美团技术团队

美团DB数据同步到数据仓库的架构与实践

萌萌 心序 成聪 ·2018-12-06 20:06

背景

在数据仓库建模中，未经任何加工处理的原始业务层数据，我们称之为ODS(Operational Data Store)数据。在互联网企业中，常见的ODS数据有业务日志数据（Log）和业务DB数据（DB）两类。对于业务DB数据来说，从MySQL等关系型数据库的业务数据进行采集，然后导入到Hive中，是进行数据仓库生产的重要环节。

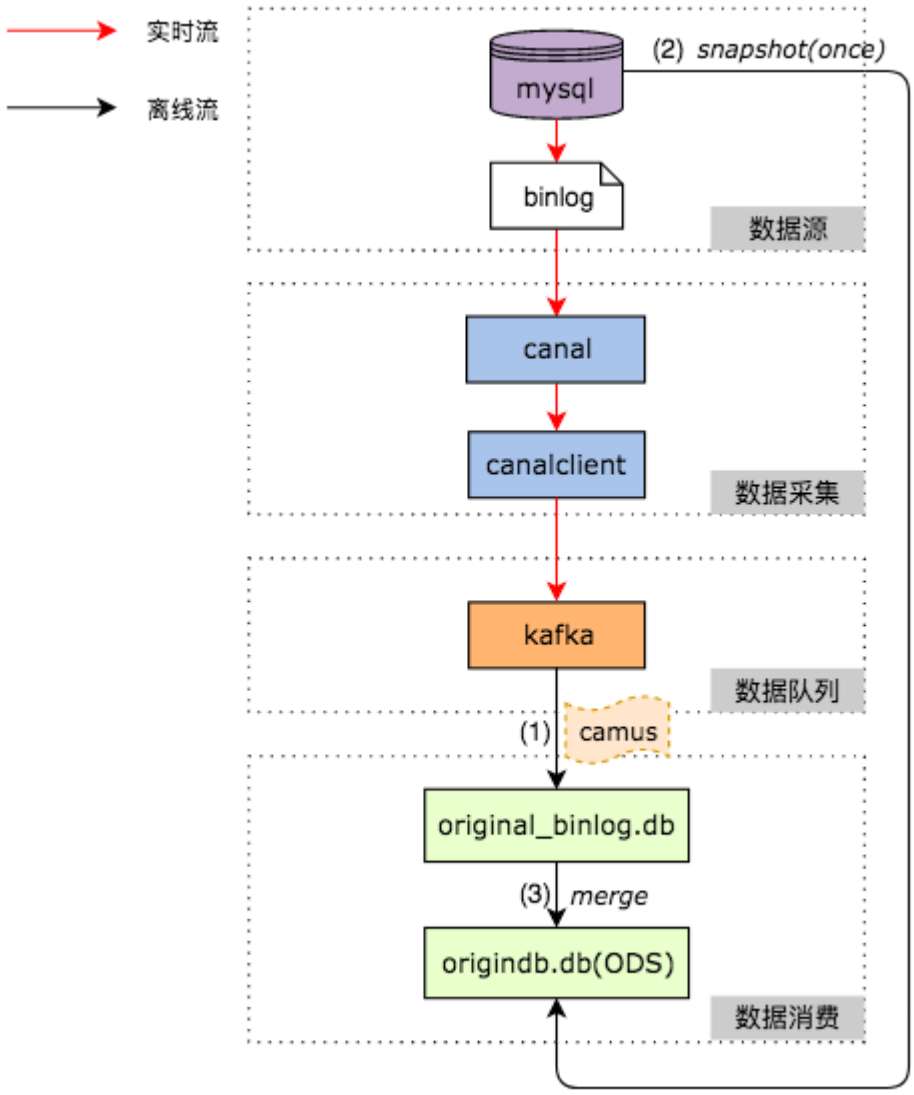
如何准确、高效地把MySQL数据同步到Hive中？一般常用的解决方案是批量取数并Load：直连MySQL去Select表中的数据，然后存到本地文件作为中间存储，最后把文件Load到Hive表中。这种方案的优点是实现简单，但是随着业务的发展，缺点也逐渐暴露出来：

- 性能瓶颈：随着业务规模的增长，Select From MySQL -> Save to Localfile -> Load to Hive这种数据流花费的时间越来越长，无法满足下游数仓生产的时间要求。
- 直接从MySQL中Select大量数据，对MySQL的影响非常大，容易造成慢查询，影响业务线上的正常服务。
- 由于Hive本身的语法不支持更新、删除等SQL原语，对于MySQL中发生Update/Delete的数据无法很好地进行支持。

为了彻底解决这些问题，我们逐步转向CDC (Change Data Capture) + Merge的技术方案，即实时Binlog采集 + 离线处理Binlog还原业务数据这样一套解决方案。Binlog是MySQL的二进制日志，记录了MySQL中发生的所有数据变更，MySQL集群自身的主从同步就是基于Binlog做的。

本文主要从Binlog实时采集和离线处理Binlog还原业务数据两个方面，来介绍如何实现DB数据准确、高效地进入数仓。

整体架构



整体的架构如上图所示。在Binlog实时采集方面，我们采用了阿里巴巴的开源项目Canal，负责从MySQL实时拉取Binlog并完成适当解析。Binlog采集后会暂存到Kafka上供下游消费。整体实时采集部分如图中红色箭头所示。

离线处理Binlog的部分，如图中黑色箭头所示，通过下面的步骤在Hive上还原一张MySQL表：

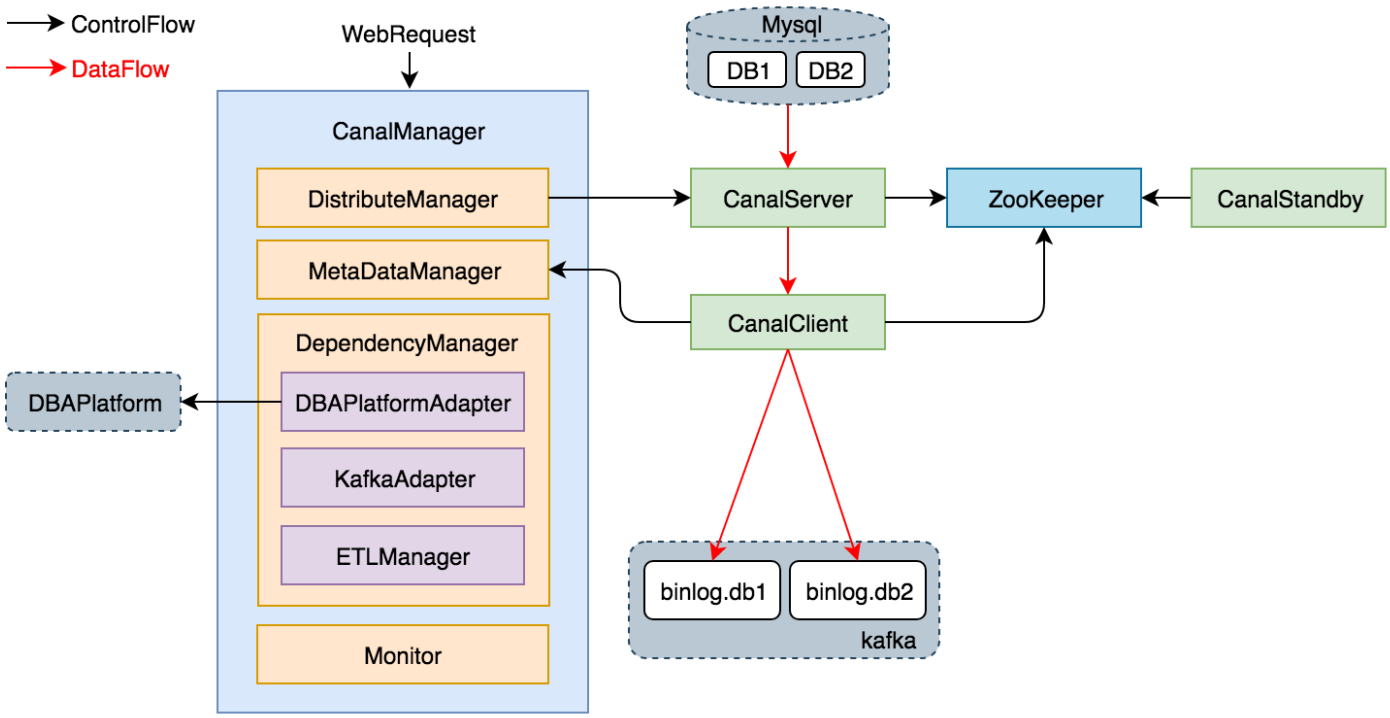
1. 采用Linkedin的开源项目Camus，负责每小时把Kafka上的Binlog数据拉取到Hive上。
2. 对每张ODS表，首先需要一次性制作快照（Snapshot），把MySQL里的存量数据读取到Hive上，这一过程底层采用直连MySQL去Select数据的方式。
3. 对每张ODS表，每天基于存量数据和当天增量产生的Binlog做Merge，从而还原出业务数据。

我们回过头来看看，背景中介绍的批量取数并Load方案遇到的各种问题，为什么用这种方案能解决上面的问题呢？

- 首先，Binlog是流式产生的，通过对Binlog的实时采集，把部分数据处理需求由每天一次的批处理分摊到实时流上。无论从性能上还是对MySQL的访问压力上，都会有明显地改善。
- 第二，Binlog本身记录了数据变更的类型（Insert/Update/Delete），通过一些语义方面的处理，完全能够做到精准的数据还原。

Binlog实时采集

对Binlog的实时采集包含两个主要模块：一是CanalManager，主要负责采集任务的分配、监控报警、元数据管理以及和外部依赖系统的对接；二是真正执行采集任务的Canal和CanalClient。



当用户提交某个DB的Binlog采集请求时，CanalManager首先会调用DBA平台的相关接口，获取这一DB所在MySQL实例的相关信息，目的是从中选出最适合Binlog采集的机器。然后把采集实例（Canal Instance）分发到合适的Canal服务器上，即CanalServer上。在选择具体的CanalServer时，CanalManager会考虑负载均衡、跨机房传输等因素，优先选择负载较低且同地域传输的机器。

CanalServer收到采集请求后，会在ZooKeeper上对收集信息进行注册。注册的内容包括：

- 以Instance名称命名的永久节点。
- 在该永久节点下注册以自身ip:port命名的临时节点。

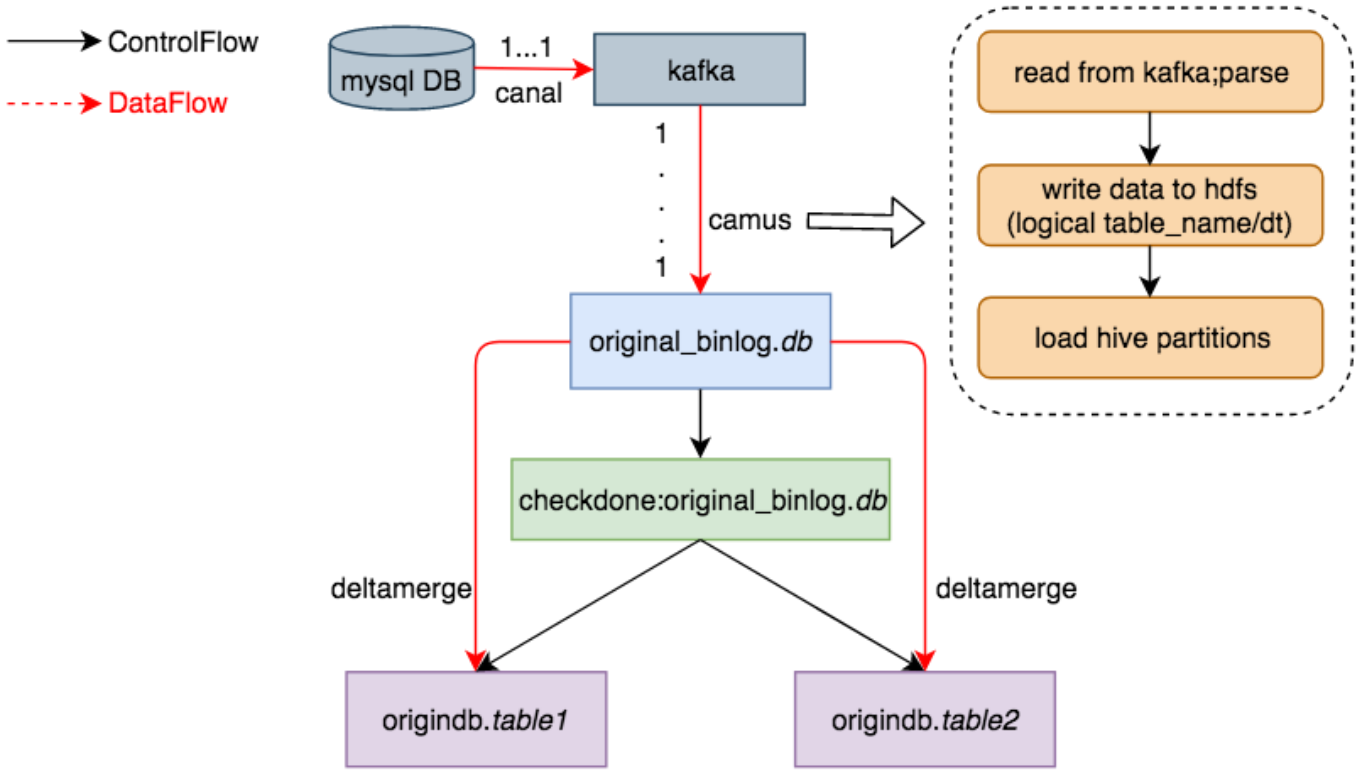
这样做的目的有两个：

- 高可用：CanalManager对Instance进行分发时，会选择两台CanalServer，一台是Running节点，另一台作为Standby节点。Standby节点会对该Instance进行监听，当Running节点出现故障后，临时节点消失，然后Standby节点进行抢占。这样就达到了容灾的目的。
- 与CanalClient交互：CanalClient检测到自己负责的Instance所在的Running CanalServer后，便会进行连接，从而接收到CanalServer发来的Binlog数据。

对Binlog的订阅以MySQL的DB为粒度，一个DB的Binlog对应了一个Kafka Topic。底层实现时，一个MySQL实例下所有订阅的DB，都由同一个Canal Instance进行处理。这是因为Binlog的产生是以MySQL实例为粒度的。CanalServer会抛弃掉未订阅的Binlog数据，然后CanalClient将接收到的Binlog按DB粒度分发到Kafka上。

离线还原MySQL数据

完成Binlog采集后，下一步就是利用Binlog来还原业务数据。首先要解决的第一个问题是把Binlog从Kafka同步到Hive上。



Kafka2Hive

整个Kafka2Hive任务的管理，在美团数据平台的ETL框架下进行，包括任务原语的表达和调度机制等，都同其他ETL类似。而底层采用LinkedIn的开源项目Camus，并进行了有针对性的二次开发，来完成真正的Kafka2Hive数据传输工作。

对Camus的二次开发

Kafka上存储的Binlog未带Schema，而Hive表必须有Schema，并且其分区、字段等的设计，都要便于下游的高效消费。对Camus做的第一个改造，便是将Kafka上的Binlog解析成符合目标Schema的格式。

对Camus做的第二个改造，由美团的ETL框架所决定。在我们的任务调度系统中，目前只对同调度队列的任务做上下游依赖关系的解析，跨调度队列是不能建立依赖关系的。而在MySQL2Hive的整个流程中，Kafka2Hive的任务需要每小时执行一次（小时队列），Merge任务每天执行一次（天队列）。而Merge任务的启动必须要严格依赖小时Kafka2Hive任务的完成。

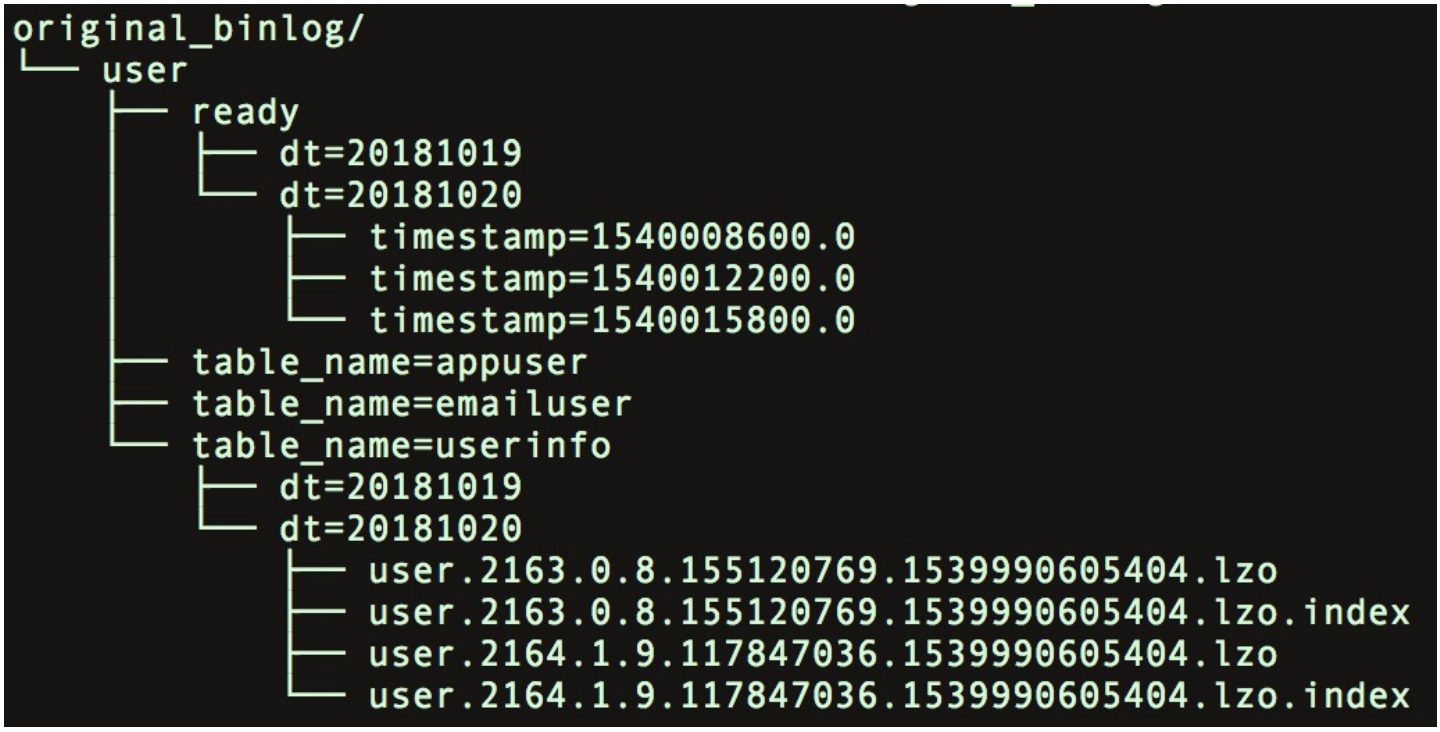
为了解决这一问题，我们引入了Checkdone任务。Checkdone任务是天任务，主要负责检测前一天的Kafka2Hive是否成功完成。如果成功完成了，则Checkdone任务执行成功，这样下游的Merge任务就可以正确启动了。

Checkdone的检测逻辑

Checkdone是怎样检测的呢？每个Kafka2Hive任务成功完成数据传输后，由Camus负责在相应的HDFS目录下记录该任务的启动时间。Checkdone会扫描前一天的所有时间戳，如果最大的时间戳已经超过了0点，就说明前一天的Kafka2Hive任务都成功完成了，这样Checkdone就完成了检测。

此外，由于Camus本身只是完成了读Kafka然后写HDFS文件的过程，还必须完成对Hive分区的加载才能使下游查询到。因此，整个Kafka2Hive任务的最后一步是加载Hive分区。这样，整个任务才算成功执行。

每个Kafka2Hive任务负责读取一个特定的Topic，把Binlog数据写入original_binlog库下的一张表中，即前面图中的original_binlog.*db*，其中存储的是对应到一个MySQL DB的全部Binlog。



上图说明了一个Kafka2Hive完成后，文件在HDFS上的目录结构。假如一个MySQL DB叫做user，对应的Binlog存储在original_binlog.user表中。ready目录中，按天存储了当天所有成功执行的Kafka2Hive任务的启动时间，供Checkdone使用。每张表的Binlog，被组织到一个分区中，例如userinfo表的Binlog，存储在table_name=userinfo这一分区中。每个table_name一级分区下，按dt组织二级分区。图中的xxx.lzo和xxx.lzo.index文件，存储的是经过lzo压缩的Binlog数据。

Merge

Binlog成功入仓后，下一步要做的就是基于Binlog对MySQL数据进行还原。Merge流程做了两件事，首先把当天生成的Binlog数据存放到Delta表中，然后和已有的存量数据做一个基于主键的Merge。Delta表中的数据是当天的最新数据，当一条数据在一天内发生多次变更时，Delta表中只存储最后一次变更后的数据。

把Delta数据和存量数据进行Merge的过程中，需要有唯一键来判定是否是同一条数据。如果同一条数据既出现在存量表中，又出现在Delta表中，说明这一条数据发生了更新，则选取Delta表的数据作为最终结果；否则说明没有发生任何变动，保留原来存量表中的数据作为最终结果。Merge的结果数据会Insert Overwrite到原表中，即图中的origindb.*table*。

Merge流程举例

下面用一个例子来具体说明Merge的流程。

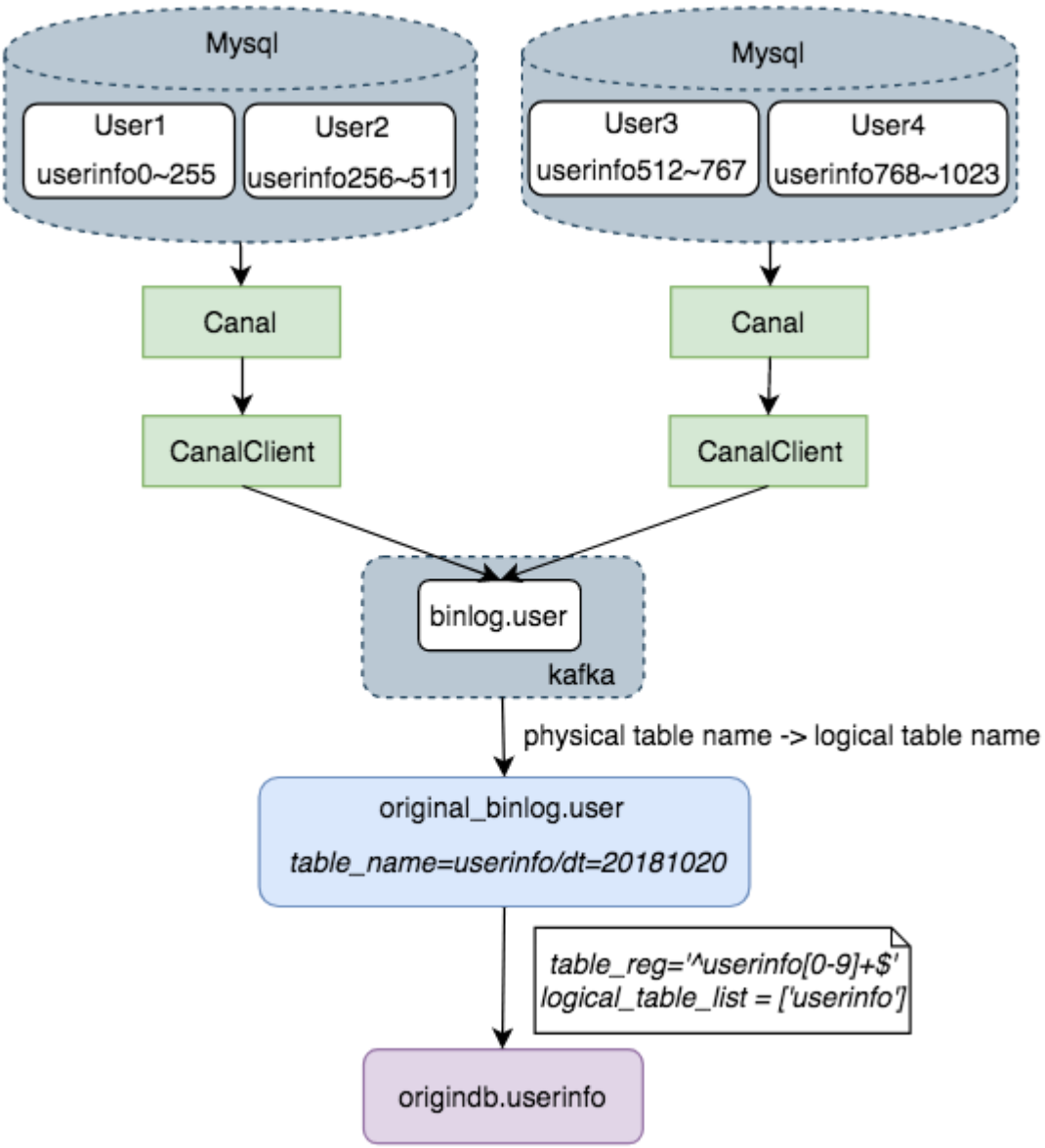
数据表共id、value两列，其中id是主键。在提取Delta数据时，对同一条数据的多次更新，只选择最后更新的一条。所以对id=1的数据，Delta表中记录最后一条更新后的值value=120。Delta数据和存量数据做Merge后，最终结果中，新插入一条数据（id=4），两条数据发生了更新（id=1和id=2），一条数据未变（id=3）。

默认情况下，我们采用MySQL表的主键作为这一判重的唯一键，业务也可以根据实际情况配置不同于MySQL的唯一键。

上面介绍了基于Binlog的数据采集和ODS数据还原的整体架构。下面主要从两个方面介绍我们解决的实际业务问题。

实践一：分库分表的支持

随着业务规模的扩大，MySQL的分库分表情况越来越多，很多业务的分表数目都在几千个这样的量级。而一般数据开发同学需要把这些数据聚合到一起进行分析。如果对每个分表都进行手动同步，再在Hive上进行聚合，这个成本很难被我们接受。因此，我们需要在ODS层就完成分表的聚合。



首先，在Binlog实时采集时，我们支持把不同DB的Binlog写入到同一个Kafka Topic。用户可以在申请Binlog采集时，同时勾选同一个业务逻辑下的多个物理DB。通过在Binlog采集层的汇集，所有分库的Binlog会写入到同一张Hive表中，这样下游在进行Merge时，依然只需要读取一张Hive表。

第二，Merge任务的配置支持正则匹配。通过配置符合业务分表命名规则的正则表达式，Merge任务就能了解自己需要聚合哪些MySQL表的Binlog，从而选取相应分区的数据来执行。

这样通过两个层面的工作，就完成了分库分表在ODS层的合并。

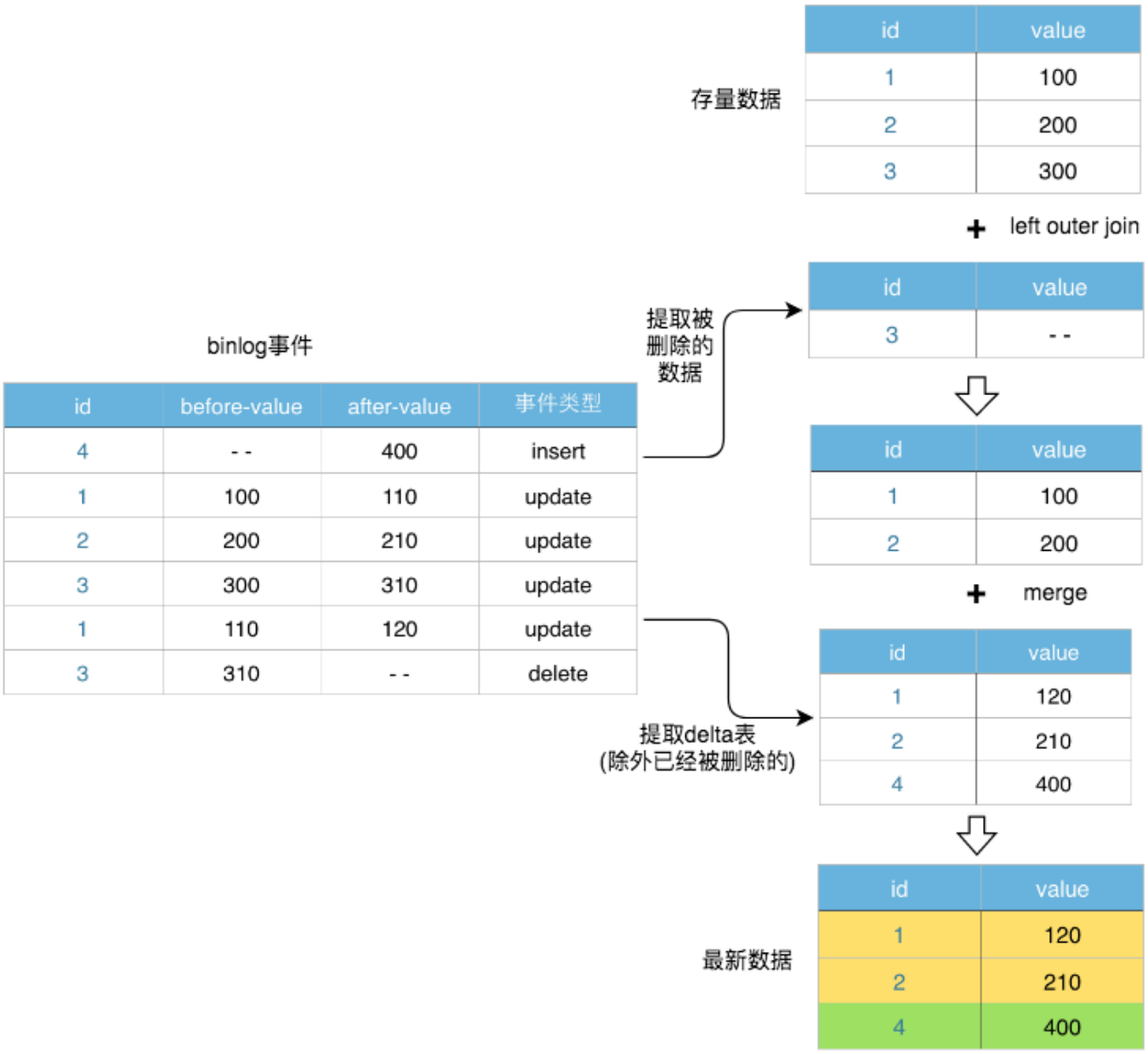
这里面有一个技术上的优化，在进行Kafka2Hive时，我们按业务分表规则对表名进行了处理，把物理表名转换成了逻辑表名。例如userinfo123这张表名会被转换为userinfo，其Binlog数据存储在original_binlog.user表的table_name=userinfo分区中。这样做的目的是防止过多的HDFS小文件和Hive分区造成的底层压力。

实践二：删除事件的支持

Delete操作在MySQL中非常常见，由于Hive不支持Delete，如果想把MySQL中删除的数据在Hive中删掉，需要采用“迂回”的方式进行。

对需要处理Delete事件的Merge流程，采用如下两个步骤：

- 首先，提取出发生了Delete事件的数据，由于Binlog本身记录了事件类型，这一步很容易做到。将存量数据（表A）与被删掉的数据（表B）在主键上做左外连接(Left outer join)，如果能够全部join到双方的数据，说明该条数据被删掉了。因此，选择结果中表B对应的记录为NULL的数据，即是应当被保留的数据。
- 然后，对上面得到的被保留下来的数据，按照前面描述的流程做常规的Merge。



总结与展望

作为数据仓库生产的基础，美团数据平台提供的基于Binlog的MySQL2Hive服务，基本覆盖了美团内部的各个业务线，目前已经能够满足绝大部分业务的数据同步需求，实现DB数据准确、高效地入仓。在后面的发展中，我们会集中解决CanalManager的单点问题，并构建跨机房容灾的架构，从而更加稳定地支撑业务的发展。

本文主要从Binlog流式采集和基于Binlog的ODS数据还原两方面，介绍了这一服务的架构，并介绍了我们在实践中遇到的一些典型问题和解决方案。希望能够给其他开发者一些参考价值，同时也欢迎大家和我们一起交流。

招聘

如果你对我们的工作内容比较感兴趣，欢迎发送简历给 wangmengmeng05@meituan.com，和我们一起致力于解决海量数据采集和传输的问题中来吧！

后台 基础研发平台 Binlog ODS 数据仓库

关注我们

微信搜索 "美团技术团队"