hosted by Alibaba Group 阿里巴巴集团 APACHE HBASE

HTAP DB—System : ApsaraDB HBase Phoenix and Spark

Yun Zhang & Wei Li

August 17,2018

Content

01 Phoenix Over ApsaraDB HBase

02 Spark & ApsaraDB HBase/Phoenix

1 Phoenix Over ApsaraDB HBase

Content

1.1 Architecture

1.2 Use Cases

1.3 Best Practice

1.4 Challenges & Improvements

1.1 Phoenix Over ApsaraDB HBase

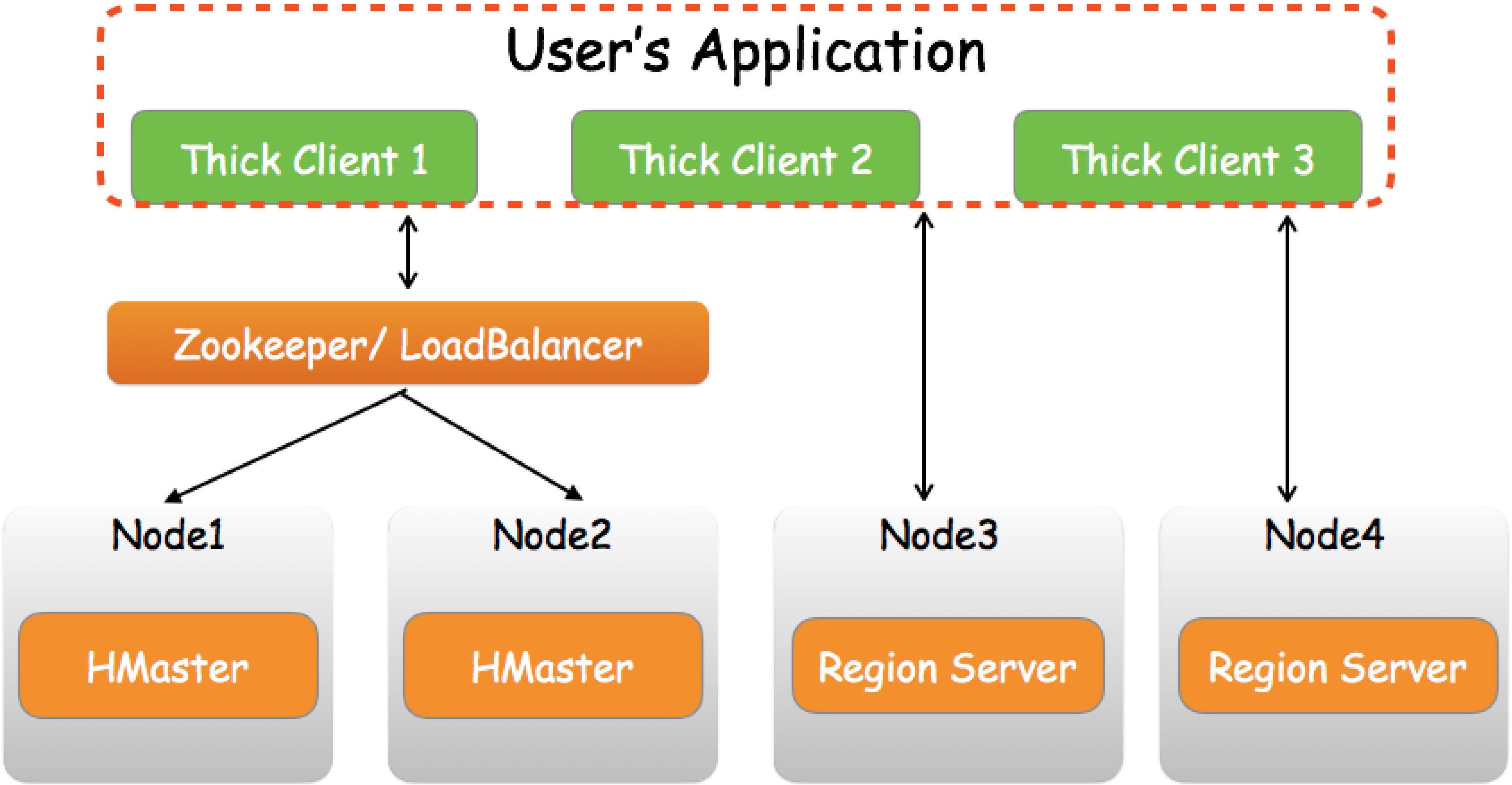
Phoenix-As-A-Service

Phoenix-as-a-service over ApsaraDB HBase

- Orientations
  - Provides OLTP and Operational analytics over ApsaraDB HBASE
- Targets
  - Make HBASE easier to use
    - JDBC API/SQL
  - Other functions
    - Secondary Index
    - Transaction
    - Multi tenancy
    - …

Phoenix Architecture

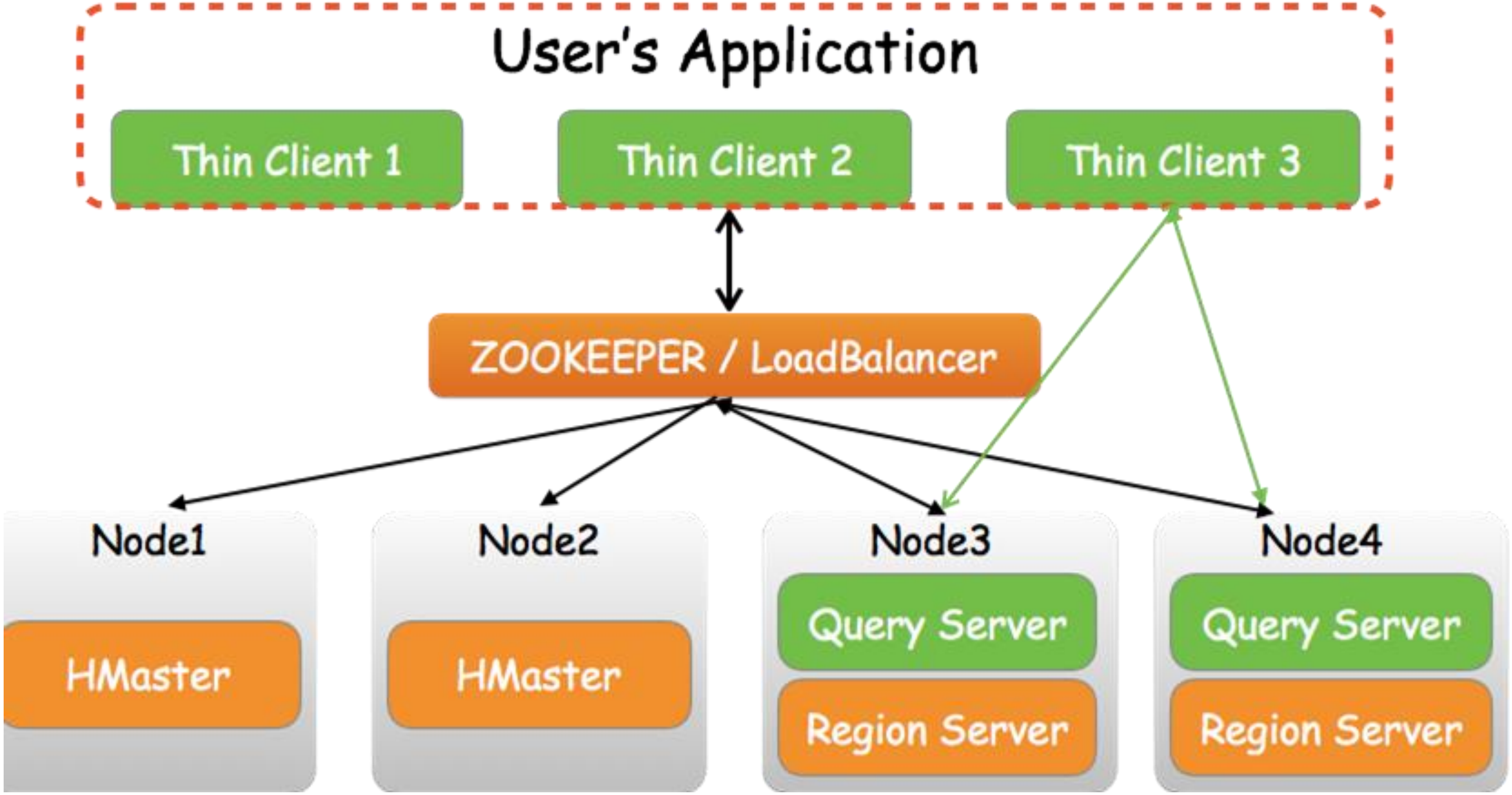
Thick Phoenix Client Architecture



Upgrades client is very painful as a cloud service!

Phoenix Architecture

Thin Phoenix Client Architecture



Lower maintenance cost as a cloud service!

1. 2 Use Cases

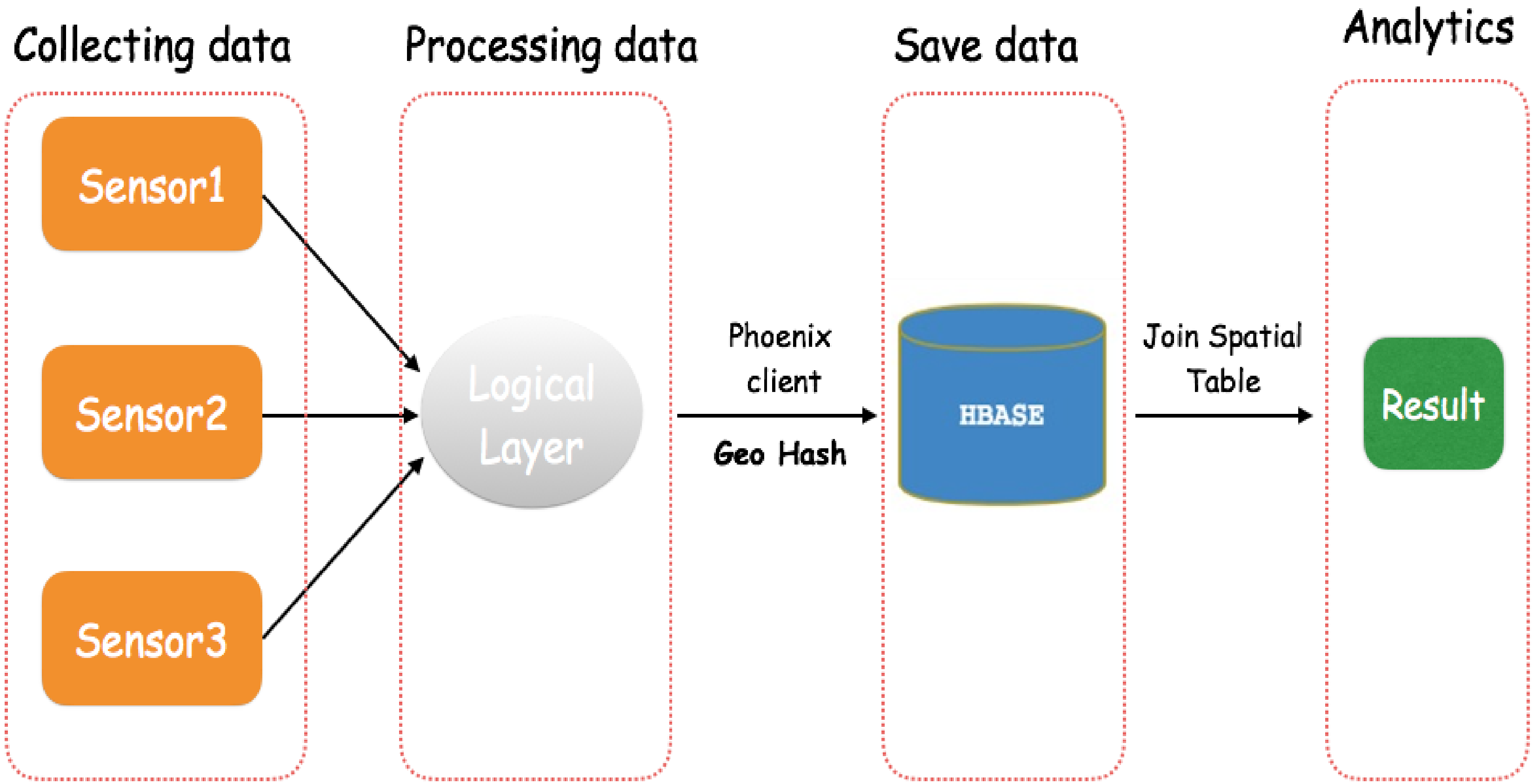
Use Case 1

LOT Scenario

- Data
  - Big table(Spatial Temporal Data) 100 million+
  - Small table(User Information) less than 1 million
- Functional Requirements
  - Hash join(big table join small table)
  - Staled table (avoid hot spotting)
  - Secondary index
- Other Requirements
  - Latency less than 2 seconds (100 vertices of polygon)
  - Scale out

Use Case 1

Architecture



Use Case 1

Query

Spatial_Temporal_Data_T

coordinate	ts	user_id	other_field
187892	1533474569	A	...
123832	1533474570	B	...
565422	1533474571	C	...
948352	1533474572	D	...
...	...	...	...

User_Data_T

user_id	ts	other_field
A	1533474569	...
B	1533474570	...
C	1533474571	...
D	1533474572	...
...	...	...

```
select * from User_Data_T as b
right join
   (select geo_hash,user_id from
   Spatial_Temporal_Data_T where
   (SPATIAL_A.position>10009 and
   SPATIAL_A.position<10011) or
   (SPATIAL_A.position>10011 and
   SPATIAL_A.position<10012) or
   (SPATIAL_A.position>10012 and
   SPATIAL_A.position<10015) or
   ...) as a
on a.user_id=b.user_id
where b.ts<152240212411 3
```

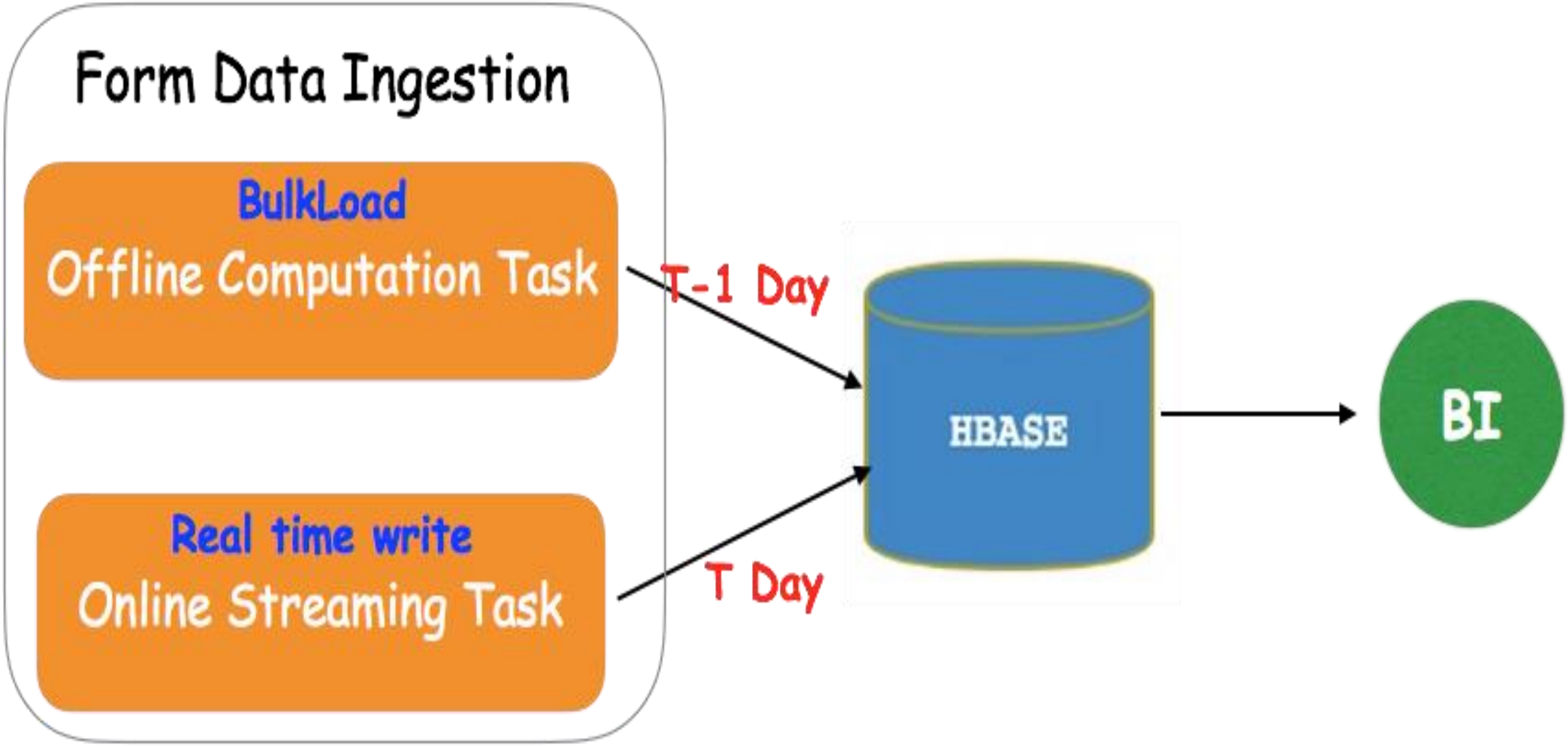
Use Case 2

Internet Company Scenario

- Data
    - 350+ million/per day
    - 500G+/per day(uncompress)
- Functional Requirements
    - Staled table (avoid hot spotting)
    - Secondary index(multidimensional analytics)
- Other Requirements
    - Latency less than 200 Millisecond
    - 6+ index tables
    - Scale out

Use Case 2

Architecture



1.3 Best Practices

Best Practices

Table Properties

1. Recommend to set **UPDATE_CACHE_FREQUENCY** when create table (120000ms as a default value)
2. Used **pre-splitting keys** is better than **slated table**.(Range scan is limited when use slate buckets)
3. **Pre-splitting** region for index table(if your data tables are salted table, index tables will inherit this property).
4. **SALT_BUCKETS** is not equal **split keys**(pre-splitting region)!!!

Best Practices

Query Hint

1. Use **USE_SORT_MERGE_JOIN** when join bigger tables.
2. Use **NO_CACHE** will avoid caching any HBase blocks loaded, which can reduce GC overhead and get better performance. it is used export data query, such as UPSERT…SELECT clause.
3. Use **SMALL** will save an RPC call on the scan, Which can reduce network overhead. it is used hit the small amount of data of query.

Best Practices

Composite Key

Data table composite key	Index table composite key

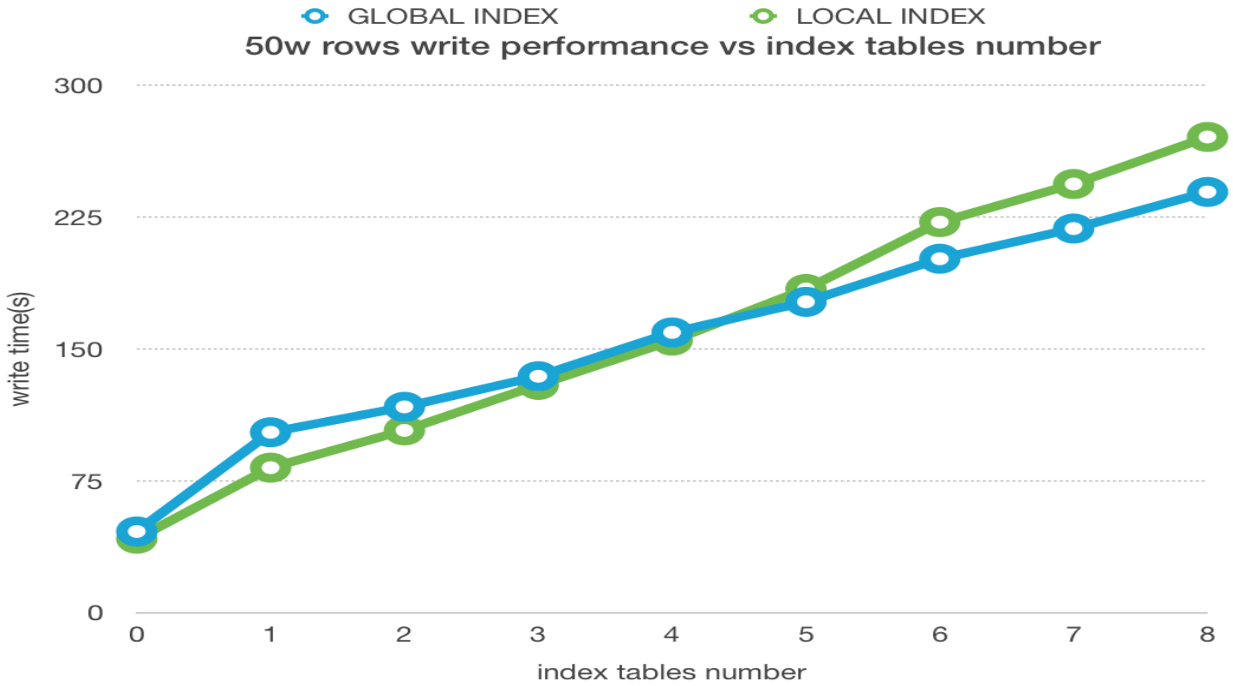
CREATE TABLE DATA_TABLE(
  A VARCHAR,
  B VARCHAR,
  C VARCHAR,
  D INTEGER,
CONSTRAINT PK PRIMARY KEY(**A, B, C**))
	CREATE INDEX IDX_NAME
	ON DATA_TABLE(**A, B, C**)

Where Conditions	Status
A=x and B=x and C=x	Best
A=x and B=x	Better
A=x	OK
B=x and C=x	Not recommended
C=x	Dangerous

Best Practices

Other Tips

1. Recommend to use global index on massive data table
2. Reasonable to use Row timestamp that affects visibility of data
3. More index tables depressed write throughput



1.4 Challenges & Improvements

Challenges

- Availability
  - Sometimes index table become unavailable.
- Stability
  - Full scan/complex queries affects cluster’s stability
- Users Complaints
  - queries can’t automatically choose the best index table
  - Using Python client get worse performance.
  - Lack of data transferring tools (data import/export)
  - Scarce monitor metrics
  - ...

Improvements

- Stability
  - Phoenix Chaos Monkey test framework
- Availability
  - Support infinite retry policy when writing index failures to avoid degrade to full scan data table.
- Producibility
  - Recognizes some full scan/complex queries and reject on the Server(Query Server) side
  - Integrate monitor platform
  - Other new features
    - Alter modify column/rename
    - Reports rate of progress When creating index

2

Spark & ApsaraDB HBase/Phoenix

Spark & ApsaraDB HBase

2.1 Overview

2.2 Architecture & Implementation

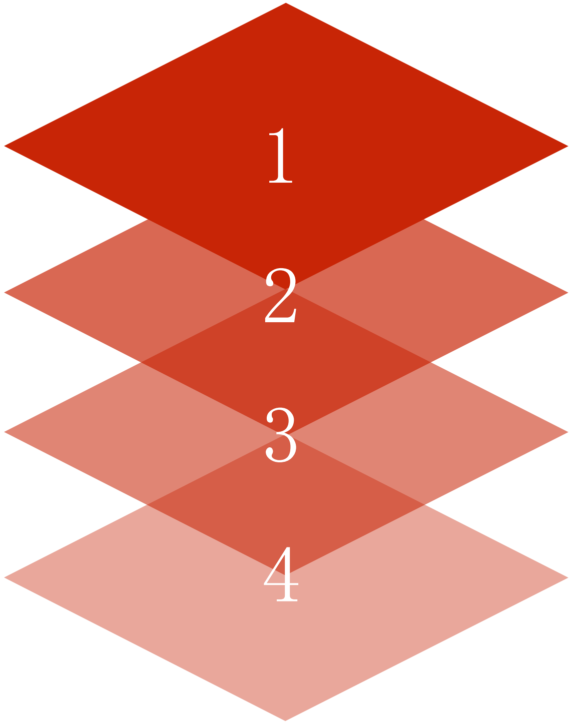
2.3 Scenario

2. 1 Overview

Overview

HBase/phoenix requirements

Why? Spark



Analysis

Phoenix not good at complex analysis

Bulkload

Users need bulkload large number of data to hbase/phoenix fastly

Elastic resource

Phoenix uses hbase coprocessors to do analysis, but hbase cluster resource has limitation

Real time etl/load

As data visibility，user need realtime etl and load to hbase/phoenix

Overview

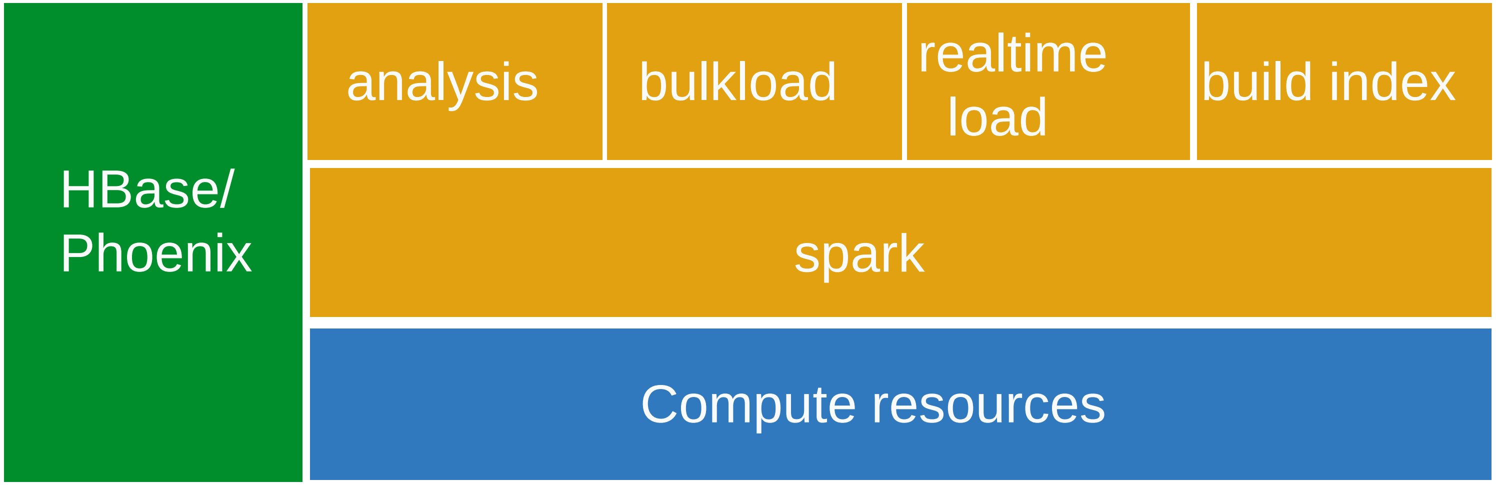
what can spark bring to ApsaraDB HBase

➢ Analysis:

- spark as a unified analytics engine，support SQL 2003
- Use dag support complex analysis

➢ Bulkload：spark can support multi datasource like jdbc, csv, elasticsearch, mongo; have elastic resource

➢ Realtime load：struct streaming easy to do etl, and load to hbase

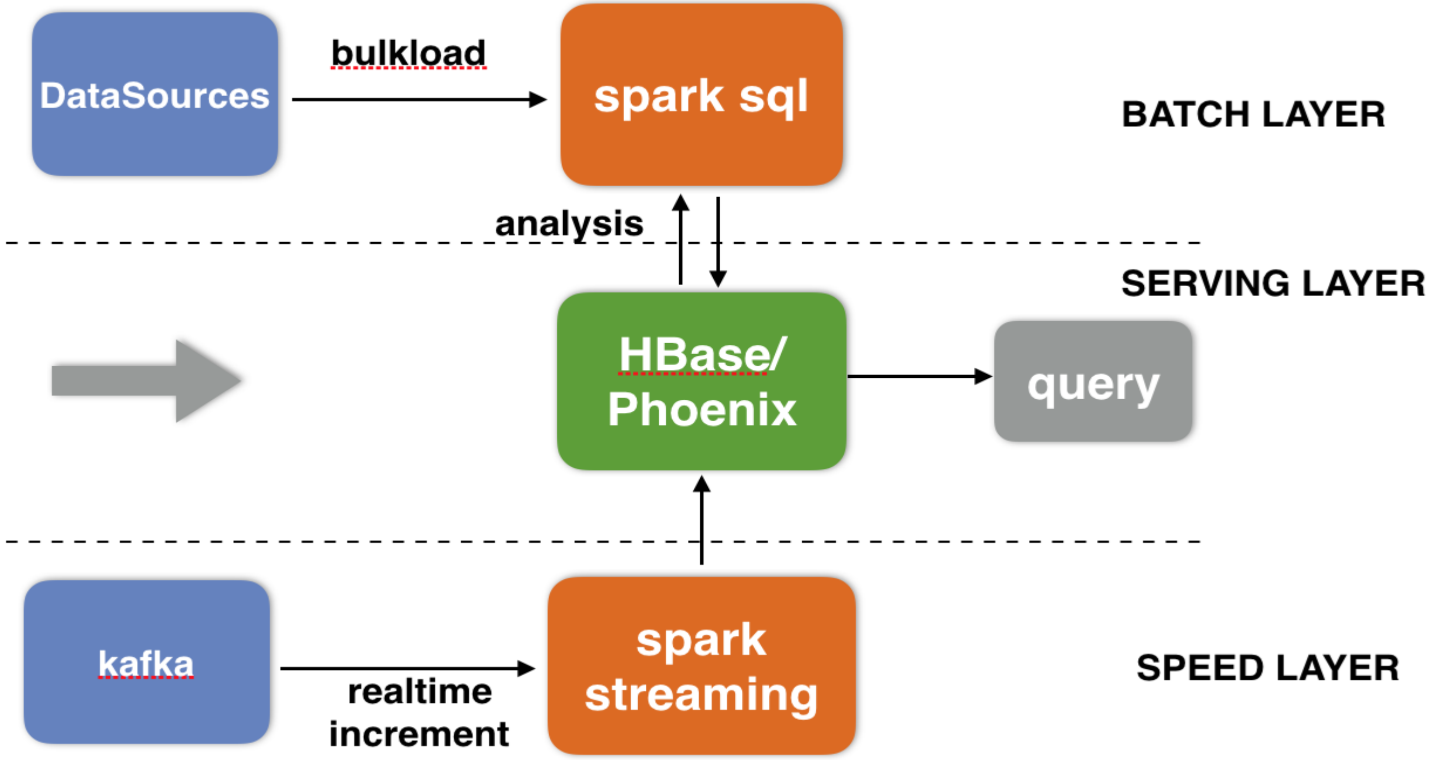


2.2 Architecture & Implementation

Architecture & implementation

Use ApsaraDB HBase and spark construct big data platform

- ➢ BATCH LAYER : use spark sql/dataset analysis HBase/Phoenix, also bulkload other datasources to HBase
- ➢ SPEED LAYER : use struct streaming etl data from kafka, and increment load into HBase
- ➢ SERVING LAYER : User query result data from HBase/Phoenix



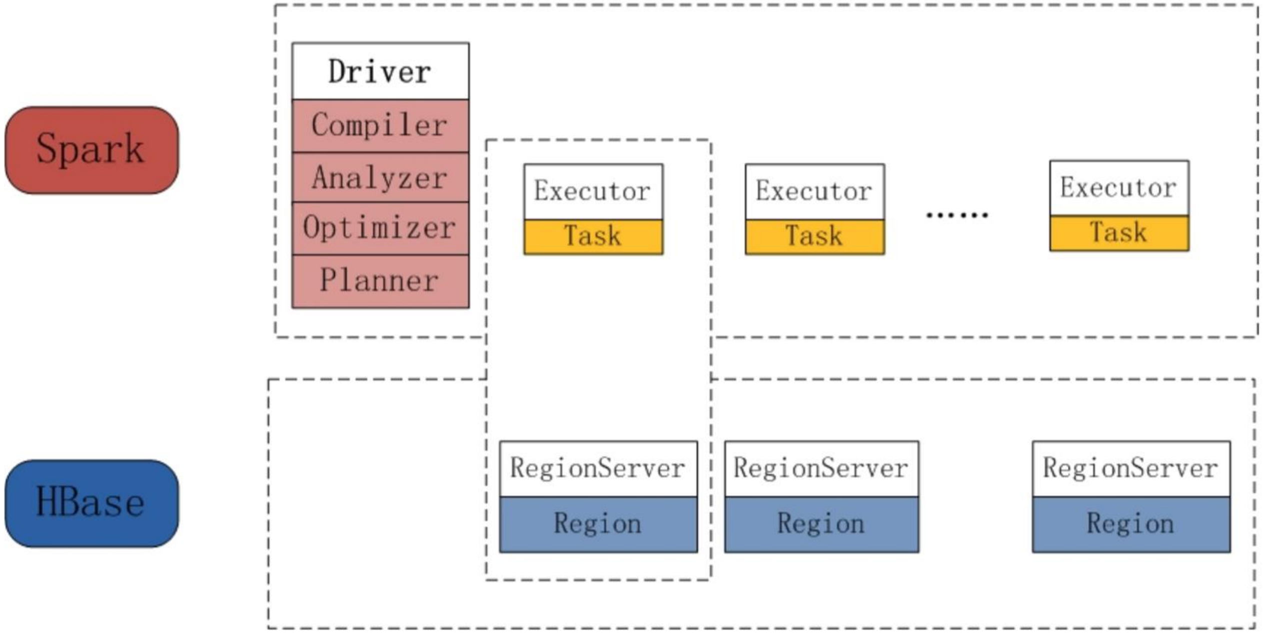
spark sql& HBase

how spark sql analysis HBase/Phoenix?

➢ SQL API：use spark sql analysis hbase，table meta in hive metastore

➢ Performance:

- distributed scan;
- sql optimize like partition pruning、column pruning、predicate pushdown；
- direct reading hifles；
- auto transform to column based storage



spark sql& HBase

demo:CREATE TABLE

```
CREATE TABLE HBaseTest USING org.apache.spark.sql.execution.datasources.hbase
OPTIONS ('catalog'=
              '{
      "table":{"namespace":"default", "name":”TestTable", "tableCoder":"PrimitiveType"},
      "rowkey":"key",
      "columns":{
         "col0":{"cf":"rowkey", "col":"key", "type":"string"},
         "col1":{"cf":"cf1", "col":"col1", "type":"boolean"},
         "col2":{"cf":"cf2", "col":"col2", "type":"double"},
         "col3":{"cf":"cf3", "col":"col3", "type":"float"},
         "col4":{"cf":"cf4", "col":"col4", "type":"int"},
         "col5":{"cf":"cf5", "col":"col5", "type":"bigint"},
         "col6":{"cf":"cf6", "col":"col6", "type":"smallint"},
         "col7":{"cf":"cf7", "col":"col7", "type":"string"},
         "col8":{"cf":"cf8", "col":"col8", "type":"tinyint"}
          }
   }’
)
```

spark sql& HBase

demo and performance

beeline> select count(col2) from HBaseTest where col0 < 'row050‘ and col2 >’10.0

- partition pruning：use col0 < ‘row050‘ to perform the needed regions
- predicate pushdown：col2 >’10.0 filter will pushdown to hbase scan
- column pruning: only scan the needed column

Performance:

- Data scale:500208
- Native HBaseRDD: HadoopRDD use TableInputFormat
- Spark SQL:spark hbase datasource

类型	时间	结果
Native HBaseRDD	14.425s	5788
Spark SQL	1.036s	5788

Spark struct streaming& HBase

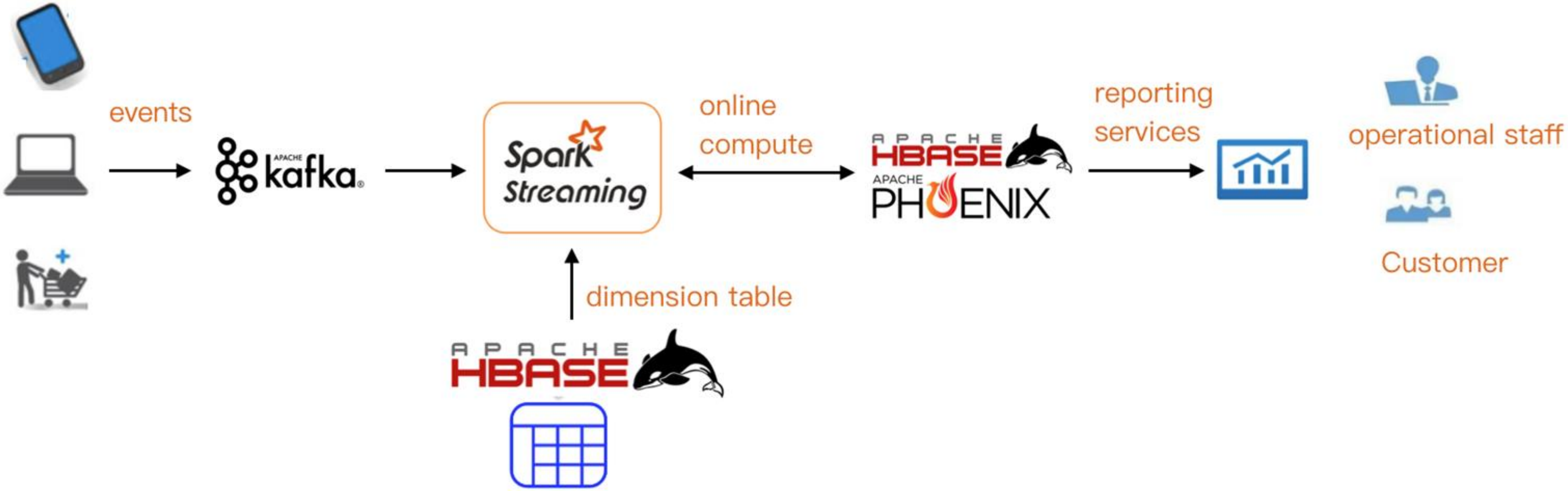
demo

```
val catalog =
  s"""{
     |"table":{"namespace":"default", "name":"structStreamingCount", "tableCoder":"PrimitiveType"},
     |"rowkey":"key",
     |"columns":{
     |"value":{"cf":"rowkey", "col":"key", "type":"string"},
     |"count":{"cf":"cf1", "col":"count", "type":"int"}
     |}
     |}""".stripMargin

val lines = spark.XXX.load()
val wordCounts = lines.as[String].flatMap(_.split(" ")).filter($"value"=!="").groupBy("value").count()

val query = wordCounts.
  writeStream.
  outputMode("update").
  format("org.apache.spark.sql.execution.datasources.hbase.HBaseSinkProvider").
  option("checkpointLocation", “xxxx").
  option("hbasecat", catalog).
  start()
```

2.3 Scenario

Scenario 1

big data online reporting services

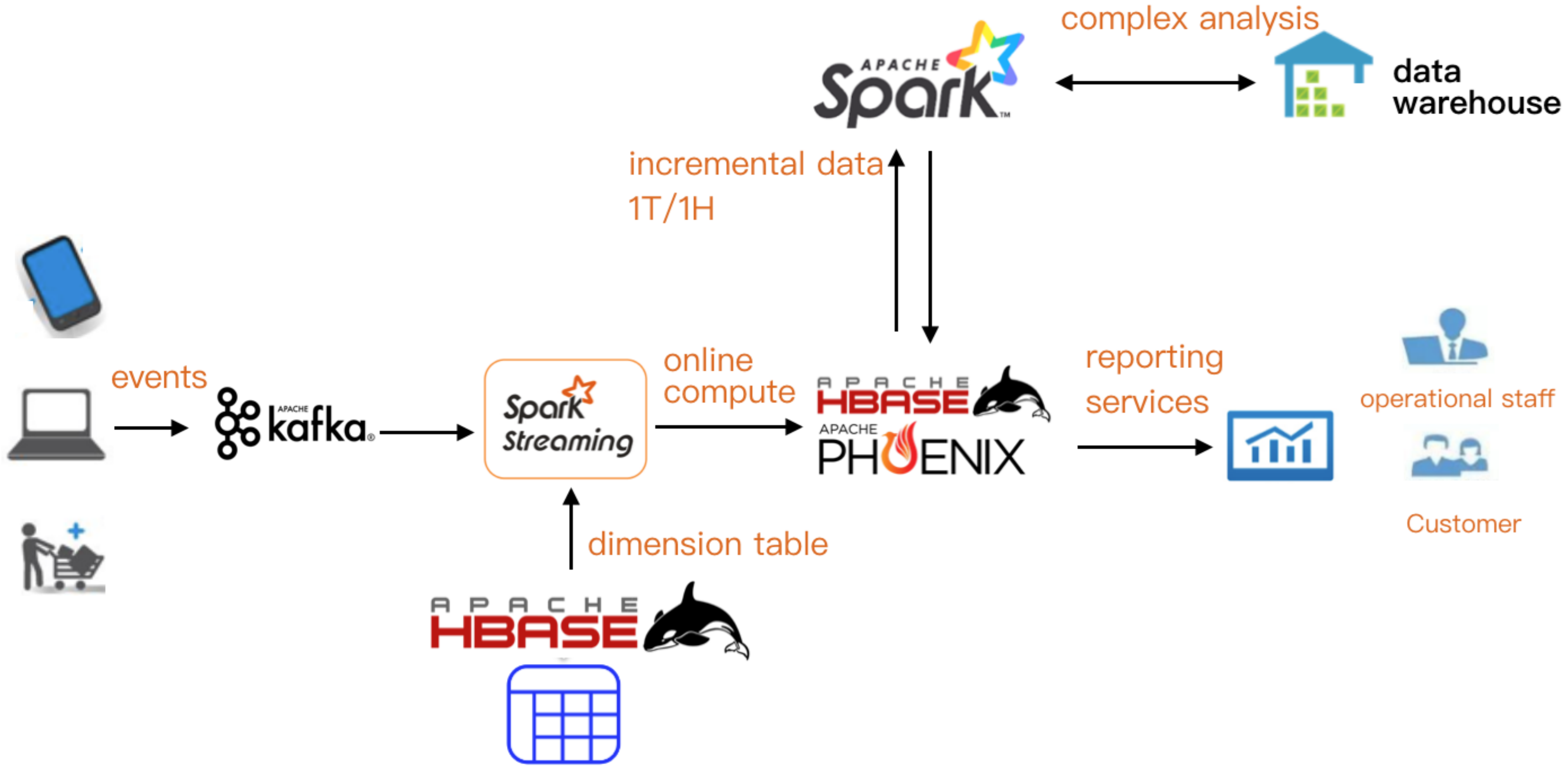


➢ Specialty
- Online service
- Dimension table

➢ Case
- Mobile game：Real-time user activity in different regions
- Business：Different types of goods pv real-time report

Scenario 2

big data complex reporting services

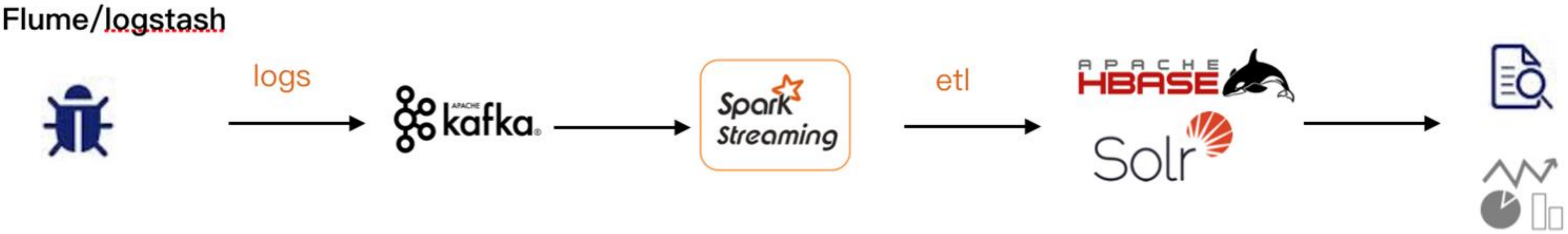


➢ Specialty
- Complex analysis
- Datawarehouse
- Quasi-real time

➢ Case
- Mobile game：Comparison of user activity in different age groups during the same period

Scenario 3

log indexes and query

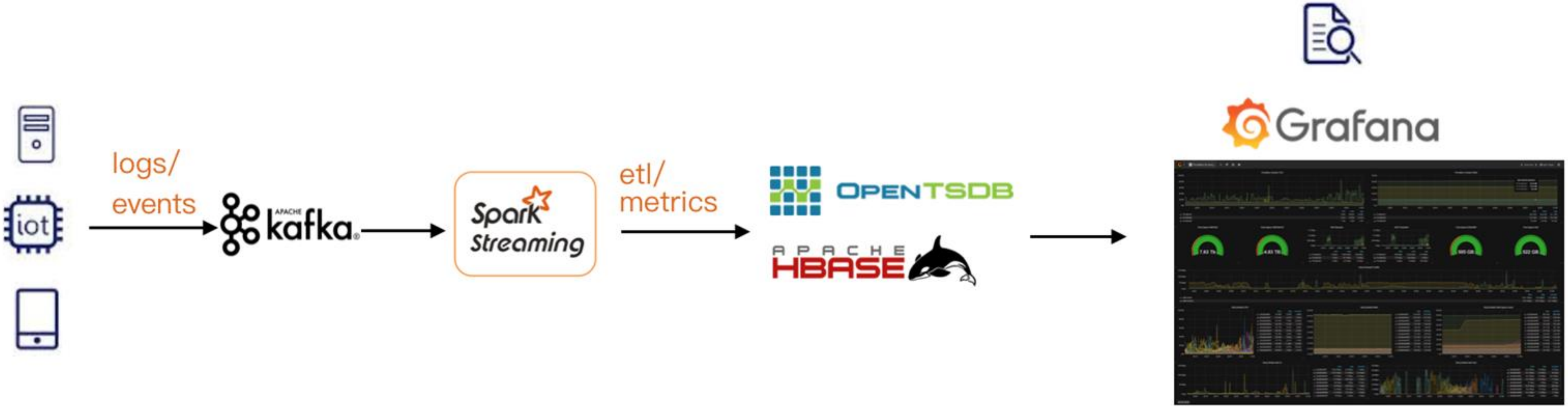


➢ Specialty
- collect logs in real time
- log indexes and query

➢ Case
- Log service system

Scenario 4

time series query and monitoring



logs/ events

Apache kafka

Spark Streaming

etl/ metrics

OpenTSDB

Apache HBase

Grafana

➢ Specialty

- Time series data
- Multi metrics
- Time series query

➢ Case

- IOT、service and business monitoring system

We are hiring!

- If you are interested in the unified online sql analysis engine
- If you are interested in the spark kernel and ecosystem

ApsaraDB HBase
DingDing Community

Yun Zhang (wechat)
Phoenix

Wei Li (wechat)
Spark

hosted by Alibaba Group 阿里巴巴集团 APACHE HBASE

Thanks