Real Time Aggregation with Kafka, Spark Streaming and ElasticSearch, scalable beyond Million RPS

Dibyendu B
Dataplayform Engineer, InstartLogic
Who We are

Making web and mobile applications fast, secure, and easy to operate.

- JS Streaming
- HTML Streaming
- Image Optimization
- Machine Learning
- Application Virtualization
- Intelligent CDN
80+ patents

$140M invested

10+ awards

- Original research
- Experts from Google, VMware, Amazon, Twitter
Dataplatfor : Streaming Channel

InstartLogic Cloud -> Event Ingestion Server -> Kafka -> Spark -> Elasticsearch

Billing API
Aggregation API
Real User Monitoring
Ad-hoc queries, offline queries
What We Aggregate

![Graph showing bandwidth usage over time]

**Browser Types**

- **Windows** (41.82%) - 1,644,668 Total
- **iOS** (22.79%) - 896,433 Total
- **Linux** (16.53%) - 649,942 Total
- **Others** (16.51%) - 649,273 Total
- **Unknown** (2.36%) - 92,678 Total

**Bandwidth Breakdown**

- **Last Mile (Original)**
  - 27.33 GB
- **Last Mile (Actual)**
  - 31.12 GB
- **Middle Mile**
  - 3.39 GB
- **First Mile**
  - 6.8 GB

**Cache Hits vs. Misses**

- Cache Hits: 1.94 million
- Cache Misses: 564.75 million

**Content Types**

- Images
- HTML
- CSS
- JavaScript
- Flash
- Others
We Aggregate on:

- Aggregate Metrics on different Dimensions for different Granularity
We have configurable way to define what all Dimension are allowed for given Granularity

This example for DAY Granularity

Similar Set Exists for

HOUR and MINUTE

Let see the challenges of doing Streaming Aggregation on large set of Dimensions across for different Granularities
Some Numbers on volume and traffic

Streaming Ingestion ~ 200K RPS
50 MB / Seconds ~ 4.3 TB / Day

Streaming Aggregation on 5 min Window.

- 60 million Access Log Entries within 5 min Batch
- ~100 Dimensions across 3 different Granularities.
- Every log entry creates ~ 100 x 3 = 300 records

*Key to handle such huge aggregations within 5 min window is to aggregate at stages.*
Multi Stage Aggregation using Spark and Elasticsearch...
Spark Fundamentals

- Executor
- Worker
- Driver
- Cluster Manager
Kafka Fundamentals

Diagram of a Kafka cluster with a producer, broker1, broker2, and broker3. The diagram also shows a consumer and a ZooKeeper for state management.
Kafka and Spark
Spark RDD ..Distributed Data in Spark

How are RDDs generated? ..Let's understand how we consume from Kafka
Apache Spark has in-built Kafka Consumer but we used a custom high performance consumer

I have open sourced Kafka Consumer for Spark Called Receiver Stream

(https://github.com/dibbhatt/kafka-spark-consumer)

It is also part of Spark-Packages : https://spark-packages.org/package/dibbhatt/kafka-spark-consumer

Receiver Stream has better control on Processing Parallelism.

Receiver Stream has some nice features like

Receiver Handler, Back Pressure mechanism,
WAL less end to end No-Data-Loss.

Receiver Stream has auto recovery mechanism from failure situations to keep the streaming channel alway up.

Contributed back all major enhancements we did in Kafka Receiver back to spark community.
Streaming Pipeline Optimization

Too many variables to tune:

- How many Kafka Partitions?
- How many RDD Partitions?
- How much Map and Reduce side partition?
- How much network Shuffle? How many stages?
- How much spark Memory, CPU cores, JVM Heap, GC overhead, memory back-pressure, Elasticsearch optimizations, bulk request, retry, bulk size, number of indices, number of shards...

And so on..
Revisit the volume

*Streaming Ingestion ~ 200K RPS peak rate and growing*

Streaming Aggregation on 5 min Window.

- **60 million** Access Log Entries within 5 min Batch
- 100 Dimensions across 3 different Granularities.
- Every log entry creates ~ 100 x 3 = 300 records
- ~ **20 billion** records to aggregate upon in a single window.

*Key to handle such huge aggregations within 5 min window is to aggregate at stages.*
Aggregation Flow

Consumer Pulls compressed access log entries Kafka

Every compressed entries has N individual logs

Every log fan-out to multiple records (dimensions/granularity)

Every record is (key, value) pair

Map: Per Partition logic

Reduce: Cross Partition logic
Stage 1: Aggregation at Receiver Handler

For each Compressed message: Aggregate
During Job run we observed Stage 1 and Stage 2 contributes to ~ 5 times reduction in object space.

E.g. with 200K RPS, 5 min batch consumes ~60 million access logs, and after Stage 1 and 2, number of aggregated logs are around ~ 12 millions.

What is the Key to aggregate upon?
Stage 3: Fan-out and per-partition aggregation: Map

During Job run we observed Stage 3 contributes to ~ 8 times increase in object space. Note: Fan-out factor is 3 x 100 = 300

After stage 1 and stage 2, number of aggregated records are around ~ 12 million. Number of records after Stage 3 ~ 80 million

What is the key for aggregation?
Stage 4: cross partition aggregation: Reduce

During Job run we observed after Stage 4, number of records reduces to ~ 500K

This number tally with the write RPS at ElasticSearch.
Multi-Stage Aggregation - In a Slide

Partition 1

Node 1

Partition 2

Node 2

Partition 3

Partition 4

Map function

Reduce function

Global aggregation

Partition Level Merge

Fan-out each records

Message level merge

Stage: 1

Stage: 2

Stage: 3

Stage: 4

Daily

Hourly

Minute

Daily

Hourly

Minute

Daily

Hourly

Minute

Daily

Hourly

Minute
Stage 5 : Elasticsearch final Stage Aggregation

- **Reason:**
  - Batch Job: late arriving logs
  - Streaming Job: Each partition could have logs across multiple hours
End to End No Data Loss without WAL

Why WAL is recommended for Receiver Mode?
How we achieved WAL Less Recovery

Keep Track of Consumed and Processed Offset

Every Block written by Receiver Thread belongs to one Kafka Partitions. Every messages written has metadata related to offsets and partition

Driver reads the offset ranges for every block and find highest offset for each Partitions. Commits offset to ZK after every Batch
Spark Back Pressure
Spark Executors Memory: JVM Which Executes Task

Storage Memory: Used for Incoming Blocks

<table>
<thead>
<tr>
<th>Batch Time</th>
<th>Input Size</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>2016/09/20 10:35:00</td>
<td>28544 records</td>
<td>queued</td>
</tr>
<tr>
<td>2016/09/20 10:30:00</td>
<td>29039 records</td>
<td>processing</td>
</tr>
</tbody>
</table>
Control System

It is a feedback loop from Spark Engine to Ingestion Logic
PID Controller

\[ u(t) = K_p e(t) + K_i \int_0^t e(t) \, dt + K_d \frac{de(t)}{dt} \]

Output = Proportional + Integral + Derivative

Output = Error Now + Errors Past + Error Future
Input Rate throttled as Scheduling Delay and Processing Delay increases
Thank You