SQL 开发任务超 50%！滴滴实时计算的演进与优化

作者介绍：梁李印，2010 年至 2014 年，阿里云 Hadoop 集群(云梯 I)负责人之一；2014 年至 2016 年，阿里云分布式图计算框架 (MaxCompute Graph) 研发；2017 年至 2019 年，滴滴出行负责实时计算及 OLAP 建设，目前负责滴滴大数据架构部。

摘要：Apache Flink 是一个分布式大数据处理引擎，可对有限数据流和无限数据流进行有状态计算。可部署在各种集群环境，对各种大小的数据规模进行快速计算。滴滴基于 Apache Flink 做了大量的优化，也增加了更多的功能，比如扩展 DDL、内置消息格式解析、扩展 UDX 等，使得 Flink 能够在滴滴的业务场景中发挥更大的作用。本文中，滴滴出行实时计算负责人、高级技术专家梁李印分享了 Apache Flink 在滴滴的应用与实践。主要内容为：
1.服务化概述
2.StreamSQL 实践
3.平台化建设
4.挑战及规则

一、服务化概述

滴滴大数据服务架构

滴滴目前基于开源的大数据生态构建了一个比较完整的大数据体系，这套体系包括了离线、实时、OLAP、HBase 生态、检索以及消息队列等。滴滴大数据体系在 Flink 的基础之上着力发展 StreamSQL，后面也会对此进行详细介绍。

滴滴流计算发展历程

如下图所示的是滴滴流计算的发展历程。在 2017 年之前，滴滴的流计算基本上采用的都是业务方自建小集群的方式，并且小集群的选型也是多种多样的，包括了 Storm、Jstorm、Spark Streaming 以及 Samza 等。滴滴的流计算技术从 2017 年开始收敛，开始主要基于 Spark Streaming 来构建服务化和平台化的大集群。

- 从 2017 年开始，滴滴开始引入了 Flink，这是因为一些对于延迟要求特别高的业务，Spark Streaming 是无法支持的。
- 2018 年，滴滴流计算开始重点支持 StreamSQL 即 Flink SQL 的 SQL 化服务，另外一方面滴滴也在 Flink CEP 投入了一定精力来解决实际应用中的一些问题。
- 到 2019 年为止，滴滴基本上完成了流计算引擎的统一，除了少量残留的历史业务之外，现在绝大多数业务都是以 Flink 为基础的，目前在滴滴通过 SQL 开发的任务也已经超过了 50%，SQL 开发成为了主流方式。

滴滴流计算业务规模

因为滴滴采用集中式业务管理，因此流计算所支持和服务的业务线达到了 50 多条。流计算集群的规模大致在千台级别，目前流计算任务数达到了 3 千多个，其中绝大多数是使用 SQL 开发的，集群每天处理的数据量会达到上万亿条。

滴滴流计算业务场景

滴滴流计算的业务场景主要可以划分为四个方面：实时监控、实时同步、实时特征以及实时业务。

- 实时监控，包括对于业务指标、导航及 POI 准确率、业务健康度以及车联网等进行实时监控。
- 实时同步，指的是数据实时地从一个地方转移到另外一个地方去，这部分包括了业务日志、数据库日志、埋点以及轨迹数据等，其中轨迹属于滴滴比较有特点的数据，这部分数据会放入到 HBase 中去。
- 实时特征，在滴滴中也属于一个比较关键的业务，因为其会直接影响派单、导航等的准确性，实时特征包括司机乘客特征、上下车特征、导航轨迹特征以及工单特征等。
- 实时业务，主要指的是会实时影响用户的流计算场景，包括司乘位置语义同步，也就是当司机接单之后能够知道乘客位置的动态变化，同时乘客也能够知道司机的位置变化，以及高危的行程检测等，比如在某个红绿灯处等待时间较长，就可能出现了异常情况需要客服介入处理，除此之外还包括个性化和服务检测等。

滴滴流计算多集群体系

随着业务的不断发展，滴滴的机房也越来越多。基于这样的情况，滴滴希望为业务提供一个多机房的统一视图。因此，滴滴选择在 YARN 的基础之上构建一个路由层，这个路由层的职责是屏蔽多个物理集群，为业务方提供一个单一的逻辑集群，通过 YARN 的划分来确定哪个业务运行在哪个机房中。在物理集群内部，又实现了隔离，因为一些业务比较重要，不希望受到其他业务的影响，因此可能需要专门划分出一批机器来管理这些业务。

此外，滴滴对于 YARN 的调度也做了一些定制，因为在滴滴内部，YARN 的实时和离线是完全分开的，而两者的差异也是比较大的，离线作业需要将机器资源全部用起来，吞吐越大越好，而实时作业却不同，所追求的是均衡，因此机器的 Load 对它的影响会比较高。

因此，滴滴一方面将 YARN 的调度改为基于 CPU 进行调度，另外一方面还做了繁忙节点的智能过滤，并且还实现了动态资源推荐，这里会为用户提供参数的推荐配置，而最终是否调整则取决于用户。

二、StreamSQL 实践

StreamSQL 的优势

StreamSQL 的实践也是滴滴最近一年以来比较重要的工作。StreamSQL 是在 Flink SQL 基础之上对于功能进行增加和完善所形成的产品。使用 StreamSQL 的优势主要包括以下 5 点：

1. 描述性语言，业务方不需要关注底层实现，只要将业务逻辑描述出来就能够达到所想要的效果。
2. 接口稳定，虽然版本一直在升级迭代，但只要 SQL 语法不变，对于用户而言就是完全透明的。

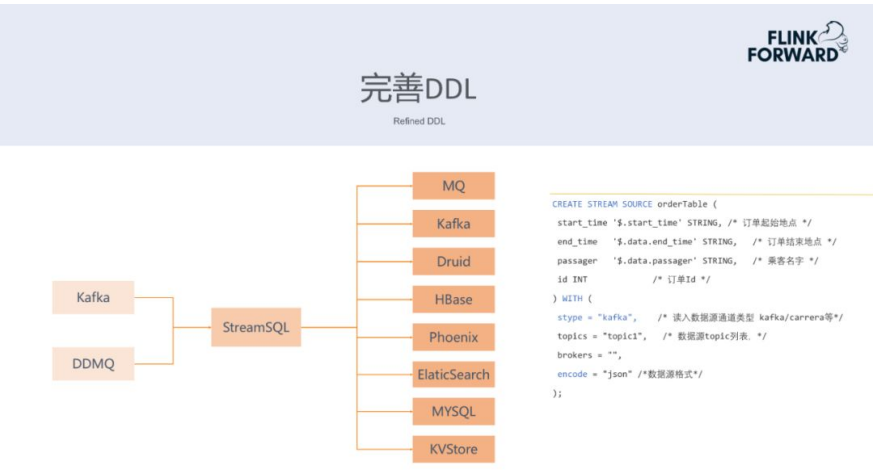
3. 问题易排查，因为代码可读性比较强，因此用户只要看懂一些语法逻辑就能够快速排查出问题。

4. 批流一体化，今天在滴滴批处理大部分使用的是 Hive SQL 和 Spark SQL，如果流处理也是用 SQL，那么在 SQL 与 SQL 之间能够实现相互结合，比如实现共享 EDF、共享 Meta Store，甚至共享一些语法等，最终实现批流一体化的效果。

5. 入门门槛低，StreamSQL 的学习入门的门槛比较低，因此受到了广大开发者的欢迎。

完善 DDL

滴滴在 StreamSQL 上也做了很多事情，其中之一就是完善 DDL。社区版本的 StreamSQL 在 DDL 这部分是不完善的，而在滴滴却将这部分实现了打通，比如打通了上游和下游的消息队列、实时存储和离线存储等，业务方或者用户方只需要创建一个 Source 或 Sink 就可以将上游和下游描述出来。



内置消息格式解析

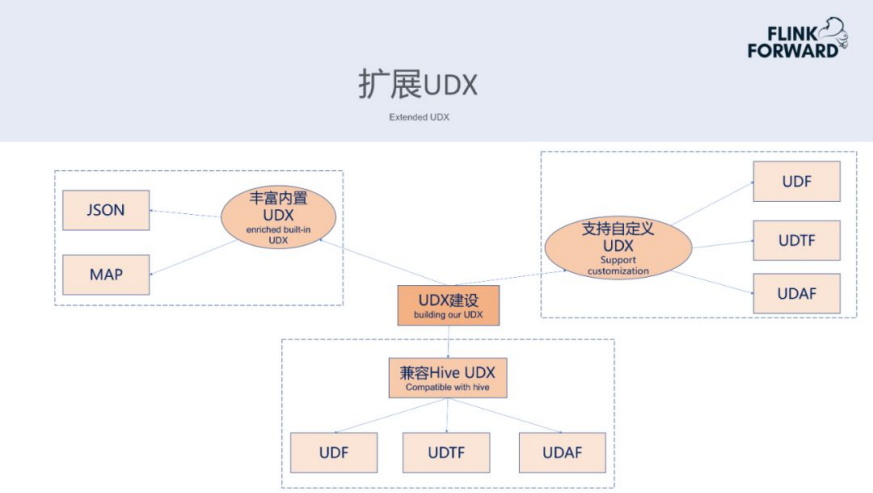
消息中数据的提取一般而言都比较复杂，比如 Binlog 日志的数据格式是非常复杂的，如果每个用户都实现一次数据提取是非常困难的，因此滴滴在 StreamSQL 中内置了消息格式解析，用户只需要创建一个 Source，并指定类型为 Binlog 就可以将相应的字段创建完成，包括数据库名称、表名称以及业务属性等，因此使用起来非常方便。

此外，滴滴在 StreamSQL 中还内置了数据去重功能。上游在数据采集等环节往往容易造成数据重复，数据重复往往会对于下游如数据监控等造成影响，因此 StreamSQL 中内置了数据去重功能，业务方只需要进行简单配置，就能够将重复的数据清洗掉。

除了数据库日志 Binlog 之外，滴滴还有很多自己定义的业务日志，包含了很多内部的业务字段，对于这些，滴滴在 StreamSQL 中也实现了自动提取日志头以及提取所有业务字段，组装成 Map 的功能。此外，滴滴的 StreamSQL 还支持通过 JSONPath 指定所需字段，而不需要通过 UDF 再获取字段，因此大大降低了使用复杂度。

扩展 UDX

一方面，滴滴对于 StreamSQL 的原生设计进行了一些扩展，主要是在 JSON 和 Map 的操作上，因为这两种类型在滴滴的应用非常广泛，因此滴滴的 StreamSQL 对于这两者的应用方法做了一些扩充。另外一方面，滴滴的 StreamSQL 还支持了自定义 UDX，用户可以自己实现一个 Function，通过类似于 Hive Function 这样的方式指定一个 Jar 包来应用到业务里面。第三方面，滴滴的 StreamSQL 还兼容了 Hive UDX，如果用户原来使用的 Hive SQL，而又想要将任务实时化，基本上逻辑不需要发生太大变化。与此同时，引用的 UDX 也不需要做什么改动，只需要加以引用就可以使用了，这样一来就有助于实现批流一体化。



Join 能力

在 Join 能力方面，滴滴的 StreamSQL 也花了很多精力来进行支持。第一方面所支持的就是 Join 比较长的跨度能力，比如滴滴的顺风车业务可能在发单到接单的跨度能够达到一个星期的时间，在这段时间之内如果 Join 都基于内存实现是不太可能的，因此滴滴在这种场景下 Join 都是放到状态里面去，通过 TTL 窗口实现。比如顺风车的 TTL 需要配成 7 天，7 天之后自动过期，这样就可以实现窗口的能力。

在维表 Join 方面，滴滴的 StreamSQL 则支持了 HBase、滴滴内部研发的 KV Store 以及 MySQL 等的 Join，这部分解决的问题包括字段补全，以及将司机 ID、司机姓名、明细等 Join 在一起方便下一个任务使用。

三、平台化建设

StreamSQL IDE

在平台化建设方面，滴滴目前已经构建了一站式 StreamSQL 开发平台。

首先，滴滴提供了 StreamSQL 的 IDE。因为目前大部分业务是基于 SQL 开发的，因此 IDE 需要承担非常重要的职责，在 StreamSQL IDE 里面除了提供了本身的 SQL 之外，还提供了 SQL 模板，如果用户想要开发流式 SQL 时不需要从零开始，只需要选择一个 SQL 模板，并在这个模板之上进行修修改改即可达到期望的结果。

此外，StreamSQL IDE 还提供了 UDF 的库，相当于一个库如果不知道具有什么含义以及如何使用，用户只需要在 IDE 上搜索到这个库，就能够找到使用说明以及使用案例。

IDE 所提供的第三个附加功能就是语法检测和智能提示，用户在写 SQL 的过程中能够为用户做出实时的语法检测并给出一定的反馈，与此同时，也能够对一些表和字段等提供一些智能提示的能力，使用起来也非常方便。

StreamSQL IDE 的第四种能力是 Debug，这种能力对于流计算而言是非常重要的，如果没有这种能力可能需要先开发、再提交、运行一段时间之后再通过结果来看是否存在问题，因此 StreamSQL IDE 提供了在线的 Debug 能力，用户可以直接上传一个文件，并构造一个大约 100 行的数据，运行之后立即能够在控制台上打印出 SQL 输出结果，并校验是否正确。

除了上传数据之外，StreamSQL IDE 还支持采样 Topic 中数据进行验证。

StreamSQL IDE 的最后一个功能就是版本管理，因为业务版本需要不断升级，而升级时也可能需要回退，因此 StreamSQL IDE 也提供了版本管理功能。

任务管控

如今，滴滴的所有流计算任务都是通过 Web 化入口提交的，从一开始的时候就没有暴露出客户端，这也保证任务的发展是全部可控的。除了 Web 化入口之外，滴滴也提供了全流程的任务生命周期管理，包括任务的提交、任务的停止以及任务的升级、回滚等。因为客户端是流计算团队自己控制的，因此参数可以自己设置，如果需要调整也可以直接在 Web 化的入口进行统一的修改。

任务运维

滴滴在任务运维方面所做的优化大致包括四个方面，首先是日志检索，因为 Flink 的日志是打在本地的，而通过 Flink UI 来看日志的体验是非常糟糕的，所以滴滴对于 Flink 的日志采集到了 Elasticsearch 集群上去，通过 Web 化界面来检索日志，方便大家调查问题。

第二个是指标监控，因为 Flink 的指标非常多，因此通过 Flink UI 来看指标的体验也非常差，因此构建了一个 Web 的报表平台，通过将指标采集到 Druid 中去，通过查询 Druid 来查询报表。

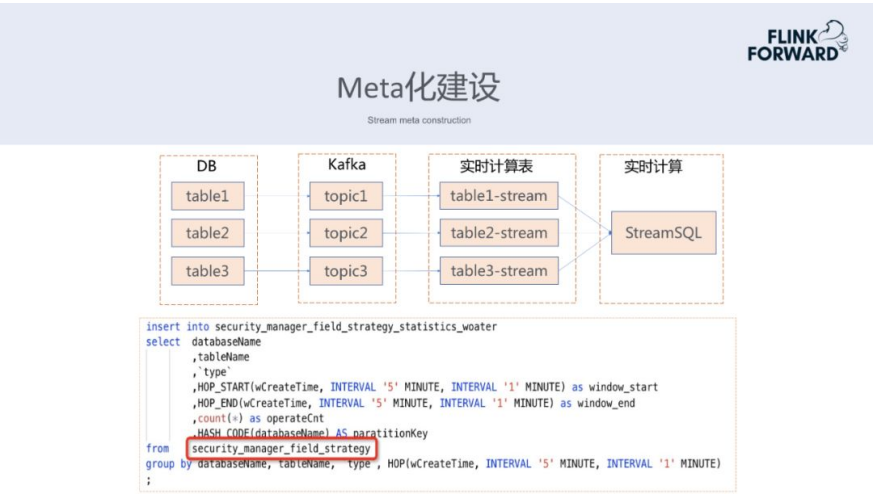
第三点就是报警，这部分比较有意思，因为基本上所有的系统都会提供报警能力，但是报警里面就会涉及到发报警和收报警的两部分人员，对于发报警的人而言，都是希望报警信息尽量多发，这样可以免责；而对于收报警的人而言，则希望尽量少收到报警，因为看报警信息也需要花费时间和精力，因此在两方面需要做到很好的平衡，既能够将该发的报警信息发出去，又不能乱发报警信息。

最后一点，滴滴在实时任务的运维方面还提供了血缘追踪的能力，因为实时任务和离线任务不同，其任务链路非常长，从采集到消息通道再到流计算以及下游的存储之间存在四到五个环节，这里面如果无法追踪，那么在查找问题时将会是灾难性的，因此需要进行血缘追踪。

Meta 化建设

滴滴目前在做的一件事情就是 Meta 化建设，流计算任务的开发需要先定义 DDL，再写 DML，这样的代价是比较大的。而无论是通过 Hive SQL 还是 Spark SQL， DDL 都已经在 Meta Store 里面了，其实是不需要重复写的。为了解决上述的问题，滴滴也正在做实时 Meta 的相关工作，希望将实时的数据比如 Kafka 的流定义成一个实时表存储到 Meta Store 里面去，用户只需要写 DML 语句即可。

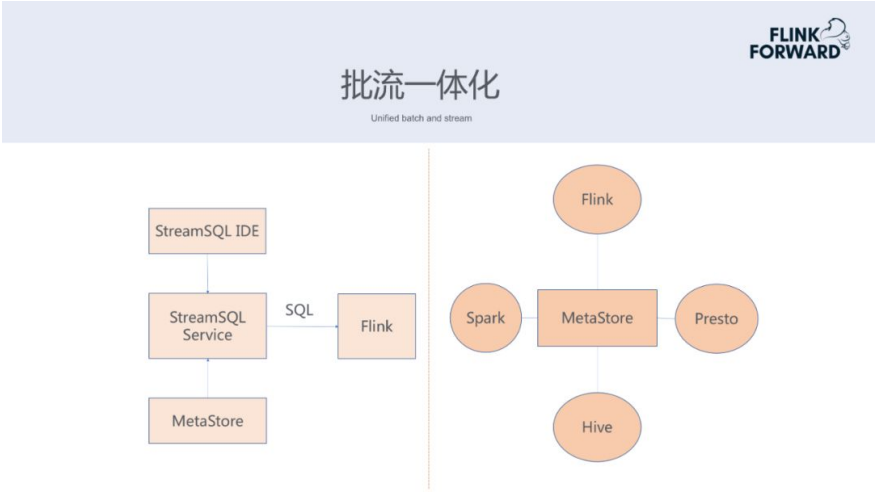
在任务提交的时候，系统会自动地将 DDL 语句补充上去，变成完成的 StreamSQL 进行提交。这是滴滴目前正在实现中的一个功能，这样的功能实现之后可以进一步降低 Meta 数据量，同时也能够进一步打通批处理业务和流处理业务，使得两者的开发步骤更加趋近相同。



批流一体化

虽然 Flink 具备批流的核心能力，但是在滴滴内部还没有完全实现。滴滴首先希望能够在产品上实现批流一体化，上面提到通过 Meta 化建设之后能够实现整个滴滴只有一个 Meta Store，无论是 Hive 的表、 Kafka 的 Topic 还是下游的 HBase 或者 ES，都能够定义到

Meta Store 里面去， Hive 能够查询 Meta Store， Spark、 Presto 以及 Flink 也都能够查询 Meta Store，这样使得整个 SQL 开发起来是完全一致的。进一步来说，可以根据 SQL 消费的 Source 来判断到底是批计算任务还是流计算任务，如此实现产品的批流一体化体验。



四、挑战与规划

面临的挑战

滴滴实时计算所面临的挑战主要有三点：

- 大状态管理：Flink 中一个比较重要的特点就是具有状态，而状态有时候会非常大，而且时间比较长，因此数据对齐需要非常大的开销。另外一点，在做 Checkpoint 的时候，磁盘 IO 会变大，进而引起机器负载升高，从而影响任务运行，因此 Checkpoint 能不能用好还是会影响服务稳定性的。因为 Checkpoint 是一个黑盒，那么想要看到 Checkpoint 内部的东西去做一些修改和验证，就需要状态诊断的能力。第三点还需要提供全链路的 Exactly Once 的能力。

- 业务高可用：在滴滴内部，越来越多的业务将 Java 或者 Golang 写的业务搬到 Flink 上去，让 Flink 来解决容错和扩展问题以及编程模型问题，这样就需要原来本地开发所有的诊断手段在 Flink 上都能够具备，比如在业务不能停止的情况下做透明升级，以及一旦出现问题之后如何快速解决等，因此快速诊断也是非常重要的能力。在之后还需要资源伸缩的能力，因为业务存在早高峰和晚高峰还有平峰，而业务量还在增长，还需要保证活动或者节假日的流量速增情况下的任务还处于稳定状态，这些都是需要解决的问题。

- 多语言：虽然今天在滴滴大部分实时任务都是通过 SQL 来开发的，但是依旧不能100%覆盖全部的场景，有些场景下是需要写代码的。Flink 提供了 Java 和 Scala 这两种 API，但这对于业务人员而言依然是不够的，因为业务大部分是 Go 语言系或者 Python 语言系的，因此滴滴希望根据社区来提供多语言的开发 Flink 的能力，比如写 SQL，而 UDF 也可以通过多语言来开发。

未来规划

最后为大家介绍滴滴在 Flink 应用和实践方面的未来规划。

- 首先，需要提供高可用的流计算服务，使得 Flink 能够具备支撑完整线上能力的机制。
- 其次，滴滴也在探索实时机器学习，原来任务是以天级别进行更新的，而在现在的模型中，借助 Flink 的能力已经能够做到10分钟到15分钟的更新，在下一步将会进一步实现秒级别的更新。
- 最后一点就是实时数仓，能否实现数仓报表的实时化，能否使得指标口径能够和离线保持一致，是否能够使得实时业务和离线业务相互补数据，都时滴滴实时计算未来的挑战。