Marmaray: Spark Ingestion and Dispersal

May 21, 2019
Omkar Joshi

omkar@uber.com

- Software engineer on Hadoop
  Data Platform team
- 2.5 years at Uber
- Hadoop Yarn committer
- Architected Object store & NFS
  solutions at Hedvig
- Enjoy gardening & hiking in free
  time

Photo from Hawaii trip!!
01 Mission
02 Overview
03 Need for Ingestion & Dispersal Framework
04 Deep Dive
05 Completeness & Data Deletion
06 Configuration & Monitoring of Jobs
07 Learnings
01 Mission
Uber Apache Hadoop Platform Team Mission

Build products to support reliable, scalable, easy-to-use, compliant, and efficient data transfer (both ingestion & dispersal) as well as data storage leveraging the Apache Hadoop ecosystem.

Apache Hadoop is either a registered trademark or trademark of the Apache Software Foundation in the United States and/or other countries. No endorsement by The Apache Software Foundation is implied by the use of this mark.
Overview

- Any Source to Any Sink
- Ease of onboarding
- Business impact & importance of data & data store location
- Suite of Apache Hadoop ecosystem tools
Open Sourced in September 2018

https://github.com/uber/marmaray

Blog Post:

https://eng.uber.com/marmaray-hadoop-ingestion-open-source/
03 Need for Ingestion & Dispersal Framework
Marmaray (Ingestion): Why?

- Raw data needed in Apache Hadoop data lake
- Ingested raw data -> Derived Datasets
- Reliable and correct schematized data
- Maintenance of multiple data pipelines
Marmaray (Dispersal): Why?

- Derived datasets in Hive
- Need arose to serve live traffic
- Duplicate and ad hoc dispersal pipelines
- Future dispersal needs
Marmaray: Main Features

- Automated schema management
- Integration w/ monitoring & alerting systems
- Fully integrated with workflow orchestration tool
- Extensible architecture
- Open sourced
Marmary: Uber Eats Use Case

Uber Eats recommendations are powered through Marmary dispersal
Hadoop Data Ecosystem at Uber
Hadoop Data Ecosystem at Uber

Apache Kafka, Cassandra, Spark, and HDFS logos are either registered trademarks or trademarks of the Apache Software Foundation in the United States and/or other countries. No endorsement by The Apache Software Foundation is implied by the use of these marks.
04 Deep Dive
High-Level Architecture

Input Storage System → Source Connector → Converter1 → Converter 2 → Sink Connector → Output Storage System

Chain of converters

Datafeed Config Store

Error Tables

Metadata Manager (Checkpoint store)

Converter1 → Converter 2

M3

Work Unit Calculator

Monitoring System
High-Level Architecture

Input Storage System

Source Connector

Converter1

Converter 2

Sink Connector

Output Storage System

Schema Service

Datafeed Config Store

Metadata Manager
(Checkpoint store)

Error Tables

Chain of converters

Work Unit Calculator

M3

Monitoring System
Schema Service

- Get Schema by Name & version
- Get Schema Service Reader
  - Reader / Decoder
    - Binary Data
    - Generic Record
  - Writer / Encoder
    - Binary Data
    - Generic Data
Metadata Manager

- **init()**
  - Called on Job start

- **Set (key, value)**
  - Called 0 or more times

- **Get(key) -> value**
  - Called 0 or more times

**Persistent Storage (ex. HDFS)**

- Called after Job finish

**In-Memory Copy**

**Different Job DAG Components**
Fork Operator

- Avoid reprocessing input records
- Avoid re-reading input records (or in Spark, re-executing input transformations)
Fork Operator & Fork Function

Input Records

Fork Function

Tagged Records

Success Filter

Schema Conforming records

Failure Filter function

Error Records

Persisted using Spark's disk/memory persistence level
Easy to Add New Source & Sink

Kafka → Data lake with GenericRecord → Hive
Kafka → Data lake with GenericRecord → S3
Kafka → Data lake with GenericRecord → Cassandra

New Source → Data lake with GenericRecord
Support Writing into Multiple Systems

Kafka → Data lake with GenericRecord → Hive Table 1, Hive Table 2
JobDag & JobDagActions

JobDAG

Job Dag Actions

- Report metrics for monitoring
- Register table in Hive
Need for Running Multiple JobDags Together

- Frequency of data arrival
- Number of messages
- Avg record size & complexity of schema

- Spark job has Driver + executors (1 or more)
- Not efficient model to handle spikes
- Too many topics to ingest. 2000+
JobManager

- Single Spark job for running ingestion for 300+ topics
- Executes multiple JobDAGs
- Manages execution ordering for multiple JobDAGs
- Manages shared Spark context
- Enables job and tier-level locking

Ingesting kafka-topic 1 (JobDAG 1)

Ingesting kafka-topic N (JobDAG N)

Waiting Q for JobDAGs
05 Completeness & Data Deletion
Completeness

Source (Kafka)

Sink (Hive)
Completeness

Why not run queries on source and sink dataset periodically?
Possible for very small datasets
Won’t work for billions of records; very expensive!!

Bucketizing records
How about creating time based buckets say for every 2min or 10min.
Count records at source and sink during every runs
Does it give 100% guarantee?? No but w.h.p. it is close to it.
Completeness - High-level Approach

- **Input Record (IR)**
- **Input Success Record (ISR)**
- **Input Error Record (IER)**
- **Output Error Record (OER)**
- **Output Records (OR)**

Diagram:
- **Kafka** → **Src Converter** → **Error Table** → **Sink Converter** → **Hoodie (Hive)**

Legend:
- IR
- IER
- OER
- OR
Previous Way of Storing Kafka Data in Apache Hadoop

Kafka topic1

- 2014
- 2015
  - 01
  - 02
- 2018
  - 08
  - 06

Latest Date Partition

Stitched Parquet files (~4GB) (~400 files per partition)

Non-stitched Parquet files (~40MB) (~20-40K files per partition)
Data Deletion (Kafka)

- Old architecture is designed to be append/read only
- No indexes
  - Need to scan entire partition to find out if record is present or not
- Only way to update is to rewrite entire partition
- GDPR requires all data to be cleaned up once user requests deletion
- This is a big architectural change
Hudi Data Layout

- Kafka Topic
- 2014
- 2015
- 2018
- Updates
- .hoodie
- ts1.commit
- ts2.commit
- ts3.commit
- f1_ts1.parquet
- f2_ts1.parquet
- f3_ts1.parquet
- f4_ts1.parquet
- f5_ts2.parquet
- f6_ts2.parquet
- f7_ts2.parquet
- f8_ts3.parquet
- ts1.commit
- ts2.commit
- ts3.commit
06 Configuration & Monitoring of Jobs
Configuration

common:
hadoop:
  fs.defaultFS: "hdfs://namenode/"
hoodie:
  table_name: "mydb.table1"
  base_path: "/path/to/my.db/table1"
  metrics_prefix: "marmaray"
  enable_metrics: true
  parallelism: 64
  source:
    topic_name: "topic1"
    max_messages: 1024
    read_parallelism: 64
kafka:
  conn:
    bootstrap.servers: "kafkanode1:9092,kafkanode2:9092"
    fetch.wait.max.ms: 1000
    socket.receive.buffer.bytes: 5242880
    fetch.message.max.bytes: 20971520
    auto.commit.enable: false
    fetch.min.bytes: 5242880
Monitoring & Alerting
Learnings

- **Spark**
  - Off heap memory usage of spark and YARN killing our containers
  - External shuffle server overloading
  - Parquet
  - Better record compression with column alignments

- **Kafka**
  - Be gentle while reading from kafka brokers

- **Cassandra**
  - Cassandra SSTable streaming (no throttling), no monitoring
External Acknowledgments

https://gobblin.readthedocs.io/en/latest/
Useful Links

https://github.com/uber/marmaray
https://github.com/uber/hudi
https://eng.uber.com/michelangelo/
https://eng.uber.com/m3/
https://eng.uber.com/marmaray-hadoop-ingestion-open-source/
Thank you

Questions: email ospo@uber.com

Follow our Facebook page:
www.facebook.com/uberopensource