Genji
Framework for building resilient near-realtime data pipelines
About Us

Swami Sundaramurthy
@swamisun (Twitter, LinkedIn)

Mark Cho
@markCho, markcho (LinkedIn)

Stream Platform Team
Data | Pinterest
Overview

• Motivation
• Challenges
• Architecture
• Use Cases
• Accomplishments
Overview

• Motivation
• Challenges
• Architecture
• Use Cases
• Accomplishments
Journey

Realtime anti-spam
Ads Reporting
Ads Reporting

Advertiser1  Advertiser2  ...  AdvertiserN
Ads Reporting

Advertiser1  Advertiser2  ...  AdvertiserN

Ad Tech Partners
Ads Reporting

Advertiser1  Advertiser2  ...  AdvertiserN

Ad Tech Partners

Google  Facebook  ...  Pinterest
Ads Reporting

Advertiser1 ➔ Ad Tech Partners ➔ Google
Advertiser2 ➔ Ad Tech Partners ➔ Facebook
... ➔ Ad Tech Partners ➔ ...
AdvertiserN ➔ Ad Tech Partners ➔ Pinterest

8 AM EST
$2.9M
(fictional number)
Ads Reporting

Advertiser1  Advertiser2  ...  AdvertiserN

Ad Tech Partners

Google  Facebook  ...  Pinterest

8 AM EST
$2.9M (fictional number)
Ads Reporting

Ad Tech Partners

Advertiser1  Advertiser2  …  AdvertiserN

Google  Facebook  …  Pinterest

$1.2 M  $1.0M  $0.7M ?

(fictional number)  (fictional number)  (fictional number)
Legacy Data Flow
Legacy Data Flow

Frontend → Backend → Kafka → S3

hourly
Legacy Data Flow

- Frontend
- Backend
- Kafka
- S3
- Daily Ads Jobs
- HBase

Flow:
- Backend to Kafka (hourly)
- Kafka to S3
- S3 to Daily Ads Jobs
- Daily Ads Jobs to HBase (8 hrs)

Timeline:
- 1 AM
Overview

• Motivation
• Challenges
• Architecture
• Use Cases
• Next Steps
• Results & Recap
Challenges

- Data
  - Low input topic cardinality
  - 20+ TB per hour
  - Jobs use few columns
Challenges

- Data
  - Low input topic cardinality
  - 20+ TB per hour
  - Jobs use few columns
- Code sharing
Challenges

- Data
  - Low input topic cardinality
  - 20+ TB per hour
  - Jobs use few columns
- Code sharing
- Schema evolution
Challenges

• Data
  • Low input topic cardinality
  • 20+ TB per hour
  • Jobs use few columns
• Code sharing
• Schema evolution
• Cascading workflow
Need for framework

**Infrastructure (AWS)**
- EC2, S3, EBS, Route53, …

**Platform (Overwatch)**
- Mesos, Marathon, Chronos, Hive, HDFS, Spark, Presto, Redash

**Framework (Genji)**
- Voracity, Structured Columnar Tables, libs

**Applications**
- Ads Reporting, Traffic
Overview

• Motivation
• Challenges
• Architecture
• Use Cases
• Accomplishments
Architecture Overview

- Stream ingestion and persistence
- Structured data store
- Downstream jobs
- Data visualization and exploration
Stream Data Ingestion

- Producers write structured data to Kafka
- Kafka acts as a buffer between online systems and offline systems
- Voracity (Spark Streaming) processes the data from Kafka and writes to Hive (HDFS)
Stream Data Ingestion

- Producers write structured data to Kafka
- Kafka acts as a buffer between online systems and offline systems
- Voracity (Spark Streaming) processes the data from Kafka and writes to Hive (HDFS)
Stream Data Ingestion

- Producers write structured data to Kafka
- Kafka acts as a buffer between online systems and offline systems
- Voracity (Spark Streaming) processes the data from Kafka and writes to Hive (HDFS)
Stream Data Ingestion

- Producers write structured data to Kafka
- Kafka acts as a buffer between online systems and offline systems
- Voracity (Spark Streaming) processes the data from Kafka and writes to Hive (HDFS)
Voracity

Streaming Data Persistence

- Spark Streaming workflow to persist data from Kafka into the data warehouse
- Provides schema validation
- Handles late arriving events based on event time
- Data projection, filtering, transformation via Spark SQL
- Columnar data conversion (Parquet)
- Loads data into unbounded, time partitioned Hive table
Voracity
Streaming Data Persistence

- Spark Streaming workflow to persist data from Kafka into the data warehouse
- Provides schema validation
- Handles late arriving events based on event time
- Data projection, filtering, transformation via Spark SQL
- Columnar data conversion (Parquet)
- Loads data into unbounded, time partitioned Hive table
Voracity

Streaming Data Persistence

• Spark Streaming workflow to persist data from Kafka into the data warehouse

• Provides schema validation

• Handles late arriving events based on event time

• Data projection, filtering, transformation via Spark SQL

• Columnar data conversion (Parquet)

• Loads data into unbounded, time partitioned Hive table
Voracity
Streaming Data Persistence

- Spark Streaming workflow to persist data from Kafka into the data warehouse
- Provides schema validation
- Handles late arriving events based on event time
- Data projection, filtering, transformation via Spark SQL
- Columnar data conversion (Parquet)
- Loads data into unbounded, time partitioned Hive table
Voracity
Streaming Data Persistence

- Spark Streaming workflow to persist data from Kafka into the data warehouse
- Provides schema validation
- Handles late arriving events based on event time
- Data projection, filtering, transformation via Spark SQL
- Columnar data conversion (Parquet)
- Loads data into unbounded, time partitioned Hive table
Unbounded Hive Tables

- Voracity produces the unbounded, time-partitioned Hive table
- Every micro batch appends data into Hive partitions

<table>
<thead>
<tr>
<th>EVENT TIME</th>
<th>DATA</th>
</tr>
</thead>
<tbody>
<tr>
<td>08:00:00</td>
<td>apple</td>
</tr>
<tr>
<td>08:05:00</td>
<td></td>
</tr>
<tr>
<td>09:00:00</td>
<td>orange</td>
</tr>
<tr>
<td>09:00:05</td>
<td></td>
</tr>
<tr>
<td>09:00:07</td>
<td>cherry</td>
</tr>
</tbody>
</table>
Unbounded Hive Tables

- Voracity produces the unbounded, time-partitioned Hive table
- Every micro batch appends data into Hive partitions

Table at 09:01:00

<table>
<thead>
<tr>
<th>EVENT TIME</th>
<th>DATA</th>
</tr>
</thead>
<tbody>
<tr>
<td>08:00:00 PARTITION</td>
<td></td>
</tr>
<tr>
<td>08:05:00</td>
<td>apple</td>
</tr>
<tr>
<td>09:00:00 PARTITION</td>
<td></td>
</tr>
<tr>
<td>09:00:05</td>
<td>orange</td>
</tr>
<tr>
<td>09:00:07</td>
<td>cherry</td>
</tr>
</tbody>
</table>

Table at 09:02:00

<table>
<thead>
<tr>
<th>EVENT TIME</th>
<th>DATA</th>
</tr>
</thead>
<tbody>
<tr>
<td>08:00:00 PARTITION</td>
<td></td>
</tr>
<tr>
<td>08:05:00</td>
<td>apple</td>
</tr>
<tr>
<td>08:07:00</td>
<td>peach</td>
</tr>
<tr>
<td>09:00:00 PARTITION</td>
<td></td>
</tr>
<tr>
<td>09:00:05</td>
<td>orange</td>
</tr>
<tr>
<td>09:00:06</td>
<td>strawberry</td>
</tr>
<tr>
<td>09:00:07</td>
<td>cherry</td>
</tr>
<tr>
<td>09:01:00</td>
<td>pineapple</td>
</tr>
</tbody>
</table>
Fact Tables

Table at 09:01:00

<table>
<thead>
<tr>
<th>EVENT TIME</th>
<th>DATA</th>
</tr>
</thead>
<tbody>
<tr>
<td>08:00:00</td>
<td>apple</td>
</tr>
<tr>
<td>08:05:00</td>
<td></td>
</tr>
<tr>
<td>09:00:00</td>
<td>orange</td>
</tr>
<tr>
<td>09:00:05</td>
<td></td>
</tr>
<tr>
<td>09:00:07</td>
<td>cherry</td>
</tr>
</tbody>
</table>

Table at 09:02:00

<table>
<thead>
<tr>
<th>EVENT TIME</th>
<th>DATA</th>
</tr>
</thead>
<tbody>
<tr>
<td>08:00:00</td>
<td>apple</td>
</tr>
<tr>
<td>08:05:00</td>
<td>apple</td>
</tr>
<tr>
<td>08:07:00</td>
<td>peach</td>
</tr>
<tr>
<td>09:00:00</td>
<td>orange</td>
</tr>
<tr>
<td>09:00:05</td>
<td>orange</td>
</tr>
<tr>
<td>09:00:06</td>
<td>strawberry</td>
</tr>
<tr>
<td>09:00:07</td>
<td>cherry</td>
</tr>
<tr>
<td>09:01:00</td>
<td>pineapple</td>
</tr>
</tbody>
</table>

- Voracity creates unbounded, time-partitioned fact tables
Dimension Tables

- Dimension tables are derived information from fact tables
- Downstream jobs read from fact tables and produce dimension tables
Downstream Jobs

- Downstream jobs contain business logic specified by the user
- Written using Spark Batch (Simple as writing SQL queries)
- Downstream jobs can be viewed as functions that transform data from fact tables to dimension tables
Framework for Downstream Jobs

- Framework continuously runs the downstream jobs to process the micro batches

- Downstream jobs computes $n$ output partitions in each batch

- Time partitions to compute in the dimension table is based on the current time

- Newly computed partitions in the downstream job overwrites the existing partitions in the dimension table
Benefits of Voracity

- Efficient network I/O
- Only Voracity needs to read directly from Kafka
Benefits of Voracity

- Efficient network I/O
- Downstream jobs read only the columns they need
- Better parallelism in HDFS compared to Kafka
Benefits of Voracity

- Efficient network I/O
- Downstream jobs read only the columns they need
- Persist raw logs in long term storage
- Separate data preparation and downstream job logic
Benefits of Micro Batching in Downstream Jobs

• Near real-time data processing (in minutes)
• Simple system architecture / computation model
• Streaming and batch jobs can share the same code
• Full power of Spark SQL
Maintaining and Operating Downstream Jobs

- Job isolation from Voracity and other jobs
- Automatic handling of late arriving events
- Idempotent computation
Overview

• Motivation
• Challenges
• Architecture

• Use Cases
• Accomplishments
Realtime Ads Analytics

- Built on top of the stream processing framework
- Multiple ads online services write data into different Kafka topics
- Voracity instance for each Kafka topic
- Structured data persisted into Hive on HDFS
- Downstream job computes ads analytics for advertisers in near real-time
- Computed metrics are stored in Hive and in HBase for serving in the Ads API
Traffic Monitoring

- Built on top of the stream processing framework
- Data is aggregated by Presto at query time
- Redash is used to visualize the data returned by Presto
Overview

• Motivation
• Challenges
• Architecture
• Use Cases
• Accomplishments
Accomplishments

• Solved near-realtime ads reporting
  • Generate reports in <1 hour
  • Productionized 10+ ads workflows
• Supports growing number of non-ads use cases
• Zero P0 incidents

• Gains
  • Developer velocity
  • Efficiency
  • Maintainability
  • Scalability
Benefits of Genji

- Near-realtime processing
- Unified batch & stream code
- Streaming consumption from Kafka
- Unbounded realtime columnar table data abstraction
Benefits of Genji

• Near-realtime processing
• Unified batch & stream code
• Streaming consumption from Kafka
  • Reduced load on Kafka brokers
  • Schema validation & evolution
  • Backfill
• Unbounded realtime columnar table data abstraction
Benefits of Genji

- Near-realtime processing
- Unified batch & stream code
- Streaming consumption from Kafka
- Unbounded realtime columnar table data abstraction
  - Business logic as SQL
  - Efficient
  - Simplify data exploration & visualization
Thank you!

Acknowledgements

Stream Platform Team
Yoni Ben-tzur
Robert Jones
Mao Geng