Cassandra Data Ingestion

Jay Zhuang & Simon Zhou
Software Engineers, Uber

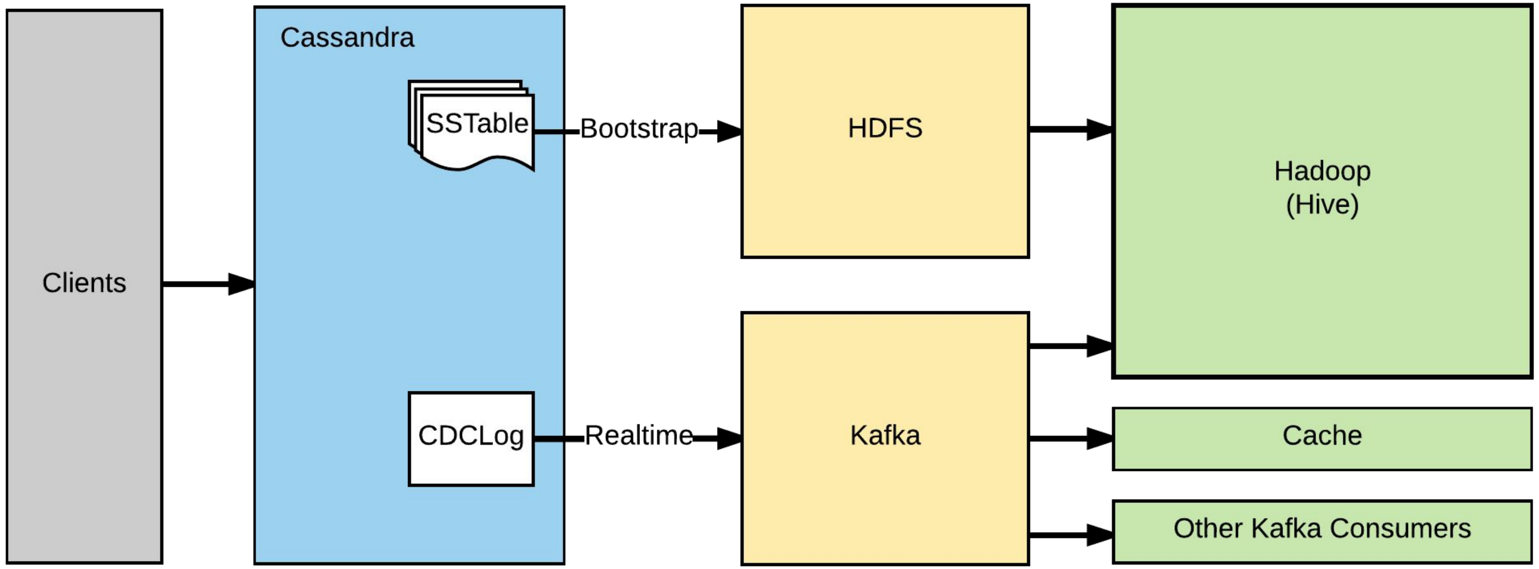
September 26, 2017

Agenda

1. Cassandra Ingestion Overview
2. Near-realtime Ingestion with CDC
    a. Current CDC implementation and limitations
    b. CDC as an interface
    c. Other Challenges
3. Bootstrap

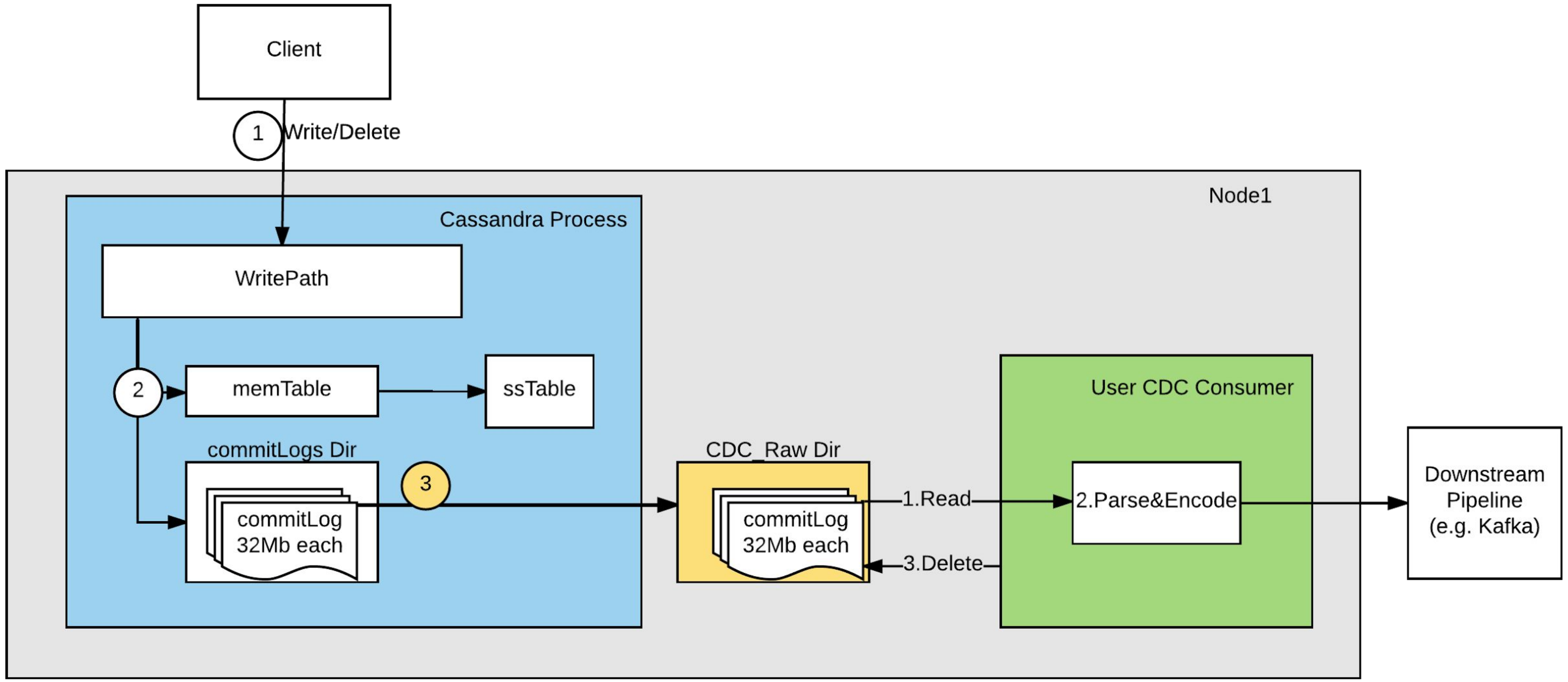
1. Cassandra Ingestion Overview

- Near RealTime Ingestion (CDC)
- Offline Ingestion (Bootstrap)



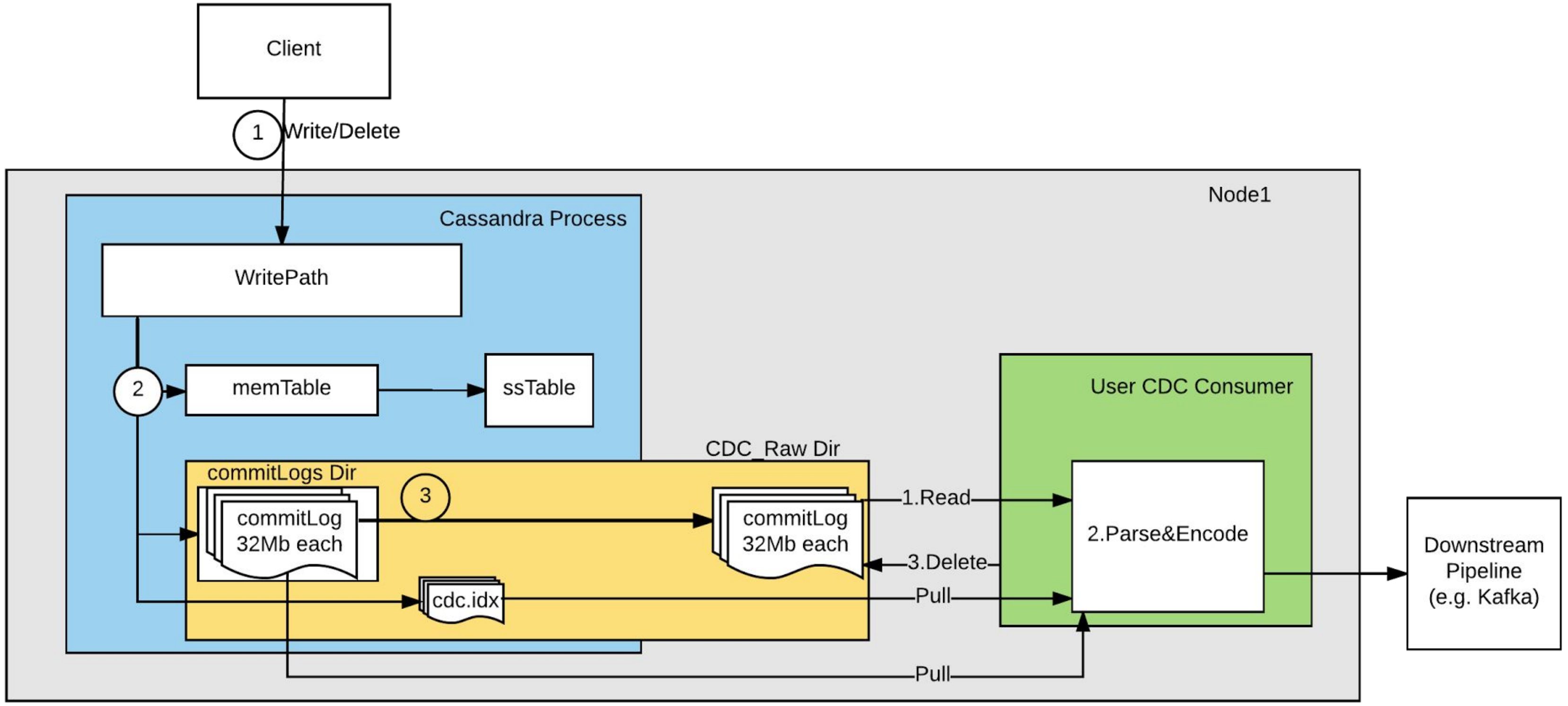
2. *Near* Real-time Ingestion with CDC

- CASSANDRA-8844



Update Lag Issue and Fix

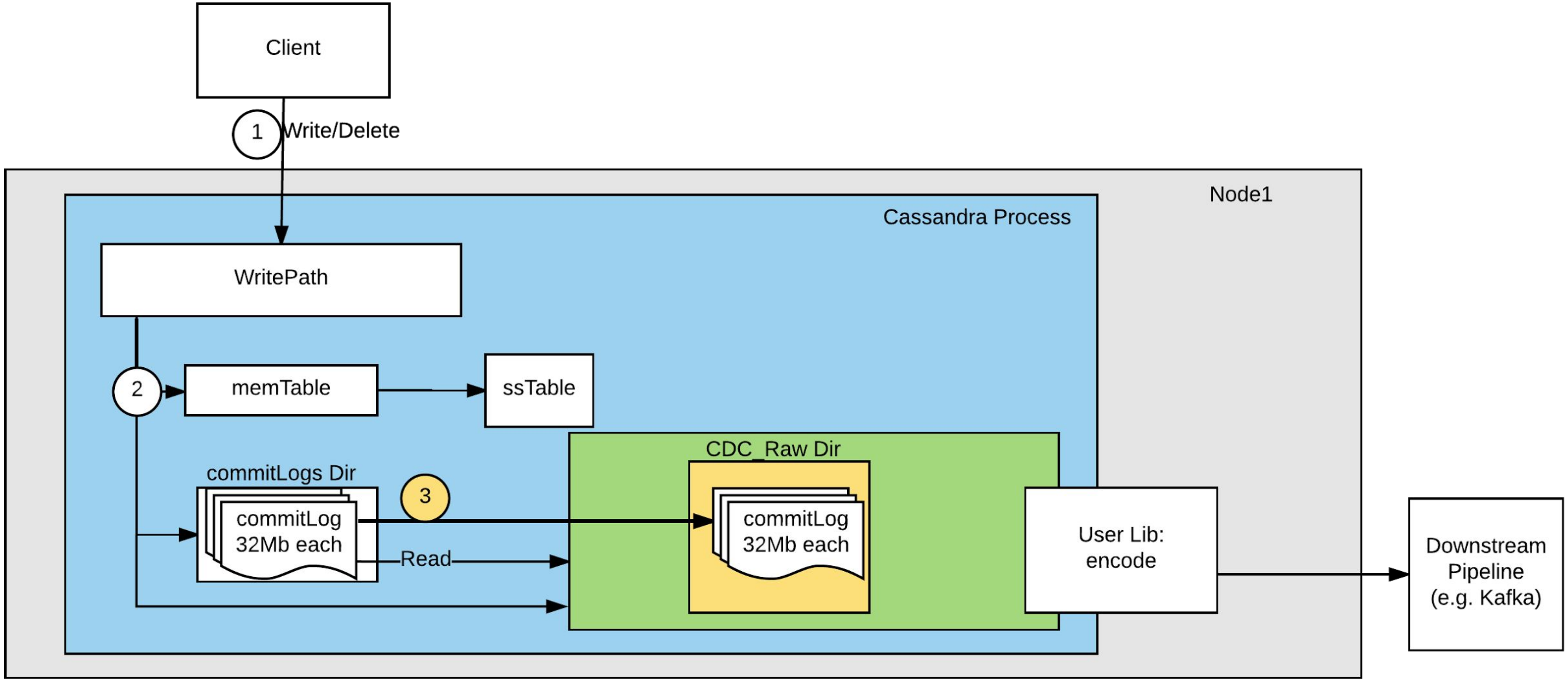
- CASSANDRA-12148



Limitations

- User interface is not friendly (with CommitLog file)
- Understand CommitLog file format (CDC log is commit log)
- Keep pulling idx files
- Read whole CommitLog whenever there’s any change
- CommitLog contains both CDC and non-CDC data

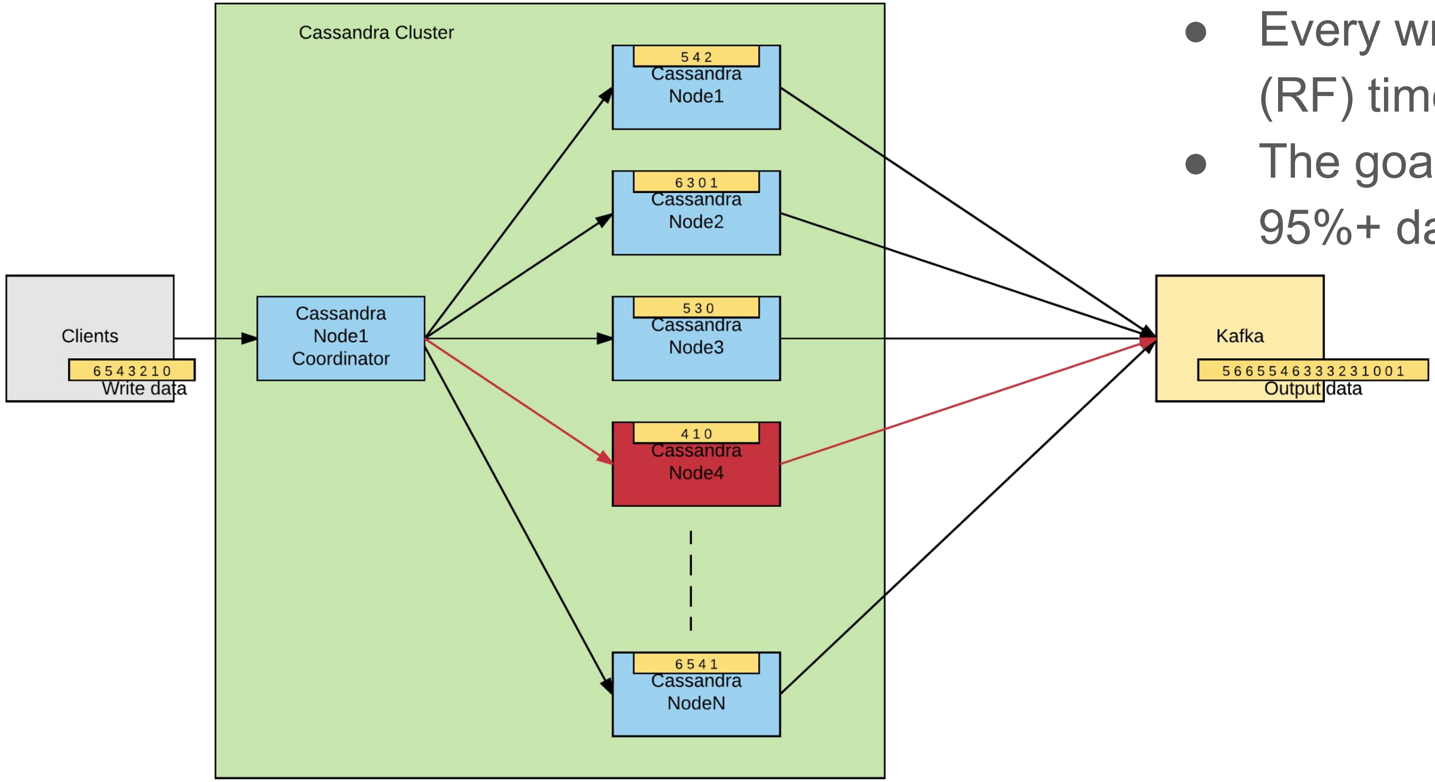
Proposal: CDC as an Interface



Pros vs. Cons

Pros	Cons
● Simplify the user interface; ● Avoid pulling idx file (get notification from CommitLogManagerCDC); ● Cassandra manages the lifecycle of commitLogs (no need to keep checking cdc_raw size); ● Easier to manage, deploy and monitor (just one process);	● Increase heap usage ● Increase CDC module internal complexity

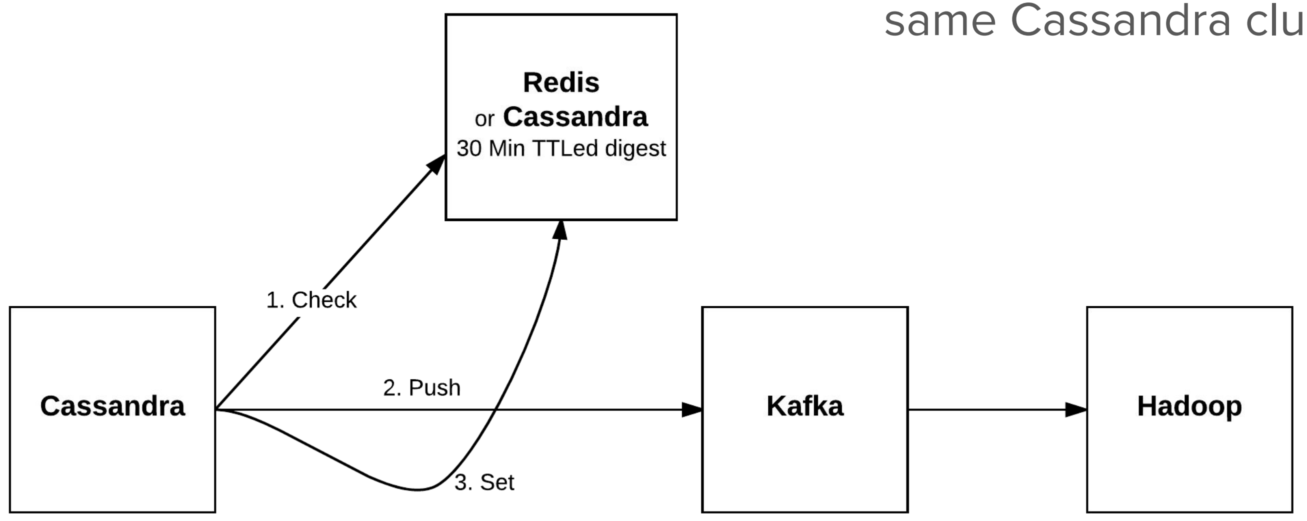
Challenge 1: Deduplication



- Every write will appear 3 (RF) times in Kafka
- The goal is to dedup 95%+ data

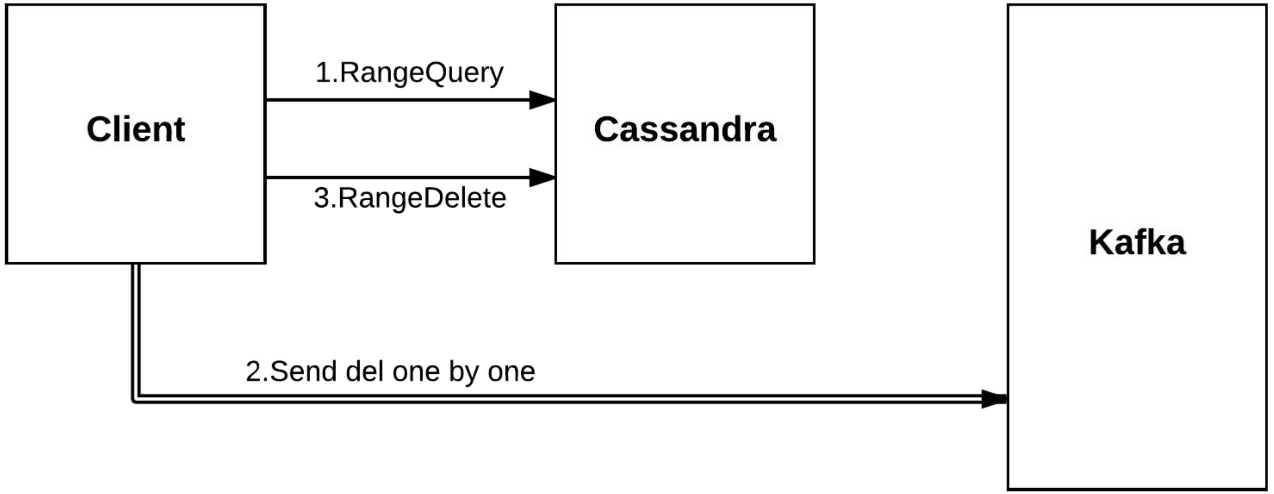
Cache Dedup

- Centralized cache system for dedup
- Store the message digest for 30 minutes
- The cache could be a keyspace from the same Cassandra cluster



Challenge 2: Range Delete

- Expand the range delete on client side: phantom data in the downstream
- Another solution is handling the range delete in downstream (by querying Hive)
  - As the CDC log order is not guaranteed, it will have the same phantom data issue
- Proposal: expand the data in CDC module (change the read path to exclude the current delete)



Challenges 3 & 4

- CDC Log compactor
  - Problem: the commit log has both CDC and non-CDC data
  - If the interface call failed, save the data to a failed-file and retry later (current our Kafka client has the same logic)
- Static column
  - Downstream system unlikely to handle it
  - No good solution but to block it

3. Bootstrap

Use cases:

- Onboard Cassandra tables with existing sstables
- Something wrong with the pipeline and re-bootstrap needs to happen
- Some tables don’t need CDC and infrequent bootstrapping is okay

Requirements:

- Bootstrap can finish within two days
- Further processing happens on HDFS as with other databases
- De-duplication

Bootstrap: Spark Cassandra Connector

Pros	Cons
● Easy to use & minimum client code. ● Native integration with Spark, which is used by our downstream job.	● Could affect production traffic. ● Read rate is slow (~30k rows/second) and may need up to 10 days to load all data. ● Hard to cover corner cases unless we read with ALL/LOCAL_ALL consistency level, or do manual repair.

Bootstrap -- Parse sstables directly from HDFS

Pros	Cons
● Straightforward	● Lots of details when parsing sstables ● Maintenance effort if storage engine updates ● Need to parse index file to process different parts of a large sstable in parallel

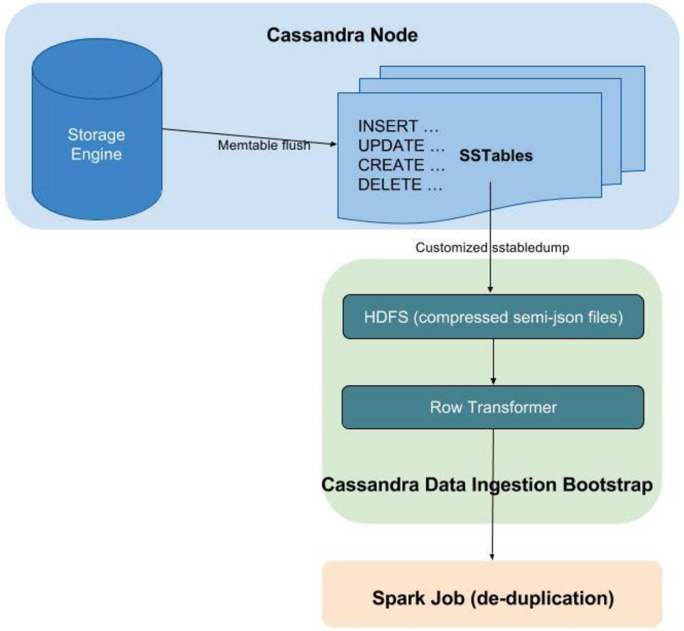
Bootstrap -- sstabledump

Pros	Cons
● Almost no implementation effort. ● Json output is easy for further processing.	● Output file is huge, 20x the original sstable file size ● Compression is ok but still Json file is not splittable ● The data in the Json output is organized in hierarchy partition -> row -> column which is not flattened

Bootstrap -- Customized sstabledump

A few improvements:

- Only include necessary info
- Compression (gzip)
- Split output of large sstable file
- Primary key encoding
- Automatic and incremental processing



UBER