字节跳动基于 Flink 的 MQ-Hive 实时数据集成

作者：字节跳动技术团队
发布于：2020 年 7 月 26 日 10:00

背景

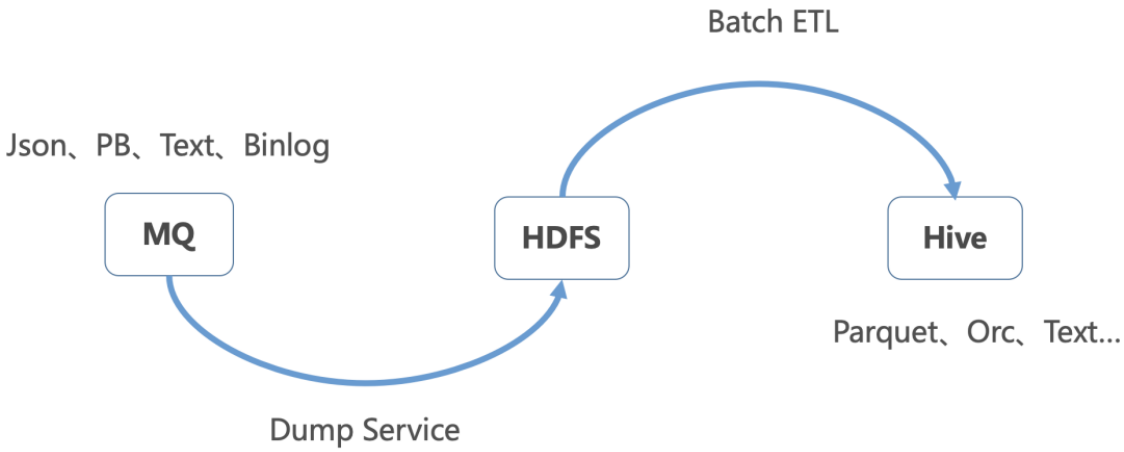
在数据中台建设过程中，一个典型的数据集成场景是将 MQ (Message Queue，例如 Kafka、RocketMQ 等) 的数据导入到 Hive 中，以供下游数仓建设以及指标统计。由于 MQ-Hive 是数仓建设第一层，因此对数据的准确性以及实时性要求比较高。

本文主要围绕 MQ-Hive 场景，针对目前字节跳动内已有解决方案的痛点，提出基于 Flink 的实时解决方案，并介绍新方案在字节跳动内部的使用现状。

已有方案及痛点

字节跳动内已有解决方案如下图所示，主要分了两个步骤：

1. 通过 Dump 服务将 MQ 的数据写入到 HDFS 文件
2. 再通过 Batch ETL 将 HDFS 数据导入到 Hive 中，并添加 Hive 分区



痛点

- 任务链较长，原始数据需要经过多次转换最终才能进入 Hive
- 实时性比较差，Dump Service、Batch ETL 延迟都会导致最终数据产出延迟
- 存储、计算开销大，MQ 数据重复存储和计算
- 基于原生 Java 打造，数据流量持续增长后，存在单点故障和机器负载不均衡等问题
- 运维成本较高，架构上无法复用公司内 Hadoop/Flink/Yarn 等现有基础设施
- 不支持异地容灾

推荐阅读

阿里巴巴 Flink 踩坑经验：如何
HDFS 压力?
2020 年 1 月 9 日

日处理数据量超 10 亿：友信
Flink 构建实时用户画像系统的实
2019 年 12 月 6 日

58 同城基于 Flink 的千亿级实时
架构实践
2020 年 1 月 8 日

Kylin 实时流处理技术探秘
2019 年 4 月 11 日

BigTable 的开源实现：HBase
2018 年 11 月 29 日

答疑课堂：MySQL 中 InnoDB 的
讲
2019 年 8 月 22 日

流计算与消息（二）：在流计
Kafka 链接计算任务
2019 年 10 月 3 日

基于 Flink 实时解决方案

优势

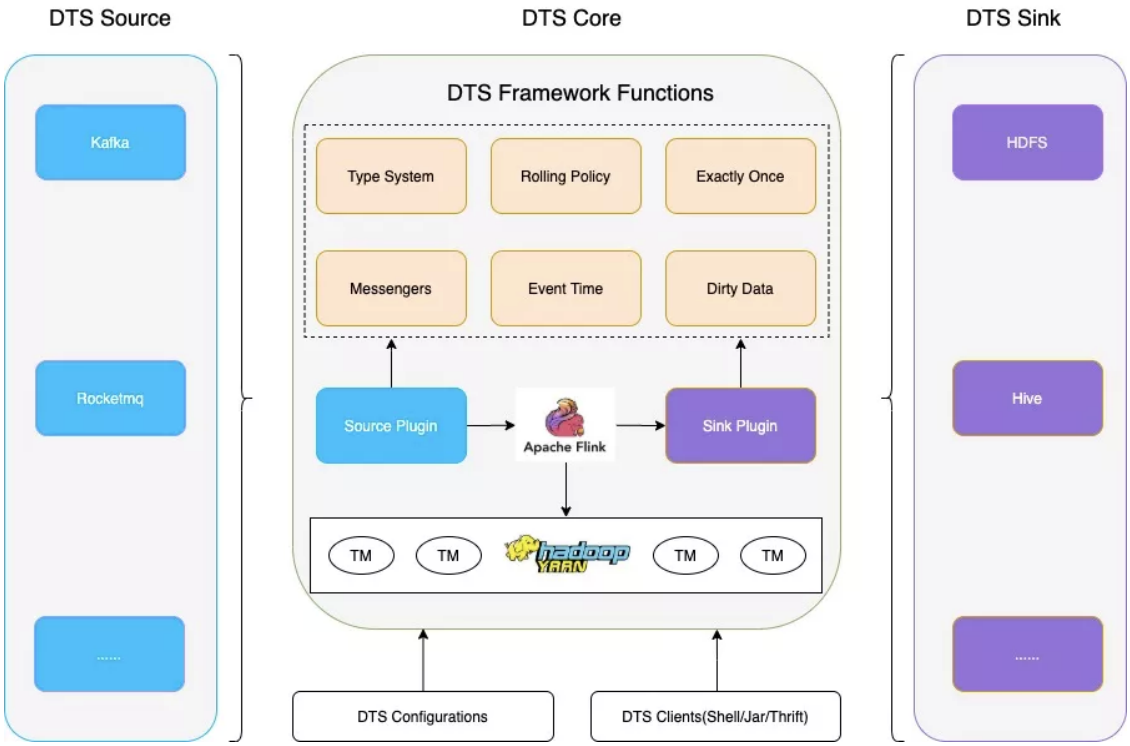
针对目前公司传统解决方案的痛点，我们提出基于 Flink 的实时解决方案，将 MQ 的数据实时写入到 Hive，并支持事件时间以及 Exactly Once 语义。相比老方案，新方案优势如下所示：

1. 基于流式引擎 Flink 开发，支持 Exactly Once 语义
2. 实时性更高，MQ 数据直接进入 Hive，无中间计算环节
3. 减少中间存储，整个流程数据只会落地一次
4. 支撑 Yarn 部署模式，方便用户迁移
5. 资源管理弹性，方便扩容以及运维
6. 支持双机房容灾

整体架构

整体架构如下图所示，主要包括 DTS(Data Transmission Service) Source、DTS Core、DTS Sink 三大模块，具体功能如下：

1. DTS Source 接入不同 MQ 数据源，支持 Kafka、RocketMQ 等
2. DTS Sink 将数据输出到目标数据源，支持 HDFS、Hive 等
3. DTS Core 贯穿整个数据同步流程，通过 Source 读取源端数据，经过 DTS Framework 处理，最后通过 Sink 将数据输出到目标端。
4. DTS Framework 集成类型系统、文件切分、Exactly Once、任务信息采集、事件时间、脏数据收集等核心功能
5. 支持 Yarn 部署模式，资源调度、管理比较弹性



DTS Dump 架构图

Exactly Once

Flink 框架通过 Checkpoint 机制，能够提供 Exactly Once 或者 At Least Once 语义。为了实现 MQ-Hive 全链路支持

Exactly-once 语义，还需要 MQ Source、Hive Sink 端支持 Exactly Once 语义。本文通过 Checkpoint + 2PC 协议实现，具体过程如下：

1. 数据写入时，Source 端从上游 MQ 拉取数据并发送到 Sink 端；Sink 端将数据写入到临时目录中
2. Checkpoint Snapshot 阶段，Source 端将 MQ Offset 保存到 State 中；Sink 端关闭写入的文件句柄，并保存当前 Checkpoint ID 到 State 中；
3. Checkpoint Complete 阶段，Source 端 Commit MQ Offset；Sink 端将临时目录中的数据移动到正式目录下
4. Checkpoint Recover 阶段，加载最新一次成功的 Checkpoint 目录并恢复 State 信息，其中 Source 端将 State 中保存的 MQ Offset 作为起始位置；Sink 端恢复最新一次成功的 Checkpoint ID，并将临时目录的数据移动到正式目录下

实现优化

在实际使用场景中，特别是大并发场景下，HDFS 写入延迟容易有毛刺，因为个别 Task Snapshot 超时或者失败，导致整个 Checkpoint 失败的问题会比较明显。因此针对 Checkpoint 失败，提高系统的容错性以及稳定性就比较重要。

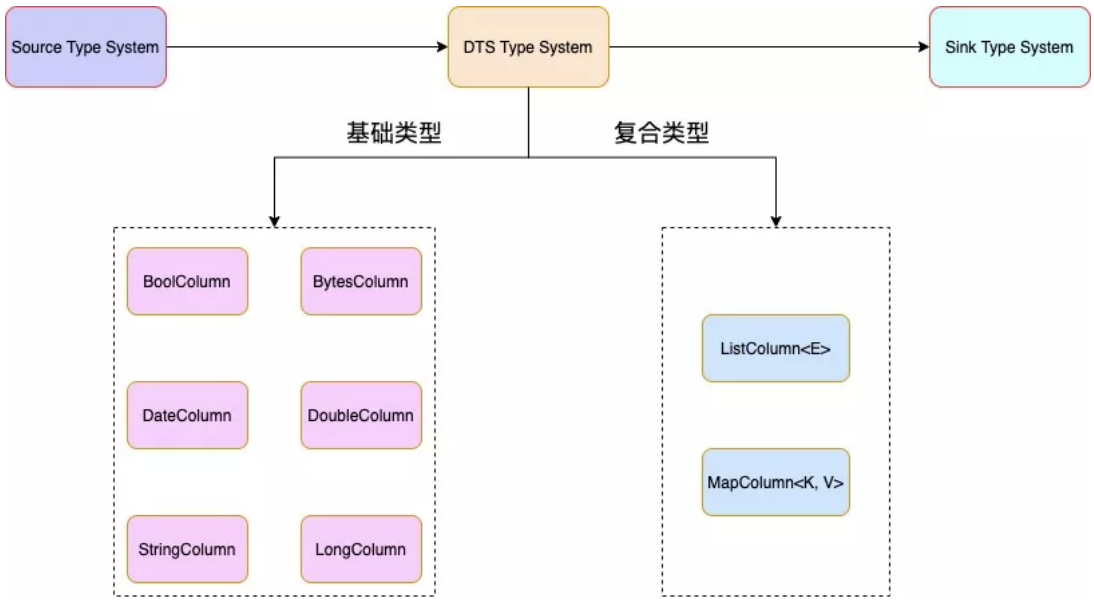
这里充分利用 Checkpoint ID 严格单调递增的特性，每一次做 Checkpoint 时，当前 Checkpoint ID 一定比以前大，因此在 Checkpoint Complete 阶段，可以提交小于等于当前 Checkpoint ID 的临时数据。具体优化策略如下：

- Sink 端临时目录为{dump_path}/{next_cp_id}，这里 next_cp_id 的定义是当前最新的 cp_id + 1
- Checkpoint Snapshot 阶段，Sink 端保存当前最新 cp_id 到 State，同时更新 next_cp_id 为 cp_id + 1
- Checkpoint Complete 阶段，Sink 端将临时目录中所有小于等于当前 cp_id 的数据移动到正式目录下
- Checkpoint Recover 阶段，Sink 端恢复最新一次成功的 cp_id，并将临时目录中小于等于当前 cp_id 的数据移动到正式目录下

类型系统

由于不同数据源支持的数据类型不一样，为了解决不同数据源间的数据同步以及不同类型转换兼容的问题，我们支持了 DTS 类型系统，DTS 类型可细化为基础类型和复合类型，其中复合类型支持类型嵌套，具体转换流程如下：

1. 在 Source 端，将源数据类型，统一转成系统内部的 DTS 类型
2. 在 Sink 端，将系统内部的 DTS 类型转换成目标数据源类型
3. 其中 DTS 类型系统支持不同类型间的相互转换，比如 String 类型与 Date 类型的相互转换



DTS Dump 架构图

Rolling Policy

Sink 端是并发写入，每个 Task 处理的流量不一样，为了避免生成太多的小文件或者生成的文件过大，需要支持自定义文件切分策略，以控制单个文件的大小。目前支持三种文件切分策略：文件大小、文件最长未更新时间、Checkpoint。

优化策略

Hive 支持 Parquet、Orc、Text 等多种存储格式，不同的存储格式数据写入过程不太一样，具体可以分为两大类：

1. RowFormat：基于单条写入，支持按照 Offset 进行 HDFS Truncate 操作，例如 Text 格式
2. BulkFormat：基于 Block 写入，不支持 HDFS Truncate 操作，例如 Parquet、ORC 格式

为了保障 Exactly Once 语义，并同时支持 Parquet、Orc、Text 等多种格式，在每次 Checkpoint 时，强制做文件切分，保证所有写入的文件都是完整的，Checkpoint 恢复时不用做 Truncate 操作。

容错处理

理想情况下流式任务会一直运行不需要重启，但实际不可避免会遇到以下几个场景：

1. Flink 计算引擎升级，需要重启任务
2. 上游数据增加，需要调整任务并发度
3. Task Failover

并发度调整

目前 Flink 原生支持 State Rescale。具体实现中，在 Task 做 Checkpoint Snapshot 时，将 MQ Offset 保存到 ListState 中；Job 重启后，Job Master 会根据 Operator 并发度，将 ListState 平均分配到各个 Task 上。

Task Failover

由于网络抖动、写入超时等外部因素的影响，Task 不可避免会出现写入失败，如何快速、准确的做 Task Failover 就显得比较重要。目前 Flink 原生支持多种 Task Failover 策略，本文使用 Region Failover 策略，将失败 Task 所在 Region 的所有 Task 都重启。

异地容灾

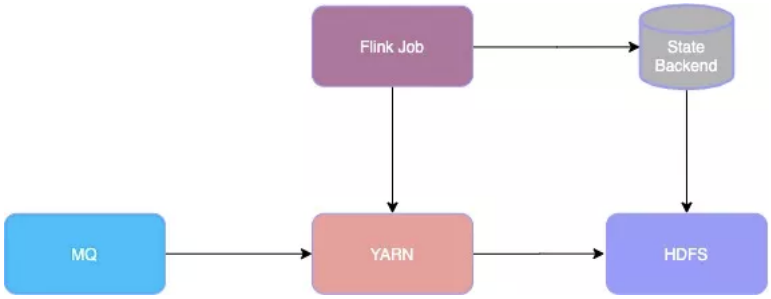
背景

大数据时代，数据的准确性和实时性显得尤为重要。本文提供多机房部署及异地容灾解决方案，当主机房因为断网、断电、地震、火灾等原因暂时无法对外提供服务时，能快速将服务切换到备灾机房，并同时保障 Exactly Once 语义。

容灾组件

整体解决方案需要多个容灾组件一起配合实现，容灾组件如下图所示，主要包括 MQ、YARN、HDFS，具体如下：

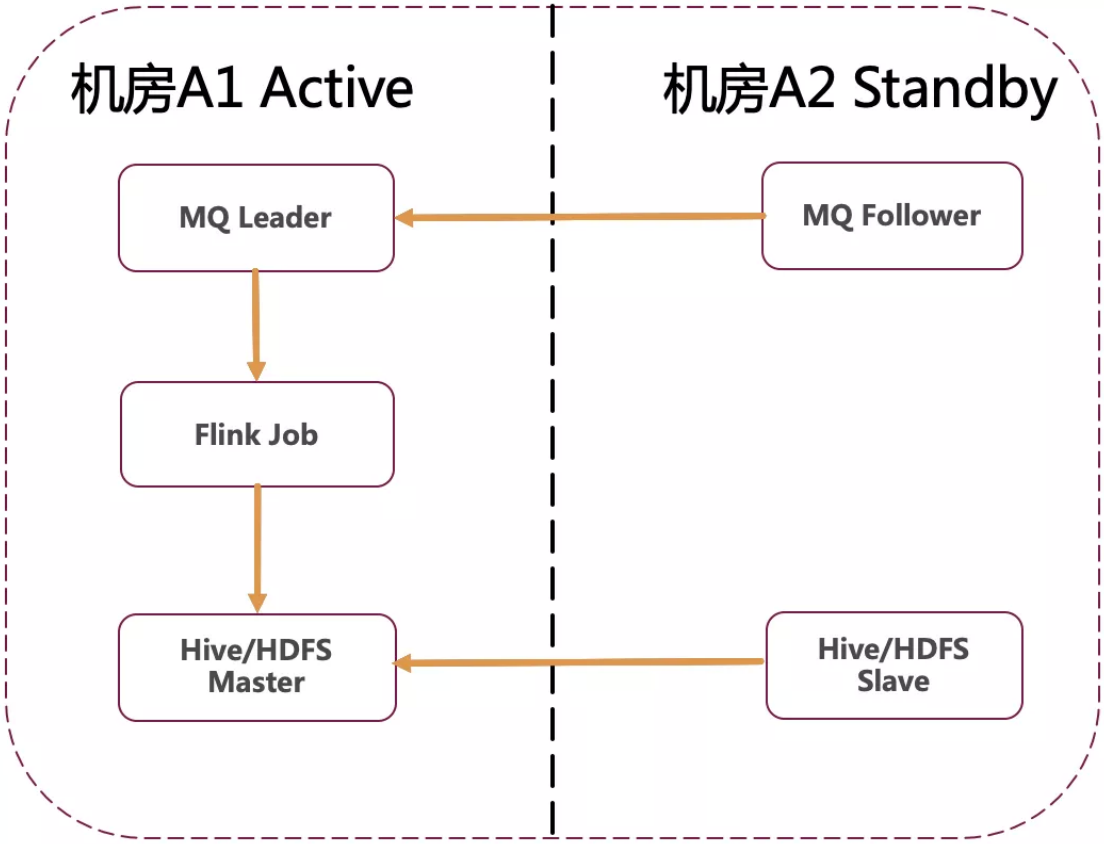
1. MQ 需要支持多机房部署，当主机房故障时，能将 Leader 切换到备机房，以供下游消费
2. Yarn 集群在主机房、备机房都有部署，以便 Flink Job 迁移
3. 下游 HDFS 需要支持多机房部署，当主机房故障时，能将 Master 切换到备机房
4. Flink Job 运行在 Yarn 上，同时任务 State Backend 保存到 HDFS，通过 HDFS 的多机房支持保障 State Backend 的多机房

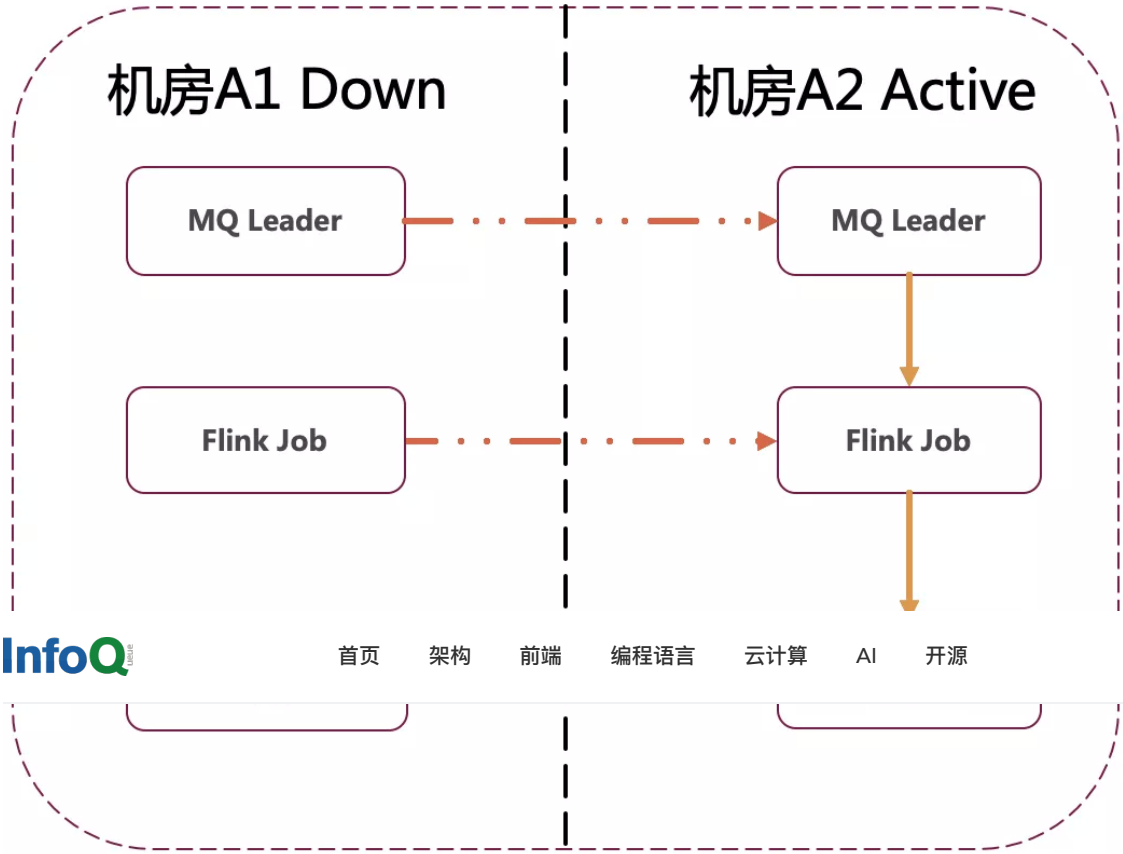


容灾过程

整体容灾过程如下所示：

1. 正常情况下，MQ Leader 以及 HDFS Master 部署在主机房，并将数据同步到备机房。同时 Flink Job 运行在主机房，并将任务 State 写入到 HDFS 中，注意 State 也是多机房部署模式
2. 灾难情况下，MQ Leader 以及 HDFS Master 从主机房迁移到备灾机房，同时 Flink Job 也迁移到备灾机房，并通过 State 恢复灾难前的 Offset 信息，以提供 Exactly Once 语义





事件时间归档

背景

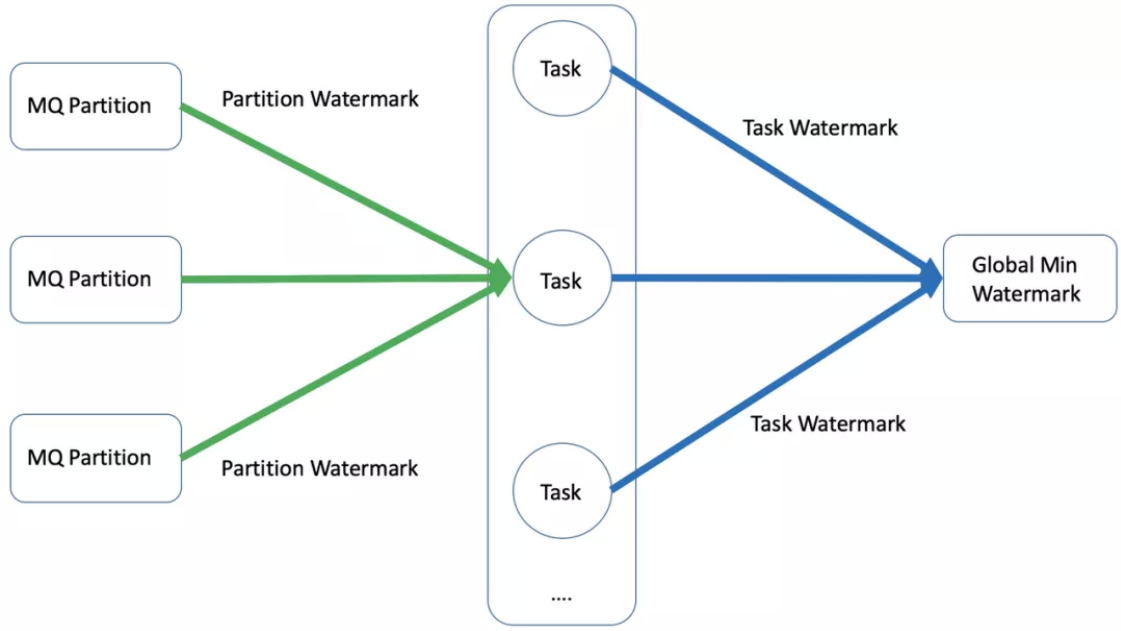
在数仓建设中，处理时间 (Process Time) 和事件时间 (Event Time) 的处理逻辑不太一样，对于处理时间会将数据写到当前系统时间所对应的时间分区下；对于事件时间，则是根据数据的生产时间将数据写到对应时间分区下，本文也简称为归档。

在实际场景中，不可避免会遇到各种上下游故障，并在持续一段时间后恢复，如果采用 Process Time 的处理策略，则事故期间的数据会写入到恢复后的时间分区下，最终导致分区空洞或者数据漂移的问题；如果采用归档的策略，会按照事件时间写入，则没有此类问题。

由于上游数据事件时间会存在乱序，同时 Hive 分区生成后就不应该再继续写入，因此实际写入过程中不可能做到无限归档，只能在一定时间范围内归档。归档的难点在于如何确定全局最小归档时间以及如何容忍一定的乱序。

全局最小归档时间

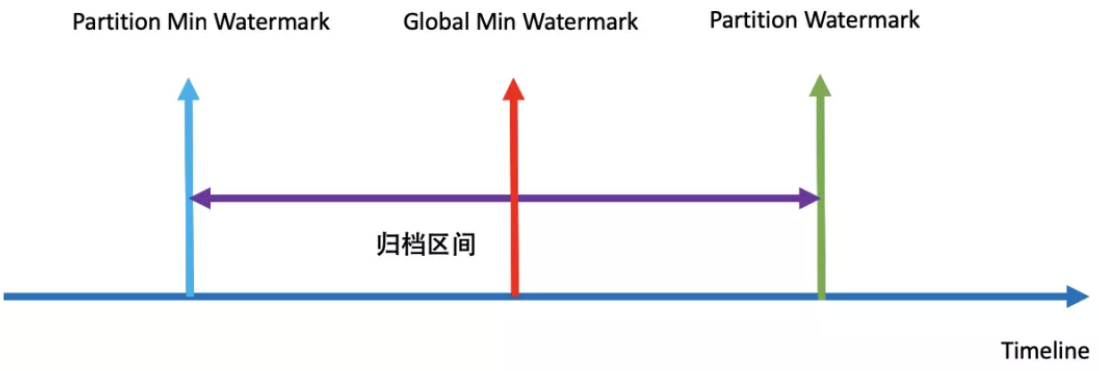
Source 端是并发读取，并且一个 Task 可能同时读取多个 MQ Partition 的数据，对于 MQ 的每一个 Parititon 会保存当前分区归档时间，取分区中最小值作为 Task 的最小归档时间，最终取 Task 中最小值，作为全局最小归档时间。



乱序处理

为了支持乱序的场景，会支持一个归档区间的设置，其中 Global Min Watermark 为全局最小归档时间，Partition Watermark 为分区当前归档时间，Partition Min Watermark 为分区最小归档时间，只有当事件时间满足以下条件时，才会进行归档：

1. 事件时间大于全局最小归档时间
2. 事件时间大于分区最小归档时间

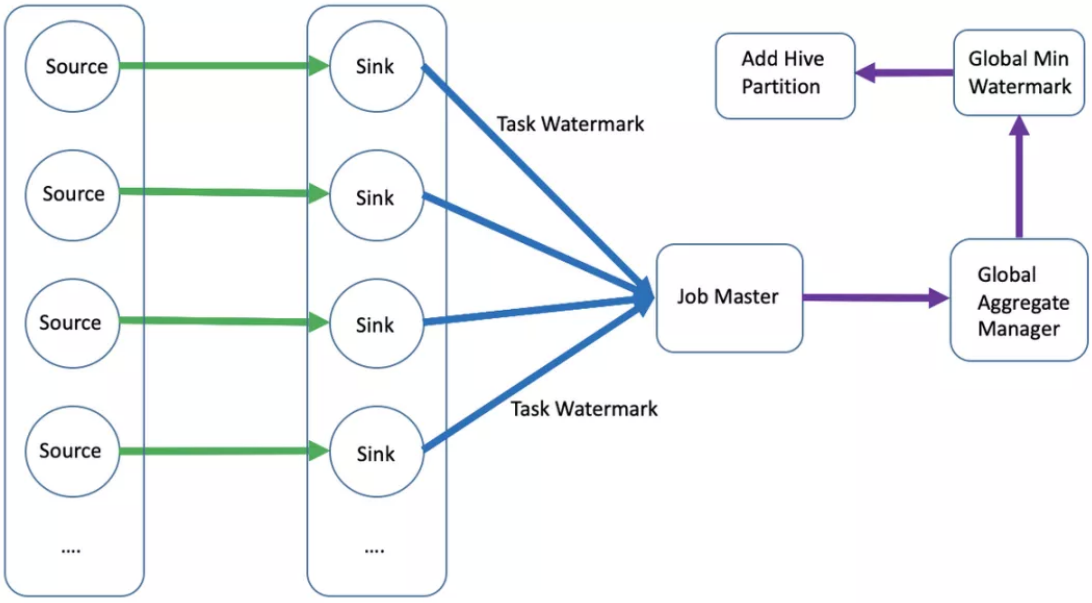


Hive 分区生成

原理

Hive 分区生成的难点在于如何确定分区的数据是否就绪以及如何添加分区。由于 Sink 端是并发写入，同时会有多个 Task 写同一个分区数据，因此只有当所有 Task 分区数据写入完成，才能认为分区数据是就绪，本文解决思路如下：

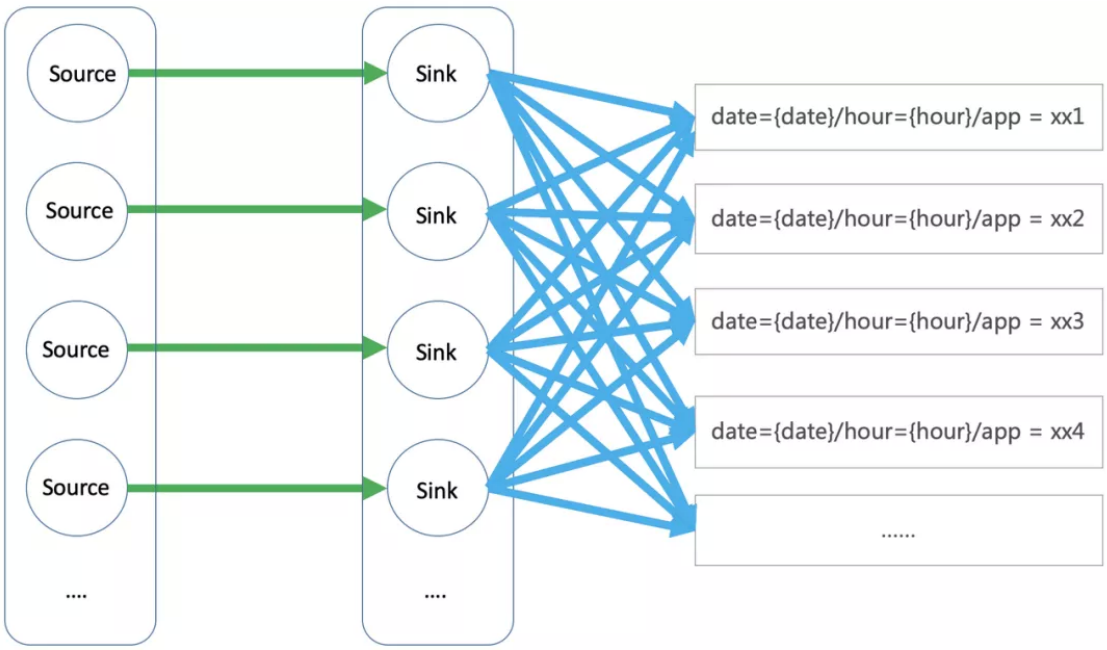
1. 在 Sink 端，对于每个 Task 保存当前最小处理时间，需要满足单调递增的特性
2. 在 Checkpoint Complete 时，Task 上报最小处理时间到 JM 端
3. JM 拿到所有 Task 的最小处理时间后，可以得到全局最小处理时间，并以此作为 Hive 分区的最小就绪时间
4. 当最小就绪时间更新时，可判断是否添加 Hive 分区



动态分区

动态分区是根据上游输入数据的值，确定数据写到哪个分区目录，而不是写到固定分区目录，例如 date={date}/hour={hour}/app={app}的场景，根据分区时间以及 app 字段的值确定最终的分区目录，以实现每个小时内，相同的 app 数据在同一个分区下。

在静态分区场景下，每个 Task 每次只会写入一个分区文件，但在动态分区场景下，每个 Task 可能同时写入多个分区文件。对于 Parque 格式的写入，会先将数据写到做本地缓存，然后批次写入到 Hive，当 Task 同时处理的文件句柄过多时，容易出现 OOM。为了防止单 Task OOM，会周期性对文件句柄做探活检测，及时释放长时间没有写入的文件句柄。

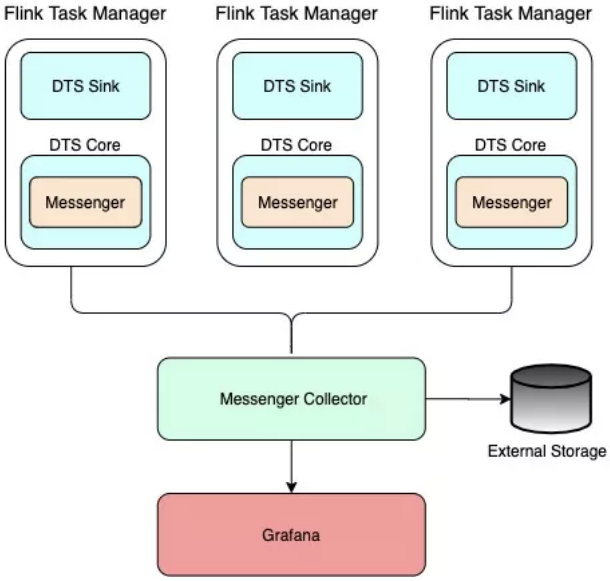


Messenger

Messenger 模块用于采集 Job 运行状态信息，以便衡量 Job 健康度以及大盘指标建设。

元信息采集

元信息采集的原理如下所示，在 Sink 端通过 Messenger 采集 Task 的核心指标，例如流量、QPS、脏数据、写入 Latency、事件时间写入效果等，并通过 Messenger Collector 汇总。其中脏数据需要输出到外部存储中，任务运行指标输出到 Grafana，用于大盘指标展示。

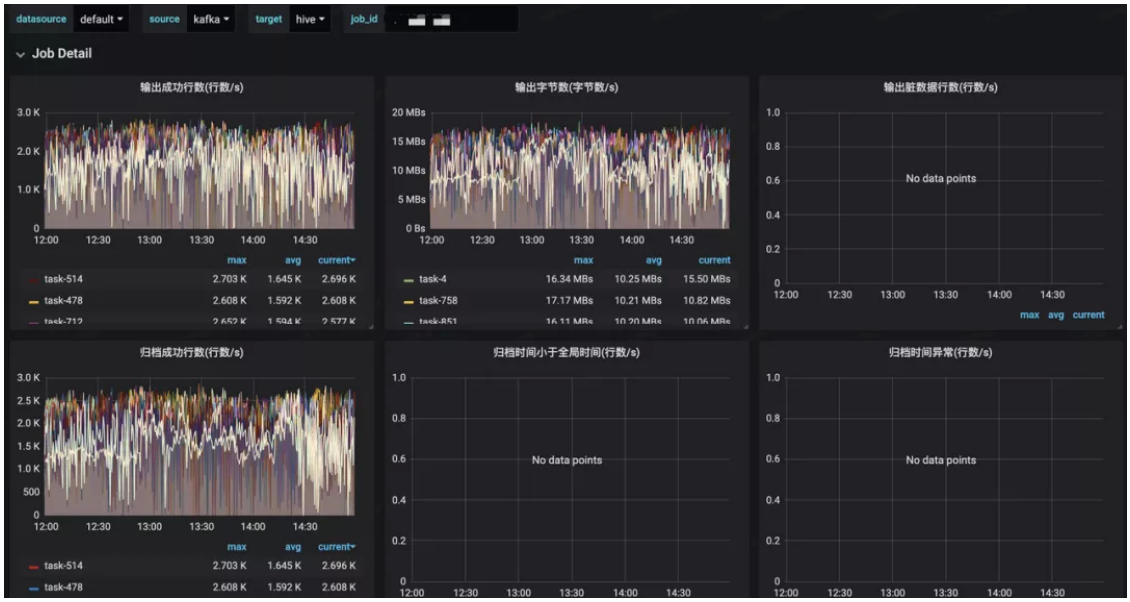


脏数据收集

数据集成场景下，不可避免会遇到脏数据，例如类型配置错误、字段溢出、类型转换不兼容等场景。对于流式任务来说，由于任务会一直运行，因此需要能够实时统计脏数据流量，并且将脏数据保存到外部存储中以供排查，同时在运行日志中采样输出。

大盘监控

大盘指标覆盖全局指标以及单个 Job 指标，包括写入成功流量和 QPS、写入 Latency、写入失败流量和 QPS、归档效果统计等，具体如下图所示：



未来规划

基于 Flink 实时解决方案目前已在公司上线和推广，未来主要关注以下几个方面：

1. 数据集成功能增强，支持更多数据源的接入，支持用户自定义数据转换逻辑等
2. Data Lake 打通，支持 CDC 数据实时导入
3. 流批架构统一，支持全量、增量场景数据集成
4. 架构升级，支持更多部署环境，比如 K8S
5. 服务化完善，降低用户接入成本

总结

随着字节跳动业务产品逐渐多元化快速发展，字节跳动内部一站式大数据开发平台功能也越来越丰富，并提供离线、实时、全量、增量场景下全域数据集成解决方案，从最初的几百个任务规模增长到数万级规模，日处理数据达到 PB 级，其中基于 Flink 实时解决方案目前已在公司内部大力推广和使用，并逐步替换老的 MQ-Hive 链路。

目前字节跳动还在持续高速发展中，我们仍旧不忘初心，砥砺前行。如果大家对本文或大数据开发场景感兴趣，包括但不限于开发平台、调度系统、数据集成、元数据和数据质量平台等，欢迎加入我们 数据平台 - 开发套件 团队，一起迎接字节跳动业务快速发展下的数据建设效率和数据治理等领域的挑战。欢迎大家投递简历，北京、上海均可，邮箱地址：**dataplatform-hr@bytedance.com** 。

参考文献

1. **Real-time Exactly-once ETL with Apache Flink**

   http://shzhangji.com/blog/2018/12/23/real-time-exactly-once-etl-with-apache-flink/

2. **Implementing the Two-Phase Commit Operator in Flink**

   https://flink.apache.org/features/2018/03/01/end-to-end-exactly-once-apache-flink.html

3. **A Deep Dive into Rescalable State in Apache Flink**

   https://flink.apache.org/features/2017/07/04/flink-rescalable-state.html

4. **Data Streaming Fault Tolerance**

   https://ci.apache.org/projects/flink/flink-docs-release-1.9/internals/stream_checkpointing.html

本文转载自公众号字节跳动技术团队（ID：toutiaotechblog）。

原文链接：

https://mp.weixin.qq.com/s/U04x9bzgWhjxQTXFgp-_GQ

2019 年 2 月 17 日

CAP 定理：三选二，架构师必须学会的取舍

作为大规模数据处理的架构师，我们应该熟知自己的系统到底应该保留CAP中的哪两项属性。

2019 年 5 月 6 日

MySQL 主从数据库同步是如何实现的?

主从同步做数据复制时，一般可以采用几种复制策略。性能最好的方法是异步复制。

2020 年 3 月 26 日

分布式计算模式之 MR：一门同流合污的艺术

今天，我以MapReduce为例，与你讲述了分布式领域中分治法的抽象模型、工作原理和实践应用。

2019 年 10 月 25 日

从 Chukwa 到 Keystone： Netflix 的数据流水线演进

2015 年 12 月，Netflix 新的数据流水线 Keystone 上线。本文将介绍近年来 Netflix 数据流水线演进的 3 个阶段。

2016 年 2 月 18 日