Fast Copy-On-Write with Apache Parquet

Xinli Shang, Mingmin Chen @ Uber Data Infra

shangxinli@apache.org

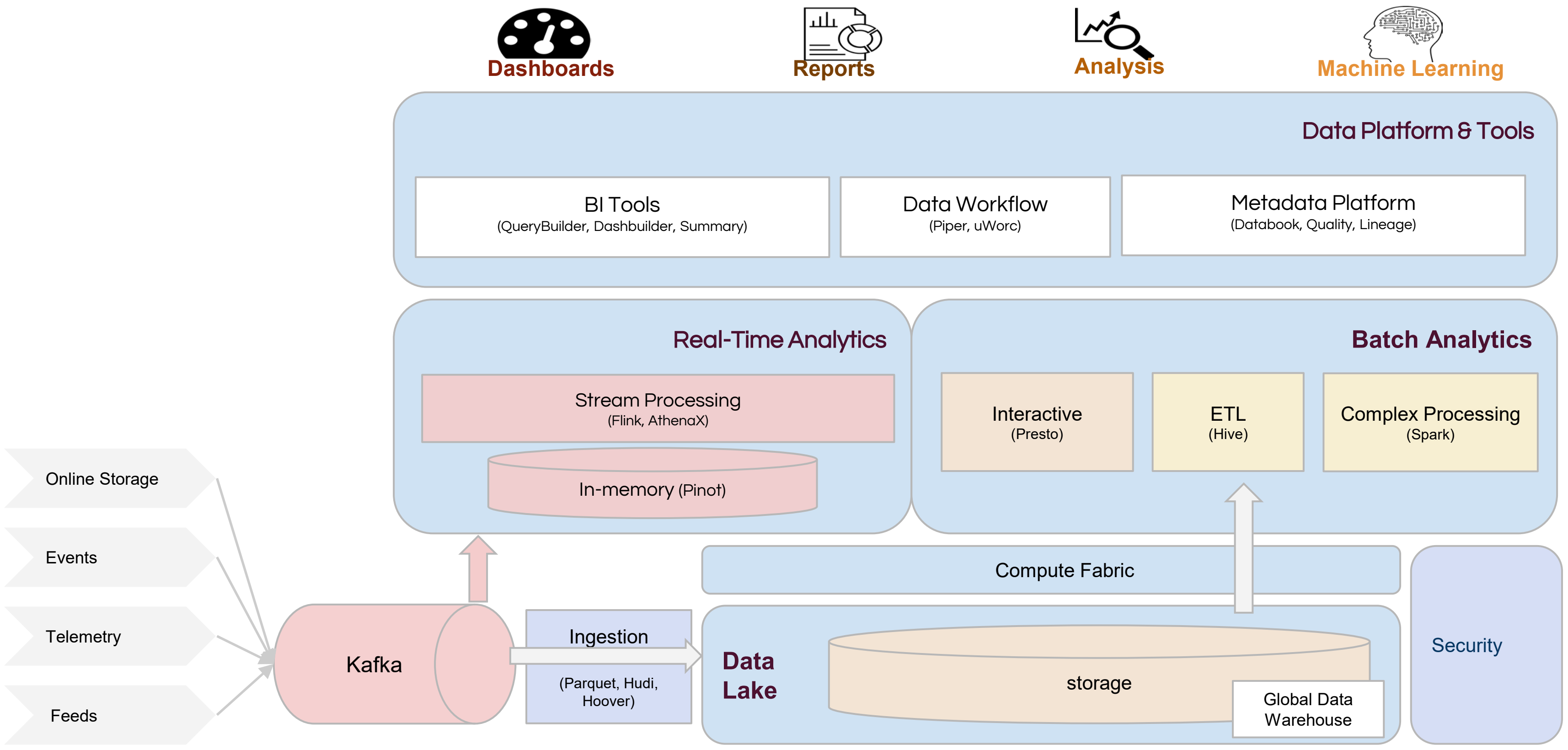
Speaker Intro

- Xinli Shang
  - Senior Manager @ Uber
  - Apache Parquet PMC chair, Presto committer
- Mingmin Chen
  - Director @ Uber Data Infra

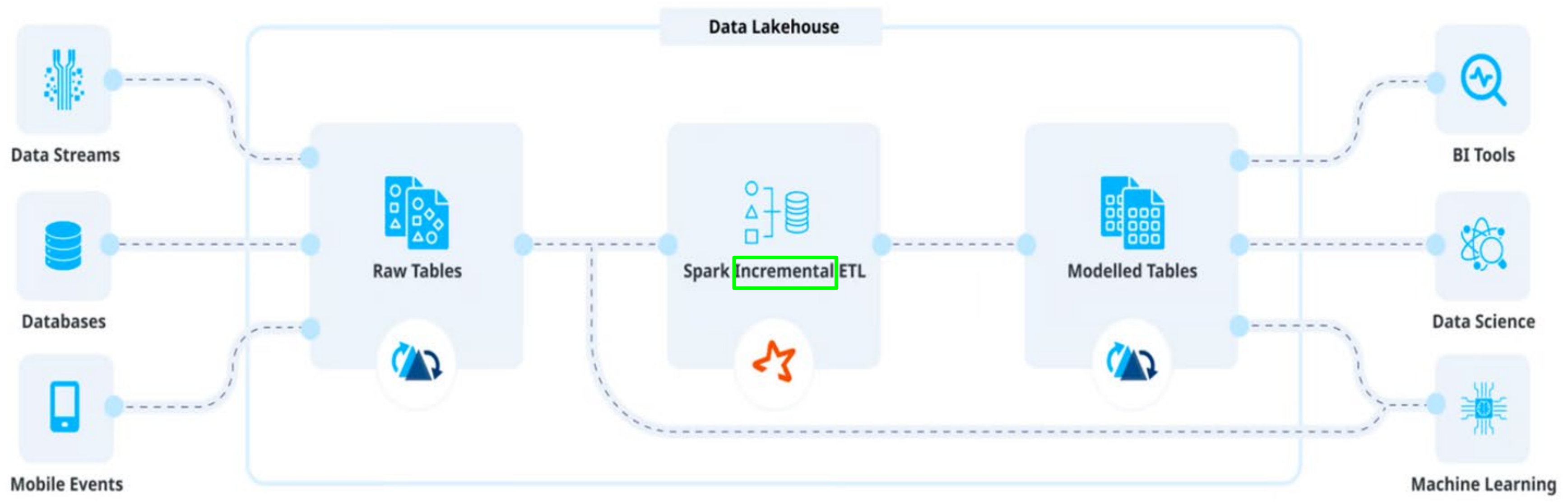
Agenda

- Uber data architecture
- Upserts challenges
- Apache Parquet introduction
- Fast Copy-On-Write within Parquet
- Conclusion & future work

Uber Data Architecture



Uber Lakehouse Platform



Updates in Lakehouse

A dataset need to be updated for different use case

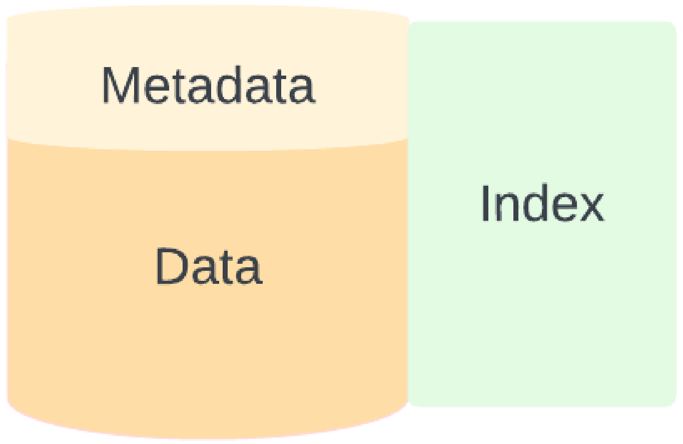
- Trip fare is changed
- Change Data Capture (CDC)
- Change data for compliance reason

Updating datasets is not that easy

- Append only system
- Structure data with compression, e.g. Parquet
- Locating affected data files is slow

Lakehouse Data Stack

- Data - A collection of files(e.g. Parquet) storing table's content
- Metadata - Info about a table schema, partition, file and snapshot details
- Index - Data structure to efficiently locate records within a table



Logic View of Table Update

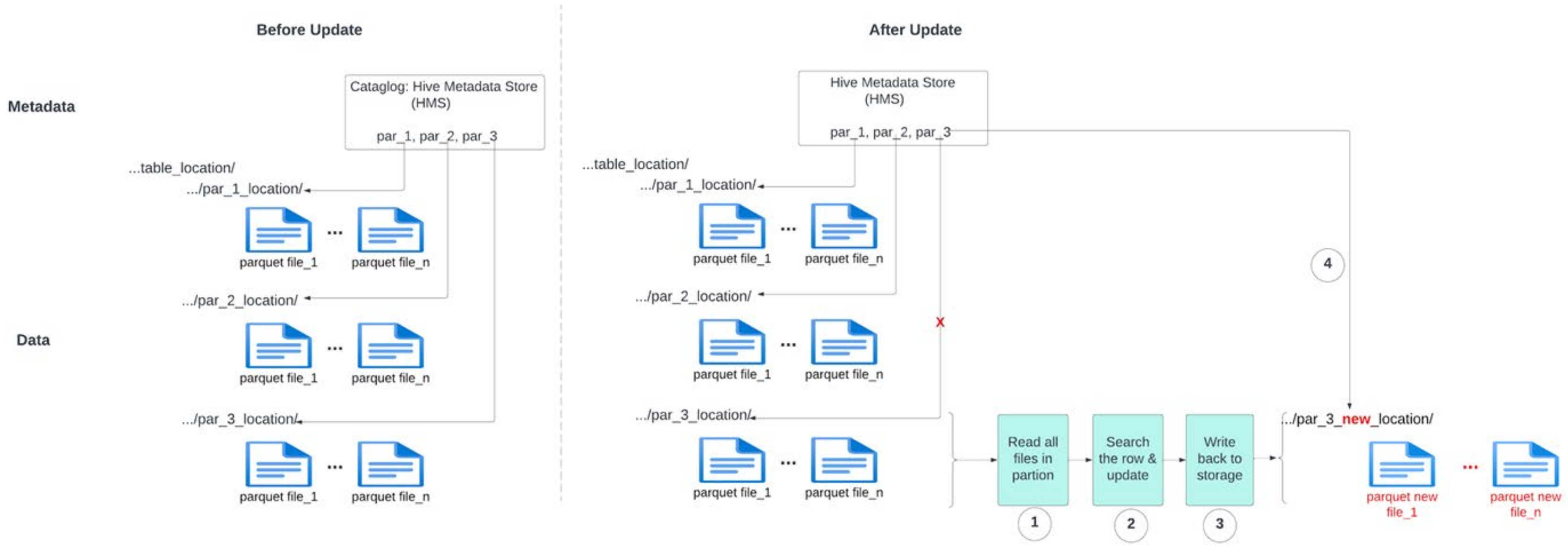
trip_uuid	…	trip_fair_tips	datestr
111-1111-1111-1111-1111111111111111	…	3	2023-06-30
22222-2222-2222-2222-22222222222222	…	8	2023-07-01

trip_uuid	…	trip_fair_tips	datestr
111-1111-1111-1111-1111111111111111	…	$\boxed{5}$	2023-06-30
22222-2222-2222-2222-22222222222222	…	8	2023-07-01

change trip_fair_tips to 5$ where trip_uuid = 111-1111..’ and datestr = ‘2023-06-30’

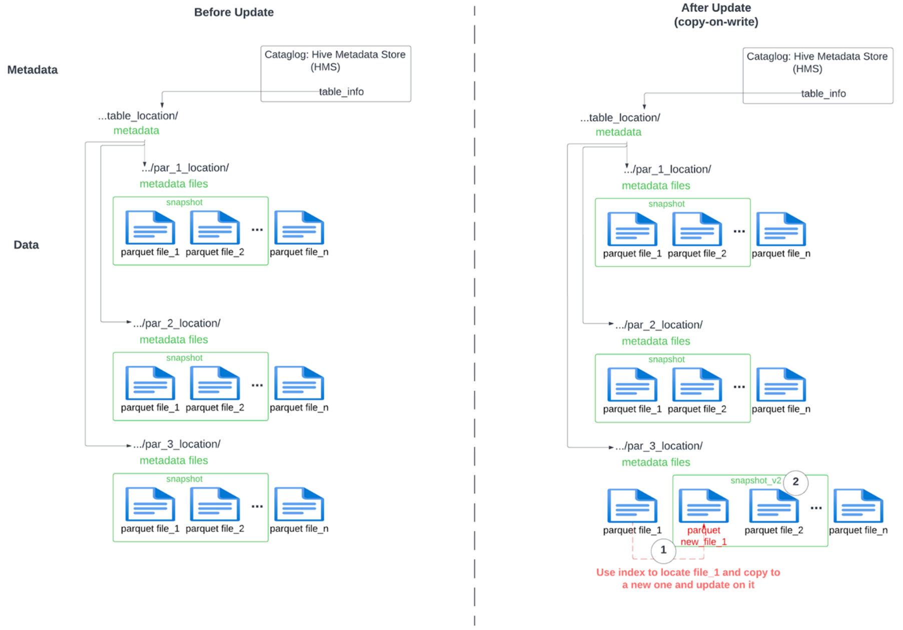
Update in Plain Hive Table Format

update trip_fair_tips to 5$ where trip_uuid = ‘345-2342…..’ and datestr = ‘2023-06-30’



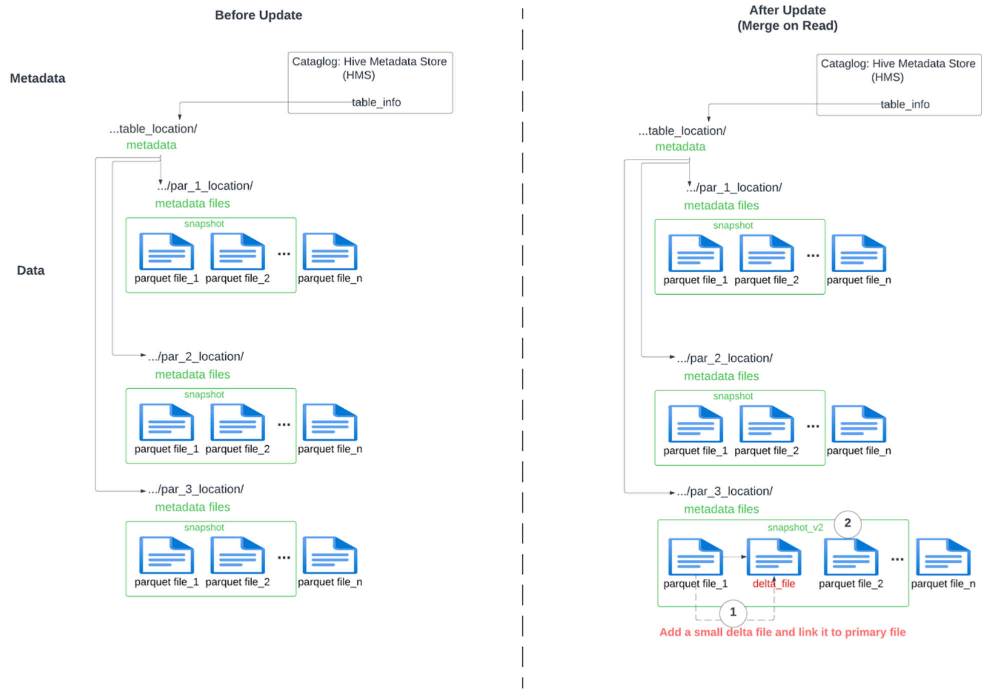
Update with Table Format (Copy-On-Write)

update trip_fair_tips to 5$ where trip_uuid = ‘345-2342…..’ and datestr = ‘2023-06-30’



Update with Table Format (Merge-On-Read)

```
update trip_fair_tips to 5$ where trip_uuid = ‘345-2342…..’ and datestr = ‘2023-06-30’
```



Comparison Copy-On-Write and Merge-On-Read

Copy-on-Write (CoW)

- Modifications create entirely new copies of the affected data
- Lead to increased storage usage
- Slower for rewriting, faster for reading

Merge-on-Read (MoR)

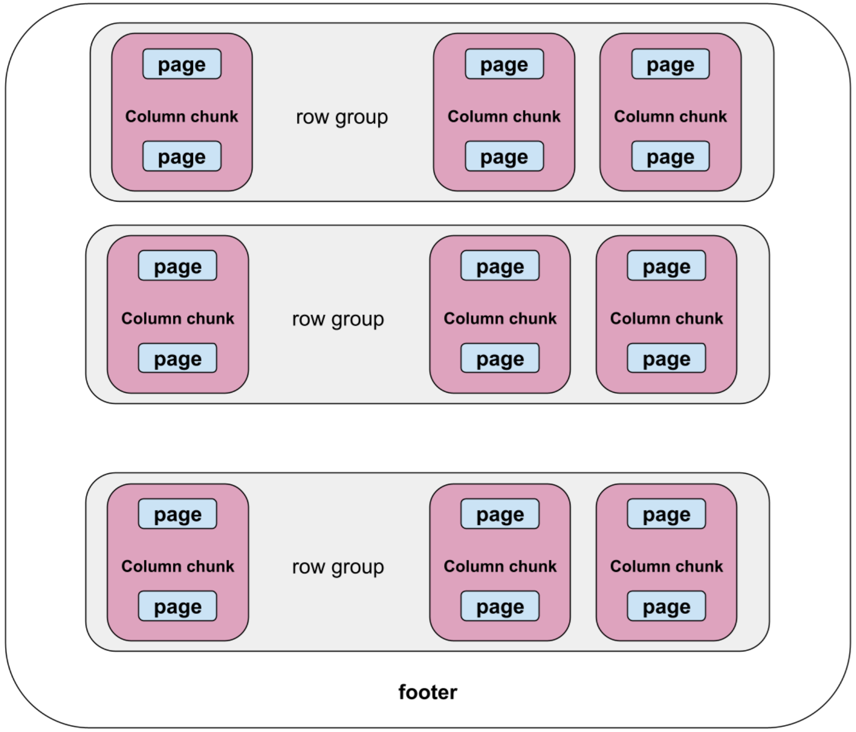
- Append changes in the form of delta files, avoiding complete rewrites
- Reader need to merge
- Slower for reading, faster for writing

Some use cases prefer copy-on-write, e.g right-to-be-forgotten

Large scale use cases of CoW is challenge!!!

Apache Parquet Introduction

- A columnar storage file format for big data processing
- Stores complex nested data structures in a highly efficient and compressed manner
- Widely used in the big data ecosystem, supporting various processing frameworks



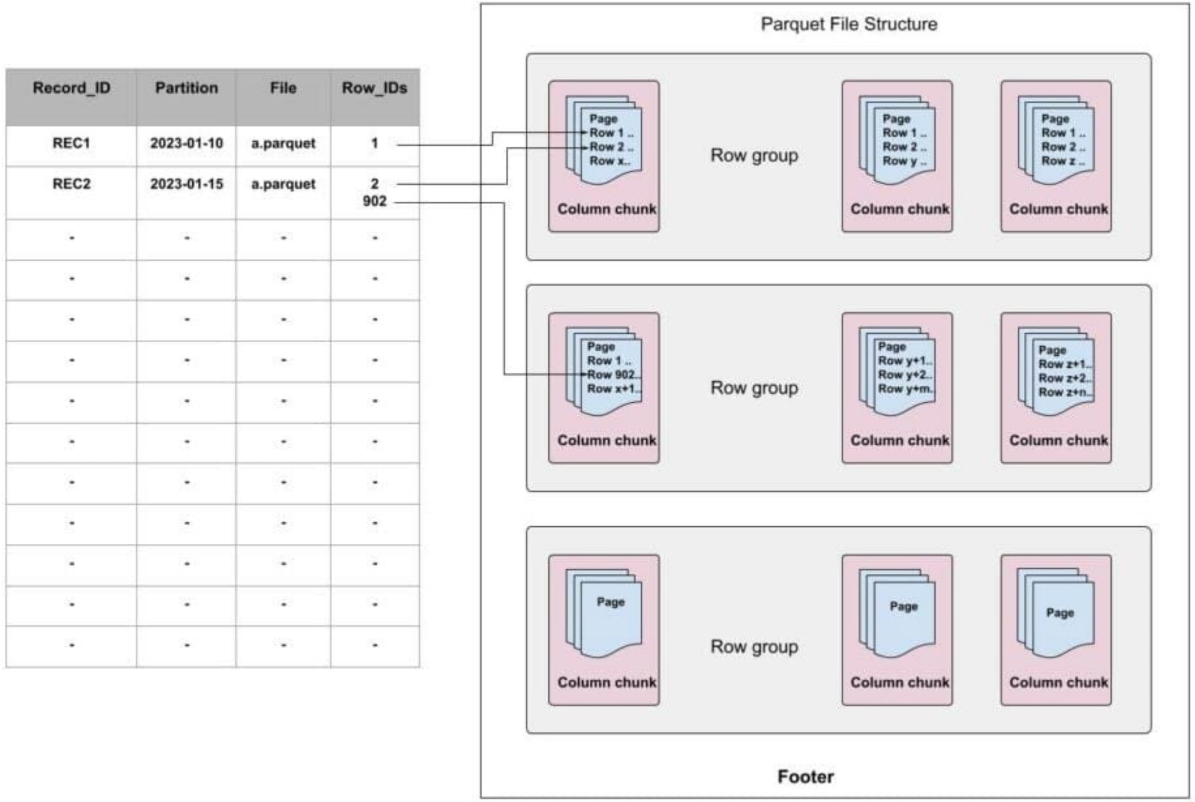
Introduce Row-Level Secondary Index

- Each entry of the index table pointing to Parquet internal structure: page, rowgroup etc

- Locate data record in a table **precisely**: which parquet page has it.

- Make fast copy-on-write possible

- But more expensive for storing index



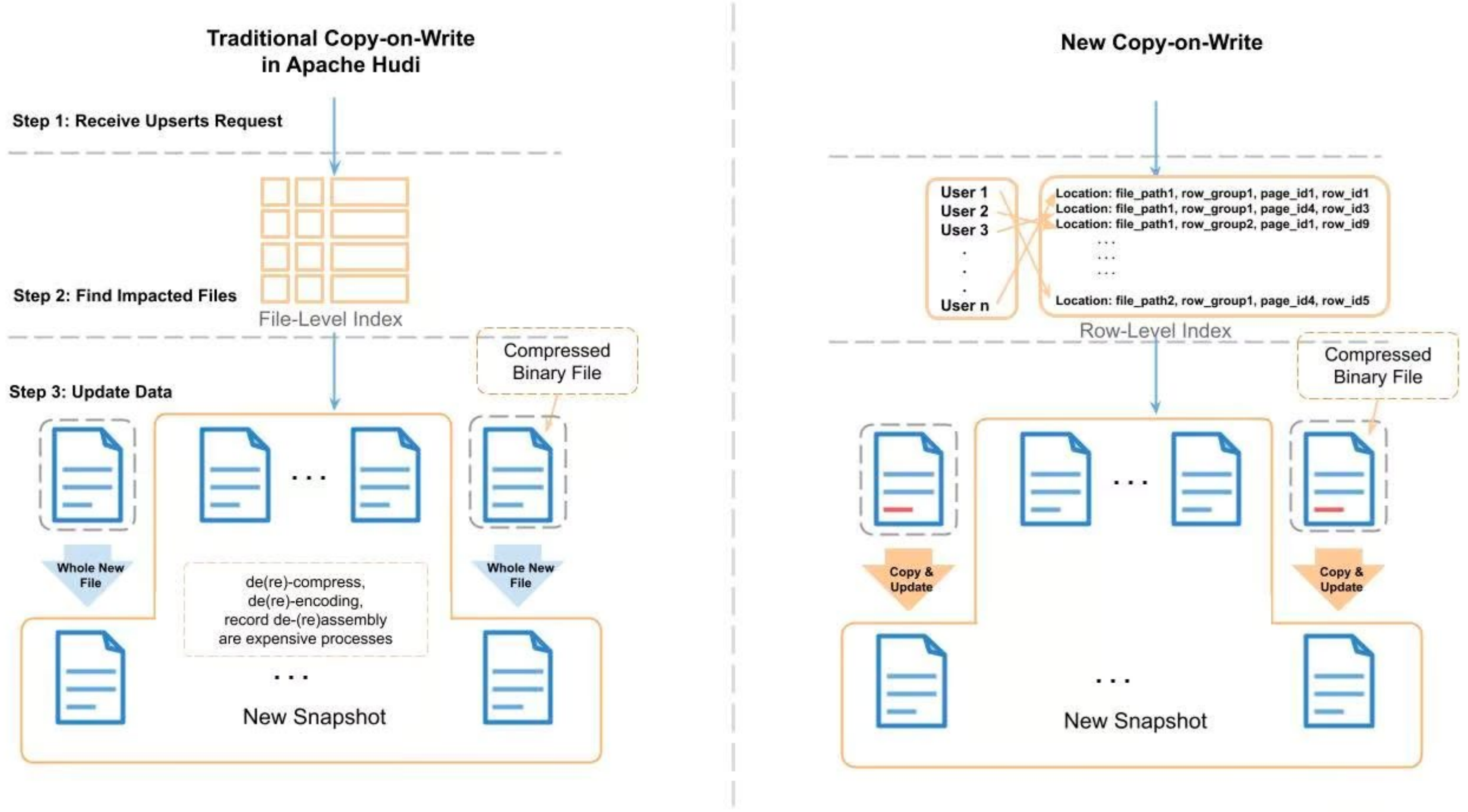
Record_ID	Partition	File	Row_IDs
REC1	2023-01-10	a.parquet	1
REC2	2023-01-15	a.parquet	2 902
-	-	-	-
-	-	-	-
-	-	-	-
-	-	-	-
-	-	-	-
-	-	-	-
-	-	-	-
-	-	-	-
-	-	-	-
-	-	-	-
-	-	-	-

Parquet File Structure
Page Row 1 .. Row 2 .. Row x..
Row group
Column chunk
Page Row 1 .. Row 2 .. Row y ..
Column chunk
Page Row 1 .. Row 2 .. Row z ..
Column chunk
Page Row 1 .. Row 902.. Row x+1..
Row group
Column chunk
Page Row y+1.. Row y+2.. Row y+m..
Column chunk
Page Row z+1.. Row z+2.. Row z+n..
Column chunk
Page
Row group
Column chunk
Page
Column chunk
Page
Column chunk
Footer

Introducing Copy-on-Write in Apache Parquet

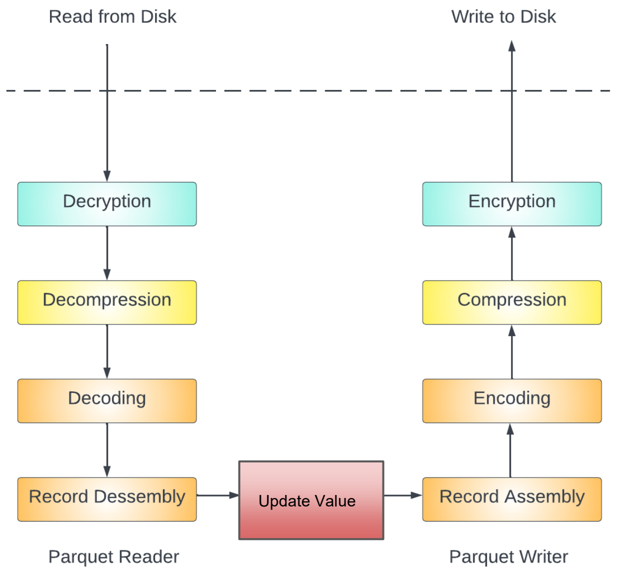
- Improvement in copying and rewriting a new parquet file
- Utilize row-level index to **accurately** locate which Parquet pages have the records to be updated
- Only decoding/decompress the pages that need to be updated
- Bytebuffer copy those not needed pages

Copy-On-Write within Apache Parquet

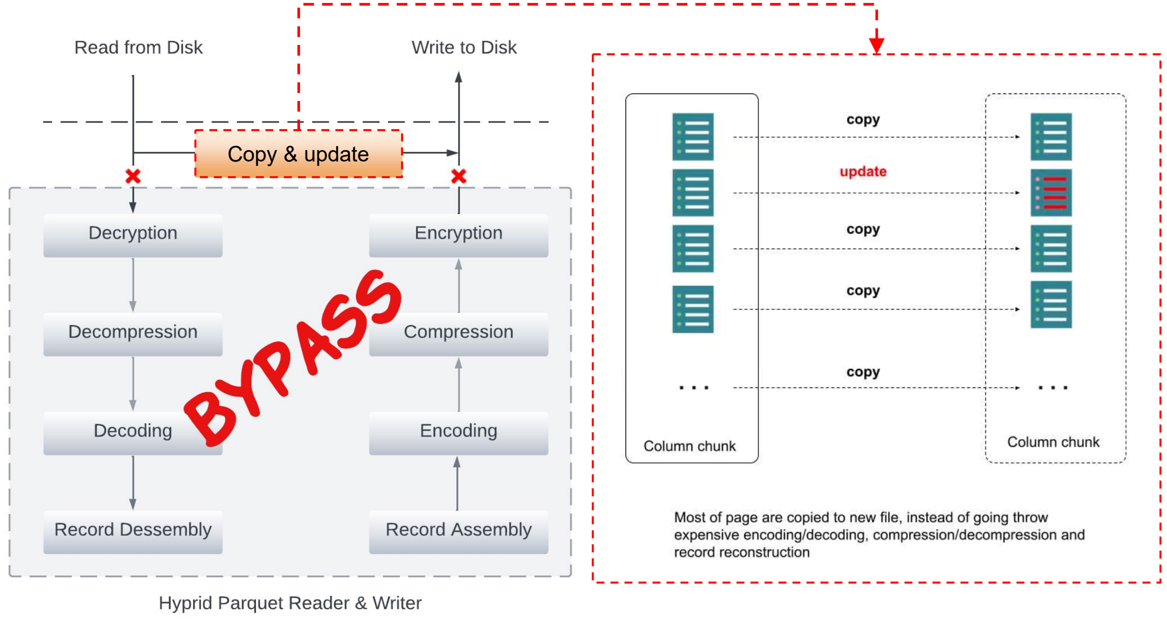

Traditional Copy-on-Write in Apache Hudi
Step 1: Receive Upserts Request
Step 2: Find Impacted Files
File-Level Index
Step 3: Update Data
Compressed Binary File
Whole New File
Whole New File
de(re)-compress, de(re)-encoding, record de-(re)assembly are expensive processes
New Snapshot
New Copy-on-Write
User 1
User 2
User 3
User n
Location: file_path1, row_group1, page_id1, row_id1
Location: file_path1, row_group1, page_id4, row_id3
Location: file_path1, row_group2, page_id1, row_id9
Location: file_path2, row_group1, page_id4, row_id5
Row-Level Index
Compressed Binary File
Copy & Update
Copy & Update
New Snapshot

Copy & Update

Traditional



New



Most of page are copied to new file, instead of going throw expensive encoding/decoding, compression/decompression and record reconstruction

Limitations

- The storage size of row-level index is pretty large
- Updates to data are not reflected in the index realtime

Conclusion

Efficient upserts are critical for data lakehouse.

Speed remains a challenge, when volume scales up

Fast copy-on-write within Apache Parquet files with row-level indexing

- Skip unnecessary data pages reads and writes efficiently
- Improve the speed of upserts

Future Work

- Improve the large storage size issue of row-level index
- Integrate the row-level index and fast copy-on-write feature to table formats

Q & A

Send questions to: shangxinli@apache.org