消息队列 Pulsar 版

最佳实践

腾讯云

文档目录

最佳实践
交易对账

最近更新时间：2021-09-26 11:18:55

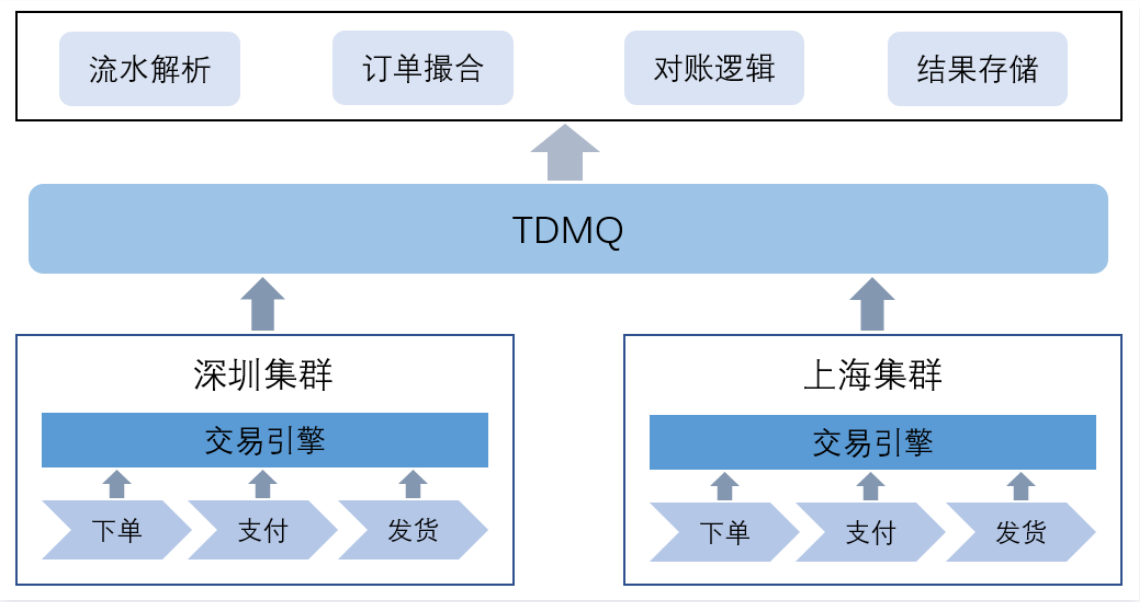
场景描述

对账是任何一个计费系统都需要的一个**辅助系统**。无论是对账作为支付的主路或是旁路系统，为了保证计费的准确性，都需要在支付过程中或支付完成之后进行对账。通过引入 TDMQ Pulsar 版，一方面保证了对账的时效性，另一方面也不会影响交易的关键路径。

遇到的问题

1. **系统解耦**
   交易涉及系统众多，系统之间需要**解耦**，避免互相影响。
2. **数据到达时间差**
   系统之间数据到达会有**时间差**，保证不同时间到达的数据能够聚合计算。
3. **数据一致性**
   保证数据**不丢失**，不会因为数据丢失造成对账结果异常。
4. **跨区域数据传输**
   系统部署分布在不同的地区，需要进行**跨区域**的数据传输。

部署架构图



问题解决

我们将根据上面提到的问题，使用 TDMQ Pulsar 版的解决方式。

1. 系统解耦问题

直观的看，为了实现各个系统之间的对账，我们可以直接将消息上报给对账系统，对账系统负责接受消息并进行对账处理。但是这里又面临一些问题，这里需要对接的系统很多，并且系统还会不断的增加。这样会花费大量的时间在系统之间的对接上，并且对现网的系统流程的侵入性就会非常大。显然，这样的设计是非常不合理的。我们通过 TDMQ Pulsar 版的引入，使得系统之间只需要和 MQ 进行统一对接，这样单个系统的问题也不会影响到其他服务。

2. 数据到达时间差问题

对账需要进行各个系统之间的数据聚合，要想进行实时的数据聚合，正常情况下数据到达的时间不会相差太大，但是因为系统之间的流程总是会有先后，当一个系统的数据延迟之后，为了能够实现数据的聚合，我们需要控制数据的读取速度，以免大量数据进入到对账系统中等待。通过 TDMQ Pulsar 版的暂时存储消息，使得同一时间的数据到达时间大致相同。

3. 数据一致性问题

TDMQ Pulsar 版提供高一致的可靠数据存储，保证数据不会丢失，同时提供高可用的服务，异常情况下能够快速自动修复。

4. 跨区域数据传输问题

TDMQ Pulsar 版提供两种方案来实现多个城市之间的数据复制，为业务层提供实时数据复制通道。

- 对于非常重要的数据需要满足跨城容灾级别的场景，可以支持多个区域之间进行强一致性的数据同步，使得消息的存储分布在不同的区域。
- 对于一些数据并不需要强一致要求的场景，TDMQ Pulsar 版提供多城市之间异步复制方案，来达到多地数据最终一致。

两种跨城同步方案的对比如下：

跨城方案	生产耗时	容灾性	存储成本
多城市强一致	高	高	少
多城市最终一致	低	低	多

通过引入 TDMQ Pulsar 版和实时计算的能力，我们将交易对账从按天的模式发展为实时对账的模式。更加快速的检测交易的准确性。

消息幂等性

最近更新时间：2023-06-30 15:33:02

应用的幂等是在分布式系统设计时必须要考虑的一个关键点。如果对幂等没有额外的考虑，那么在业务出现处理失败的情况时，可能出现重复消费相同的消息，从而导致出现不符合业务预期的情况。为了避免上述异常，消息队列的消费者在接收到消息后，有必要根据业务上的唯一 Key 对消息做幂等处理。

什么是消息幂等

定义

当业务多次消费到同一条消息的结果与消费一次的结果是相同的，同时多次消费同一条消息并未对业务系统产生任何负面影响，那么这个消费者的处理过程就是幂等的。

场景示例

举个例子，在银行支付系统的场景下，当消费端消费到扣款消息后，系统将对订单执行扣款操作，扣款金额为1元。如果因由于网络不稳定等一些原因导致扣款消息再次被消费到，如果最终的业务结果是只扣款一次，扣费1元，且用户的扣款记录中对应的订单只有一条扣款流水，并没有多次被多次扣费，那么这次扣款操作是符合要求的，整个消费过程实现了消费幂等。

适用场景

发送消息引起消息重复场景

生产者在发送一条消息后，服务端接收成功并完成持久化，如果此时出现了网络闪断或者客户端重启等异常导致服务端对客户端应答失败，此时生产者由于没有收到服务端的确认消息，从而尝试再次发送消息，这时会导致后续消费者会收到两条内容相同但 Message ID 不同的消息。

消费消息引起消息重复场景

消费端消费到了消息并完成业务处理，当消费端给服务端返回 ACK 的时候，此时网络发生异常。当消费端再次来消费消息时，会再次消费到已被处理过的消息，消费者收到了两条内容相同并且 Message ID 也相同的消息。

处理方案

根据以上两种场景可以看出消息重复会出现以下两种情况：

- 可能出现不同的 Message ID 对应的消息内容相同。
- 可能是相同的 Message ID 同时消息内容相同。

所以不建议以 Message ID 作为处理依据，建议以业务的唯一标识作为幂等处理的依据。例如支付场景可以将订单号，作为幂等处理的依据。消费到消息后，取出业务中的订单号，业务根据订单号进行判断是否被处理。

代码示例

```
public static class Order {
    public String orderId;
    public String orderData;
}
```

生产端

```
Producer<Order> producer =
client.newProducer(Schema.AVRO(Order.class)).create();
producer.newMessage().value(new Order("orderid-12345678", "orderData")).send();
```

消费端

```
Consumer<Order> consumer =
client.newConsumer(Schema.AVRO(Order.class)).subscribe();
Order order = consumer.receive().getValue();
String key = order.orderId;
```

获取到业务的唯一标识 orderId 后，对其进行去重操作。

常见的去重方式

利用数据库进行去重

业务上的幂等操作可以添加一个过滤的数据库，例如设置一个去重表，也可以在数据库中通过唯一索引来去重。
举一个例子，现在要根据订单流转的消息在数据库中写一张订单 Log 表，可以把订单 ID 和修改时间戳做一个唯一索引进行约束。
当消费端消费消息出现相同内容的消息，会多次去订单 Log 表中进行写入，由于添加了唯一索引，除了第一条之外，后面的都会失败，这就从业务上保证了幂等，即使消费多次，也不会影响最终的数据结果。

设置全局唯一消息 ID 或者任务 ID

调用链 ID 也可以应用在这里。在消息生产的时候向每条消息中添加一个唯一 ID，消息被消费后，在缓存中设置一个 Key 为对应的唯一 ID，代表数据已经被消费，当消费端去消费时，就可以根据这条记录，来判断是否已经处理过。

消息压缩

最近更新时间：2023-11-02 15:45:52

背景描述

由于 Pulsar 限制消息最大为5MB，消息体过大将会导致消息发送失败。这时需要客户端将大消息进行压缩，以支持20MB大小的消息体发送。

Pulsar 大消息处理

Pulsar 的消息最大限制默认是 5MB，Producer 发送的消息大小超过5MB会导致消息发送失败。如果客户端发送单条消息大小超过该限制，我们可以采用如下两种方式来处理：

- Chunk Message：Pulsar 提供 Chunk Message 功能，开启 Chunk 机制时，客户端能够自动对大消息进行拆分，并保证消息的完整性，在 Consumer 能自动整合消息。
- 压缩消息：主要是对消息数据中相同字符序列进行替换，来压缩消息的大小。Pulsar 支持 LZ4、ZLIB、ZSTD、SNAPPY 四种压缩算法。

我们这里推荐压缩消息对大消息进行处理。

压缩算法分析比较

算法介绍

- LZ4

  LZ4 是一种无损数据压缩算法，可以提供极快的压缩和解压缩速度，对于 CPU 占用少。
- ZLIB

  ZLIB 压缩算法是一种常用的无损数据压缩技术，它可以有效地减少收发数据的大小，从而提高网络传输效率和网络容量。ZLIB 压缩算法是基于 Lempel-Ziv 压缩算法的一种变体，可以将原始数据压缩到原来的一半大小以下，并且支持压缩和解压缩操作。
- ZSTD

  ZSTD 压缩算法是一种 Huffman 编码的压缩算法，是 LZ77 的一种变种，可以针对不同数据进行有效压缩。它是一种实时编码算法，在处理大数据时可以更快速、更高效地压缩数据。相比其他压缩算法，ZSTD 在提高数据压缩率的同时兼顾压缩速度。
- SNAPPY

  SNAPPY 压缩是一种无损压缩技术，它依赖于 LZ77 原理来实现压缩效果。SNAPPY 压缩的核心原理是：只要数据流找到两个字符串之间的重复，就会用一组更短的代码来表示这个字符串，这样就可以减少数据流的大小。

算法对比

压缩算法	压缩比	压缩速度	解压速度

ZLIB 1.2.11 -1	2.743	110MB/S	400MB/S
LZ4 1.8.1	2.101	750MB/S	3700MB/S
ZSTD 1.3.4-1	2.877	470MB/S	1380MB/S
SNAPPY 1.1.4	2.091	530MB/S	1800MB/S

- 吞吐量：LZ4 > SNAPPY > ZSTD > ZLIB
- 压缩比：ZSTD > ZLIB > LZ4 > SNAPPY
- 物理资源方面，SNAPPY 算法占用的网络带宽最多，ZSTD 算法占用的网络带宽最少。

各压缩算法测试

测试结果

> **注意：**
> 以下测试结果仅供参考。压缩效果，需要根据具体消息体内容来验证。

消息大小	消息	压缩算法	topic 监控消息大小	客户端消息压缩耗时	消息发送耗时
5M	随机消息体	LZ4（阀值5MB）	9.95MB	31ms	0.049ms
		ZLIB	7.26MB	31ms	0.038ms
		ZSTD	8.20MB	31ms	0.039ms
		SNAPPY（阀值5MB）	9.70MB	33ms	0.046ms
6M	随机消息体	ZLIB（阀值6MB）	8.71MB	35ms	0.044ms
		ZSTD（阀值6MB）	9.84MB	35ms	0.046ms
20M	相同消息体	LZ4	0.16MB	41ms	0.006ms
		ZLIB	0.20MB	42ms	0.006ms
		ZSTD	0.01MB	42ms	0.003ms
		SNAPPY	2.47MB	41ms	0.021ms

40M	相同消息体	LZ4	0.32MB	123ms	0.008ms
		ZLIB	0.39MB	122ms	0.008ms
		ZSTD	0.01MB	124ms	0.004ms
		SNAPPY	4.95MB	123ms	0.036ms
80M	相同消息体	LZ4	0.63MB	241ms	0.009ms
		ZLIB	0.39MB	244ms	0.01ms
		ZSTD	0.01MB	243ms	0.004ms
		SNAPPY（阀值80M）	9.9MB	243ms	0.056ms
160M	相同消息体	LZ4	1.26MB	484ms	0.013ms
		ZLIB	1.56MB	479ms	0.016ms
		ZSTD	0.03MB	481ms	0.004ms
320M	相同消息体	LZ4	2.5MB	1035ms	0.03ms
		ZLIB	3.1MB	1008ms	0.027ms
		ZSTD	0.03MB	949ms	0.004ms
585M	相同消息体	LZ4	4.59MB	1705ms	0.027ms
		ZLIB	5.67MB	1733ms	0.03ms
		ZSTD	0.11MB	1722ms	0.006ms

总结：

- 在纯随机数据流中，四种压缩算法压缩效率都不是很高。消息大小超过5MB，四种压缩算法都无法将消息压缩到5MB以内。
- 在重复数据较多的数据流中，四种压缩算法可以实现很高的压缩速率，其中 LZ4、ZLIB、ZSTD 压缩算法可以实现将600MB内的消息压缩到5MB以内。

消息压缩 Demo 及使用测试

消息压缩 Demo 可参见：tdmq-sdk-demo 。

使用测试

生产端调用参数：

```
java -jar tdmq-sdk-demo-1.0-SNAPSHOT-jar-with-dependencies.jar pulsar://xxxx:6650
eyJrZXlJZCI6ImRlZmF1bHRfa2V5SWQiLCJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJzdXBlcnVzZXIif
Q.dYcCfp4XrdWRKdKaWylobY-_xEExfRCi1pMvNyZXbqU
pulsar-78ra8ownxb7d/BigMSGSpace/BigMSGTopic subname 1 500 0 1 20480 1 0
```

消费端调用参数：

```
java -jar tdmq-sdk-demo-1.0-SNAPSHOT-jar-with-dendencies.jar pulsar://xxxx:6650
eyJrZXlJZCI6ImRlZmF1bHRfa2V5SWQiLCJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJzdXBlcnVzZXIifQ
.dYcCfp4XrdWRKdKaWylobY-_xEExfRCi1pMvNyZXbqU
pulsar-92d7w2mjwmv9/BigMessSpace/BigMessTopic subname 1 500 1
```