Taming large state for real-time joins

Sonali Sharma & Shriya Arora

Netflix
Waiting for your data be like ....
“I love waiting for my data”

- said no stakeholder ever!
Sonali Sharma
- Senior Data engineer, Data Science and Engineering, Netflix
- Build data products for personalization
- Building low latency data pipelines
- Deal with PB scale of data

Shriya Arora
Coming up in the next 40 minutes

- Use case for a stateful streaming pipeline
- Concept and Building blocks of streaming apps
- Data join in a streaming context (windows)
- Challenges in building low latency pipeline
Use case for streaming pipeline
Netflix Traffic

1 trillion events per day

100 PB of data stored on cloud
Recommendations everywhere!

Popular on Netflix

Spanish-Language TV Shows

Comedies
Which artwork to show?
Signal: Take Fraction

Take Fraction = 1 / 3
Making a case for streaming ETL

- Real time Reporting
- Real time Alerting
- Faster training of ML models
- Computational gains
Recap: Use case

- Join Impression events with playback events in real time to calculate take fraction
- Train model faster and on fresher data
- Convert large batch data processing pipeline to a stateful streaming pipeline
Concepts and Building Blocks
Modern stream processing frameworks
Bounded vs Unbounded Data

Batch data at rest, hard boundaries

Stream data is unbounded
Solution: Windows

Windows split the stream into buckets of finite size, over which we can apply computations.

```
stream.keyBy(...)  
  .window(...)  ✗
  [.trigger(...)]
  [.allowedLateness(...)]
  .reduce/aggregate/fold/apply()

stream.join(otherStream)  
  .where(<KeySelector>)
  .equalTo(<KeySelector>)
  .window(<WindowAssigner>)  ✗
  .apply(<JoinFunction>)
```
# Event time vs processing time

<table>
<thead>
<tr>
<th>Clock</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event time</td>
<td>[Red]</td>
<td>[Blue]</td>
<td>[Green]</td>
<td>[Yellow]</td>
<td>[Orange]</td>
</tr>
<tr>
<td>Processing time</td>
<td>[Red]</td>
<td>[Blue]</td>
<td>[Green]</td>
<td>[Yellow]</td>
<td>[Orange]</td>
</tr>
</tbody>
</table>
Out-of-order and late-arriving events

Events from the Netflix apps

Event time windows

Processing time windows

Ingestion pipeline

1st burst of events
2nd burst of events
Solution: Watermark

A watermark is a notion of input completeness with respect to event time. Watermarks act as a metric of progress when processing an unbounded data source.
Slowly changing dimensions

Enriching stream with dimensional data

Raw streams

Combined streams

API calls for enrichment

Movie Metadata (Hive or data map)

Enriched stream
Fault tolerance

- Checkpoint
  - Snapshot of metadata and state of the app
  - Helps in recovery

Checkpoint interval

Checkpoint {n}  Checkpoint {n-1}

Newer records  Older records

Event time
Check point interval

Interval should have cover duration and pauses with buffer
Recap: Concepts and Building blocks

- Handling unbounded data, define boundaries using Windows
- Event time processing
- Handle out of order and late arriving events using Watermarks
- Enrich data in stream using external calls
- Fault tolerance is very important for streaming applications
Making a stream join work
Data Flow Architecture

Impression stream

Playback stream

Transform + AssignTs

.keyBy

Reduce

Output

Transform + AssignTs

Hive

Data Flow Architecture

Transform + AssignTs

Parse (raw -> T) → Filter (T -> T) → AssignTs (t.getTs())
Joining streams: Keyed Streams

DataStream

→

.keyBy

KeyedStream
Stream joins in Flink: Maintaining State

- Events need to be held in-memory for user-defined intervals of time for meaningful aggregations
- Data held in memory needs to be cleared when no longer needed
Aggregating streams: Windows

Windows split the stream into buckets of finite size, over which we can apply computations.

Stream volume: 200k/s/region

Repeating values for same keys: 3–4
Aggregating streams

Can the events be summarized as they come?
Updating state: CoProcess Function

ValueState<T>

K1
I + P +

K3
I ++ P

K4
P

Impressions
Playback
Composite Type
Stream joins in Flink: Updating State

- Timers
  - Flink’s TimerService can be used to register callbacks for future time instants.

```
Timer service
```
```
processElement()
```
```
onTimer()
```
```
State
```
```
Aggregated elements
```
Recap

Impression stream

Playback stream

kafka

Transform + AssignTs .keyBy

Transform + AssignTs .keyBy

Summarize

Output

Challenges
Challenge: Data Correctness

- **Trade-offs**
  - Latency v/s completeness

- **Duplicates**
  - Most streaming systems are at-most-once
  - de-duplication explodes state

- **Data validation**
  - Real-time auditing of data
  - How to stop the incoming flow of bad data?
Challenge: Operations

Visibility into event time progression
Challenge: Operations

- Visibility into state
- Monitoring checkpoints
- Periodic Savepoints
- Intercepting RocksDB metrics
Challenge: Data recovery

- Replaying from Kafka
  - Checkpoints contain offset information
  - Different streams have different volumes

- Replaying from Hive
  - Kafka retention is expensive
  - Easier for stateless applications
Solution: Replaying from Kafka

- Ingestion time filtering
  - Read all input streams from earliest
  - Netflix Kafka producer stamps processing time
  - Filter out events based on processing time

```java
stream.filter(e => e.ingestionTs > T2 &&
            e.ingestionTs < T7 )
```

T0  T1  T2  T3  T4  T5  T6  T7  T8  T9  T10

System went down  System came back up
Challenge: Region failovers

- Event time is dependent on incoming data
- Force moving the watermark via a `maxInactivity` parameter

```java
@override
public final Watermark getCurrentWatermark() {
    long currentSystemTime = System.currentTimeMillis();
    // this guarantees that the watermark never goes backwards.
    if (currentSystemTime > lastRecordSeenTimestamp + idleTimeoutMillis) {
        this.currentMaxTimestamp = currentSystemTime;
    }

    long potentialWM = currentMaxTimestamp - maxOutOfOrderness;
    if (potentialWM >= lastEmittedWatermark) {
        lastEmittedWatermark = potentialWM;
    }

    return new Watermark(lastEmittedWatermark);
}
```
Challenges we are working on

- State Schema Evolution
- Application level De-duplication
- Auto Scaling and recovery
- Replaying and Restating data
Finally
What sparked joy

● Fresher data for Personalization models
● Enhanced user experience
● Enable stakeholders for early decision making
● Save on storage and compute costs
● Real-time auditing and early detection of data gaps
Questions?

Join us!

Twitter: @NetflixData