The big data real-time analysis platform (THOR)

Weihai Shen
wshen24@asu.edu

Goals

Analyze real-time DLU (daily launch user) & DNU (daily new user) data of users for ByteDance Inc.

1. The reference value for product marketing. Regulate the marketing launch based on the real-time performance of each launch channel without having to wait until the next day

2. Real-time hot events. Analyze the impact of hot events ahead

3. Expose potential issues as soon as possible. Cut the loss if any exception of DLU & DNU is triggered when a new version is launched

4. …

5. ...

Size of input stream

Rows: 7 ~ 8 billion rows within a day and still in increase with multiple dimensions, such as channel, mobile brand, operation system, location, ...

Write qps: 1 million/s at its peak

Functionality: get the dlu & dnu with arbitrary combinations of limited dimensions, SQL style would be expressed as this:

```
SELECT count(distinct(device_id)) as dlu FROM data WHERE os = “iOS” and location = “AZ”
```

Alternatives - MySQL

The first consideration: MySQL

1. Fail to meet significantly high write qps due to its master-cluster architecture
2. Being impossible to create a suitable index for arbitrary combinations of dimensions
3. It is not built for big data scenario
4. ...

Alternatives - Druid

Druid almost is a perfect product is designed to quickly ingest massive quantities of event data, and provide low-latency queries on top of the data

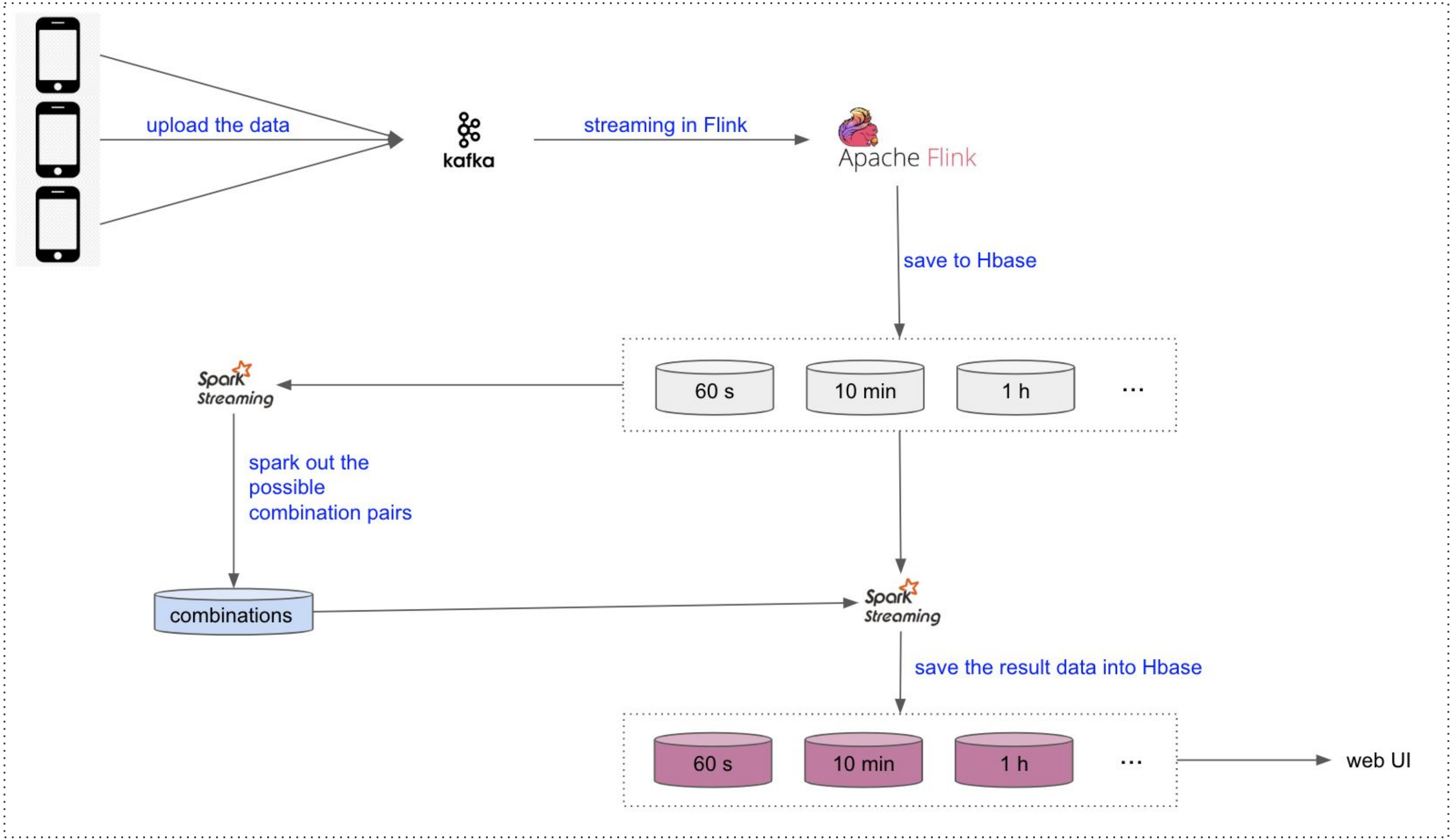
However, druid only provides approximate values for the distinct count due to speed consideration. At ByteDance, accuracy is the highest priority and approximate aggregation is unacceptable [1].

References

1. Aggregations https://druid.apache.org/docs/latest/querying/aggregations

Our choice

1. Using Hbase as data storage
2. Using Spark to pre-calculate all possible results
3. Using Flink to process data streaming
4. Using Python to provide RESTful API
5. ….



Overall Design

Procedure of overall design

1. Upload data to the kafka clusters from clients
2. Flink or strom consumes kafka and dumps rows into detail hbase table, as this

```
{“key”: “23#20170101#13#298789”, “value”: “os:ios, os_version:0.98, brand:xiaomi, …”}
```

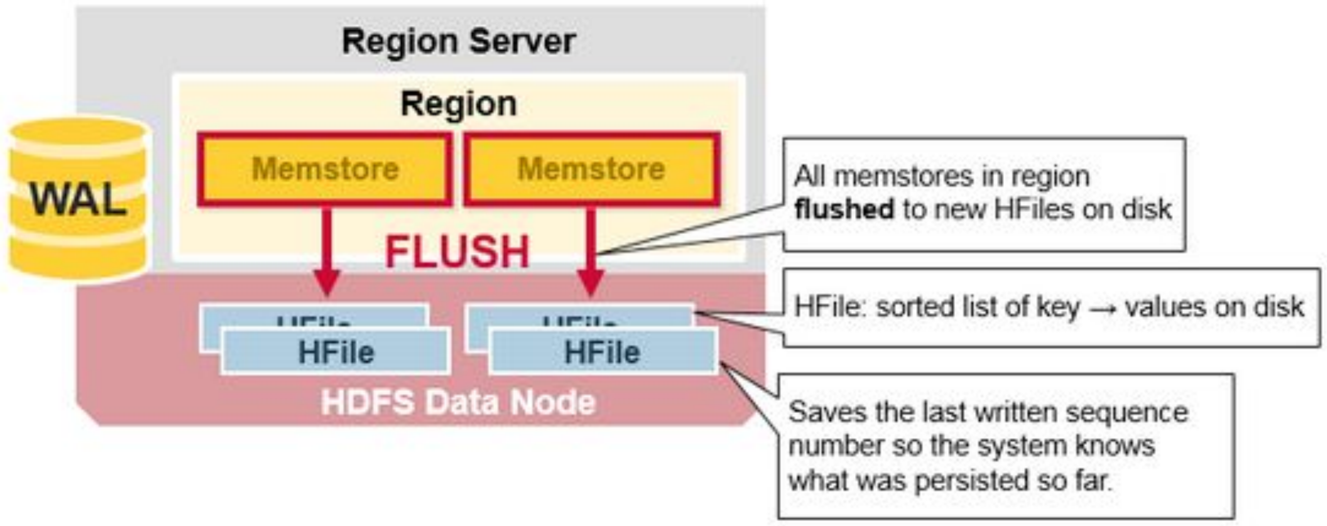
3. Pre-calculate all possible results from detail table to result hbase

```
{“key”: “23#20170101#13#os:android#brand:xiaomi#-”, “value”: “dlu:19023”}
```

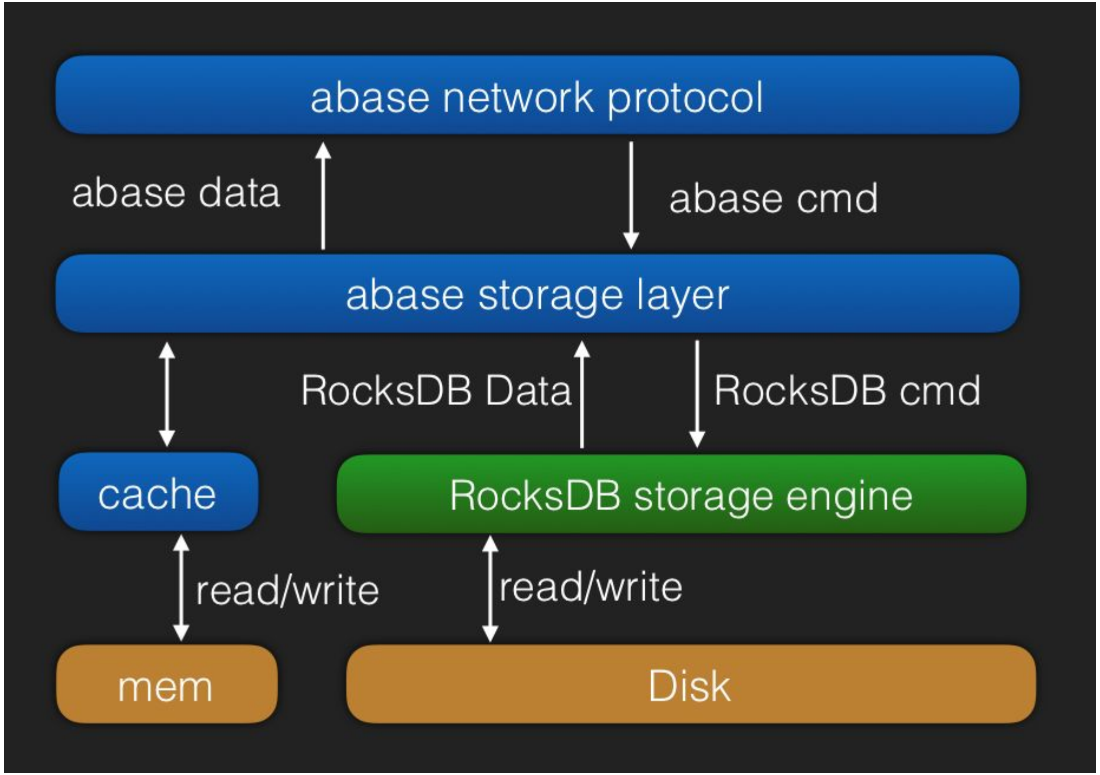
4. Python retrieves data from result hbase
5. Browser renders UI

Hbase in the THOR

- In fact, Bytedance builds a new big-data key-value software called Abase based on rocksdb, https://github.com/facebook/rocksdb, similar to hbase usage
- WAL: Write-Ahead-Log. Great Performance on Writing with huge data



Abase - layered architecture



In general, each abase instance follows master-slave pattern based RocksDB incremental WAL file

Reference

1. RocksDB WAL https://github.com/facebook/rocksdb/wiki/Write-Ahead-Log

Merge two Kafka streams

In order to get new user information, we need to merge two kafka streams. The key is using Window [1] of Flink, The data from two kafka streams with the same device_id and within the same time window will be merged to a new record

Reference

1. Window https://ci.apache.org/projects/flink/flink-docs-stable/dev/stream/operators/windows.html

Row key in the abase

Abase data stores consist of one or more tables, which are indexed by row keys. Data is stored in rows with columns, and rows can have multiple versions. By default, data versioning for rows is implemented with time stamps

In the THOR platform, all secrets lies on row key design

Row_key of detail abase table

Key

- {salt}#{date_format}#{app_id}#{device_id}

Value

- dimension values

Salt

- Distributed the data evenly to different region servers
- The dimension values from the same device are able to be distributed to the same region server
- Partition data into 1000 regions

Dimension values

- Pattern: brand:Meizu|os:Android|os_version:0.12

Row_key of result abase

Key

- {salt}#{date_format}#{app_id}#{dimension_whence_str}#{optional time}

Value

- Integer (DLU or DNU)

Salt

- {salt} = hash(date_format + app_id + dimension_whence_str} % 10
- Distributed the result data evenly to different region server

Dimension_whence_str

- Pattern: dimension_key_a:value|dimension_key_b:value|dimension_key_c:value

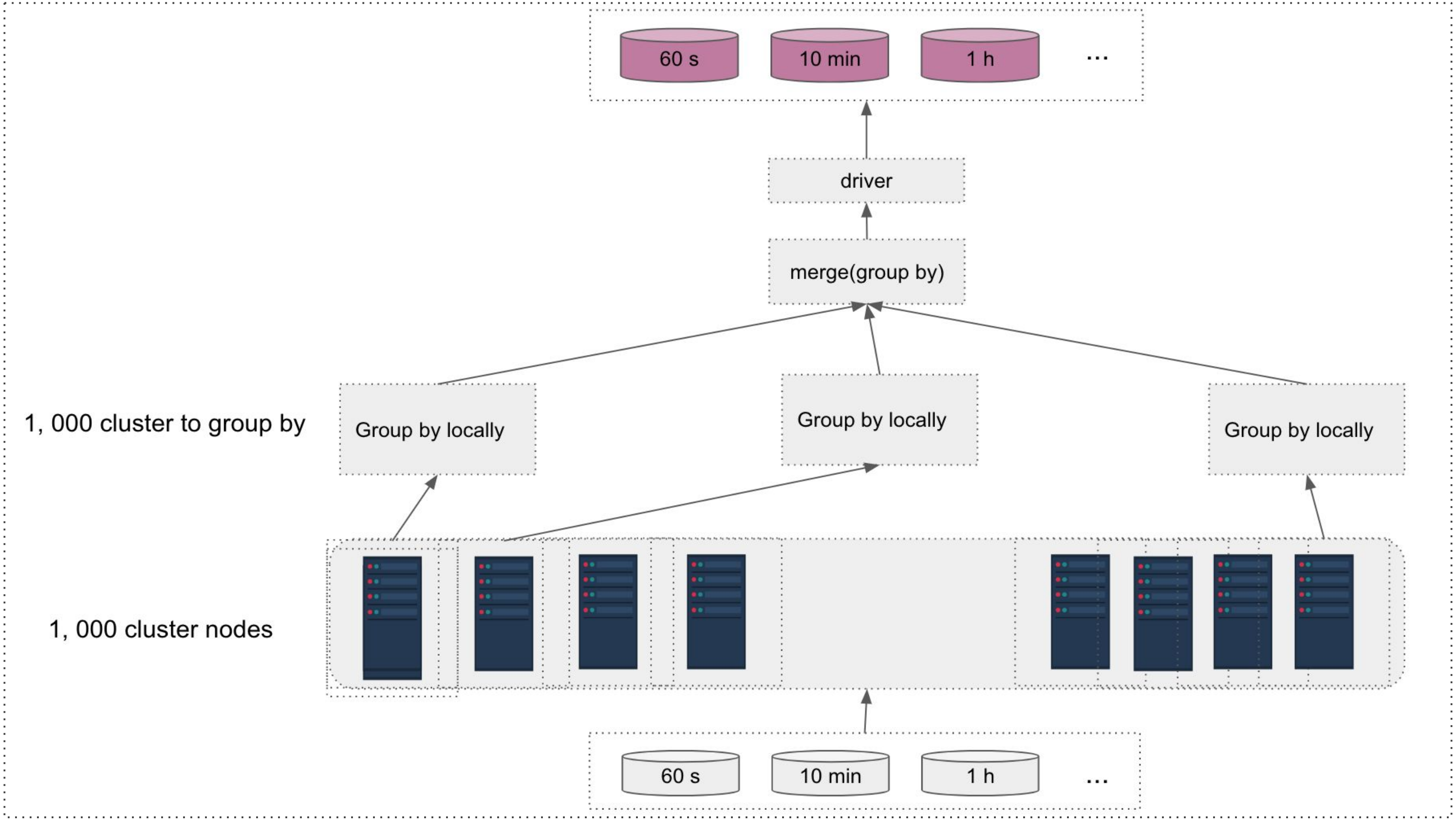
Row_key of result abase

Date_format

- **Day**: %Y%m%d, **hour**: %Y%m%d_%H, **10 minutes**: %Y%m%d_%H%M

Optional time

- **Day**: “”, **hour**: %H, **10 minutes**: %H%M

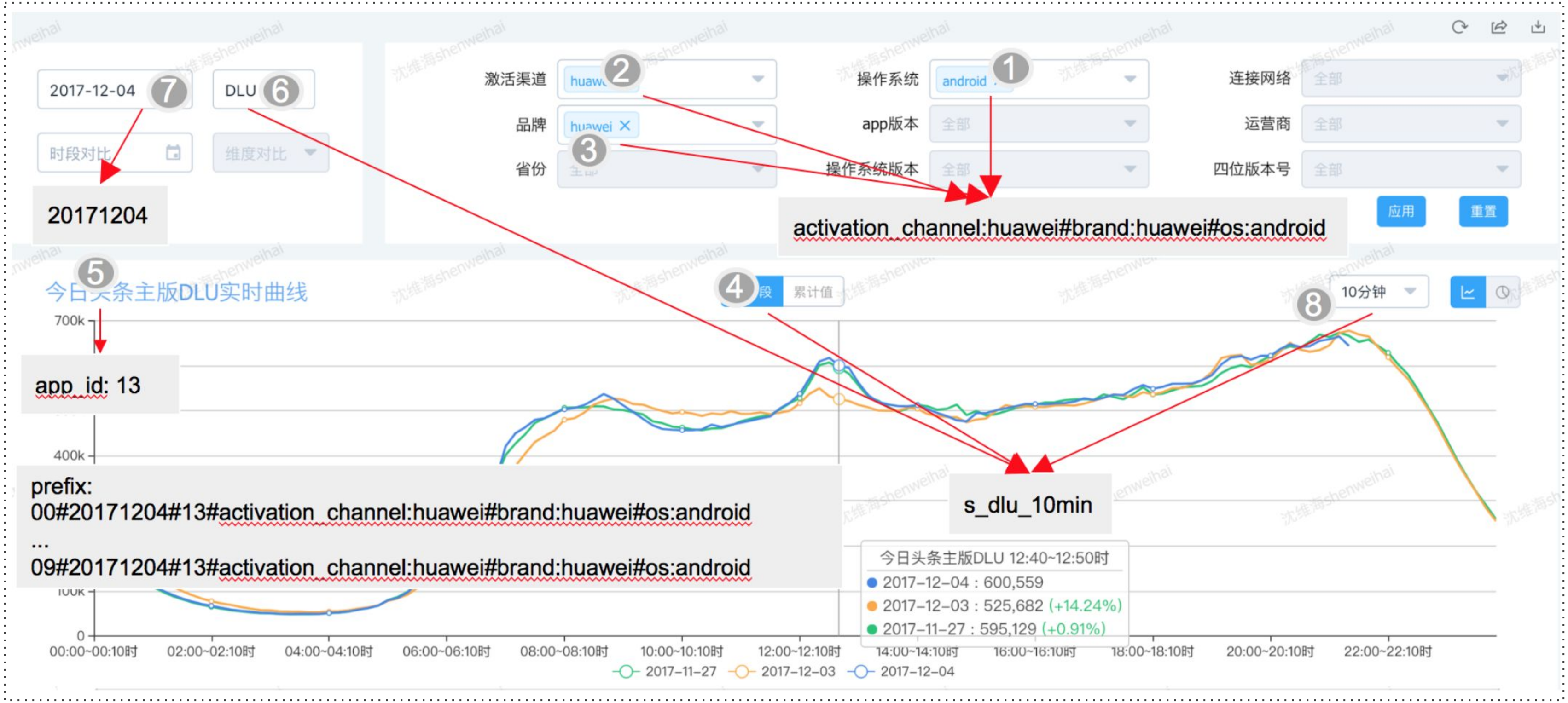


Spark aggregate data

Procedure of aggregation

1. Start 1000 spark executors periodically
2. Aggregate data such as distinct count on each executor
3. Merge aggregated data from 1000 executors to local driver
4. Re-aggregate 1000 aggregated data on demand
5. Store all re-aggregated data into result hbase tables

One query



2017-12-04
7
DLU
6
时段对比
维度对比
20171204
激活渠道
huawei
2
品牌
huawei
3
省份
操作系统
android
1
app版本
全部
操作系统版本
全部
连接网络
全部
运营商
全部
四位版本号
全部
应用
重置
activation_channel:huawei#brand:huawei#os:android
5
今日头条主版DLU实时曲线
4
累计值
10分钟
8
700k
400k
100k
0
app_id: 13
prefix:
00#20171204#13#activation_channel:huawei#brand:huawei#os:android
...
09#20171204#13#activation_channel:huawei#brand:huawei#os:android
s_dlu_10min
今日头条主版DLU 12:40~12:50时
2017-12-04 : 600,559
2017-12-03 : 525,682 (+14.24%)
2017-11-27 : 595,129 (+0.91%)
00:00~00:10时
02:00~02:10时
04:00~04:10时
06:00~06:10时
08:00~08:10时
10:00~10:10时
12:00~12:10时
14:00~14:10时
16:00~16:10时
18:00~18:10时
20:00~20:10时
22:00~22:10时
2017-11-27
2017-12-03
2017-12-04

One query

- Using section 4, 6, 8 determines the corresponding Abase table
- Using section 1, 2, 3, 5, 6, 7 combines the row_key's prefix:

```
{salt} #00#20171204#13#activation_channel:huawei#brand:huawei#os:android
```

- Get the result data by row key prefix

Future work

- Add configuration center
- Support recovery mechanism
- Utilize Apache Druid to support more useful functionalities
- Add the intelligent alarm module
- Hbase clusters Read/Write Splitting
- ...