Flink SQL 1.9.0 技术内幕和最佳实践

伍翀（云邪）· 阿里巴巴 / 技术专家

Kafka X Flink Meetup 深圳 - 2019年08月31日

个人介绍

- 伍 翀（云邪，Jark）
- Apache Flink Committer
- 阿里巴巴 Blink SQL 团队

云邪的博客

专注分享Flink、Spark
等大数据相关技术。
欢迎关注，共同进步！

http://wuchong.me

CONTENT

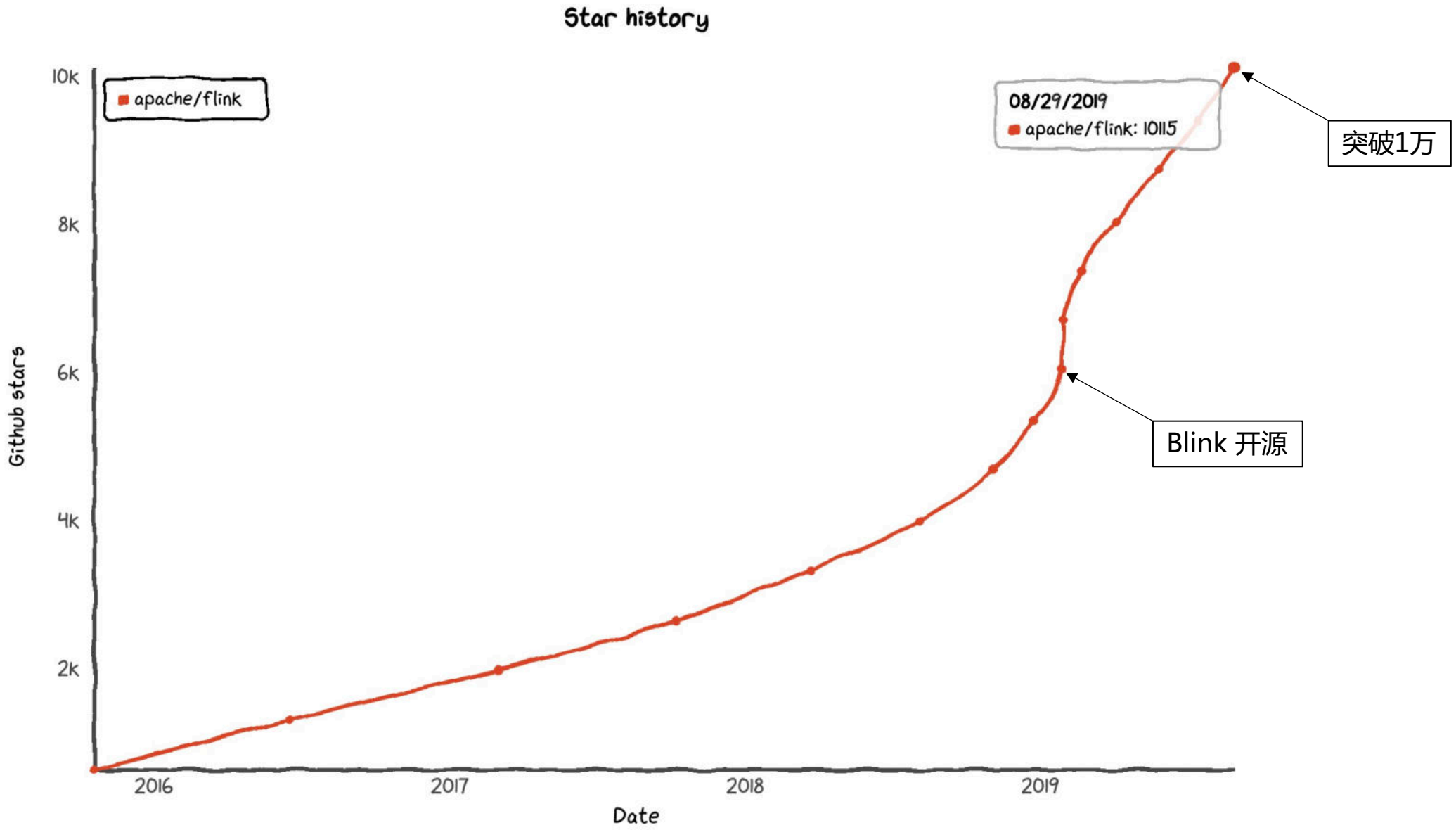
目录 >>

Flink 1.9.0 的里程碑意义

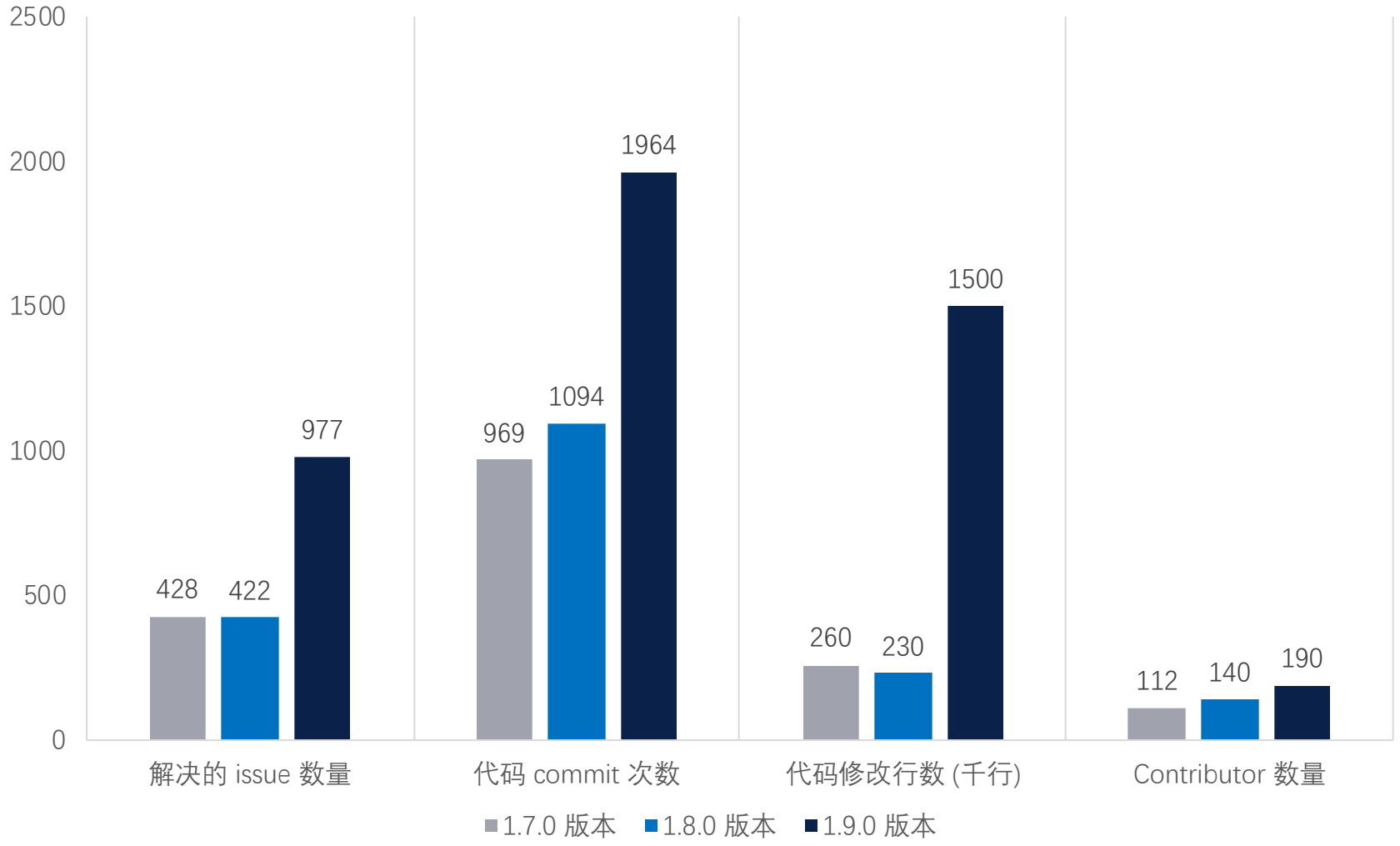
2019.01 Blink 开源并贡献给 Apache Flink

2019.08 Blink 合并入 Flink 后首次发版

Flink 1.9.0 的里程碑意义



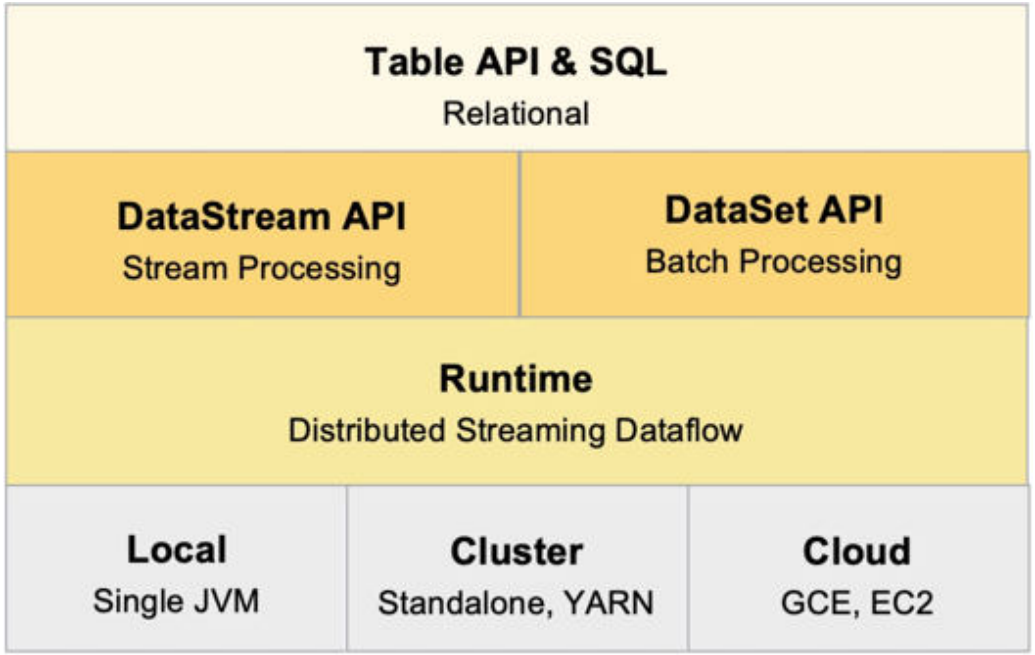
Flink 1.9.0 的里程碑意义



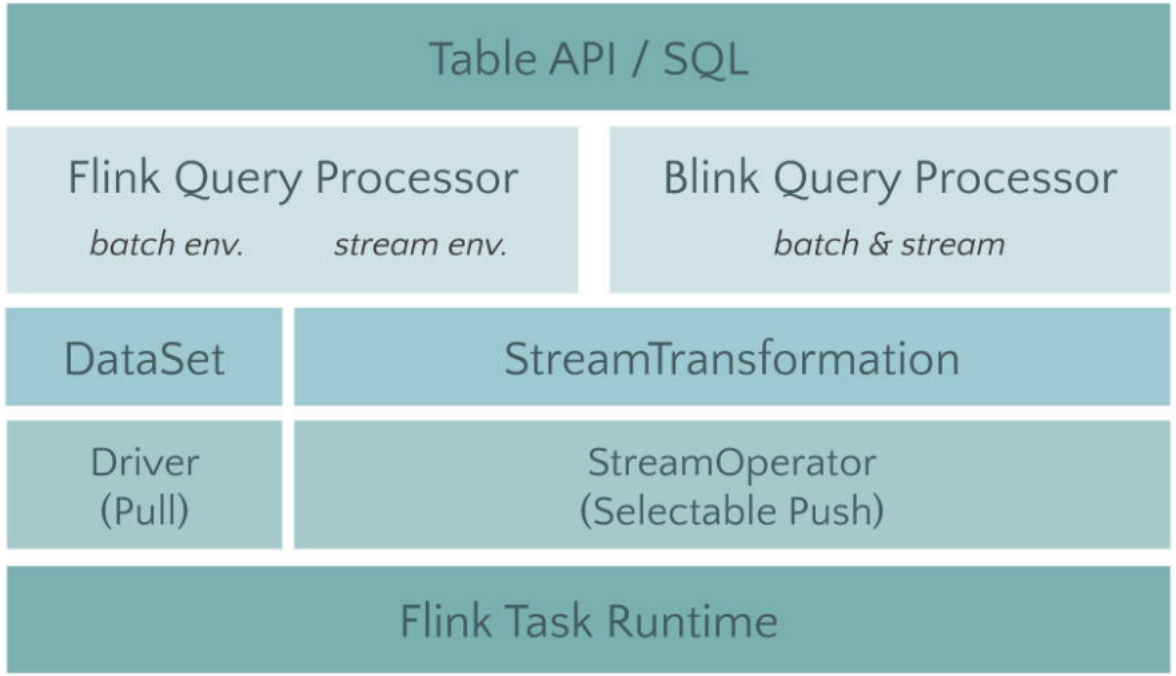
02

新架构和新功能

架构升级



Flink 1.8



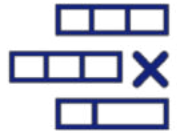
Flink 1.9

新功能一览

SQL DDL

重构类型系统

TopN



高效流式去重

维表关联

MiniBatch

多种解
热点手段

完整的
批处理支持

Python
Table API

Hive 集成

新功能一览

SQL DDL

重构类型系统

TopN

高效流式去重

维表关联

MiniBatch

多种解
热点手段

完整的
批处理支持

Python
Table API

Hive 集成

新功能一览

SQL DDL

重构类型系统

TopN

高效流式去重

维表关联

MiniBatch

多种解
热点手段

完整的
批处理支持

Python
Table API

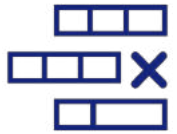
Hive 集成

新功能一览

SQL DDL

重构类型系统

TopN



高效流式去重

维表关联

MiniBatch

多种解
热点手段

完整的
批处理支持

Python
Table API

Hive 集成

新功能一览

SQL DDL

重构类型系统

TopN

高效流式去重

维表关联

MiniBatch

多种解
热点手段

完整的
批处理支持

Python
Table API

Hive 集成

新功能一览

SQL DDL

重构类型系统

TopN

高效流式去重

维表关联

MiniBatch

多种解
热点手段

完整的
批处理支持

Python
Table API

Hive 集成

新功能一览

SQL DDL

重构类型系统

TopN

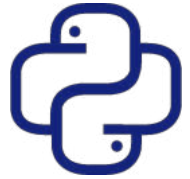
高效流式去重

维表关联

MiniBatch

多种解
热点手段

完整的
批处理支持



Python
Table API

Hive 集成

新功能一览

SQL DDL

重构类型系统

TopN

高效流式去重

维表关联

MiniBatch

多种解
热点手段

完整的
批处理支持

Python
Table API

Hive 集成

03

如何使用 Blink Planner

如何使用 Blink Planner

```
<dependency>
  <groupId>org.apache.flink</groupId>
  <artifactId>flink-table-api-scala-bridge_2.11</artifactId>
  <version>1.9.0</version>
  <scope>provided</scope>
</dependency>

<dependency>
  <groupId>org.apache.flink</groupId>
  <artifactId>flink-table-planner-blink_2.11</artifactId>
  <version>1.9.0</version>
  <scope>provided</scope>
</dependency>
```

如果在IDE中运行，要删除 provided。
如果打包提交，要确保有 provided。

```
val settings = EnvironmentSettings.newInstance().useBlinkPlanner().inStreamingMode().build();
val env = StreamExecutionEnvironment.getExecutionEnvironment()
val tEnv = StreamTableEnvironment.create(bsEnv, bsSettings)
tEnv.sqlUpdate(…)
tEnv.execute()
```

04

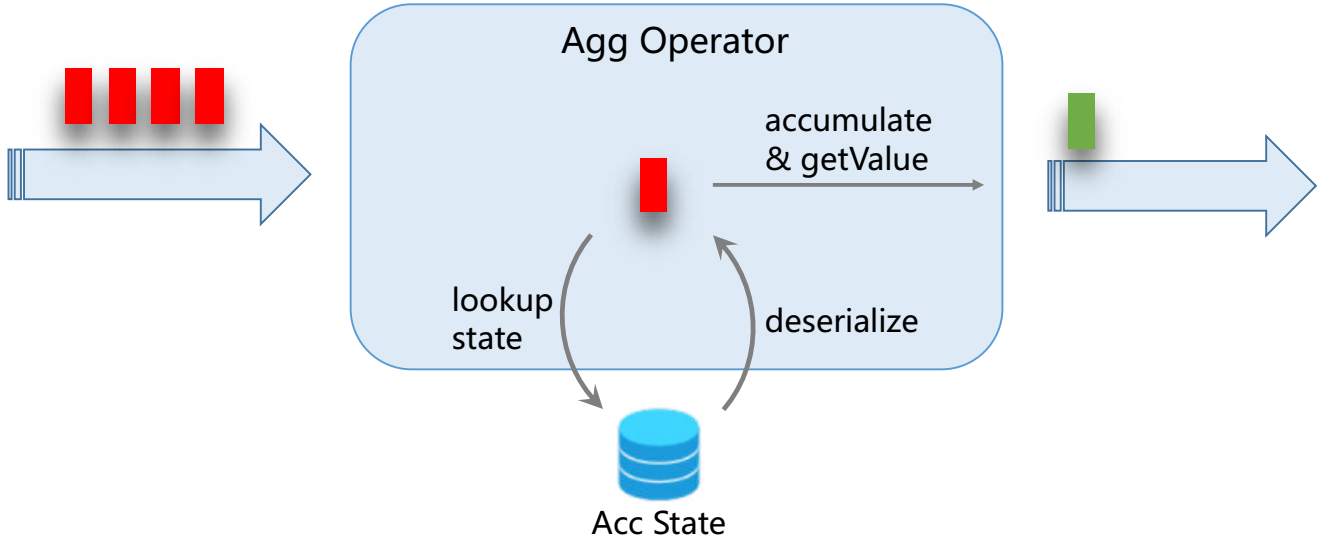
性能优化指南

举个栗子

统计每小时全网的访问量。

每条数据状态访问

```
SELECT
  DATE_FORMAT(ts, 'yyyy-MM-dd HH') hour,
  COUNT(*) AS pv
FROM user_log
GROUP BY DATE_FORMAT(ts, 'yyyy-MM-dd HH')
```

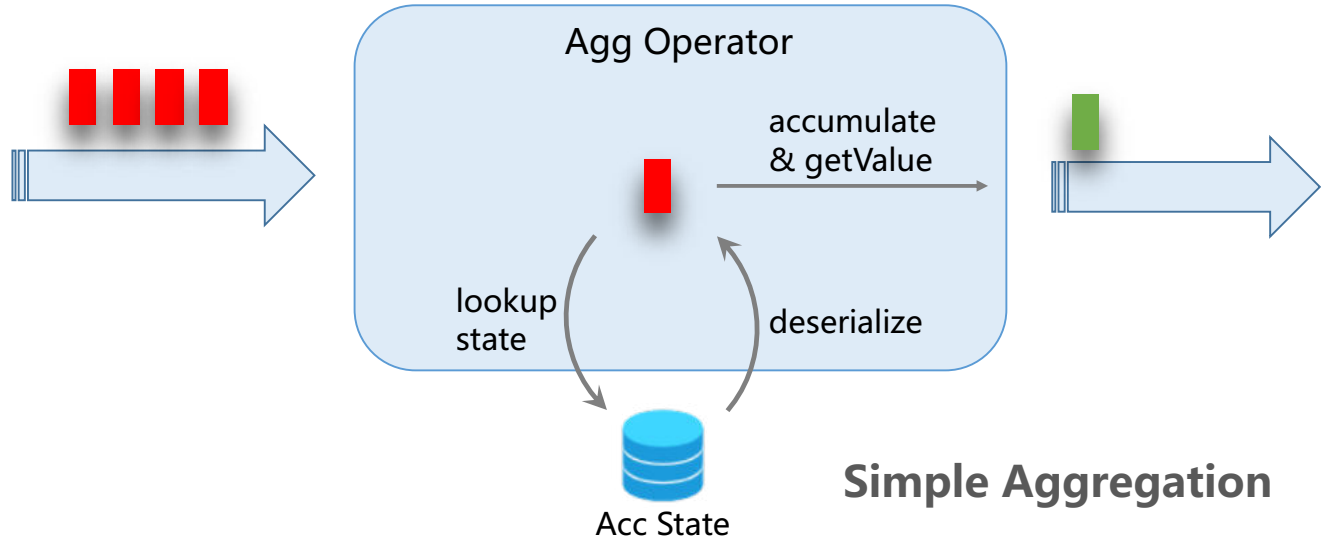


MiniBatch 优化

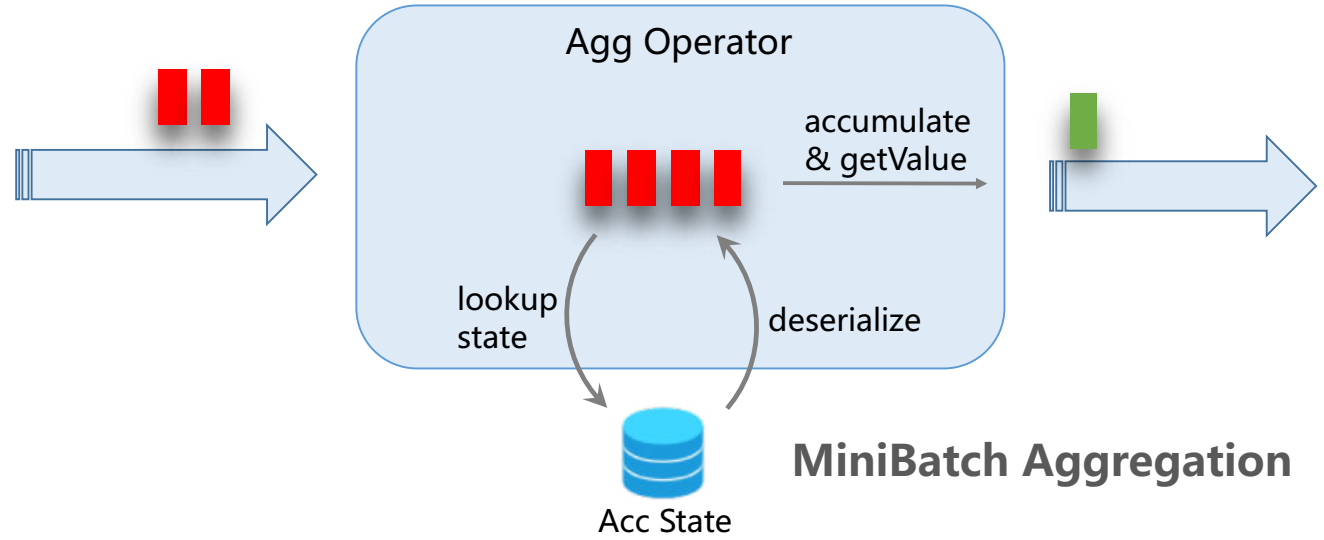
MiniBatch 的优势：

1. 提升吞吐
2. 减少 state 访问
3. 减少序列化/反序列化的开销
4. 减少输出，降低对下游压力

```
# 开启 mini-batch
table.exec.mini-batch.enabled=true
# mini-batch的时间间隔，即作业需要额外忍受的延迟
table.exec.mini-batch.allow-latency=5s
# 一个节点中允许最多缓存的数据
table.exec.mini-batch.size=5000
```

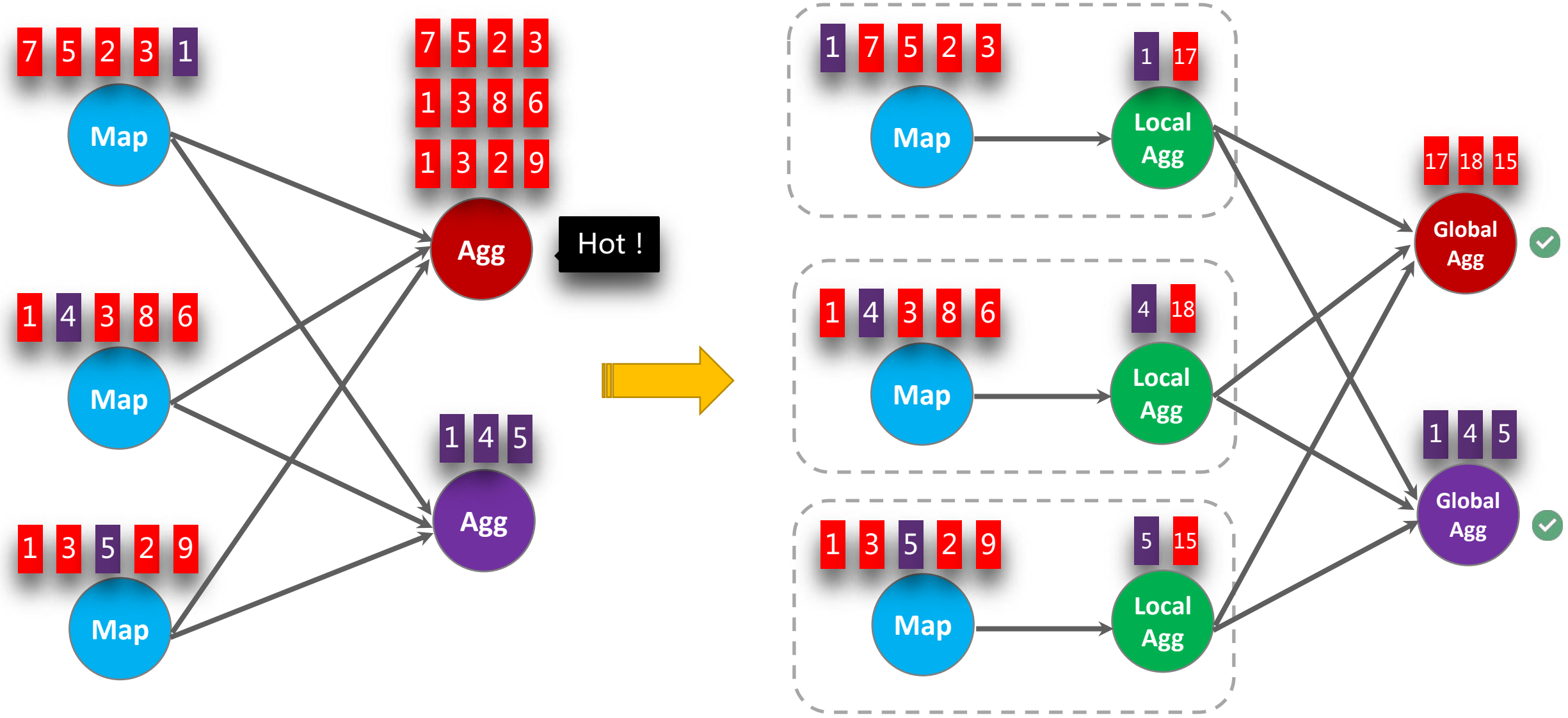


Simple Aggregation



MiniBatch Aggregation

Local-Global 优化



Local-Global 优化

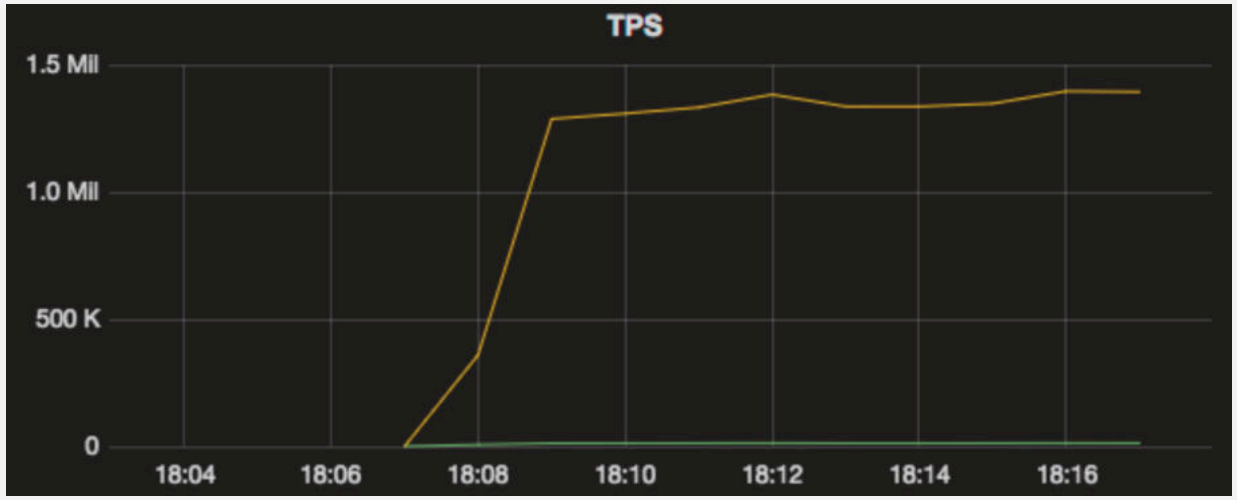
Local-Global 的优势：

1. 对于**普通聚合**，性能提升明显
2. 缓解数据倾斜
3. 减少网络传输

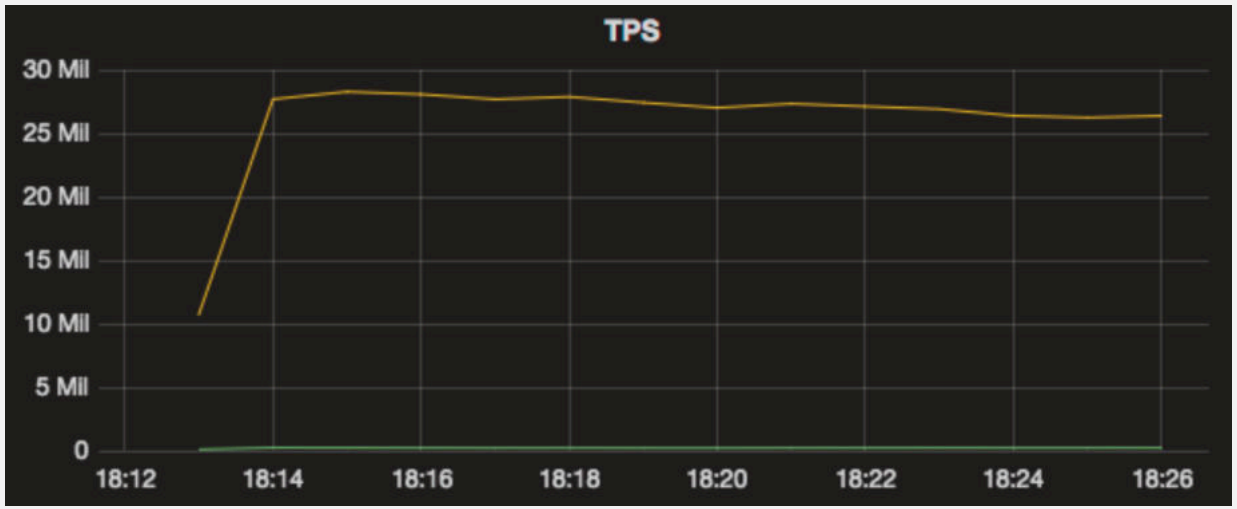
```
# 开启两阶段，即 local-global 优化
table.optimizer.agg-phase-strategy=TWO_PHASE

# 还需开启 mini-batch
table.exec.mini-batch.enabled=true
table.exec.mini-batch.allow-latency=5s
table.exec.mini-batch.size=5000
```

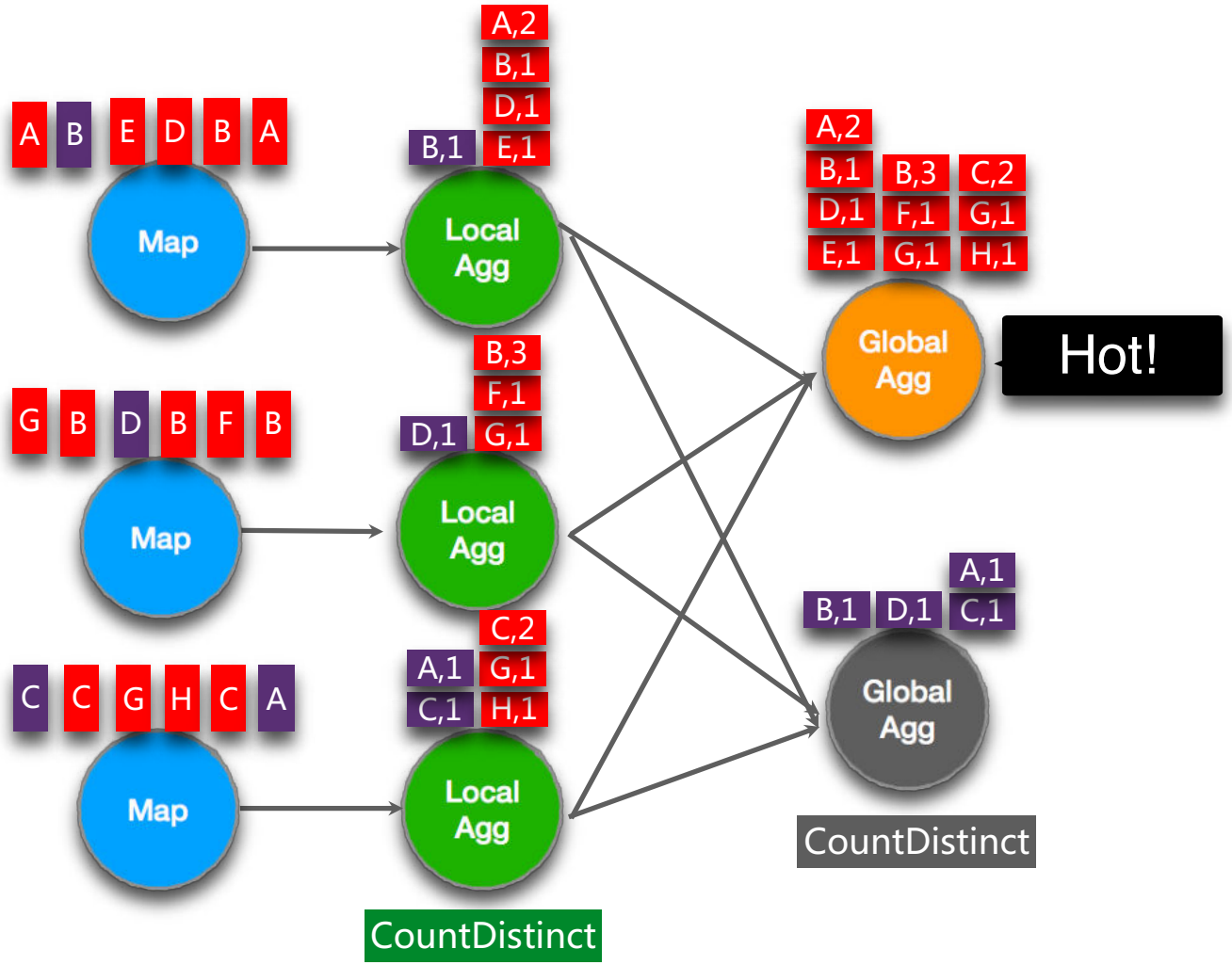
优化前



优化后



DISTINCT 数据倾斜问题



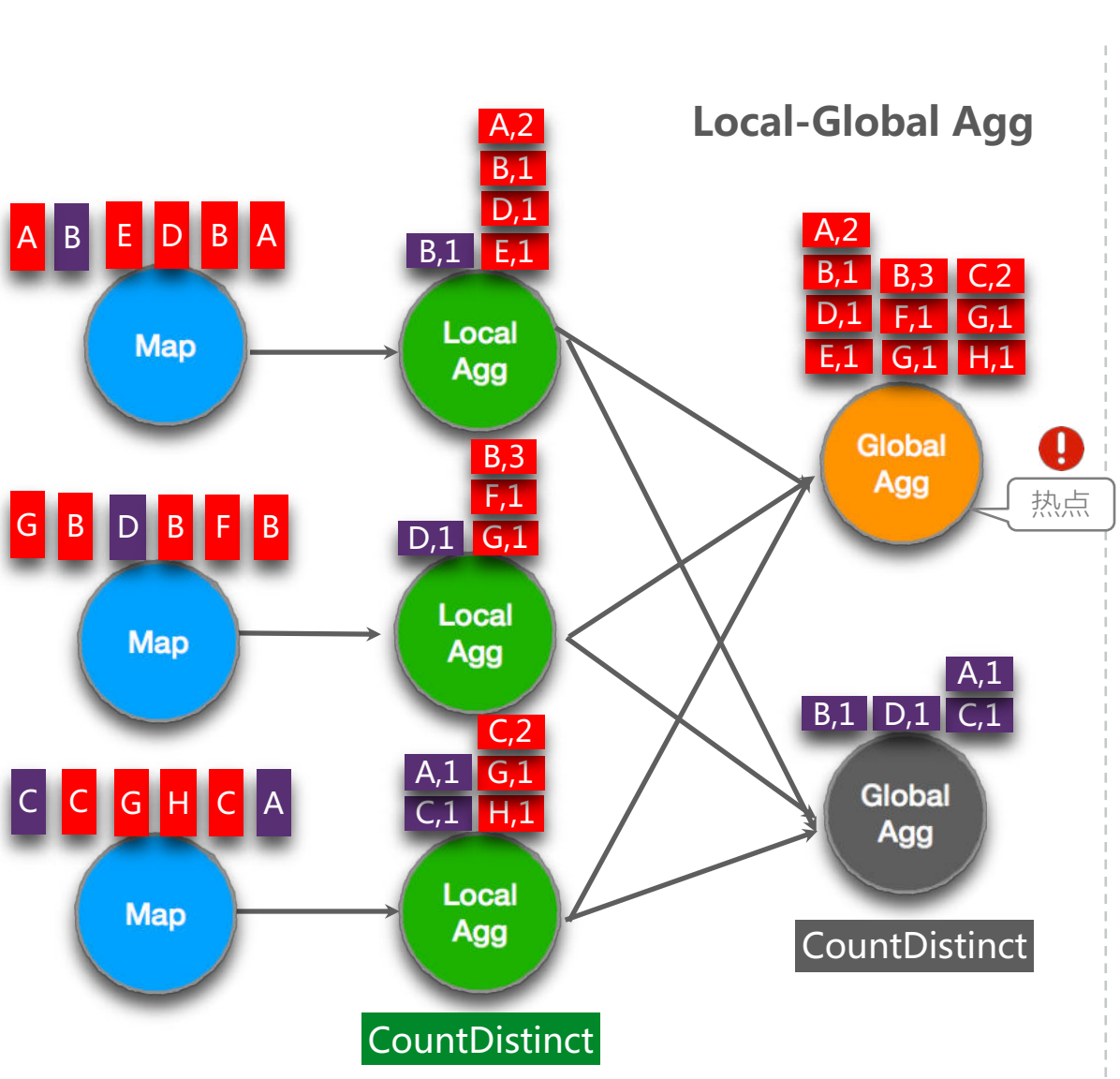
对于中间结果会存储明细的agg，

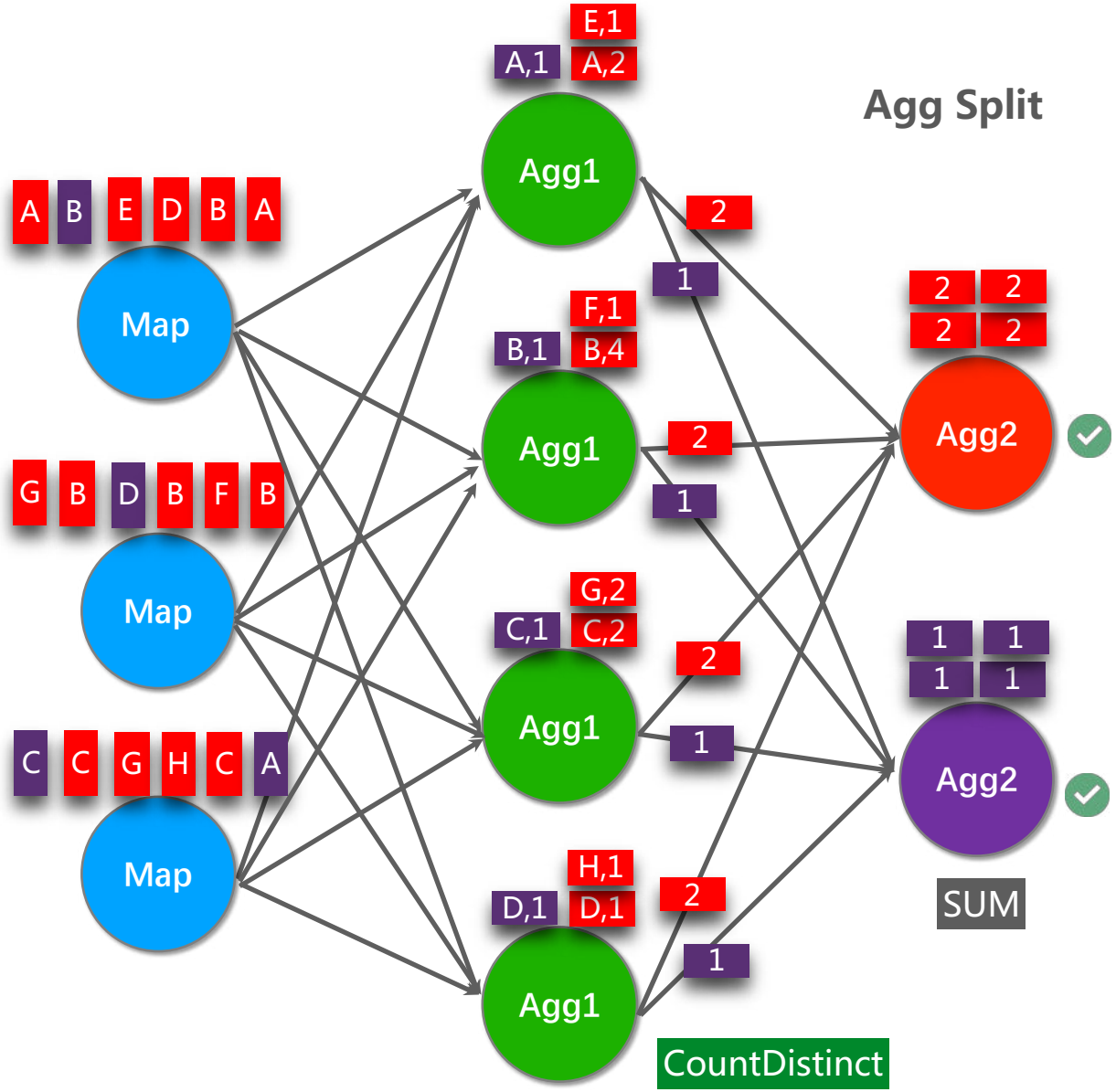
（例如 count distinct）

LocalGlobal无法有效解决热点问题

```
SELECT
  hour,
  COUNT(DISTINCT user) AS uv
FROM T
GROUP BY hour
```

COUNT DISTINCT 自动拆分





COUNT DISTINCT 自动拆分

1. 解决 COUNT DISTINCT 数据倾斜问题
2. 亦适用于多 DISTINCT 场景

```
COUNT(DISTINCT a)
SUM(DISTINCT b)
MAX(DISTINCT b)
```

```
# 开启 distinct agg 切分
table.optimizer.distinct-agg.split.enabled=true
```

```
SELECT
  hour,
  COUNT(DISTINCT user) AS uv
FROM T
GROUP BY hour
```

Simple Aggregation

```
SELECT hour, SUM(cnt) AS uv
FROM (
  SELECT hour, COUNT(DISTINCT user) AS cnt
  FROM T
  GROUP BY hour, MOD(HASH_CODE(user), 1024)
) GROUP BY hour
```

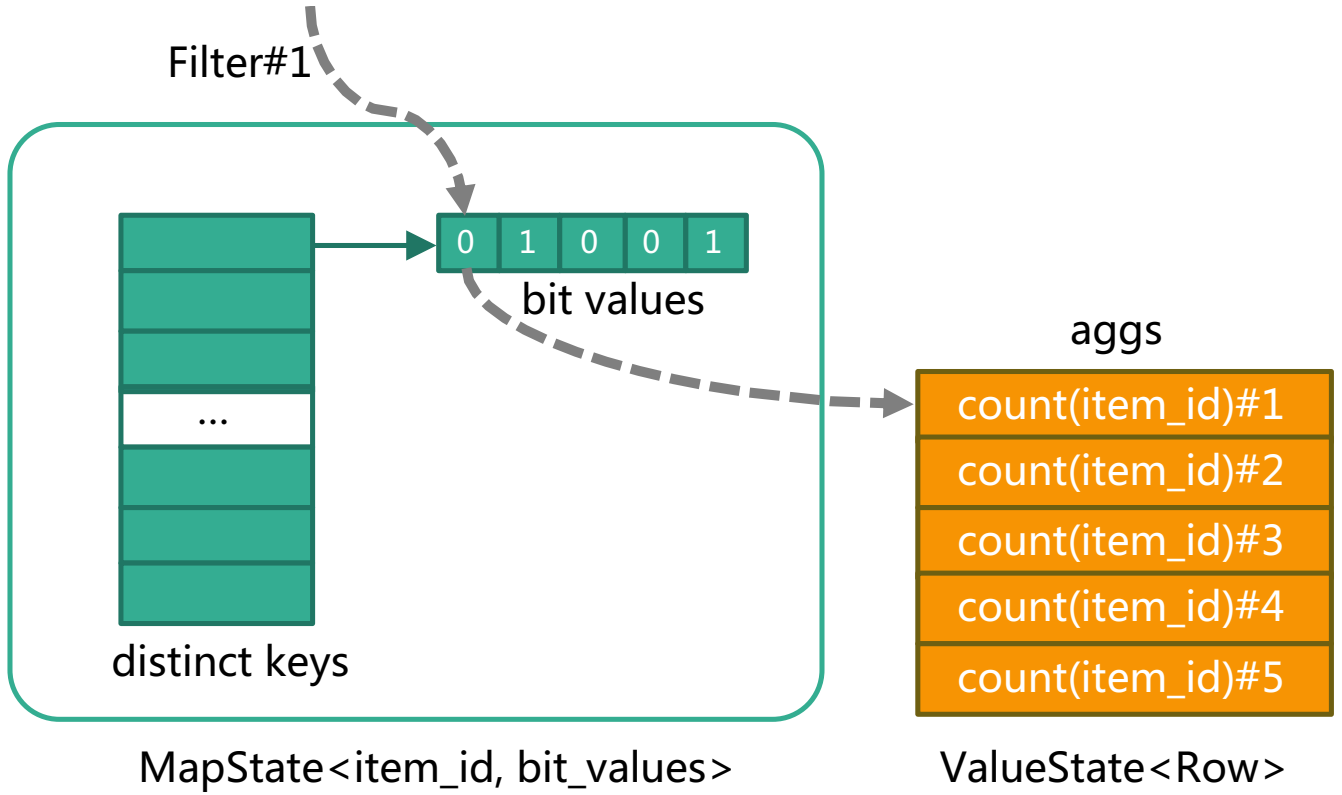
拆分后 Aggregation

大量 COUNT DISTINCT 场景的优化

```sql
SELECT
  date_time,
  shop_id,
  COUNT (DISTINCT item_id) AS item_col1,
  COUNT (DISTINCT item_id) FILTER (WHERE flag IN ('iphone')) AS item_col2,
  COUNT (DISTINCT item_id) FILTER (WHERE flag IN ('android')) AS item_col3,
  COUNT (DISTINCT item_id) FILTER (WHERE flag IN ('pc')) AS item_col4,
  COUNT (DISTINCT item_id) FILTER (WHERE flag IN ('wap')) AS item_col5,
  COUNT (DISTINCT item_id) FILTER (WHERE flag IN ('other')) AS item_col6,
  COUNT (DISTINCT item_id) FILTER (WHERE flag IN ('iphone', 'android')) AS item_col7,
  COUNT (DISTINCT item_id) FILTER (WHERE flag IN ('pc', 'other')) AS item_col8,
  COUNT (DISTINCT item_id) FILTER (WHERE flag IN ('iphone', 'android', 'wap')) AS item_col9,
  COUNT (DISTINCT item_id) FILTER (WHERE flag IN ('iphone', 'android', 'wap', 'pc', 'other')) AS item_col10,
  COUNT (DISTINCT visitor_id) AS visitor_col1,
  COUNT (DISTINCT visitor_id) FILTER (WHERE flag IN ('iphone')) AS visitor_col2,
  COUNT (DISTINCT visitor_id) FILTER (WHERE flag IN ('android')) AS visitor_col3,
  COUNT (DISTINCT visitor_id) FILTER (WHERE flag IN ('pc')) AS visitor_col4,
  COUNT (DISTINCT visitor_id) FILTER (WHERE flag IN ('wap')) AS visitor_col5,
  COUNT (DISTINCT visitor_id) FILTER (WHERE flag IN ('other')) AS visitor_col6,
  COUNT (DISTINCT visitor_id) FILTER (WHERE flag IN ('iphone', 'android')) AS visitor_col7,
  COUNT (DISTINCT visitor_id) FILTER (WHERE flag IN ('pc', 'other')) AS visitor_col8,
  COUNT (DISTINCT visitor_id) FILTER (WHERE flag IN ('iphone', 'android', 'wap')) AS visitor_col9,
  COUNT (DISTINCT visitor_id) FILTER (WHERE flag IN ('iphone', 'android', 'wap', 'pc', 'other')) AS visitor_col10
FROM logs
GROUP BY date_time, shop_id
```

大量 COUNT DISTINCT 场景的优化

```
SELECT
  key,
  count(distinct item_id) filter (…),
  count(distinct item_id) filter (…),
  count(distinct item_id) filter (…),
  count(distinct item_id) filter (…),
  count(distinct item_id) filter (…)
FROM T
GROUP BY key
```



节省了约1倍的 state size, 4倍的额外读写

实际测试性能提升1倍

05

Demo

FLINK FORWARD

Organized by 阿里云

Created by ververica

Flink Forward Asia

全球最大的 Apache Flink 官方会议

预计 2000+ 参会人员， 2019年11月28-30日 @北京国家会议中心

国内外一线厂商悉数参与

阿里巴巴、腾讯、字节跳动、intel、 DellEMC 、Uber、美团点评、Ververica ...

大会官网，查看更多

云邪的博客

专注分享Flink、Spark等大数据相关技术。

欢迎关注，共同进步！

THANKS