hulu

Apache Hudi 101

Table of Contents

hulu

What is Hudi

Apache Hudi(Hadoop Update, Delete & Incremental) is a storage framework over DFS(HDFS or cloud store).

The main abstraction of Hudi is a Hudi table, which provides

- Fast & scalable near real-time stream ingestion
- Record-level update/delete support
- Optimized storage format for analytical workload
- Snapshot/Incremental query support

A quick tour of Hudi

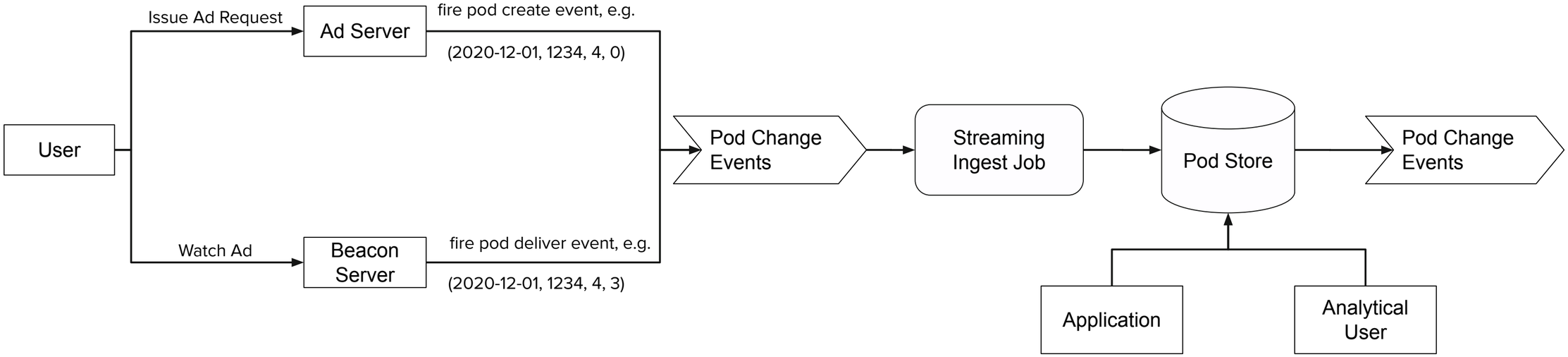
Suppose you want to build a Pod Store to track all pods' supply & delivery, i.e. you want to store & update below information for all pods.

```
class Pod(createTime: Date,
          guid: String,
          filledSlots: Int,
          deliveredSlots: Int,
          lastUpdateTs: Long)
```

A quick tour of Hudi

With the help of Hudi, you can end up with a following pipeline.

- A change event queue to capture all pod change events
- A spark streaming job to ingest changes to a Hudi table in near real-time
- Application & analytical user can query the latest snapshot or a snapshot of a specific time
- A downstream pipeline can execute an increment query to chain the pod change events



A quick tour of Hudi

In the streaming ingest job, you can use Hudi Spark Datasource Writer to write a data frame as a Hudi table.

```
df.write.format("hudi")
  .option(PRECOMBINE_FIELD_OPT_KEY, "lastUpdateTs")
  .option(RECORDKEY_FIELD_OPT_KEY, "guid")
  .option(PARTITIONPATH_FIELD_OPT_KEY, "createTime")
  .option(TABLE_NAME, "pod_store")
  .mode(Append)
  .save(basePath)
```

A quick tour of Hudi

Application or analytical user can query the Hudi table by Hudi Datasource Reader. Hudi table have a meta-column to provide the record commit time.

```
val df = spark.read.format("hudi")
                  .load(basePath + "/*/*")
df.createOrReplaceTempView("pod_store")

spark.sql("""select
               guid,
               filledSlots,
               deliveredSlots,
               _hoodie_commit_time
            from pod_store where createDate >= today()
          """).show()
```

hulu

A quick tour of Hudi

User also can execute an incremental query to get the changed records in a time range.

```
val df = spark.read.format("hudi")
  .option(QUERY_TYPE_OPT_KEY, QUERY_TYPE_INCREMENTAL_OPT_VAL)
  .option(BEGIN_INSTANTTIME_OPT_KEY, beginTime)
  .option(END_INSTANTTIME_OPT_KEY, endTime)
  .load(basePath)
```

By setting begin time to "000" (denoting the earliest commit time), user can query a table snapshot at a specific time.

```
val df = spark.read.format("hudi")
  .option(QUERY_TYPE_OPT_KEY, QUERY_TYPE_INCREMENTAL_OPT_VAL)
  .option(BEGIN_INSTANTTIME_OPT_KEY, "000")
  .option(END_INSTANTTIME_OPT_KEY, endTime)
  .load(basePath)
```

hulu

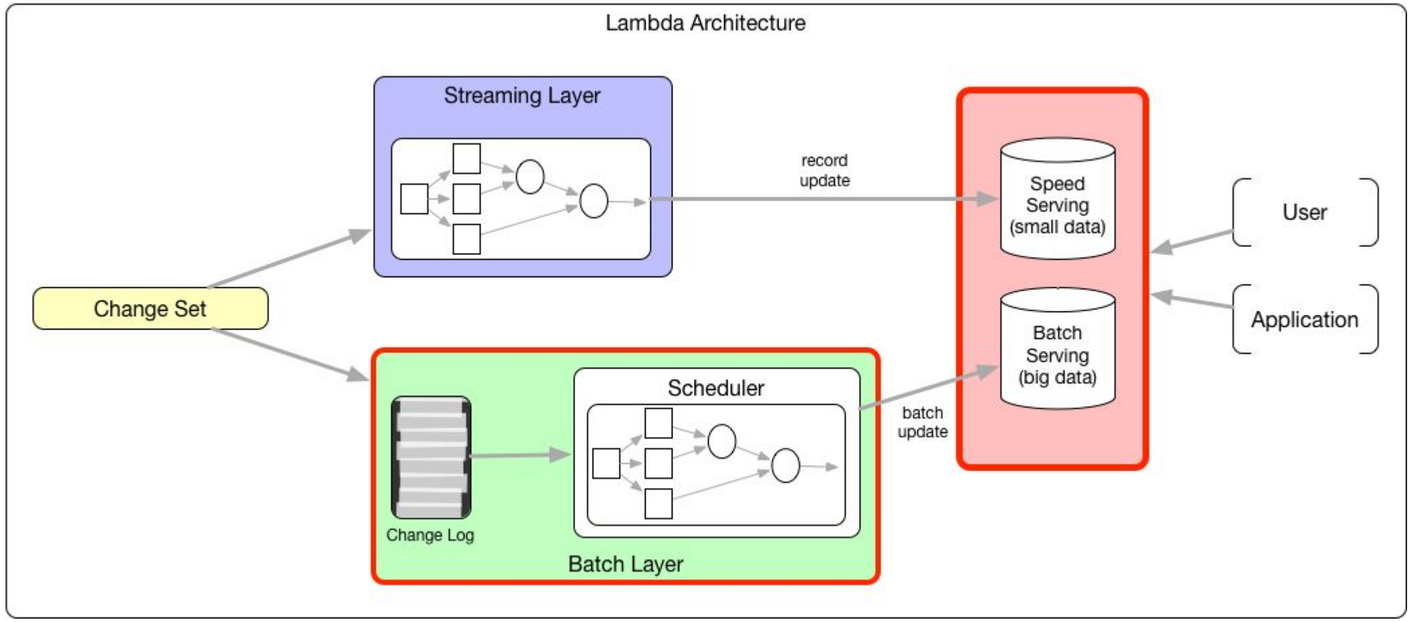
Why we need to use Hudi

By using a Hudi table rather than a plain Parquet/ORC table, we gain the benefits

- Record-level update/delete capability
- Faster data ingestion by low IO cost
- Optimized storage by small files & data skew avoidance

Why we need to use Hudi

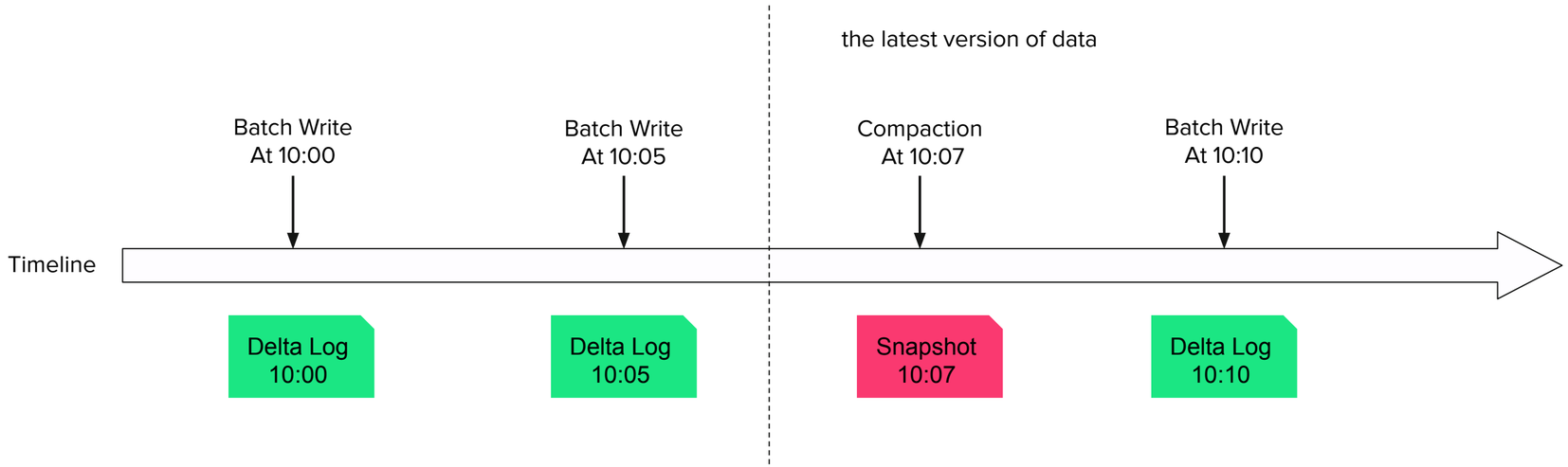
On the top of these benefits, we can have HDFS as a unified serving layer, which can simplify data processing architecture much!



In a traditional data processing architecture, the batch processing & streaming processing varies in update style, data volume & query performance. Developers have to use different serving layer for each(normally use HDFS for batch layer & use a database for streaming layer). As Hudi make HDFS satisfy the streaming processing requirement, HDFS can be the unified serving layer for both batch processing & streaming processing.

How Hudi achieve these

- Hudi ingest data as a series of small batches
- For each batch, Hudi write data as a delta log & put a record in metadata
- Delta logs can be compacted to a snapshot to improve query performance
- Hudi maintain its metadata as a "timeline", which records all batches & compactions with their complete time
- By the metadata & snapshots & delta logs, Hudi can provide instantaneous views at any specific time



How can we use Hudi

- hudi-spark - a spark library which provides a custom datasource to write/read data as a Hudi table, and sync metadata to Hive optionally.

- HoodieDeltaStreamer - a spark app to ingest data from a kafka topic or a change stream of DFS/Hive table, to a Hudi table.

- Hudi Hive Bundle - a hive library which provides custom InputFormat to read data of a Hudi table. This library also can be used to integrate with Hive-compatible query engines like Spark SQL/Presto/Impala.

- Hudi Utils Bundle - CLI & spark apps used to provide utilities like compaction, clean...

hulu

Hudi Use Cases

- Near real-time Data Ingestion
  Hudi enable faster & efficient data ingestion on HDFS, while also avoid small file/data skew problems.

- Near Real-time Analytics
  Hudi enable real-time analytics with a minutes level of data freshness.

- Incremental Processing Pipeline
  By Hudi incremental queries, Hudi can be used to build a chained workflow to process data incrementally, which improve much processing efficiency & end-to-end latency.

- Data Dispersal From DFS
  By Hudi incremental queries, changed data on HDFS can be efficiently dispersed to other systems, without relying on a redundant storage(like a kafka).

hulu

What Hudi can't do

- Real-time ingestion
  Considering its complex ingest path, Hudi should be unsuitable for real-time ingestion.
  Metadata also will be a problem in this case.

- OLTP Workload
  It's not efficient to query a specific row of a Hudi table, since Hudi doesn't build any index,
  nor use memory to cache data.

hulu

THANK YOU