eBay Real-time OLAP Engine Built on Apache Kylin

May 2017
What is Apache Kylin

Apache Kylin is an open source Distributed Analytics Engine designed to provide SQL interface and multi-dimensional analysis (OLAP) on Hadoop supporting extremely large datasets, initiated and continuously contributed by eBay Inc.

Kylin fills an important gap – OLAP on Hadoop
NRT Streaming in Apache Kylin 1.6
Current Design Limitation

- Minutes level build latency
- Create many tiny HBase tables
- Highly depends on Kafka
- Hard to combine Hive and Kafka together
Real-time OLAP

Milliseconds Query Latency + Milliseconds Data Preparation Delay
The Challenge

Remember Kylin is about pre-aggregations, it successfully reduced the query latencies by spending more time and resources on data preparation.

But for real-time OLAP, cubing time and data visibility latency is critical. Real-time applications are more sensitive to resource usage also.

Can we pre-build the cubes in real-time in a more cost effective manner? This is hard but still doable.
The New Solution

We divide the unbounded incoming streaming data into 3 stages, the data come into different stages are all queryable.

Unbounded streaming events

InMem Stage

Continuously InMem Aggregations

On Disk Stage

Flush to disk, columnar based storage and indexes

Full Cubing Stage

Full cubing with MR or Spark, save to HBase.
New Streaming Components

- Monitoring and Management
- Coordinator
- Query Engine
- Build Engine
- Streaming Receiver
  - Source Connector
  - Columnar Store
- Metadata Store
How Streaming Cube Engine Works

Streaming Coordinator

Build Engine

new streaming cube request

1. Streaming Coordinator

2. Streaming Receivers Cluster

3. Streaming Sources

4. Build Engine

5. Streaming Receivers Cluster

6. Streaming Coordinator

7. Build Engine

8. HBase

ReplicaSet1, ReplicaSet2, ReplicaSet3, ReplicaSet4, ReplicaSet5
Core concepts

- Hierarchy Data Storage
- On Disk Columnar Store
- Segment Window
- Active Segments
- Immutable Segments
- Replica Set
- Check Point
Hierarchy Data Storage

Cube

Segment

Fragment

Meta data

Meta data

Meta data

Meta data
Column Based Fragment File Design

Dimensions: D1, D2... Dn.
Metrics: M1, M2... Mm.
RawRecords: R1, R2... Rr.
Dimension 1 Dictionary

Dimension 1 Raw Records

Inverted Index

<table>
<thead>
<tr>
<th>Dim_1Value_1</th>
<th>Dict_1Value_1</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Dim_1Value_x</td>
<td>Dict_1Value_x</td>
</tr>
</tbody>
</table>

Row 1's Dim 1 Value

...  

Row r's Dim 1 Value

Dic_1Value_1  

Dic_1Value_1's occurrence of Rows

Dic_1Value_x  

Dic_1Value_x's occurrence of Rows

Offset_1

Offset...

Offset_x

R_1 Occu  

R_2 Occu  

...  

R_r Occu
Dimensions: Site, Category, Date
Metrics: SPS, PV.

RawRecords:

<table>
<thead>
<tr>
<th>Site</th>
<th>Category</th>
<th>Date</th>
<th>SPS</th>
<th>PV</th>
</tr>
</thead>
<tbody>
<tr>
<td>US</td>
<td>Collectibles</td>
<td>8/27/16</td>
<td>11757</td>
<td>204143</td>
</tr>
<tr>
<td>US</td>
<td>Collectibles</td>
<td>8/28/16</td>
<td>10949</td>
<td>192350</td>
</tr>
<tr>
<td>GE</td>
<td>Fashion</td>
<td>8/28/16</td>
<td>5338</td>
<td>186659</td>
</tr>
<tr>
<td>US</td>
<td>Collectibles</td>
<td>8/29/16</td>
<td>9609</td>
<td>186283</td>
</tr>
<tr>
<td>US</td>
<td>Fashion</td>
<td>8/27/16</td>
<td>15174</td>
<td>177123</td>
</tr>
<tr>
<td>US</td>
<td>Fashion</td>
<td>8/28/16</td>
<td>14612</td>
<td>172552</td>
</tr>
<tr>
<td>GE</td>
<td>Home &amp; Garden</td>
<td>8/27/16</td>
<td>7336</td>
<td>171364</td>
</tr>
<tr>
<td>US</td>
<td>Fashion</td>
<td>8/29/16</td>
<td>13927</td>
<td>171235</td>
</tr>
<tr>
<td>GE</td>
<td>Home &amp; Garden</td>
<td>8/28/16</td>
<td>8615</td>
<td>169846</td>
</tr>
<tr>
<td>US</td>
<td>Home &amp; Garden</td>
<td>8/29/16</td>
<td>4560</td>
<td>166506</td>
</tr>
</tbody>
</table>
Segment Window

Segments are divided by event time.

New created segments are first in Active state.

After a preconfigured data retention period, no further events coming, the segment will be changed to Immutable state and then write to remote HDFS.

Long latency events can’t fit into active segments will write to long latency segment.
Replica Set

The concept of Replica Set is similar to many other distributed systems like Kafka, Mongo, Kubernetes, etc.

Replica Set ensures the replications for HA purpose.

Streaming receiver instances in the same replica set have the exact same local state.

Streaming receive instances are pre-allocated to join the ReplicaSet groups.
<table>
<thead>
<tr>
<th>Date Time</th>
<th>Topic Name</th>
<th>Partition #</th>
<th>Offset</th>
<th>SeqID of Seg_LL</th>
<th>SeqID of Active Segments</th>
</tr>
</thead>
<tbody>
<tr>
<td>2016/10/01 12:00:00</td>
<td>Topic_1</td>
<td>1</td>
<td>x</td>
<td>I</td>
<td>Seg_4, J</td>
</tr>
</tbody>
</table>

**Kafka**

**Check Point**

**Topic_1 Part_1**

Last Checkpoint

Current Offset

In Memory

Active Fragments

Raw Record_1

 RR_x

 RR_x+1

 RR_y

 Seg_LL

 1 ... I

 Seg_4

 1 ... J

 Seg_3

 1 ... K
Extended Query Engine

SQL Query

Query Engine

Steaming Coordinator

Streaming Receivers Cluster

Cube Storage (HBase)
Extended Query Engine

GTScanner Interface

InMemoryScanner  FragmentScanner  HBaseScanner

Unbounded streaming events

Raw Record_1
...
RR_x
RR_x+1
...
RR_y

In Memory

Active Fragments

Seg_LL
1 ... I

Seg_4
1 ... J

Seg_3
1 ... K

Segment LL
1 ... L

Seg_2
1 ... M

Seg_1
1 ... N

HBase

Open to Write
Close to Process
Processed
Cubing Job

Coordinator submit MR jobs incrementally to build the full cube when all the streaming receiver instances convert their segment to Immutable for that window.

The MapReduce jobs will do all the dirty work like merge all fragment files together, build dictionaries, build cubes, generate Hfiles and bulkload to Hbase.

Incrementally merge jobs can be configured to merge hourly segments to daily and then to weekly segments to avoid too many Hbase tables.
<table>
<thead>
<tr>
<th>Job Description</th>
<th>Start Time</th>
<th>Duration</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Build Dimension Dictionaries For Streaming Job</td>
<td>2017-05-10 23:49:45 PST</td>
<td>6.37 min</td>
<td>Complete</td>
</tr>
<tr>
<td>Save Cube Dictionaries</td>
<td>2017-05-10 23:50:47 PST</td>
<td>6.12 min</td>
<td>Complete</td>
</tr>
<tr>
<td>Build Base Cuboid Data For Streaming Job</td>
<td>2017-05-10 23:50:48 PST</td>
<td>4.48 min</td>
<td>Complete</td>
</tr>
</tbody>
</table>
Use Case

**TOTAL CUBE COUNT**
11
More Details

**QUERY COUNT**
84,023
More Details

**AVG QUERY LATENCY**
0.15 s
More Details

**JOB COUNT**
223
More Details

**AVG BUILD TIME PER MB**
7.00 sec
More Details

**AVG CUBE EXPANSION**
4.33
More Details

---

Query Count by Project

Query Count by Day
Thanks!