Apache Flink 在米哈游的落地实践

摘要：本文是来自米哈游大数据部对于Flink在米哈游应用及实践的分享。

本篇内容主要分为四个部分：

1.背景介绍

2.实时平台建设

3.实时数仓和数据湖探索

4.未来发展与展望

作者：实时计算负责人 张剑

背景介绍

米哈游成立于2011年，致力于为用户提供美好的、超出预期的产品与内容。公司陆续推出了多款高品质人气产品，包括《崩坏学园2》、《崩坏3》、《未定事件簿》、《原神》，动态桌面软件《人工桌面》以及社区产品《米游社》，并围绕原创IP打造了动画、漫画、音乐、小说及周边等多元产品。总部位于中国上海，并在新加坡、美国、加拿大、日本、韩国等国家和地区进行全球化布局。

Flink在米哈游大数据发展过程中，一直扮演着重要角色。自实时计算平台建立以来，Flink作为实时计算引擎，经历了多个发展阶段，实时计算平台也在不断的迭代完善。在米哈游内部，实时计算平台被称作Mlink，主要以Flink为主，兼容Spark Streaming任务。从起初的Flink Jar包任务为主，发展到以Flink Sql为主，不断的降低的使用门槛和提高了任务的开发效率；从起初的基础的Flink任务开发，发展到跨区域、跨云厂商的任务多版本管理，满足了业务发展的需求。在发展的过程中，米哈游不断的关注着社区的发展，并同社区和阿里云同学保持密切的联系。

Mlink主要是基于Yarn资源管理的计算平台，支持了数仓、算法、推荐、风控、大屏等业务。任务数1000+，Sql任务占比80%左右。使用的Yarn Vcores超5000核，内存10T左右。其中单个任务峰值吞吐在500万QPS。每天吞吐的数据规模超千亿。

实时平台建设

遇到的问题

在Flink探索发展的过程中，都会遇到Flink使用的一些痛点，大家遇到的，同样，我们在探索和实

践过程中也有所感触。总结起来，大概以下五个方面：一是Jar任务的开发成本高，对不熟悉Flink代码的同学使用成本过高，同时，Jar任务维护成本高，一些代码逻辑的改动会涉及到从新打包、上传，上线等动作。二是任务管理功能缺失，其中多租户、历史版本回溯、开发版本和线上版本管理、UDF管理、血缘管理是实时平台管理的重要内容。三是Flink引擎本身管理，主要涉及到多Flink版本管理，任务参数配置、常用Connector的二次开发、多资源环境管理等问题。四是任务的告警监控管理，任务问题诊断。五是同离线数仓互通，包括Hive Catalog管理，实时和离线调度依赖管理等。

上面的五个问题，可能是普遍的问题，所以各家公司都会基于内部自建或者开源项目二次开发，来满足自身任务开发管理需求。对于米哈游，除了上述五个问题，还存在跨区域，跨云厂商中遇到的问题需要解决，主要是跨区域之后，任务上线和提交效率，跨云厂商，资源环境不一致等。

解决方案

实时平台建设主要围绕如上问题。目前实时平台架构如下：

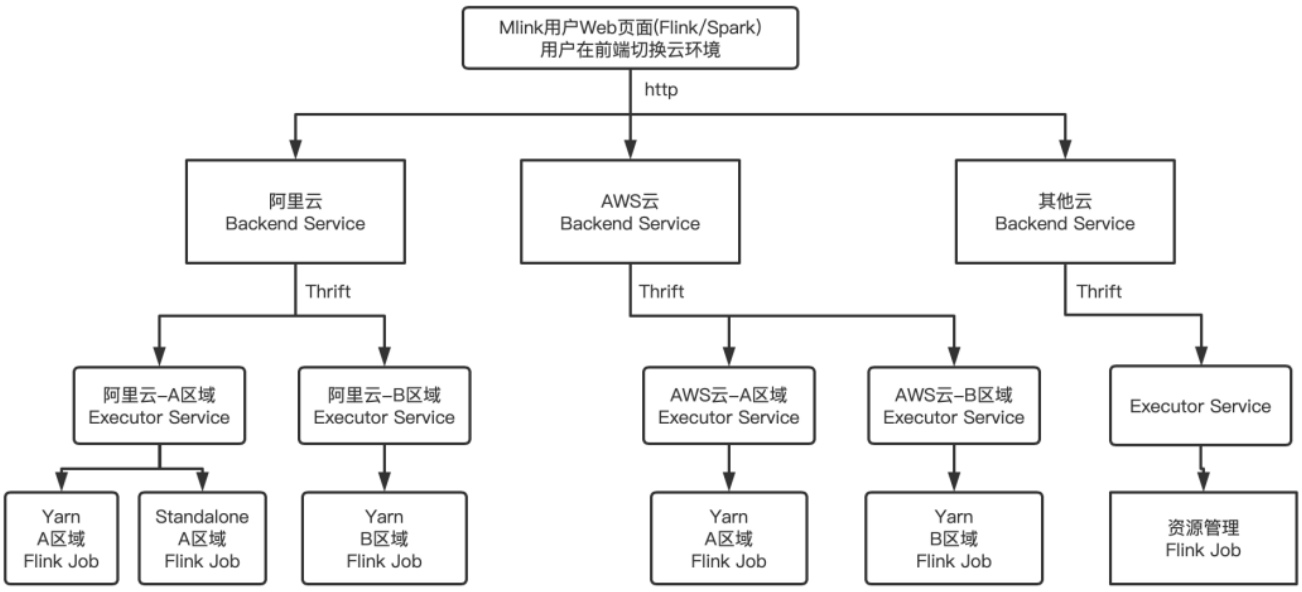

图1：多云多环境实时平台架构

前端控制云环境的切换。Backend Service主要负责用户权限管理、任务的多版本管理、血缘管理，任务运维，任务上下线，任务监控和告警等工作。Executor Service主要负责任务解析、任务提交运行、任务下线和同各类资源管理器交互等工作。其中，Backend Service到Executor Service通过Thrift协议通信，Executor Service的实现可以多语言扩展。架构设计主要解决跨地区跨云厂商问题，实现任务管理和任务运行之间解耦。

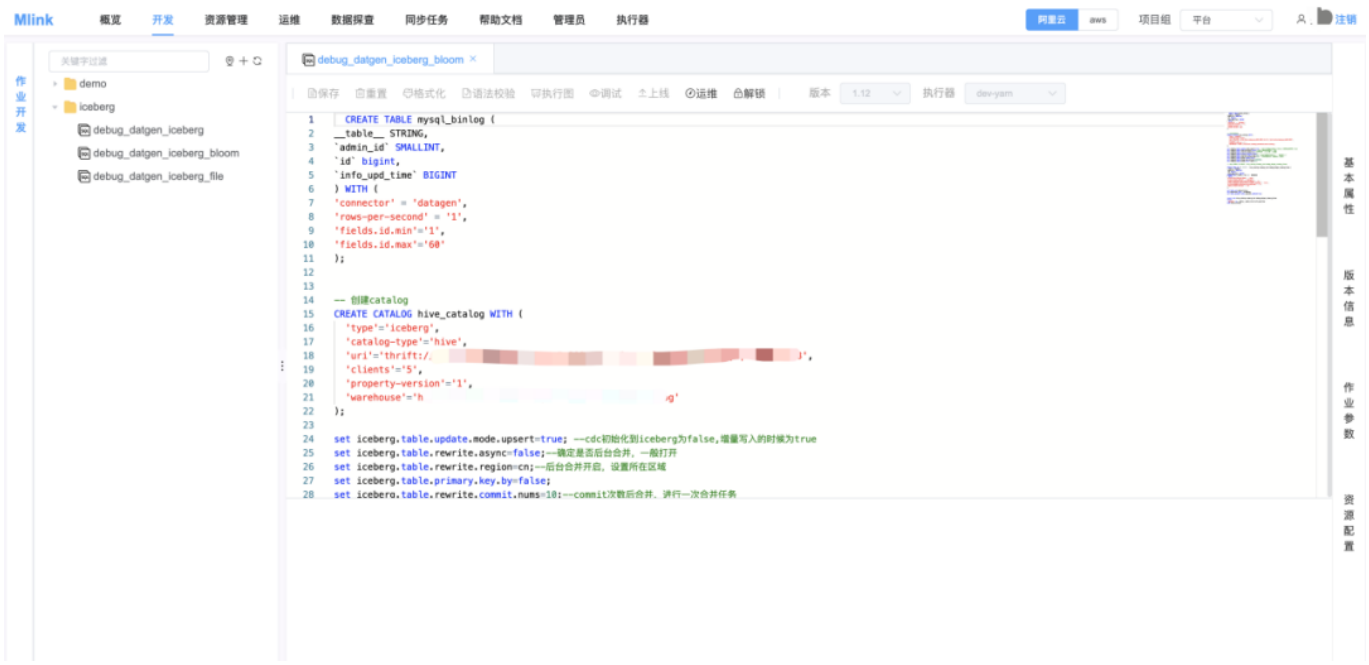

图2：Mlink平台开发页面

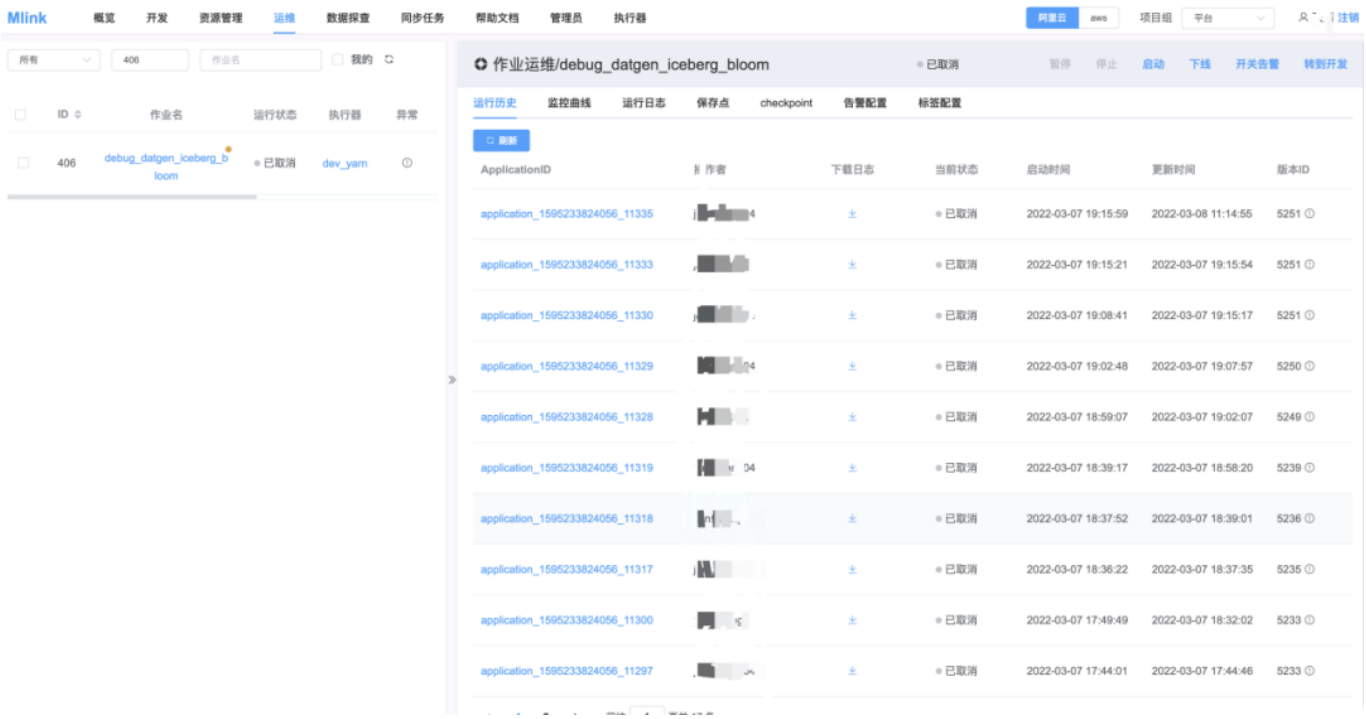

图3：Mlink平台运维页面

图4：Mlink平台同步任务页面

Mlink实时计算平台主要设计了概览、开发、资源管理、运维、数据探查、同步任务、用户管理和执行器管理等模块。其中开发页面主要是用户编写任务和参数配置，包含历史版本管理等内容。资源管理主要是Jar包任务和UDF管理。运维主要是任务启停、任务运行监控、任务告警配置等。数据探查部分主要是预览部分数据功能，比如Kafka Topic支持按分区、按时间或者Offset预览数据。同步任务主要是为了方便管理同步任务，比如CDC到Iceberg一键同步和运行管理。执行器负责Executor的运维工作，包括Executor上下线，健康状态监控等。

遇到的挑战

平台建设和迭代过程中，我们也遇到了不少的挑战，也产生了一些比较好的实践。主要分享四个方面。

Executor Service开发和维护方面

Executor主要涉及到Jar和Sql任务解析提交部分。一开始的方案为了解决跨地区传输效率问题，特别是大的jar包传输，由后端进行任务解析，最后传输job graph到Executor，Executor再通过资源管理器Api提交，这个因为后端解析环境不一致问题，部分任务解析过程中会存在action动作，特别是涉及到hive表和Iceberg表部分。最后采用后端不执行，改由Executor解析的方案。Executor在解析过程中，遇到了Executor在运行很长一段时间后，会出现元空间OOM的情况。这个主要是因为Executor不断的加载任务需要Class类，会导致使用的元空间内存不断增加。这个主要是通过任务解析完成之后，卸载类加载器和堆GC设置来解决。

监控方面

监控采用的是Influxdb加Grafana的方案。在随着任务量的不断增加，Influxdb存储的Series超过百万，影响监控查看的稳定性，查询响应缓慢。一是扩展Influxdb，执行端通过一致性hash的方案，分配任务Metric上报到不同Influxdb。本身通过对Flink任务上报Metric进行一定程度的精简。其次在监控上，比如Kafka消费监控，目前是支持消费条数的延迟监控，自定义了Kafka消费延迟时间的监控，主要是采集了Kafka最慢并行度消费的时间，能够反映Kafka消费的最大延迟时间，能够反映某个时间点的数据一定被消费了。

图5：Grafana监控示例

Connector二次开发方面

在CDC1.0版本基础上迭代，支持Mysql采集的时候动态扩展字段和基于时间启动消费位点、采集的库表、位点等Schema信息。在CDC2.0版本基础上，增加了全量读取库表流控和不需要MySQL开启Binlog的全量初始化功能。其中多CDC实例同步可能会对上游Mysql造成压力，采用了Kafka作为数据中转，根据库表主键字段作为Topic的Key，保证Binlog的顺序，在下游不会出现数据乱序。

Iceberg作为数据湖方案，改造的点主要是Iceberg V2表的支持上面，也就是Upsert表。建立Iceberg管理中心，会根据合并策略定期优化和清理，Flink写入主要保证在CDC到Iceberg V2表顺序性，在如何减少Delete File上，在Iceberg写入上增加了BloomFilter的支持，能够显著减少Delete File大小。Iceberg管理中心，支持了V2表合并和Flink提交冲突问题。

Clickhouse方面，重构了Clickhouse写入代码，优化了Clickhouse的写入性能，支持了本地表和分布式表写入。

数据入湖和离线调度方面

实时平台集成了Iceberg，并支持Iceberg Hadoop、Hive、Oss、S3多种Catalog。CDC到Iceberg入湖链路已经在部门生产业务上线使用。在数据入湖或者入仓中，如果下游表有被离线数仓用到的地方，都会有依赖调度问题，离线任务何时启动？目前我们主要通过计算任务的延迟时间和Checkpoint时间来确保数据已经入仓入湖。以CDC或者Kafka到Iceberg为例。首先采集CDC端采集延迟时间，Kafka采集最慢并行度延迟时间，同时采集任务Checkpoint时间。现在的Checkpoint完成，Iceberg版本不一定会更新，基于此，对Iceberg写入进行了改造。这样一个同步任务，如果CDC采集端没有延迟，Checkpoint也已经完成，可以保证某个小时的数据一定已经入仓。实时平台提供任务延迟查询接口。离线调度以此接口为调度依赖节点。这样就保证了离线任务启动时候，入仓数据的完整性。

实时数仓和数据湖探索

实时数据采集，目前主要是三条链路。一是日志类型，主要是通过Filebeat采集写入Kafka，Es作为Filebeat的监控。二是Api接口上报服务，后端接入Kafka。三是CDC采集全量加增量Mysql数据，写入Kafka或者直接写入Iceberg。之前是采用Canal作为增量采集方案，现在已经全部改为了CDC。

实时数仓架构设计和业内基本一致，包括ods、dwd、dws层，之后输出到各应用系统，比如Clickhouse、Doris、Mysql、Redis等。目前主要以Kafka作为中间承载，也在探索Iceberg作为中间层的使用。Iceberg虽然具有流读功能，但是流读时候数据的顺序性问题，一直没有较好的方案解决，我们也是在探索过程中。探索的主要方向有两个，一是将Kafka和Iceberg作为混合Source方案，Flink任务读取混合Source之后，基于Iceberg快照记录的Kafka位点，确定读取范围和切换点。二是社区Flip-188提出的引入Dynamic Table存储实现。Flink内置表由两部分组成，LogStore和FileStore。LogStore将满足消息系统的需要，而FileStore是列式格式文件系统。在每个时间点，LogStore和FileStore都会为最新写入的数据存储完全相同的数据（LogStore有TTL），但物理布局不同。

在实时数仓探索方面，主要是CDC到Iceberg入湖任务，已经在生产上使用。其中主要解决了四个问题：一是CDC采集问题，特别是多库多表采集，会集中采集到Kafka，减少多个CDC任务对同一数据库影响。二是Iceberg支持V2表写入，包括写入的索引过滤减少Delete文件，Iceberg管理中心合并和提交冲突。三是支持分库分表的数据校验和数据延迟检查。四是一键式任务生成。对于用户而言，只需要填写数据库相关信息，目标Iceberg表库名和表名，并支持使用Kafka中转，避免多个CDC实例采集同一个数据库实例。通过上述四个问题的解决，能够达到数据库数据分钟级数据入湖，入湖的数据校验和数据延迟依赖达成，方便下游离线任务调度启动。

图6：数据入湖链路

未来发展与展望

主要有四点：

一是Flink动态表存储能够尽快实现落地，实现真正的实时数仓和流表一体。

二是Flink任务动态扩缩容、基于任务诊断的主动资源调整、细粒度资源调整。

三是Flink对批任务的读写优化，目前批任务Flink的使用面不如Spark，如果未来能够在此补足，可以做到流批操作一个引擎，开发成本会显著降低。

四是Flink加数据湖更好的落地推广。