DTCC

2017第八届中国数据库技术大会

DATABASE TECHNOLOGY CONFERENCE CHINA 2017

大数据实时处理架构实践

朱健

提纲

- 实时计算简介
- 一个工程实践
- 系统设计建议
- 讨论和展望

什么是实时计算

- 低延时的流式数据处理
  - 离线计算的补充
  - 业务发展和技术进步的必然需求
- 关键点
  - 分布式流式数据
  - 低延迟

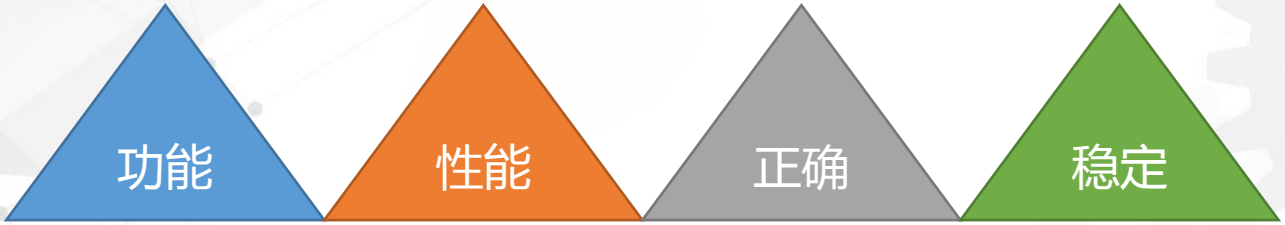
实时计算的要求

如此多的选择

samza

构建实时计算系统难点

- 坏消息：四座大山



- 误区：不是离线任务的实时化

一个工程实践例子

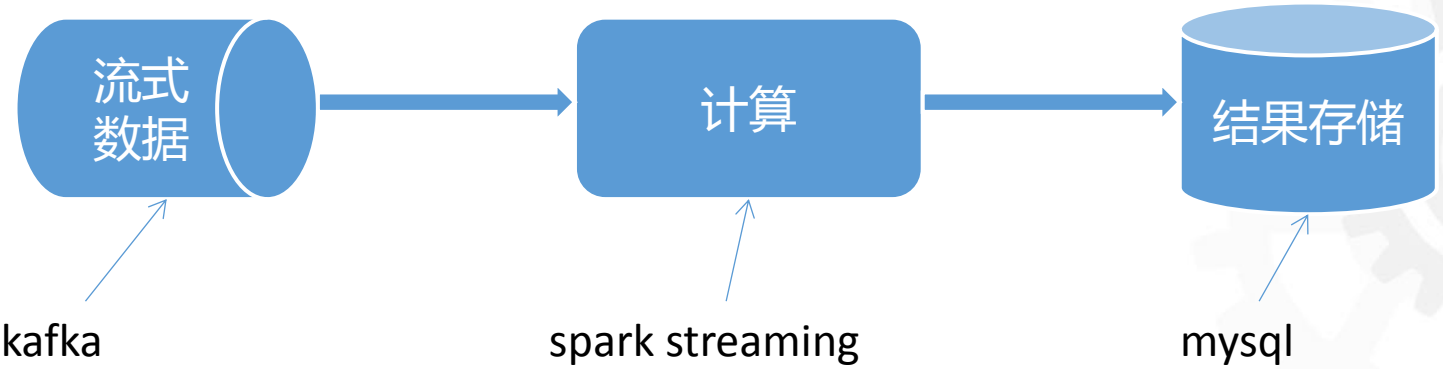
业务需求和挑战

- 实时呈现广告主展示、点击、消耗数据

- 数据量大：10w+ QPS
- 数据延时低：一分钟之内
- 数据准确无误
- 高可靠7x24

系统构想图

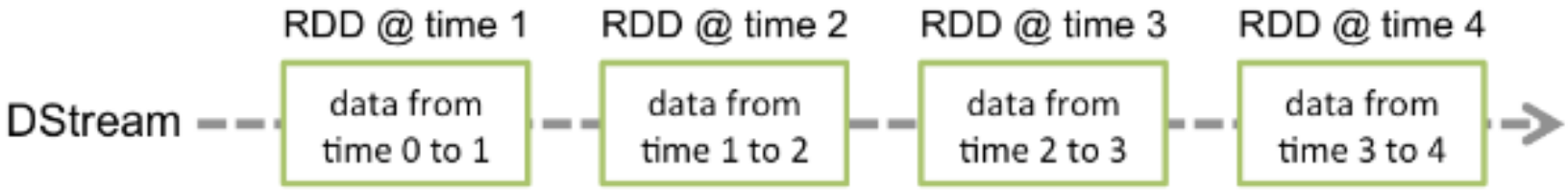
离线计算



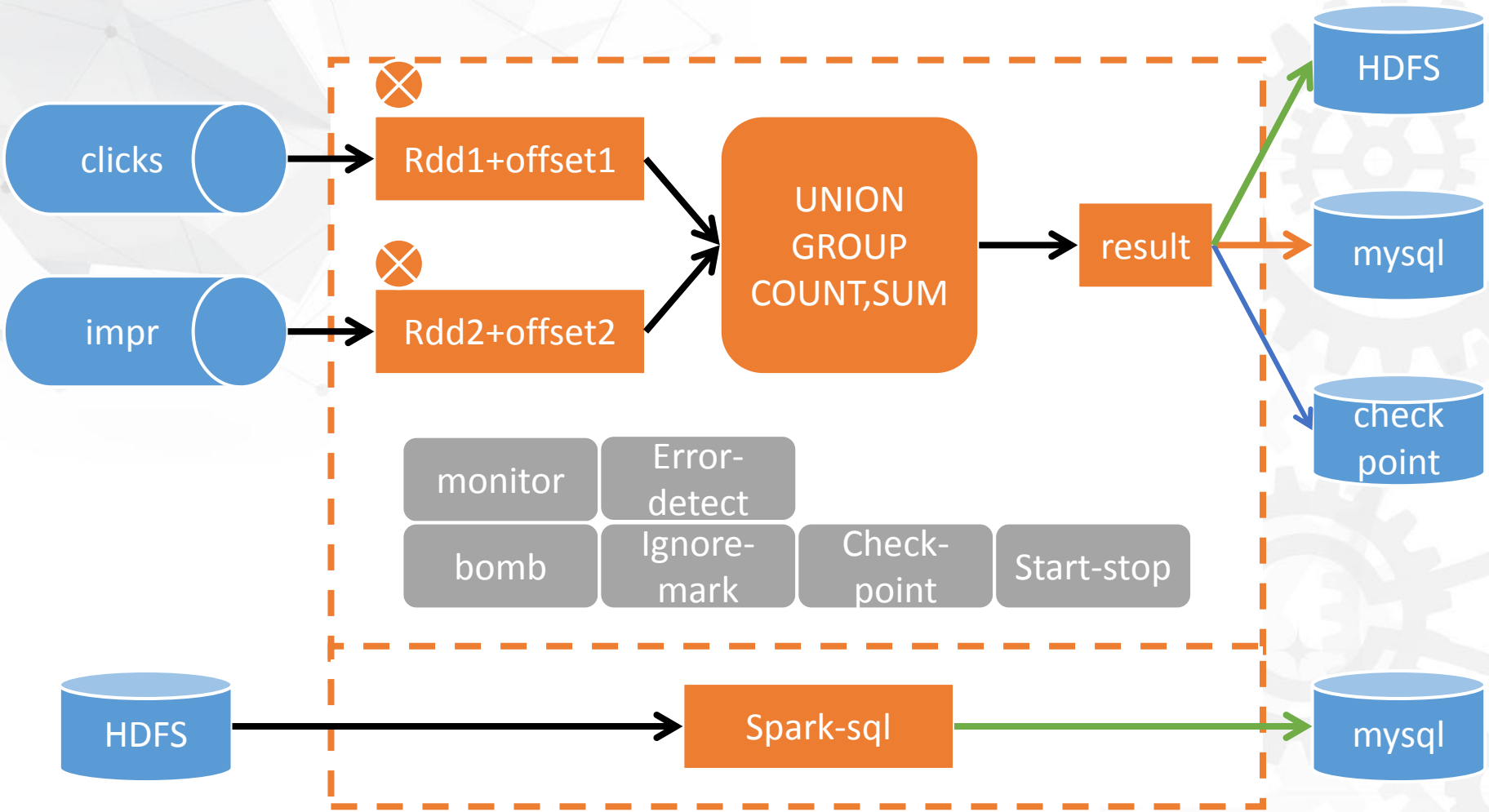
实时计算

Spark streaming简介

RDD：分布式的可恢复数据集合，spark基于此做运算
Dstream：把stream离散化成单个的RDD，运行spark引擎

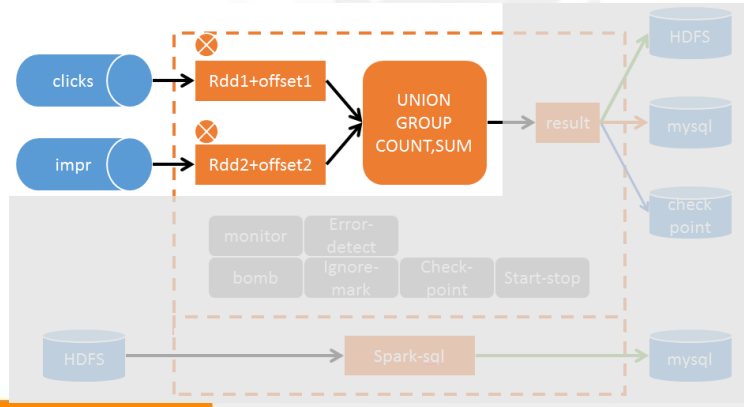


实际的系统架构


clicks
impr
Rdd1+offset1
Rdd2+offset2
UNION
GROUP
COUNT,SUM
result
HDFS
mysql
check
point
monitor
Error-
detect
bomb
Ignore-
mark
Check-
point
Start-stop
HDFS
Spark-sql
mysql

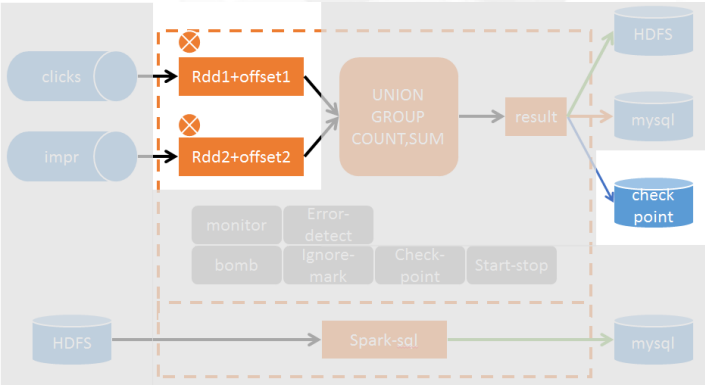
功能：多个流的处理

- 设计思路
  - 输入2个流：点击、展现
  - 点击和展现映射为2个表，执行Spark-SQL求出结果
- 难点
  - Spark streaming对多个流支持很弱
- 方案
  - 用transformWith函数绕过
  - 导致spark-ui监控数据无效
  - 无法处理超过2个流



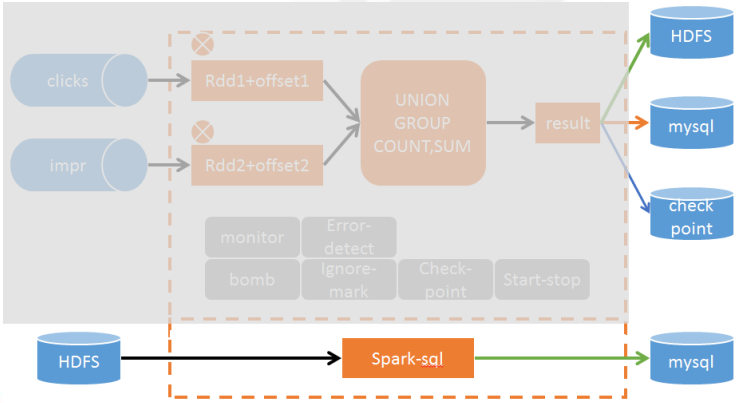
正确：数据读取

- 要求：数据源的数据不漏取、不重复取
- 难点
  - 流式数据的实时变化
- 方案
  - 选取kafka作为数据源
  - spark-streaming保证exactly-once
  - 持久化每个batch处理的kafka数据的offset
  - 注意：最后持久化offset



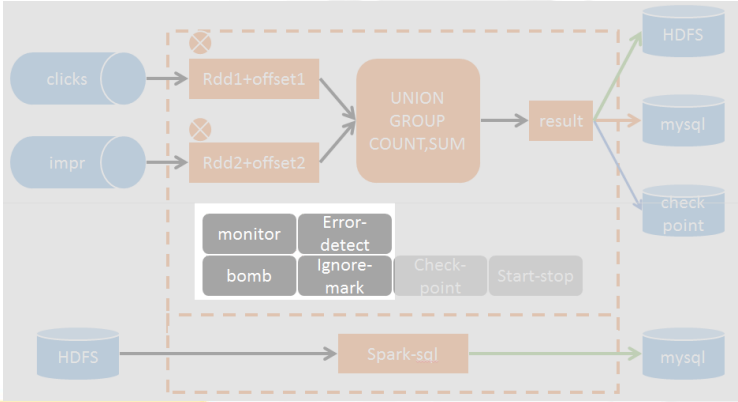
正确：数据产出

- 要求：结果不丢失不重复
- 难点
  - 分布式场景下，难以做到exactly-once
- 方案
  - 把数据输出变成幂等操作，每个batch的结果有唯一ID
  - HDFS覆盖写，checkpoint对应的id的数据才有效
  - Mysql只update一次
    - 引入HDFS存储准确的数据
    - 采用lambda架构，离线纠正数据



异常：错误处理

- 基本原则：快速失败重启，报警，人工介入
- 系统异常
  - 依据可靠度和业务，定义最小系统：kafka、spark、HDFS
  - 最小系统的失败，直接退出
    - 可能需要诊断系统：Batch卡死
  - 其他系统的失败，报警并跳过：mysql出错
- 程序异常
  - 严肃对待全部异常
  - 仔细合理的处理异常
    - 异常数据：跳过，报警
    - 批次执行失败：退出
  - 未知异常处理方法



异常：预案

- 系统异常预案
  - Kafka：退出，依赖于上游的预案
  - HDFS：可以选择切换其他HDFS或者本地磁盘
  - Mysql：主从备份
  - Spark：双集群
- 提供程序上的支持
  - 程序配置化

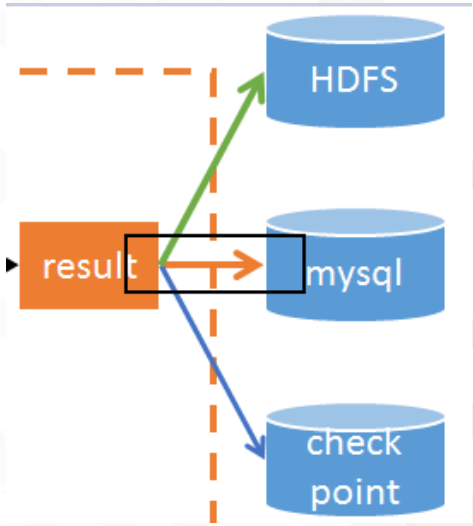
异常：失败恢复

- 难点：分布式状态恢复很难
  - 需要分布式持久化组件支持
- 方案：基于kafka-offset的无状态系统
  - 不使用spark的状态系统和checkpoint机制
    - 状态量很大：占用内存，稳定性、一致性问题
    - 不支持程序升级
  - 提供守护进程，退出自动重启
  - 好处：系统简单，快速重启，且数据一致
  - 坏处：需要编码实现，维护数据一致性

是否就可以高枕无忧？
提供7x24小时服务？

性能：慢节点问题

- Spark streaming 慢节点问题
  - 开启推测执行
  - 影响：mysql的update会重复
  - Spark的推测执行不能限定在某一个阶段，spark也不提供commit接口（与MR不同）
- 方案
  - 开启推测执行
  - 让task_attempt_id=0的任务写mysql
  - 其他attempt任务等待

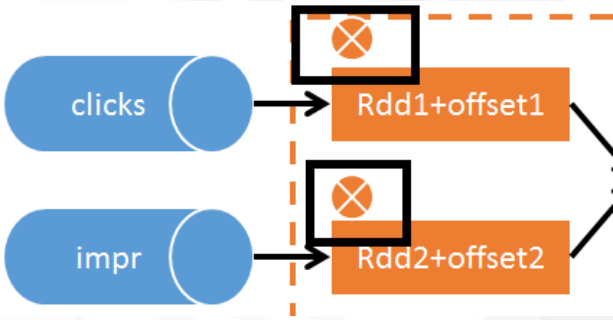


性能：提高吞吐量

- 问题
  - 因为checkpoint机制的顺序问题，导致批次之间无法并发
  - Spark在一个批次内顺序执行任务DAG
- 方案
  - Spark的自有特性，难以改变
  - 可以考虑基于streaming receiver的方式读取kafka

稳定：平稳处理数据

- 问题：突然数据量剧增
  - 比如，spark集群故障，恢复后kafka数据积压
  - Spark内存资源有限，无法容纳全部延迟数据
  - Spark中间spill磁盘文件有2G的限制
- 方案：输入限流
  - Spark的kafka集成方案有内置参数：spark.streaming.kafka.maxRatePerPartition和spark.streaming.backpressure.enabled
  - 局限：全局限速，无法针对某个输入流



其他：如何升级

- 问题：如何优雅升级，同时保持程序状态
  - spark的checkpoint有无法升级的问题
- 方案
  - 基于kafka-offset的无状态系统
  - 设置自动退出机制，运行完一个完整的批次后退出。

系统设计建议

建议1

- 是否真的需要实时计算
  - 实时系统是复杂的、昂贵的
- 功能性
  - 多数据流支持、高阶API
- 低延时和吞吐量
  - 大数据、秒级延迟不建议选用spark，可以考虑storm和flink
- 数据准确性
  - 实时计算框架本身是否支持exactly-once
  - 输入源是否支持exactly-once
  - 输出组件是否支持exactly-once

建议2

- 程序状态管理
  - 依靠计算框架，会有加载延迟，升级风险
  - 自己实现外部状态管理，较复杂
- 程序异常处理和恢复
  - 实时系统需要仔细设计和编码
  - 程序如何处理异常
  - 程序处理不了的，人工预案是什么
- 升级
- 流控
- 监控

讨论和展望

讨论-状态存储问题

- 用户数据状态存储
  - 方案一：使用计算框架存储
    - 优点：使用简单，一致性保证
    - 缺点：不成熟，难以处理大数据，可能有潜在的不一致风险，初始化压力大，程序升级困难，难以纠正
  - 方案二：使用合适的成熟的外部存储组件
    - 优点：外部存储组件更成熟、稳定，功能更丰富，与程序状态解耦，减轻程序负担
    - 缺点：复杂，自己保证数据一致性
  - 个人建议
    - 大数据量和高一致性要求的应用，优先选择外部状态存储

讨论-数据一致性

- 框架内部数据一致性（exactly-once）
  - 对计算框架的基本要求
- 框架和外部组件的数据一致性
  - 计算框架的exactly-once不等于应用系统的exactly-once
  - 分布式一致性问题的解决办法：重试
    - 要求幂等操作，比如赋值
    - 非幂等操作会导致数据错误，比如累加
  - 方案一：把非幂等操作转化为幂等操作
    - 有时候并不可行
  - 方案二：提供基于唯一id的操作
    - 要求：操作是顺序的
    - 做法：保留上一次操作的id。如果本次id等于上次id，则忽略，否则执行操作并保留本次id
    - 问题：存储器往往不支持

需要生态支持

- 实时计算框架发展迅速，但是周边生态支持甚少
- 呼吁（我们50%的时间都在解决这些问题）：
  - 数据源系统：提供数据可重入机制（像kafka的offset机制）
  - 数据存储系统：提供幂等操作机制（基于id的操作）

Q&A

THANKS

SequeMedia 盛拓传媒 ITPUB ChinaUnix