链滴

流批一体在京东的探索与实践

作者：fc13240

原文链接：https://ld246.com/article/1656928216789

来源网站：链滴

01 整体思考

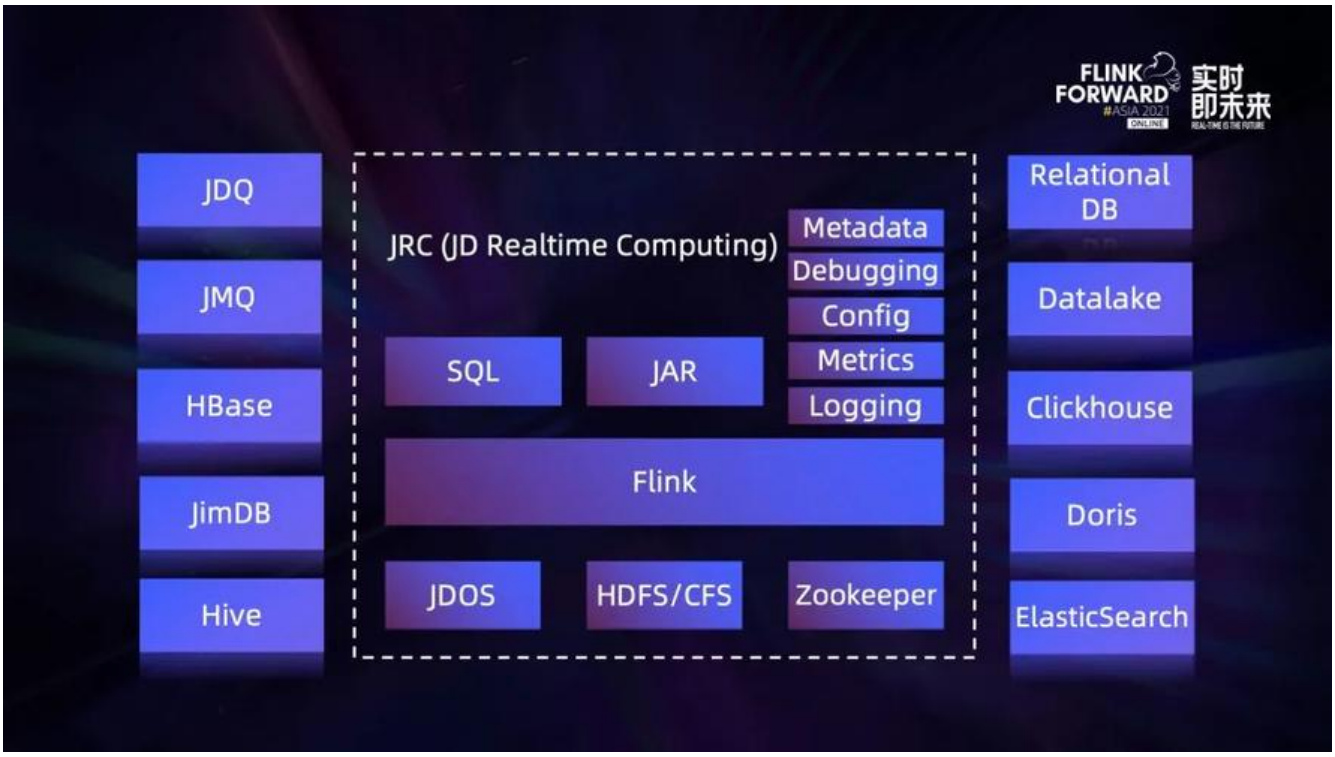
提到流批一体，不得不提传统的大数据平台 —— Lambda 架构。它能够有效地支撑离线和实时的数开发需求，但它流和批两条数据链路割裂所导致的高开发维护成本以及数据口径不一致是无法忽视的陷。

通过一套数据链路来同时满足流和批的数据处理需求是最理想的情况，即流批一体。此外我们认为流一体还存在一些中间阶段，比如只实现计算的统一或者只实现存储的统一也是有重大意义的。

以只实现计算统一为例，有一些数据应用的实时性要求比较高，比如希望端到端的数据处理延时不超一秒钟，这对目前开源的、适合作为流批统一的存储来说是一个很大的挑战。以数据湖为例，它的数可见性与 commit 的间隔相关，进而与 Flink 做 checkpoint 的时间间隔相关，此特性结合数据处理路的长度，可见做到端到端一秒钟的处理并不容易。因此对于这类需求，只实现计算统一也是可行的通过计算统一去降低用户的开发及维护成本，解决数据口径不一致的问题。

在流批一体技术落地的过程中，面临的挑战可以总结为以下 4 个方面：

- 首先是数据实时性。如何把端到端的数据时延降低到秒级别是一个很大的挑战，因为它同时涉及到算引擎及存储技术。它本质上属于性能问题，也是一个长期目标。
- 第二个挑战是如何兼容好在数据处理领域已经广泛应用的离线批处理能力。此处涉及开发和调度两层面的问题，开发层面主要是复用的问题，比如如何复用已经存在的离线表的数据模型，如何复用用已经在使用的自定义开发的 Hive UDF 等。调度层面的问题主要是如何合理地与调度系统进行集成。
- 第三个挑战是资源及部署问题。比如通过不同类型的流、批应用的混合部署来提高资源利用率，以如何基于 metrics 来构建弹性伸缩能力，进一步提高资源利用率。
- 最后一个挑战也是最困难的一个：用户观念。大多数用户对于比较新的技术理念通常仅限于技术交或者验证，即使验证之后觉得可以解决实际问题，也需要等待合适的业务来试水。这个问题也催生了些思考，平台侧一定要多站在用户的视角看待问题，合理地评估对用户的现有技术架构的改动成本以用户收益、业务迁移的潜在风险等。



上图是京东实时计算平台的全景图，也是我们实现流批一体能力的载体。中间的 Flink 基于开源社区本深度定制。基于该版本构建的集群，外部依赖包含三个部分，JDOS、HDFS/CFS 和 Zookeeper。

- JDOS 是京东的 Kubernetes 平台，目前我们所有 Flink 计算任务容器化的，都运行在这套平台之；
- Flink 的状态后端有 HDFS 和 CFS 两种选择，其中 CFS 是京东自研的对象存储；
- Flink 集群的高可用是基于 Zookeeper 构建的。

在应用开发方式方面，平台提供 SQL 和 Jar 包两种方式，其中 Jar 的方式支持用户直接上传 Flink 应 Jar 包或者提供 Git 地址由平台来负责打包。除此之外我们平台化的功能也相对比较完善，比如基础元数据服务、SQL 调试功能，产品端支持所有的参数配置，以及基于 metrics 的监控、任务日志查询。

连接数据源方面，平台通过 connector 支持了丰富的数据源类型，其中 JDQ 基于开源 Kafka 定制，要应用于大数据场景的消息队列；JMQ 是京东自研，主要应用于在线系统的消息队列；JimDB 是京自研的分布式 KV 存储。

在当前 Lambda 架构中，假设实时链路的数据存储在 JDQ，离线链路的数据存在 Hive 表中，即便算的是同一业务模型，元数据的定义也常常是存在差异的，因此我们引入统一的逻辑模型来兼容实时线两边的元数据。

在计算环节，通过 FlinkSQL 结合 UDF 的方式来实现业务逻辑的流批统一计算，此外平台会提供大量公用 UDF，同时也支持用户上传自定义 UDF。针对计算结果的输出，我们同样引入统一的逻辑模型屏蔽流批两端的差异。对于只实现计算统一的场景，可以将计算结果分别写入流批各自对应的存储，保证数据的实时性与先前保持一致。

对于同时实现计算统一和存储统一的场景，我们可以将计算的结果直接写入到流批统一的存储。我们择了 Iceberg 作为流批统一的存储，因为它拥有良好的架构设计，比如不会绑定到某一个特定的引擎。

原文链接：流批一体在京东的探索与实践

在兼容批处理能力方面，我们主要进行了以下三个方面的工作：

第一，复用离线数仓中的 Hive 表。

以数据源端为例，为了屏蔽上图左侧图中流、批两端元数据的差异，我们定义了逻辑模型 gdm_order m 表，并且需要用户显示地指定 Hive 表和 Topic 中的字段与这张逻辑表中字段的映射关系。这里映关系的定义非常重要，因为基于 FlinkSQL 的计算只需面向这张逻辑表，而无需关心实际的 Hive 表与opic 中的字段信息。在运行时通过 connector 创建流表和批表的时候，逻辑表中的字段会通过映射系被替换成实际的字段。

在产品端，我们可以给逻辑表分别绑定流表和批表，通过拖拽的方式来指定字段之间的映射关系。这模式使得我们的开发方式与之前有所差异，之前的方式是先新建一个任务并指定是流任务还是批任务然后进行 SQL 开发，再去指定任务相关的配置，最后发布任务。而在流批一体模式下，开发模式变了首先完成 SQL 的开发，其中包括逻辑的、物理的 DDL 的定义，以及它们之间的字段映射关系的指，DML 的编写等，然后分别指定流批任务相关的配置，最后发布成流批两个任务。

第二，与调度系统打通。

离线数仓的数据加工基本是以 Hive/Spark 结合调度的模式，以上图中居中的图为例，数据的加工被为 4 个阶段，分别对应数仓的 BDM、FDM、GDM 和 ADM 层。随着 Flink 能力的增强，用户希望把GDM 层的数据加工任务替换为 FlinkSQL 的批任务，这就需要把 FlinkSQL 批任务嵌入到当前的数据工过程中，作为中间的一个环节。

为了解决这个问题，除了任务本身支持配置调度规则，我们还打通了调度系统，从中继承了父任务的赖关系，并将任务自身的信息同步到调度系统中，支持作为下游任务的父任务，从而实现了将 FlinkS L 的批任务作为原数据加工的其中一个环节。

第三，对用户自定义的 Hive UDF、UDAF 及 UDTF 的复用。

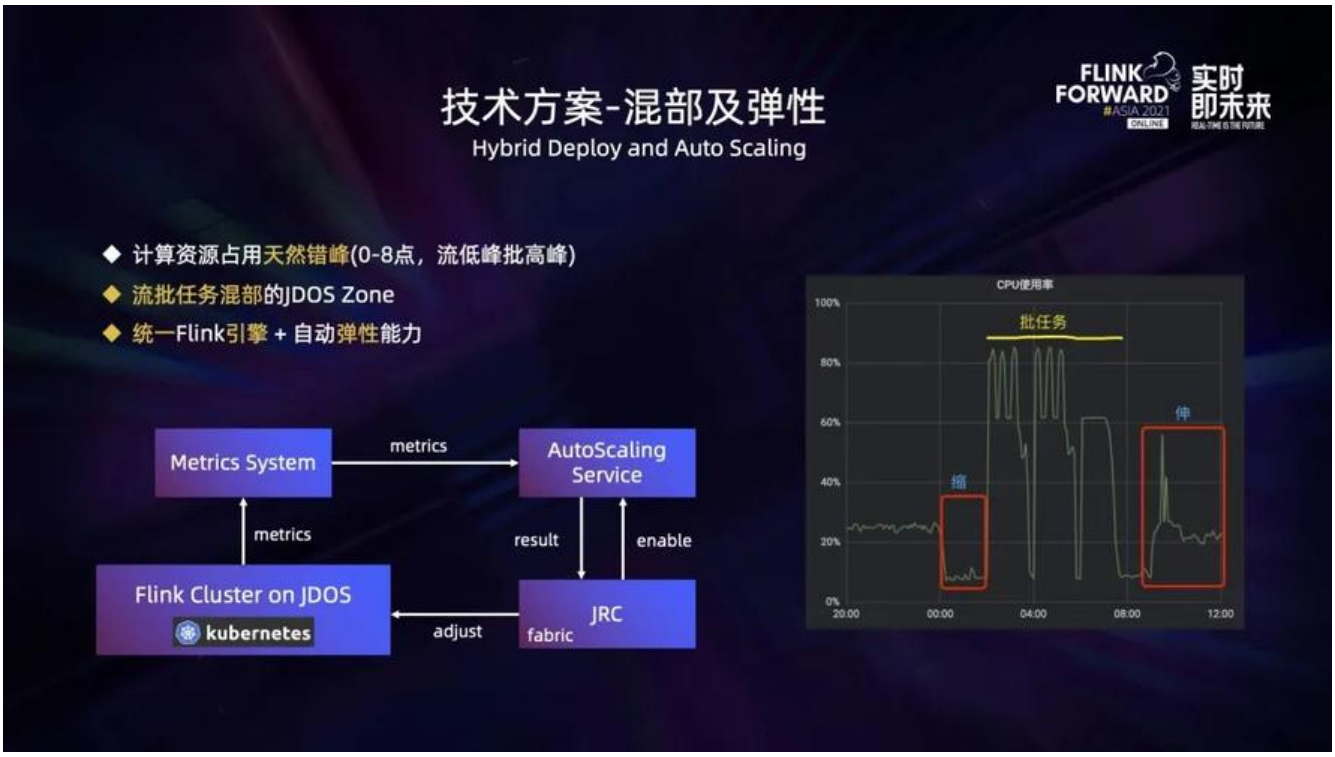
对于现存的基于 Hive 的离线加工任务，如果用户已经开发了 UDF 函数，那么最理想的方式是在迁移link 时对这些 UDF 进行直接复用，而不是按照 Flink UDF 定义重新实现。

在 UDF 的兼容问题上，针对使用 Hive 内置函数的场景，社区提供了 load hive modules 方案。如用户希望使用自己开发的 Hive UDF，可以通过使用 create catalog、use catalog、create functio，最后在 DML 中调用的方式来实现, 这个过程会将 Function 的信息注册到 Hive 的 Metastore 中。平台管理的角度，我们希望用户的 UDF 具备一定的隔离性，限制用户 Job 的粒度，减少与 Hive Met store 交互以及产生脏函数元数据的风险。

此外，当元信息已经被注册过，希望下次能在 Flink 平台端正常使用，如果不使用 if not exist 语法通常需要先 drop function，再进行 create 操作。但是这种方式不够优雅，同时也对用户的使用方有限制。另一种解决方法是用户可以注册临时的 Hive UDF，在 Flink1.12 中注册临时 UDF 的方式是reate temporary function，但是该 Function 需要实现 UserDefinedFunction 接口后才能通过后的校验，否则会注册失败。

所以我们并没有使用 create temporary function，而是对 create function 做了一些调整，扩展了 Ex FunctionModule，将解析出来的 FunctionDefinition 注册到 ExtFunctionModule 中，做了一次 Jo 级别的临时注册。这样的好处就是不会污染 Hive Metastore，提供了良好的隔离性，同时也没有对户的使用习惯产生限制，提供了良好的体验。

不过这个问题在社区 1.13 的版本已经得到了综合的解决。通过引入 Hive 解析器等扩展，已经可以把现 UDF、GenericUDF 接口的自定义 Hive 函数通过 create temporary function 语法进行注册和使。



资源占用方面，流处理和批处理是天然错峰的。对于批处理，离线数仓每天 0 点开始计算过去一整天数据，所有的离线报表的数据加工会在第二天上班前全部完成，所以通常 00:00 到 8:00 是批计算任大量占用资源的时间段，而这个时间段通常在线的流量都比较低。流处理的负载与在线的流量是正相的，所以这个时间段流处理的资源需求是比较低的。上午 8 点到晚上 0 点，在线的流量比较高，而这时间段批处理的任务大部分都不会被触发执行。

基于这种天然的错峰，我们可以通过在专属的 JDOS Zone 中进行不同类型的流批应用的混部来提升源的使用率，并且如果统一使用 Flink 引擎来处理流批应用，资源的使用率会更高。

同时为了使应用可以基于流量进行动态调整，我们还开发了自动弹性伸缩的服务 (Auto-Scaling Servie)。它的工作原理如下：运行在平台上的 Flink 任务上报 metrics 信息到 metrics 系统，Auto-Scaling Service 会基于 metrics 系统中的一些关键指标，比如 TaskManager 的 CPU 使用率、任务的背压况等来判定任务是否需要增减计算资源，并把调整的结果反馈给 JRC 平台，JRC 平台通过内嵌的 fabri 客户端将调整的结果同步到 JDOS 平台，从而完成对 TaskManager Pod 个数的调整。此外，用户以在 JRC 平台上通过配置来决定是否为任务开启此功能。

上图右侧图表是我们在 JDOS Zone 中进行流批混部并结合弹性伸缩服务试点测试时的 CPU 使用情。可以看到 0 点流任务进行了缩容，将资源释放给批任务。我们设置的新任务在 2 点开始执行，所以 2 点开始直到早上批任务结束这段时间，CPU 的使用率都比较高，最高到 80% 以上。批任务运行结后，在线流量开始增长时，流任务进行了扩容，CPU 的使用率也随之上升。

更多内容请看：

https://blog.stackanswer.com/articles/2022/07/01/1656663988707.html