text
timestamp	bigint
snapshot	uuid[]
expression	text
table_id	text
partition_desc	text
version	integer

data_commit_info

file_ops	data_file_op[]
commit_op	text
committed	boolean
timestamp	bigint
table_id	text
partition_desc	text
commit_id	uuid

table_name_id

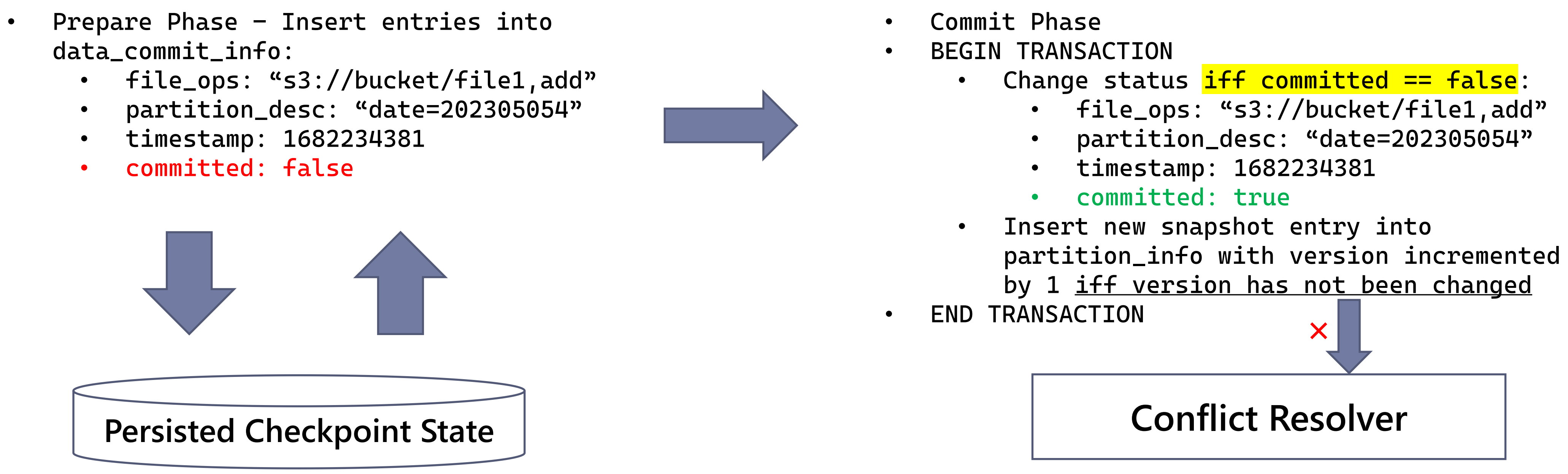
table_id	text
table_name	text
table_namespace	text

table_path_id

table_id	text
table_namespace	text
table_path	text

Centralized Metadata Management

- Two-phase Commit Protocol
  - Executed during batch write or stream checkpoint in Spark/Flink



- Prepare Phase – Insert entries into data_commit_info:
  - file_ops: “s3://bucket/file1,add”
  - partition_desc: “date=20230505”
  - timestamp: 1682234381
  - committed: false

- Commit Phase
- BEGIN TRANSACTION
  - Change status iff committed == false:
    - file_ops: “s3://bucket/file1,add”
    - partition_desc: “date=20230505”
    - timestamp: 1682234381
    - committed: true
  - Insert new snapshot entry into partition_info with version incremented by 1 iff version has not been changed
- END TRANSACTION

Centralized Metadata Management

- Fine-grained write conflict resolving with PG's transaction
  - Retry: Compatible write but version changed, retry with newest version + 1
    - Concurrent Append, Merge
  - Reorder: Create a new snapshot with current commit in the middle
    - Concurrent compaction or update with no other unresolvable conflict
  - Concurrent Updates are unresolvable and fail

Operation	Append	Merge	Compaction	Update
Append	Retry	X	Retry	Retry
Merge	X	Retry	Reorder	Retry
Compaction	Reorder	Reorder	Ignore	Ignore
Update	Reorder	Reorder	Overwrite	Fail

  - Guaranteed atomicity while improving concurrency

Centralized Metadata Management

- Auto Schema Evolution
  - Automatically update schema during write
  - Enabling schema change on the fly (without stop-the-world DDL operation)
  - Automatic read schema reconciliation
    - Add Column: Old data padded with null during read
    - Drop Column: Old data's column filtered out during read
- Snapshot Read, Rollback and Cleanup
  - Each snapshot is associated with a UTC timestamp
  - Read newest snapshot by default
  - APIs to access older snapshot with human-readable timestamp string

Native IO Layer

Native IO Layer

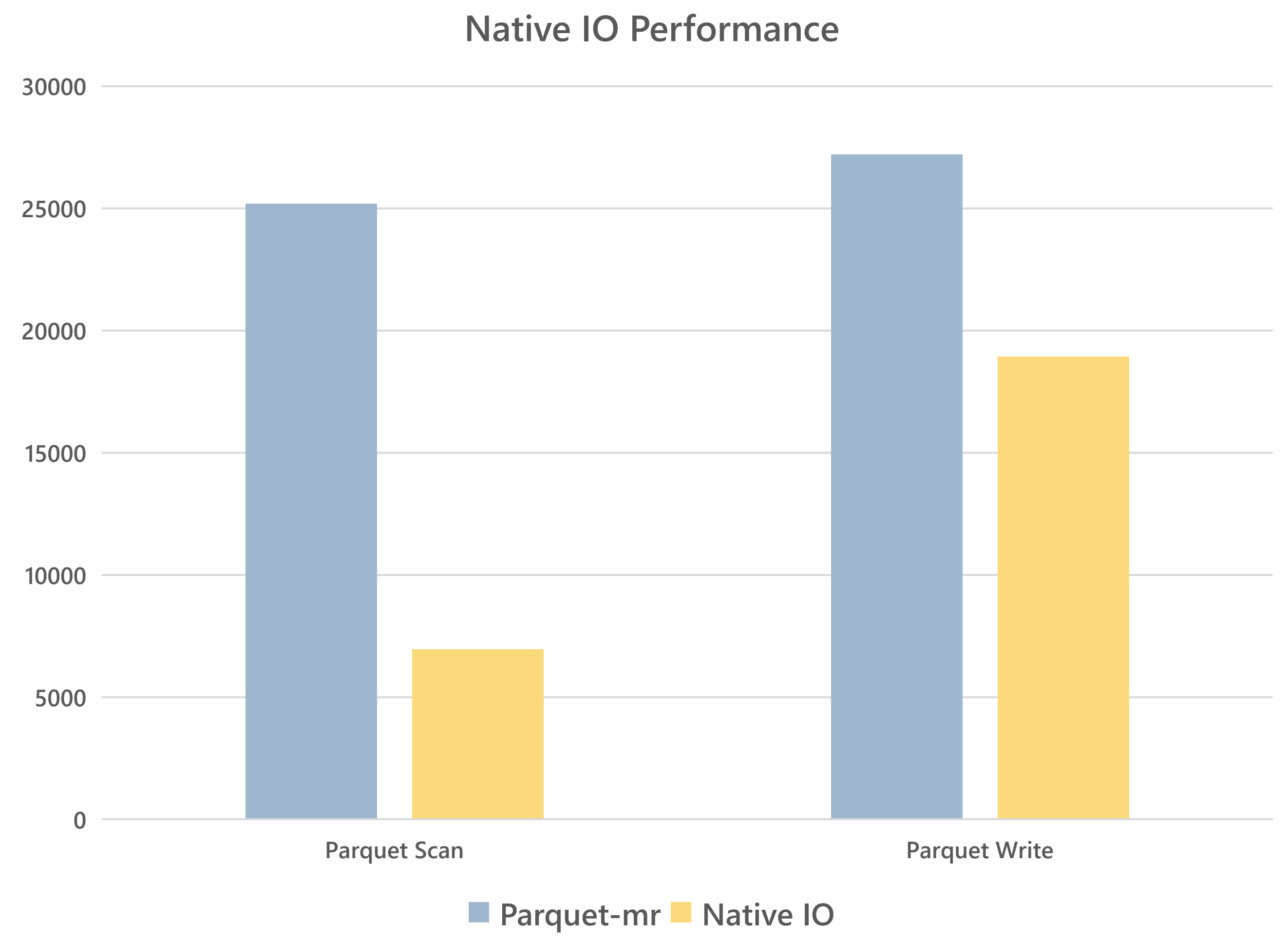
Design Principles:

1. Encapsulate read/write logics for upsert and merge on read
    1. Simple interfaces for reading/writing parquet files to/from object storage and HDFS
2. Easier integration
    1. Integrate with various data&ai compute engines
    2. Provide vectorized reader & writer if the engine needs
    3. Provide C, Java, Python wrappers
3. Cloud native
    1. Optimize for high latency r/w
    2. Limit cpu/memory usage

Implementation:

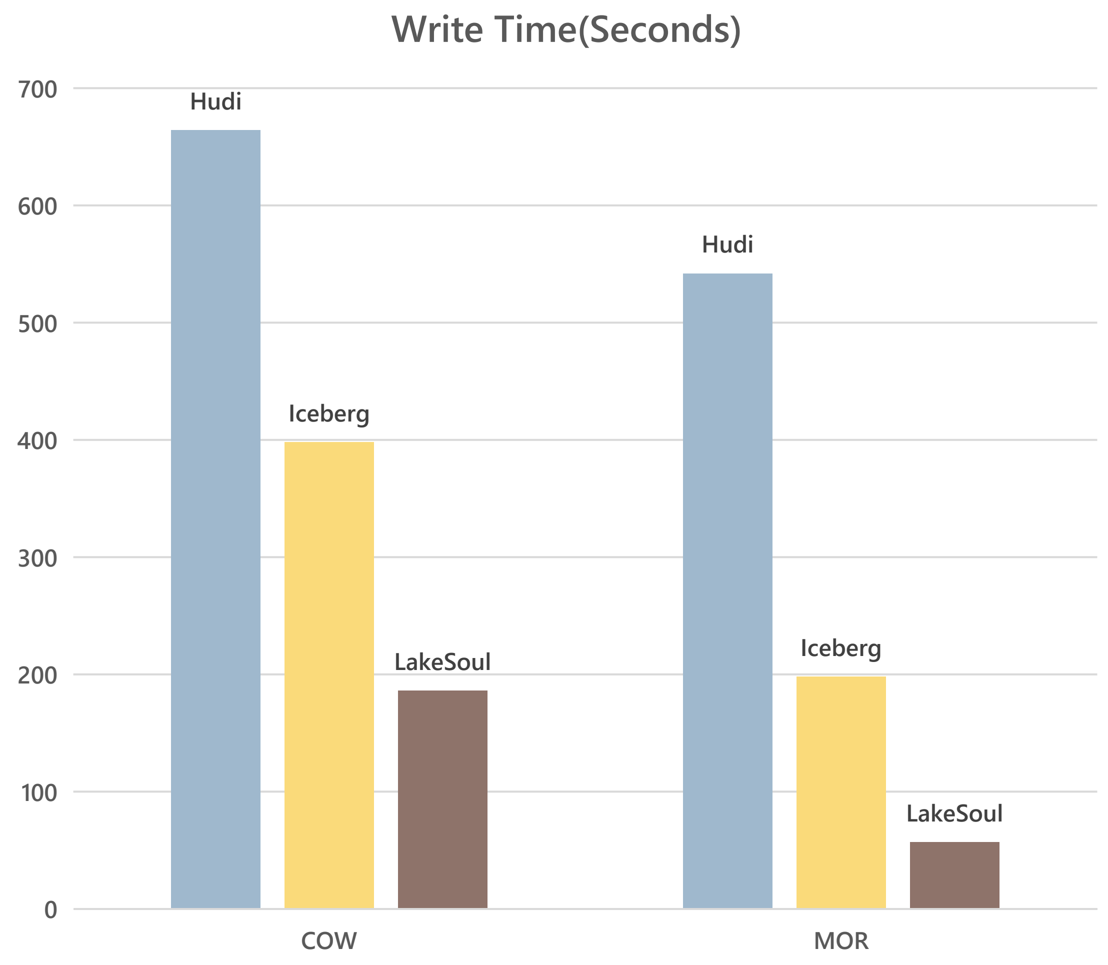
1. Async reader, writer in Rust, with arrow-rs and arrow-dataFusion
    1. Apache Arrow Recordbatch as memory format
    2. Async Writer: async sort and multi-part upload in background IO threads
    3. Async Reader: Sorted merge from async file streams with parquet row group prefetch and large request splitting
2. C interface and Java/Python wrappers through jnr-ffi/ctypes
3. Spark DataSource V2 & Flink DynamicTableFactory implementations for both batch and stream

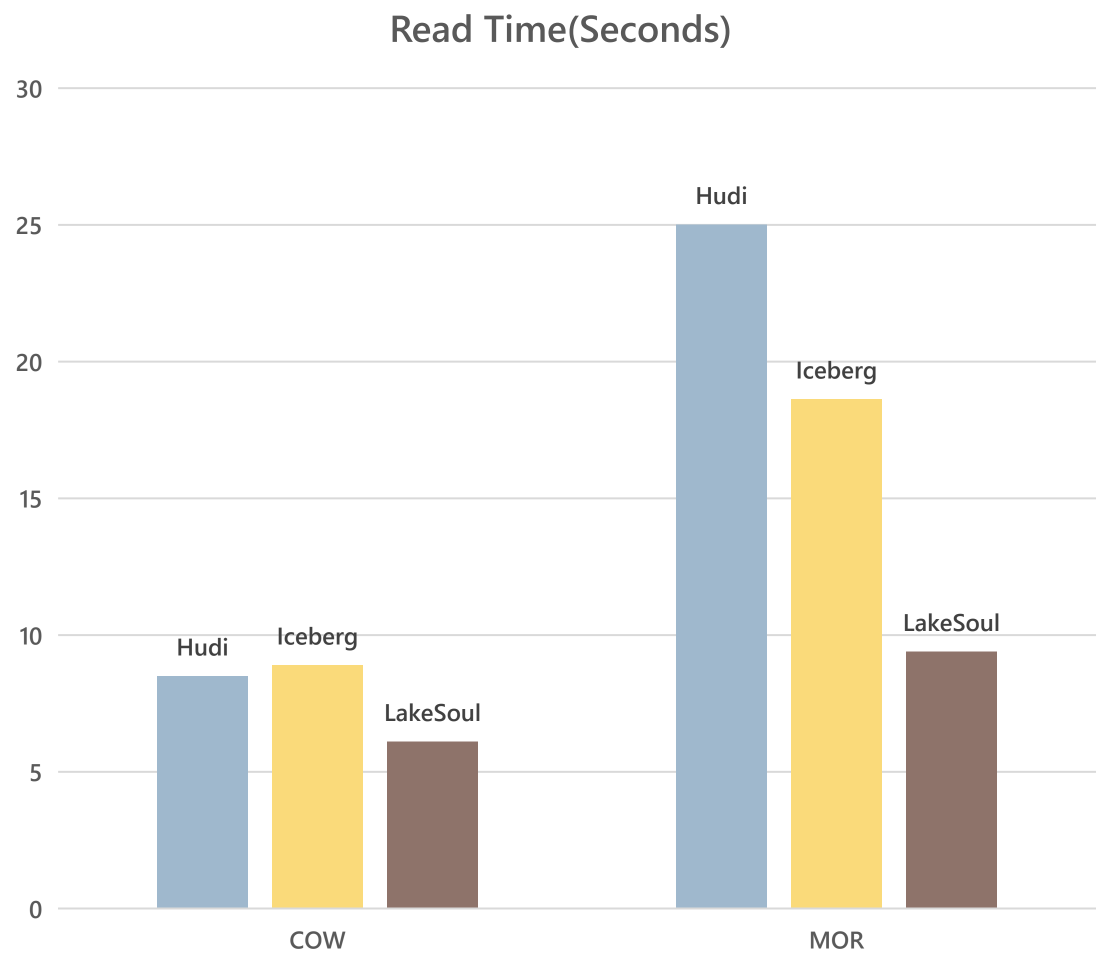
Native IO Benchmarks



Benchmark source code available at:
https://github.com/meta-soul/LakeSoul/tree/main/lakesoul-spark/src/test/scala/org/apache/spark/sql/lakesoul/benchmark/io

Native IO Benchmarks





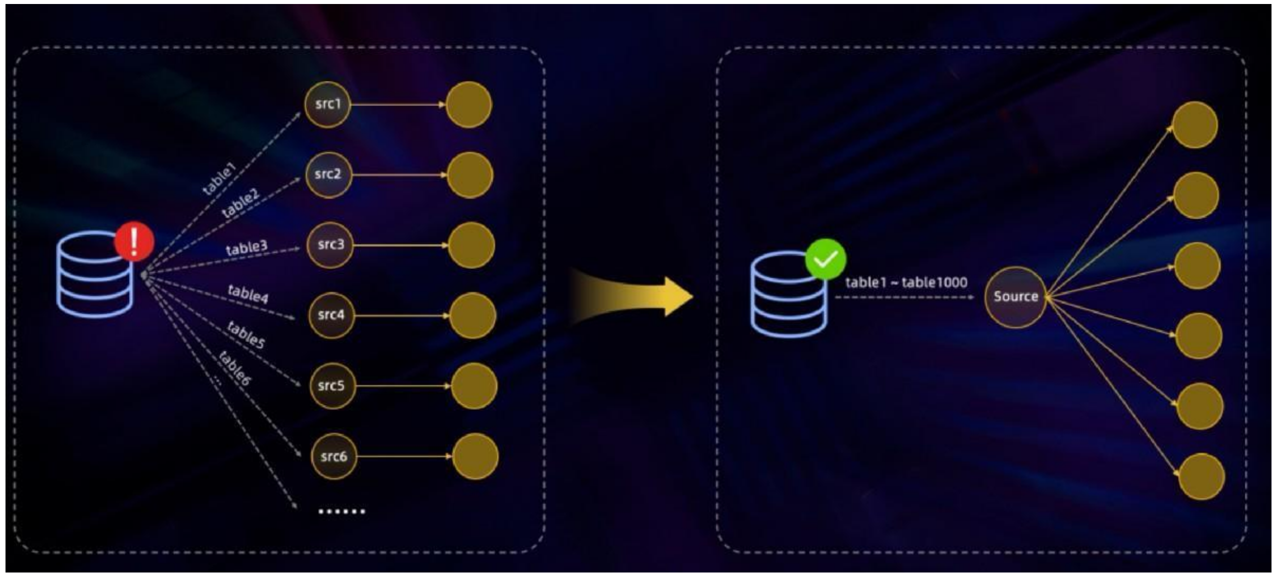
Benchmark data and source code available at:
https://github.com/meta-soul/ccf-bdci2022-datalake-contest-examples/tree/mor
https://github.com/meta-soul/ccf-bdci2022-datalake-contest-examples/tree/cow

Benchmark settings:
- **Environment: Spark 3.3.1, 4 cpu 16G, OpenJDK 11**
- **Write 10 millions rows initially, then upsert 10 times for 2 millions rows each (with 1 million existing PKs each)**
- **Merge on Read without compaction**

Streaming Pipeline

Streaming Data Ingestion

- Synchronize multiple tables from RDBMS (MySQL etc.) and multiple topics from message queue (Kafka)
  - In ONE Flink/Spark stream job
  - CDC stream ingestion
  - Auto new table/topic discovery
  - Auto schema change sync
  - End-to-end exactly once guarantee



Streaming Incremental Pipeline

- Native Support for Changelog Format
  - Added a “row_kind” column in storage format
  - “row_kind” column’s enum values: “update”, “insert”, “delete”
  - Sink Debezium/Flink CDC streams, etc. into LakeSoul’s changelog format
  - Incremental and continuous read from LakeSoul table as changlog stream source

```
INSERT INTO lakesoul_table SELECT * FROM mysql_cdc_stream;

SELECT sum(revenue) FROM lakesoul_table
/*+ OPTIONS('readstarttime'='2023-04-21 10:00:00','readtype'='incremental')*/
GROUP BY city;
```

Row Kind	city (pk)	revenue
+I	BJ	100
+I	SH	200
U	BJ	150
-D	SH	

Efficient Incremental Compute Pipeline

city	sum(rev)
BJ	100
SH	200

city	sum(rev)
BJ	150
SH	200

city	sum(rev)
BJ	150

Streaming Join

- Multi streams join without engine's state
  - Reduce maintenance overhead of large stateful stream job
  - Reduce compute overhead of full join among large tables
  - Achieve higher throughput with lower latency

Stream A

PK	Field 1	Field 2
key1	1	"abc"

Stream B

PK	Field 1	Field 3	Field 4
key2	2	9.99	"xyz"

Stream C

PK	Field 2	Field 1	Field 3
key1	"def"	3	0.99

- Native support for heterogeneous stream upserts with same pk
- Turn join job into 3 upsert jobs
- Merge on read according to target table's schema

PK	Field 1	Field 2	Field 3	Field 4
key1	3	"def"	0.99	null
key2	2	null	9.99	"xyz"

Target Table

Stream A

PK_A	Field 1	Field 2	FK_B
key1	1	"abc"	key2

Stream B

PK_B	Field 3	Field 4
key2	2	9.99

- Turn join job into
  - 1 Upsert (from stream A)
  - 1 broad cast join of B's increment with A and upsert

PK_A	PK_B	Field 1	Field 2	Field 3	Field 4
key1	key2	1	"abc"	2	9.99

Target Table

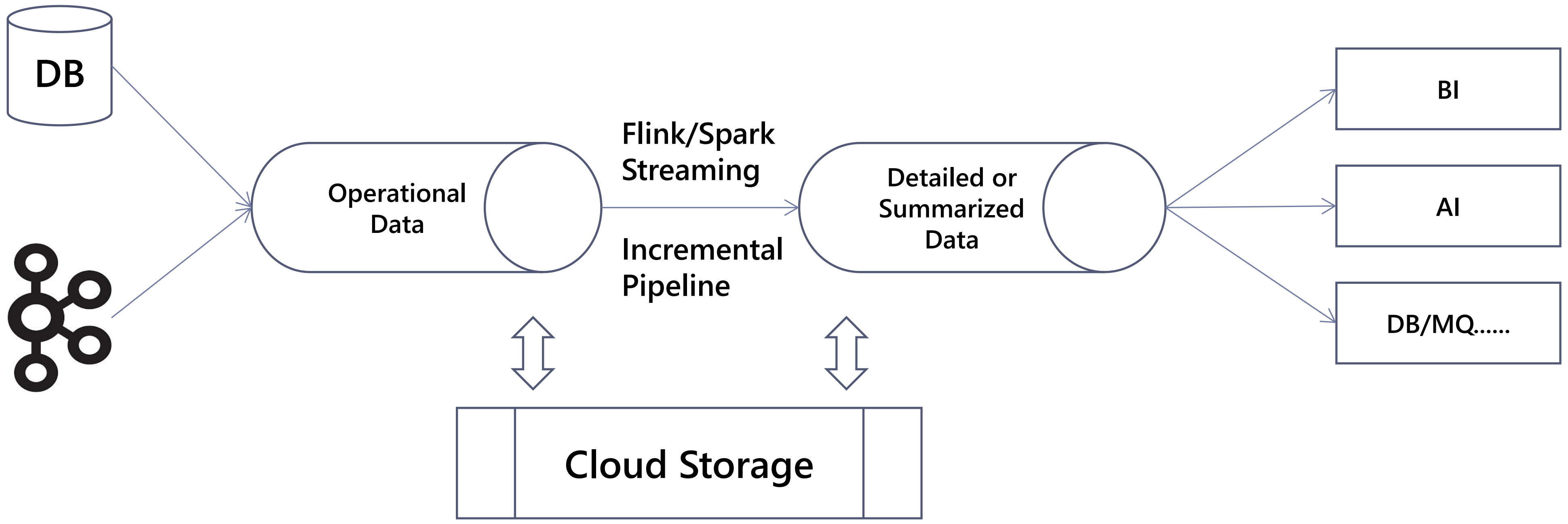
Automatic Disaggregated Compaction

- Rely on PG's trigger-notify-listen mechanism
- Define a trigger function in PLSQL in PG
- Triggerred whenever new data committed and a customizable condition met (e.g. 10 commits since last compaction)
- Listen to the triggerred events for all tables and invoke compaction in one Spark job
- Auto scaling with Spark's dynamic executor allocation

Applications

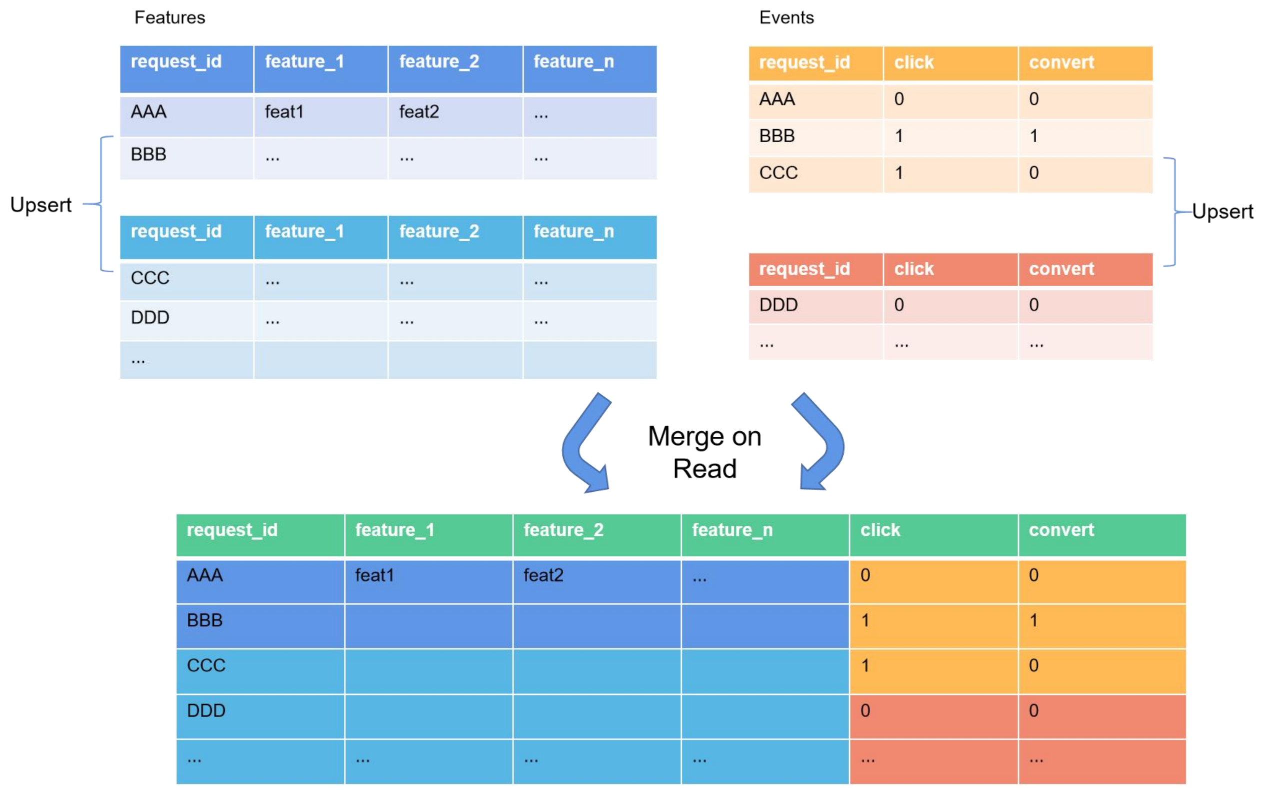
Building End-to-end Real-time LakeHouse

- Build Lakehouse on LakeSoul
  - Incremental, streaming pipeline without extra time-based scheduling
  - “Unlimited” storage
  - Historical data can be accessed and updated
  - Run BI/AI on the Lakehouse
  - Pipe data to external systems



Building Real-time Machine Learning Datasets

- Tabular datasets for machine learning
  - Solve decision making problems
  - Classification, forecasting, recommendations
- Use LakeSoul to build tabular datasets in real-time
  - Concat features and labels from multiple streams
  - Feed data to machine learning frameworks directly, including Spark's MLLib, Flink ML and PyTorch
  - Enable online learning by using LakeSoul table as stream source



Features

request_id	feature_1	feature_2	feature_n
AAA	feat1	feat2	...
BBB	...	...	...

request_id	feature_1	feature_2	feature_n
CCC	...	...	...
DDD	...	...	...
...			

Events

request_id	click	convert
AAA	0	0
BBB	1	1
CCC	1	0

request_id	click	convert
DDD	0	0
...	...	...

request_id	feature_1	feature_2	feature_n	click	convert
AAA	feat1	feat2	...	0	0
BBB				1	1
CCC				1	0
DDD				0	0
...	...	...	...	...	...

Current Community State

- Opensourced in December 2021 under Apache License V2
- 1295 Stars, 289 forks on Github
- 11 Contributors. 4 from other organizations
- Early adoptions from aviation and banking companies and one research lab

Possible Collaboration with LF AI & Data Projects

- Integrate data lineage with OpenLineage and Marquez
- Provide batch and stream source to data and feature processing projects including Sparklyr and Feast
- Build tabular training datasets for ML systems including PyTorch, Angel ML and FATE

Future Plans

- Data Warehousing
  - Streaming State Table
  - SQL to streaming pipeline translation
  - Data lineage
  - Built-in RBAC
- Echosystem
  - PyArrow reader
  - Presto Connector
  - More DB sources
  - Kafka Connector sink
  - Logstash sink
- Performance
  - Improve sorted stream merge speed
  - Minor compaction
  - Integrate with compute engine’s vectorization optimization
  - Local disk cache

Thank You!

We are requesting your support to host LakeSoul in LF AI & Data as a Sanbox Project