第五章 实时技术

为什么需要“实时技术”?

在大数据系统中，离线批处理技术虽然可以满足非常多的数据使用需求，但是每天的信息是瞬息万变的，数据价值是具有时效性的，在一条数据产生的时候，如果不能及时处理并在业务系统中使用，就不能让数据保持最高的“新鲜度”和价值最大化。

一、简介

相对于离线批处理技术，业务诉求更希望能拿到实时数据，这样就能在第一时间拿到经过加工后的数据，以便实时监控当前业务状态并做出运营决策，引导业务往好的方向发展。

比如网站上一个访问量很高的广告位，需要实时监控广告位的引流效果，如果转化率非常低的话，运营人员就需要及时更换为其他广告，以避免流量资源的浪费。

数据时效性：

- **离线**：在今天（T）处理N天前（T-N，N≥1）的数据，延迟时间粒度为“天”
- **准实时**：在当前小时（H）处理N小时前（H-N，N>0，如0.5小时、1小时等）的数据，延迟时间粒度为“小时”
- **实时**：在当前时刻处理当前的数据，延迟时间粒度为“秒”

离线和准实时都可以在批处理系统中实现（如Hadoop、Spark等），只是他们的调度周期不一样，而**实时数据需要再流式处理系统中完成。**

流式数据处理技术是指业务系统每产生一条数据，就会立刻被采集并实时发送到流式任务中进行处理，不需要定时调度任务来处理数据。

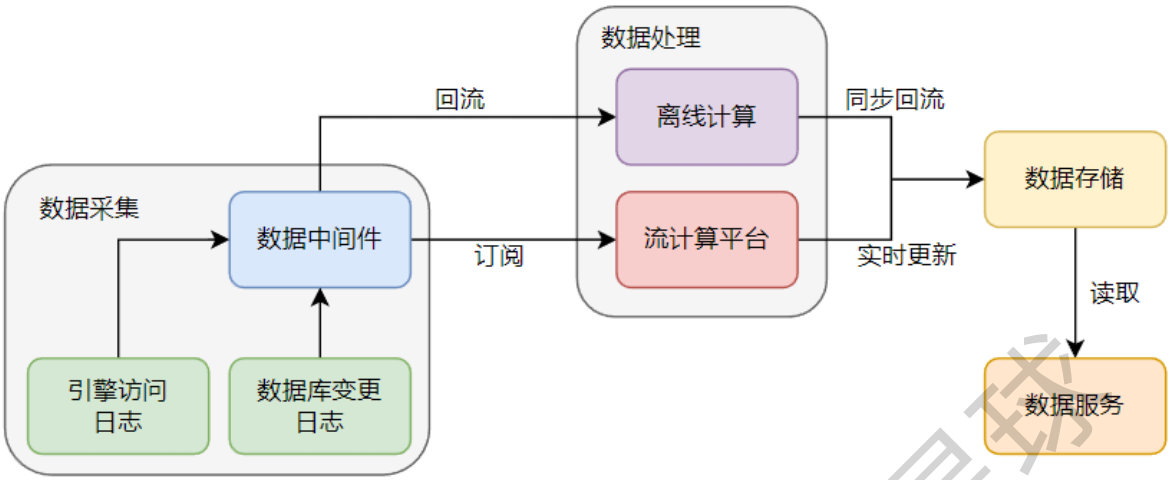
流式数据处理的特征：

- 数据处理的**实效性高**，延时粒度在秒级甚至毫秒级
- 属于**常驻任务**，任务一旦启动就会一直运行
    - 实时处理和离线处理最主要的差别：流式任务的数据源是无界的，而离线任务的数据源是有界的
- **性能要求高**，如果处理吞吐量跟不上采集吞吐量，最后得到的数据就失去了实时的特征
- 存在**应用局限性**，实时数据处理不能替代离线处理

二、流式技术架构

在流式计算技术中，需要各个子系统之间相互依赖形成一条数据处理链路，才能产出结果最终对外提供实时数据服务。

流式技术架构图



数据采集和数据服务部分实时和离线是公用的，因为在这两层中都不需要关心数据的时效性。这样才能做到数据源的统一，避免流式处理和离线处理的不一致。

根据功能划分子系统：

- **数据采集**：数据源头，实时采集业务的日志，将日志数据传输到数据中间件，供下游实时使用
- **数据处理**：数据采集到中间件后，将数据拉取到流式计算系统的任务重进行加工处理
- **数据存储**：数据被实时加工处理（比如聚合、清洗等）后，写入（增量操作）到在线服务的存储系统中，供下游调用方使用
- **数据服务**：在存储系统上架设统一的数据服务层（比如提供HSF接口、HTTP服务），用于获取实时计算结果

2.1 数据采集

实时采集的数据都来自于业务服务器，从数据种类方面可划分为两种：

- **数据库变更日志**，比如MySQL的binlog日志、HBase的hlog志、 OceanBase的变更日志、Oracle的变更日志等
- **引擎访问日志**，比如用户访问网站产生的Apache擎日志、搜索引擎的接口查询日志等

因为要考虑吞吐量和系统压力，并不是日志新增一条记录就采集一次，而是基于以下原则，对数据进行分批处理：

- **数据大小限制**：当达到限制条件时，把目前采集到的新数据作为一批（例如512KB一批）
- **时间阈值限制**：当时间达到一定条件时，也会把目前采集到的新数据作为一批，避免在数据量少的情况下一直不采集（例如30秒写一批）

上面其中一个条件满足时，就会采集一批新数据到数据中间件中。

2.2 数据处理

通过数据中间件获取到实时数据进行实时加工处理。比如Spark Streaming、Flink等。

书上提到阿里使用的是阿里云提供的StreamCompute系统。StreamCompute系统提供了SQL语义的流式数据分析能力（StreamSQL）、传统的开发模式以及流计算开发平台。

实时任务的典型问题：

- **去重指标**，可分为两种情况来看，**可以采用布隆过滤器或基数估计算法**：
  - 精确去重。在这种情况下，明细数据是必须要保存下来的，当遇到内存问题时，可以通过数据倾斜来进行处理，把一个节点的内存压力分到多个节点上。
  - 模糊去重。在去重的明细数据量非常大，而业务的精度要求不的情况下，可以使用相关的去重算法，把内存的使用 量降到千分之一甚至万分之一，以提高内存的利用率。
- **数据倾斜**，单个节点处理能力有限，此时需要**对数据进行分桶处理**：

    - 去重指标分桶：通过对去重值进行分桶Hash，相同的值一定会被放在同一个桶去重，最后再把每个桶里面的值进行加和就得到总值，这里利用了每个桶的CPU和内存资源。
    - 非去重指标分桶：数据随机分发到每个桶中，最后再把每个桶的值汇总，主要利用的是各个桶的CPU能力。
- **事务处理**，系统的不稳定性必然会导致数据的处理有可能出现失败的情况，不过流计算系统几乎都提供了**数据自动ACK、失败重发以及事务信息等机制**。
    - 超时时间：由于数据处理是按照批次来进行的，当一批数据处理超时时，会从拓扑的spout端重发数据。另外，批次处理的数据量不宜过大，应该增加一个限流的功能（限定一批数据的记录数或者容量等），避免数据处理超时。
    - 事务信息：每批数据都会附带一个事务ID的信息，在重发的情况下，让开发者自己根据事务信息去判断数据第一次到达和重发时不同的处理逻辑。
    - 备份机制：开发人员需要保证内存数据可以通过外部存储恢复，因此在计算中用到的中间结果数据需要备份到外部存储中。

2.3 数据存储

存储系统中的数据一般分为以下三种：

- **中间计算结果**：一些状态的保存，用于在发生故障时进行恢复
- **最终结果数据**：通过ETL处理后的实时结果数据，可被下游直接使用
- **维表数据**：在离线计算系统中，通过同步工具导入到在线存储系统中，供实时任务来关联实时流数据

因为实时任务时多线程处理的，**所以数据存储系统必须能够比较好的支持多并发读写，并且延时需要在毫秒级才能满足实时的性能要求。**在实践中，一般使用HBase、Tair、MongoDB等**列式存储系统。**由于**这些系统在写数据时是先写内存再落磁盘，因此写延时在毫秒级，读请求也有缓存机制，重要的是多并发读时也可以达到毫秒级延时。**

这些系统的缺点也是比较明显的，比如HBase，一张表须要有rowkey，而rowkey的规则限制了读取数据的方式，但是HBase一张表能存储几TB甚至几十TB的数据，而关系型数据库必须要分库分表才能实现这个量级的数据存储。因此，**对于海量数据的实时计算，一般会采用非关系型数据库，以应对大量的多并发读写。**

1、表名设计

设计规则：汇总层标识 + 数据域 + 主维度 + 时间维度

例如：dws_trd_slr_dtr ，表示汇总层交易数据，根据卖家（slr）主维度+0点截至当日（dtr）进行统计汇总。

优点：表名直观，所有主维度相同的数据都放在一张物理表中，避免表数量过多，难以维护。

2、rowkey设计

设计规则： MD5 + 主维度 + 维度标识 + 子维度1 + 时间维度 + 子维度2

例如：卖家ID的MD5前四位 + 卖家ID + app + 一级类目ID + ddd + 二级类目ID。

以MD5的前四位作为rowker的第一部分，可以把数据散列，让服务器整体负载是均衡的，避免热点问题。

在上面的例子中，卖家ID属于主维度 ，在查数据时是必传的。每个统计维度都会生成一个维度标识，以便在 rowkey上做区分。

2.4 数据服务

实时数据落地到存储系统中后，使用方通过统一的数据服务获取到实时数据。

使用统一的数据服务的优点：

- 不需要直连数据库，数据源等信息在数据服务层维护，这样当存储系统迁移时，对下游是透明的
- 调用方只需要使用服务层暴露的接口，不需要关心底层取数逻辑的实现
- 屏蔽存储系统间的差异，统一的调用日志输出，便于分析和监控下游使用情况

三、流式数据模型

3.1 数据分层

在流式数据模型中，数据模型整体上分为五层。

1、ODS层

属于**操作数据层**。是直接从业务系统采集过来的最原始数据，包含了所有业务的变更过程，数据粒度也是最细的。比如“订单粒度的变更过程，一笔订单有多条记录”。

2、DWD层

在ODS层基础上，根据业务过程建模出来的实时**事实明细层**。比如“订单粒度的支付记录，一笔订单只有一条记录”。

3、DWS层

实时**通用汇总层**。订阅明细层的数据后，会在实时任务中计算各个维度的汇总指标。如果维度是各个垂直业务线通用的，则会放在实时通用汇总层，作为通用的数据模型使用。比如“卖家的实时成交金额，一个卖家只有一条记录，并且指标在实时刷新”。

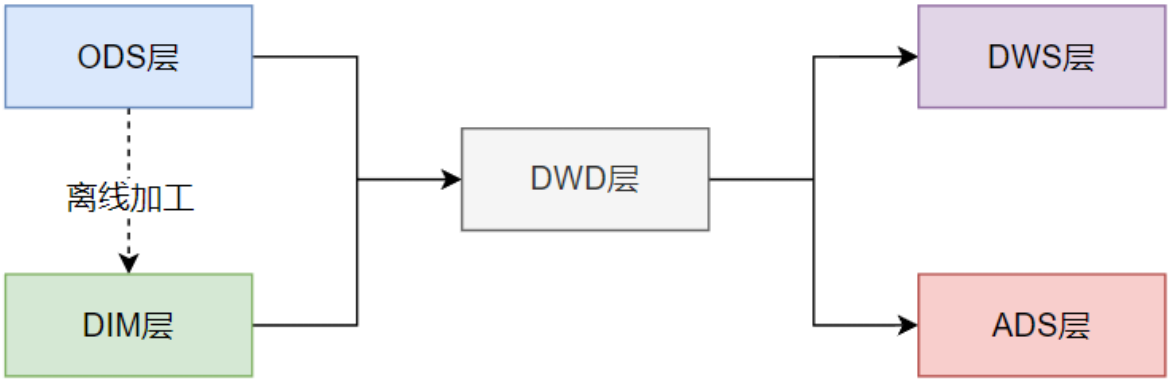
4、ADS层

个性化维度汇总层，对于不是特别通用的统计维度数据会放在这中，这里计算只有自身业务才会关注的维度和指标，眼其他业务线般没有交集，常用于一些垂直创新业务中。比如“外卖地区的实时成交金额，只有外卖业务使用”。

5、DIM层

实时维表层。该层的数据基本上都是从离线维表层导出来的，抽取到在线系统中供实时应用调用。这一层对实时应用来说是静态的，所有的ETL处理工作会在离线系统中完成。比如“订单商品类目和行业的对应关系维表”。

各层之间的数据流向：



ODS层到DIM层的ETL处理是在离线系统中进行的，处理完成后会同步到实时计算所使用的存储系统。

ODS层和DWD层会放在数据中间件中，供下游订阅使用。

DWS层和ADS层会落地到在线存储系统中，下游通过接口调用的形式使用。

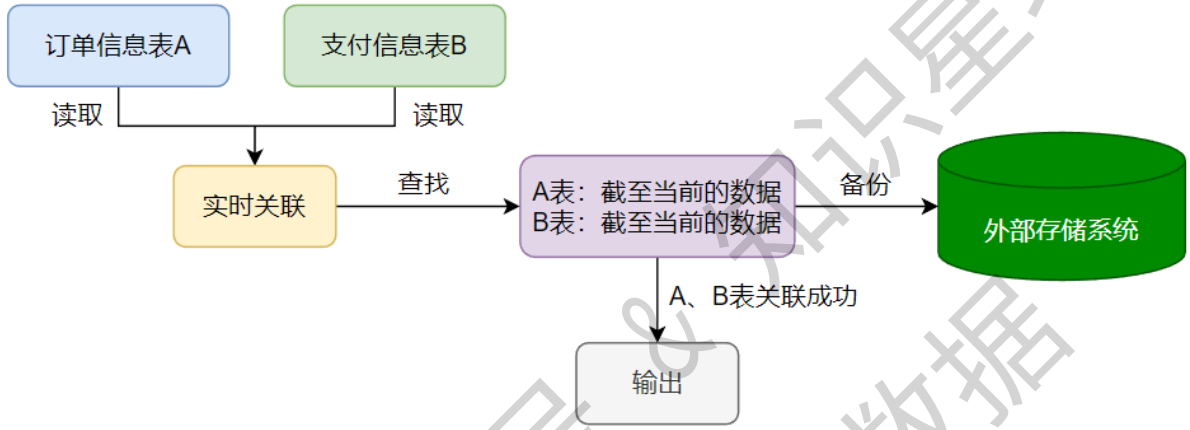
3.2 多流关联

在流式计算中常常需要把两个实时流进行主键关联，以得到对应的实时明细表。

在离线系统中两个表关联是非常简单的，因为离线计算在任务启动时已经可以获得两张表的全量数据，只要根据关联键进行分桶关联就可以了。

流式计算跟离线计算不一样，数据的到达是一个增量的过程，并且数据到达的时间是不确定的和无序的，因此在数据处理过程中会涉及中间状态的保存和恢复机制等细节问题。

多流关联的一个关键点就是需要相互等待，只有双方都到达了，才能关联成功。



比如A表和B表使用ID进行实时关联，如A表的某条数据到达，到B表的全量数据中查找，如果能查找到，说明可以关联上，拼接成一条记录直接输出到下游。但是如果关联不上，则需要放在内存或外部存储中等待，直到B表的记录也到达。

在上述关联流程中，不管是否关联成功，内存中的数据都需要备份到外部存储系统中，在任务重启时，可以从外部存储系统中恢复内存数据这样才能保证数据不丢失。因为在重启时，任务是续跑的，不会重新跑之前的数据。

3.3 维表使用

在实时计算中，关联维表一般会使用当前的实时数据（T）去关联T-2的维表数据，相当于在T的数据到达之前需要把维表数据准备好，并且一般是一份静态的数据。

之所以要在实时计算中这样做，是出于以下几点原因：

数据无法及时准备好。T-1的维表数据一般不能在零点准备就绪，因此去关联T-2维表。

无法准确获取全量的最新数据。维表一般是全量的数据，如果需要实时获取到当天的最新维表数据，则需要T-1的数据 + 当天变更才能获取到完整的维表数据。

数据存在无序性。维表作为实时流输入，维表数据获取是比较困难的，因为实时应用永远也不知道什么时候是最新的。

五、大促挑战和保障

4.1 大促特征

1、毫秒级延时需求

大促期间，业务方和用户都会对实时数据非常关注，特别是在跨过零点的时候，第一个实时数字的跳动对业务方来说意义重大，预示着大促狂欢节真正开始。

2、洪峰明显

特别是零点开售时的峰值陡峰，是非常明显的，一般是日常峰值的几十倍。

3、高保障性

因为关注的人非常多，所以在大促期间一般都要求高保障性。对于强保障的数据，需要做多链路冗余，当任何一条链路出现问题时，都能够切换到备链路。

4、公关特性

大促期间，数据及时对公众披露是一项重要的工作，这时候要求实时计算的数据质量非常高。

大促是一场对数据计算的高吞吐量、低延时、高保障性、高准确性的挑战。

4.2 大促保障

1、如何进行实时任务优化

1）独占资源和共享资源的策略

如果一个任务在运行时80%以上的时间都需要去抢资源，这时候就需要考虑给它分配更多的独占资源，避免抢不到CPU资源导致吞吐量急剧下降。

2）合理选择缓存机制，尽量降低读写库次数

让最热和最可能使用的数据留在内存中，读写库次数降低后，吞吐量自然就上升了。

3）计算单元合并，降低拓扑层级

4）内存对象共享，避免字符拷贝

在海量数据处理中，大部分对象都是以字符串形式存在的，在不同线程间合理共享对象，可以大幅降低字符拷贝带来的性能消耗，不过要注意内存溢出问题。

5）在高吞吐量和低延时间取平衡

当把多个读写库操作或者ACK操作合并成一个时，可以大幅降低因为网络请求带来的消耗，不过也会导致延时高一些，在业务上衡量进行取舍。

2、如何进行数据链路保障

实时数据的处理链路非常长，为了保障实时数据的可用性，需要对整条计算链路都进行多链路搭建，做到多机房容灾，甚至异地容灾。

3、如何进行压测

分数据压测和产品压测两步进行。

数据压测主要是蓄洪压测，就是把几个小时甚至几天的数据积累下来，并在某个时段全部放开，模拟“双11”洪峰流量的情况。

产品压测则包括产品本身压测和前端页面稳定性测试。

产品本身压测：收集大屏服务端的所有读操作的URL ，通过压测平台进行压测流量回放，按照QPS: 500次 / 秒的目标进行压测

前端页面稳定性测试：将大屏页面在浏览器中打开，并进行24小时的前端页面稳定性测试。监控大屏前端JS对客户端浏览器的内存、CPU等的消耗，检测出前端JS内存泄漏等问题并修复，提升前端页面的稳定性。