27 | Pulsar的存储计算分离设计：全新的消息队列设计思路

李玥 · 消息队列高手课

你好，我是李玥。

之前的课程，我们大部分时间都在以 RocketMQ、Kafka 和 RabbitMQ 为例，通过分析源码的方式，来讲解消息队列的实现原理。原因是，这三种消息队列在国内的受众群体非常庞大，大家在工作中会经常用到。这节课，我给你介绍一个不太一样的开源消息队列产品：Apache Pulsar。

Pulsar 也是一个开源的分布式消息队列产品，最早是由 Yahoo 开发，现在是 Apache 基金会旗下的开源项目。你可能会觉得好奇，我们的课程中为什么要花一节课来讲 Pulsar 这个产品呢？原因是，Pulsar 在架构设计上，和其他的消息队列产品有非常显著的区别。我个人的观点是，Pulsar 的这种全新的架构设计，很可能是消息队列这类中间件产品未来架构的发展方向。

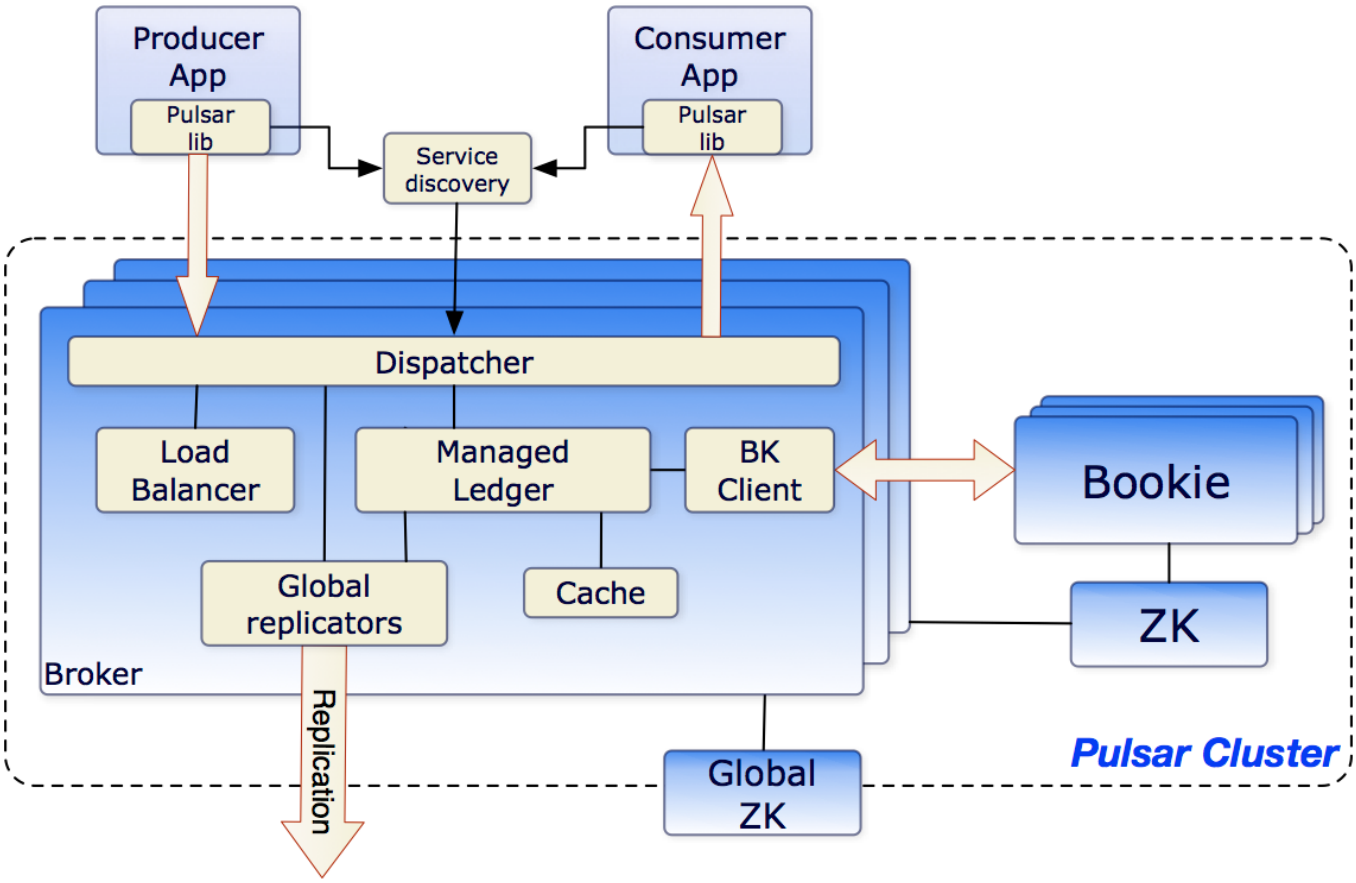
接下来我们一起看一下，Pulsar 到底有什么不同？

Pulsar 的架构和其他消息队列有什么不同?

我们知道，无论是 RocketMQ、RabbitMQ 还是 Kafka，消息都是存储在 Broker 的磁盘或者内存中。客户端在访问某个主题分区之前，必须先找到这个分区所在 Broker，然后连接到这个 Broker 上进行生产和消费。

在集群模式下，为了避免单点故障导致丢消息，Broker 在保存消息的时候，必须也把消息复制到其他的 Broker 上。当某个 Broker 节点故障的时候，并不是集群中任意一个节点都能替代这个故障的节点，只有那些“和这个故障节点拥有相同数据的节点”才能替代这个故障的节点。原因就是，每一个 Broker 存储的消息数据是不一样的，或者说，每个节点上都存储了状态（数据）。这种节点称为“有状态的节点（Stateful Node）”。

Pulsar 与其他消息队列在架构上，最大的不同在于，它的 Broker 是无状态的（Stateless）。也就是说，**在 Pulsar 的 Broker 中既不保存元数据，也不存储消息**。那 Pulsar 的消息存储在哪儿呢？我们来看一下 Pulsar 的架构是什么样的。



这张 Pulsar 的架构图来自 Pulsar 的官方文档，如果你想了解这张架构图的细节，可以去看官方文档中的Architecture Overview。我来给你解读一下这张图中我们感兴趣的重点内容。

先来看图中右侧的 Bookie 和 ZK 这两个方框，这两个方框分别代表了 BookKeeper 集群和 ZooKeeper 集群。ZooKeeper 集群的作用，我在《24 | Kafka 的协调服务 ZooKeeper：实现分布式系统的“瑞士军刀》这节课中专门讲过，在 Pulsar 中，ZooKeeper 集群的作用和在 Kafka 中是一样的，都是被用来存储元数据。BookKeeper 集群则被用来存储消息数据。

那这个 BookKeeper 又是什么呢？BookKeeper 有点儿类似 HDFS，是一个分布式的存储集群，只不过它的存储单元和 HDFS 不一样，在 HDFS 中存储单元就是文件，这个很好理解。而 BookKeeper 的存储单元是 Ledger。这个 Ledger 又是什么呢？

这里再次吐槽一下国外程序员喜欢发明概念、增加学习成本这个坏习惯。其实 Ledger 就是一段 WAL（Write Ahead Log），或者你可以简单地理解为某个主题队列的一段，它包含了连续的若干条消息，消息在 Ledger 中称为 Entry。为了保证 Ledger 中的 Entry 的严格顺序，Pulsar 为 Ledger 增加一次性的写入限制，Broker 创建一个 Ledger 后，只有这个 Broker 可以往 Ledger 中写入 Entry，一旦 Ledger 关闭后，无论是 Broker 主动关闭，还是因为 Broker 宕机异常关闭，这个 Ledger 就永远只能读取不能写入了。如果需要继续写入 Entry，只能新建另外一个 Ledger。

请你注意一下，这种“一次性写入”的设计，它的主要目的是为了解决并发写入控制的问题，我在之前课程中讲过，对于共享资源数据的并发写一般都是需要加锁的，否则很难保证数据的一致性。对于分布式存储来说，就需要加“分布式锁”。

但我们知道，分布式锁本身就很难实现，使用分布式锁对性能也会有比较大的损失。这种“一次性写入”的设计，只有创建 Ledger 的进程可以写入数据，Ledger 这个资源不共享，也就不需要加锁，是一种很巧妙的设计，你在遇到类似场景的时候可以借鉴。

消息数据由 BookKeeper 集群负责存储，元数据由 ZooKeeper 集群负责存储，Pulsar 的 Broker 上就不需要存储任何数据了，这样 Broker 就成为了无状态的节点。

虽然 Broker 是无状态的，不存储任何的数据，但是，在一个特定的时刻，每一个主题的分区，还是要落在某个具体的 Broker 上。不能说多个 Broker 同时读写同一个分区，因为这样是没有办法保证消息的顺序的，也没有办法来管理消费位置。

再来看图中左侧最大的那个 Broker 方框，在 Broker 中包含了几个重要的模块。Load Balancer 负责动态的分配，哪些 Broker 管理哪些主题分区。Managed Ledger 这个模块负责管理本节点需要用到的那些 Ledger，当然这些 Ledger 都是保存在 BookKeeper 集群中的。为了提升性能，Pulsar 同样采用用了一个 Cache 模块，来缓存一部分 Ledger。

Pulsar 的客户端要读写某个主题分区上的数据之前，依然要在元数据中找到分区当前所在的那个 Broker，这一点是和其他消息队列的实现是一样的。不一样的地方是，其他的消息队列，分区与 Broker 的对应关系是相对稳定的，只要不发生故障，这个关系是不会变的。而在 Pulsar 中，这个对应关系是动态的，它可以根据 Broker 的负载情况进行动态调整，而且由于 Broker 是无状态的，分区可以调整到集群中任意一个 Broker 上，这个负载均衡策略就可以做得非常简单并且灵活。如果某一个 Broker 发生故障，可以立即用任何一个 Broker 来替代它。

那在这种架构下，Pulsar 又是如何来完成消息收发的呢？客户端在收发消息之前，需要先连接 Service Discovery 模块，获取当前主题分区与 Broker 的对应关系，然后再连接到相应 Broker 上进行消息收发。客户端收发消息的整体流程，和其他的消息队列是差不多的。比较显著的一个区别就是，消息是保存在 BookKeeper 集群中的，而不是本机上。数据的可靠性保证也是 BookKeeper 集群提供的，所以 Broker 就不需要再往其他的 Broker 上复制消息了。

图中的 Global replicators 模块虽然也会复制消息，但是复制的目的是为了在不同的集群之间共享数据，而不是为了保证数据的可靠性。集群间数据复制是 Pulsar 提供的一个特色功能，具体可以看一下 Pulsar 文档中的geo-replication这部分。

存储计算分离的设计有哪些优点？

在 Pulsar 这种架构下，消息数据保存在 BookKeeper 中，元数据保存在 ZooKeeper 中，Broker 的数据存储的职责被完全被剥离出去，只保留了处理收发消息等计算的职责，这就是一个非常典型的“存储计算分离”的设计。

什么是存储计算分离呢？顾名思义，就是将系统的存储职责和计算职责分离开，存储节点只负责数据存储，而计算节点只负责计算，也就是执行业务逻辑。这样一种设计，称为存储计算分离。存储计算分离设计并不新鲜，它的应用其实是非常广泛的。

比如说，所有的大数据系统，包括 Map Reduce 这种传统的批量计算，和现在比较流行的 Spark、Flink 这种流计算，它们都采用的存储计算分离设计。数据保存在 HDFS 中，也就是说 HDFS 负责存储，而负责计算的节点，无论是用 YARN 调度还是 Kubernetes 调度，都只负责“读取 – 计算 – 写入”这样一种通用的计算逻辑，不保存任何数据。

更普遍的，**我们每天都在开发的各种 Web 应用和微服务应用，绝大多数也采用的是存储计算分离的设计**。数据保存在数据库中，微服务节点只负责响应请求，执行业务逻辑。也就是说，数据库负责存储，微服务节点负责计算。

那存储计算分离有什么优点呢？我们分两方面来看。

对于计算节点来说，它不需要存储数据，节点就变成了无状态的（Stateless）节点。一个由无状态节点组成的集群，管理、调度都变得非常简单了。集群中每个节点都是一样的，天然就支持水平扩展。任意一个请求都可以路由到集群中任意一个节点上，负载均衡策略可以做得非常灵活，可以随机分配，可以轮询，也可以根据节点负载动态分配等等。故障转移（Failover）也更加简单快速，如果某个节点故障了，直接把请求分配给其他节点就可以了。

对比一下，像 ZooKeeper 这样存储计算不分离的系统，它们的故障转移就非常麻烦，一般需要用复杂的选举算法，选出新的 leader，提供服务之前，可能还需要进行数据同步，确保新的节点上的数据和故障节点是完全一致之后，才可以继续提供服务。这个过程是非常复杂而且漫长的。

对于计算节点的开发者来说，可以专注于计算业务逻辑开发，而不需要关注像数据一致性、数据可靠性、故障恢复和数据读写性能等等这些比较麻烦的存储问题，极大地降低了开发难度，提升了开发效率。

而对于存储系统来说，它需要实现的功能就很简单，系统的开发者只需要专注于解决一件事就可以了，那就是“如何安全高效地存储数据？”并且，存储系统的功能是非常稳定的，比如像

ZooKeeper、HDFS、MySQL 这些存储系统，从它们诞生到现在，功能几乎就没有变过。每次升级都是在优化存储引擎，提升性能、数据可靠性、可用性等等。

接下来说存储计算分离这种设计的缺点。

俗话说，背着抱着一样沉。对于一个系统来说，无论存储和计算是不是分离的，它需要完成的功能和解决的问题是一样的。就像我刚刚讲到的，Pulsar 的 Broker 相比于其他消息队列的 Broker，各方面都变的很简单。这并不是说，存储计算分离的设计能把系统面临的各种复杂的问题都解决了，其实一个问题都没解决，只是把这些问题转移到了 BookKeeper 这个存储集群上了而已。

BookKeeper 依然要解决数据一致性、节点故障转移、选举、数据复制等等这些问题。并且，存储计算分离之后，原来一个集群变成了两个集群，整个系统其实变得更加复杂了。

另外，存储计算分离之后，系统的性能也会有一些损失。比如，从 Pulsar 的 Broker 上消费一条消息，Broker 还需要去请求 BookKeeper 集群读取数据，然后返回给客户端，这个过程至少增加了一次网络传输和 n 次内存拷贝。相比于直接读本地磁盘，性能肯定是要差一些的。

不过，对于业务系统来说，采用存储计算分离的设计，它并不需要自己开发一个数据库或者 HDFS，只要用现有的成熟的存储系统就可以了，所以相当于系统的复杂度还是降低了。相比于存储计算分离带来的各种优点，损失一些性能也是可以接受的。

因此，对于大部分业务系统来说，采用存储计算分离设计，都是非常划算的。

小结

这节课我们一起分析了 Apache Pulsar 的架构，然后一起学习了一下存储计算分离的这种设计思想。

Pulsar 和其他消息队列最大的区别是，它采用了存储计算分离的设计。存储消息的职责从 Broker 中分离出来，交给专门的 BookKeeper 存储集群。这样 Broker 就变成了无状态的节点，在集群调度和故障恢复方面更加简单灵活。

存储计算分离是一种设计思想，它将系统的存储职责和计算职责分离开，存储节点只负责数据存储，而计算节点只负责计算，计算节点是无状态的。无状态的计算节点，具有易于开发、调度灵活的优点，故障转移和恢复也更加简单快速。这种设计的缺点是，系统总体的复杂度更高，性能也更差。不过对于大部分分布式的业务系统来说，由于它不需要自己开发存储系统，采用存储计算分离的设计，既可以充分利用这种设计的优点，整个系统也不会因此变得过于复杂，综合评估优缺点，利大于弊，更加划算。

思考题

课后请你想一下，既然存储计算分离这种设计有这么多的优点，那为什么除了 Pulsar 以外，大多数的消息队列都没有采用存储计算分离的设计呢？欢迎在评论区留言，写下你的想法。

感谢阅读，如果你觉得这篇文章对你有一些启发，也欢迎把它分享给你的朋友。

精选留言 (18)

DFighting
2019-10-25

其实存储计算分离在数据的相关性不大的情况下优势会很明显，我理解这种是对数据和计算进行了一种操作和对象的切分，在大多数情况下，操作和对象的耦合度不高，可以使用，因为无状态、单一功能这就是微服务架构的优势。但是这里有个天然的缺陷就是牺牲了大量的磁盘读写和网络IO的性能，如果计算和对象耦合度过大，也就是计算需要频繁读写多数据源的数据，那这种就不如传统的MQ了。不过随着数据量的增大，读写计算分离应该是趋势，如果数据和计算耦合度过高，优先思路应该是重新划分业务模块和整合数据源，把耦合相关的劣势转换为优势，不过这一点很难，需要业务专家、技术专家、架构专家和数据专家等的一起努力。

作者回复: 👍

共 2 条评论>

44

好好写代码
2019-09-26

消息队列大部分应用的场景还是需要快速去消费数据，吞吐量比持久化优先级高。

共 1 条评论>

20

亚洲舞王.尼古拉斯赵...

2019-10-11

我有一下的几点想法：

1.性能方面的考虑：像文中说的，使用计算存储分离的设计方式，原本broker只需要在本地进行消息查找，但是现在却需要连接到另一个集群中进行查找消息，增加了网络耗时，一旦并发大，带宽占满了，性能就会明显下降。

2.成本原因：

这里的成本包括两方面：一方面是部署成本，本来只需要一个集群部署，现在我还好增加一个存储集群，增加了使用者的钱方面的成本；另一方面是使用者的学习成本，新引入一个集群，无论使用的是什么存储系统，如果想要更好的解决可能出现的问题，使用者都要去学习这个新的存储系统，增加了使用者的学习成本。

共 1 条评论>

13

DFighting

2019-10-25

读了文章才发现，我们使用Flink做ETL真的很不适合，因为Flink适合计算，存储应该另外设计，系统设计存在明显的缺陷

共 2 条评论>

6

峰

2019-12-25

本来就是数据管道结果管道还要套层管道，计算本来就很轻，分离出去一种浪费的感觉。

5

Jxin

2019-11-10

回答问题：

1.mq这一侧的计算复杂度和存储管理难度都未到做更细拆分的程度。

2.在这些mq出来前，k8s这种容器编排的方案还没有。做结构拆分势必会增加复杂度，导致入门门槛提高，进而就不易于推广。而开源项目，亲和易推广也是很重要的指标。

自己的想法：

存储层在引入k8s的pv，pvc管理后，存储的管理复杂度就可以跟业务开发说再见了。毕竟上云后，这块东西就由云服务商承接走了。这样有利于更轻量的开发，亲和拥抱变化的市场环境。所以老师说是趋势。

4

boyxie
2019-10-30

可以用在对实时性要求不高的应用，比如定时任务类型的，可以存储海量数据

2

游弋云端
2019-09-26

个人觉得从微服务的思想来讲，专业的人干专业的事，进行服务的细粒度拆分，不做大而全的系统，这样架构清晰，可扩展性强。其他消息队列适合快速部署，可以做到对外依赖少的精简配置，例如kafka要去ZK，要支持单节点配置等，这与Pulsar是不同的。在后续的微服务、容器等演进的趋势下，个人认为Pulsar会更具有竞争力。目前来看，只能说各有千秋，各有不同的应用场景。

2

leslie
2019-09-26

不一样的思路：非常喜欢老师这种方式，不仅仅是基于当下，而是持久；让我觉得不仅仅是一个User，而是在User中去扩展。

就像之前学计算机组成原理是徐老师曾经提过基于CPU的MQ：当时问徐老师为什么有了CPU还需要有GPU？老师让我去看David Patterson老爷爷的文章，包括现在阿里做的GPU减负的是内存，应当是差不多的不一样的思路吧。

感谢老师分享不一样的新东西：让我们不仅仅基于当下、研究当下，甚至可以在MQ的方向上扩展研究下去。期待老师后面的继续分享。

2

thecode
2021-11-09

这里有一篇 Pulsar 和 Kafka 性能测试对比的文章

https://www.infoq.cn/article/xeyeeenny5cg0pgyxevd
https://github.com/streamnative/openmessaging-benchmark/blob/master/blog/benchmarking-pulsar-kafka-a-more-accurate-perspective-on-pulsar-performance.pdf



1

小红帽
2023-03-15 来自广东

一方面，计算与存储分离这种思想并没有提高生产和消费性能，
另一方面，从业务上来说也很少有需要像iot这种千万级客户端通信的场景，这10年都是电子商务的各种变形的应用为主！
基于这种计算和存储的特点和业务场景，决定了市场上用得较多的还是这种成熟的计算与存储一体的架构。


wuhang202
2021-12-07

首先消息队列需要满足高并发下的低延时，hdfs为离线大数据批量计算而生，无法满足，mysql无法直接支持海量数据，因此需要消息队列定制化自己的存储。
此外，增加依赖，也增加了中间件入门、维护及使用的复杂度，作为中间件要尽可能保证最小依赖，例如rocketmq自带服务注册发现的nameserver，kafka将要移除zookeeper。


Heaven
2021-02-20

好处是,读写分离,专注传输,集群状态变更速度快,利用现成的存储系统开发效率高
坏处是,增加网络传输,降低性能,读写性能低
从上面来看,好处的确不少,但是我们现阶段对于MQ的要求主要就是性能,在性能提升不上去的情况下,好处多也没啥用,就好比是容器界的gVisor,虽然代表着一种发展方向,但是由于性能问题,应用面仍然上不去


远鹏
2020-03-28

计算存储分离加大了可控性，无论是对于开发还是运维来说都是一件好事


张天屹
2019-09-27

老师你好，请教一个本章的题外问题，对于严格的局部顺序性要求场景，比如十个分区，十个消费者线程，一一对应是可以保证的。但是绝不能一对多/批量预取，这个不设置多线程消费

就可以了，但是在进程角度的话，为了保证顺序性，一个分区也只能被一个进程消费，那就没办法做消费者集群了吗，只有一个单体的消费者服务的话，可靠性是不是不太好，如果宕机了下游就完全挂了，这种情况怎么办呢?

作者回复: 使用消息队列来保证严格顺序时，对于消费者是没有要求的。不需要单线程或者单进程，所以使用消费者集群是完全没有问题的，依然可以保证严格顺序。

消费者唯一需要保证的和接收普通消息是一样的，就是：先执行消费逻辑，再确认消费。

共 3 条评论>

Better me
2019-09-26

消息队列本身作为中间层应该以提升更好的性能为导向，但目前Pulsar的存储计算分离架构等于在中间层在分层，在架构上可以做到服务化单一职责，以及无状态的broker节点，但却有少许的性能损耗，如果以后能解决性能损耗的问题，那么存储计算分离架构必然是未来的趋势

A9
2019-09-26

是否使用存储计算分离也是一种trade off的体现。采用分离的架构，整个架构上模块更加清晰，能够降低开发成本，提升开发效率。不采用存储计算分离的架构，可能开发难度较大，不过整体效率更高。对于开发消息中间件来说，尤其是之前消息队列诞生时，计算机性能普遍性能交叉，可能牺牲一点开发的效率，获取更高的性能才是更重要的。
另外，最近流计算的流行，同样的存储计算分离架构可能有能直接进行配合（并不太了解流计算）
总结来说，算是诞生的时代背景占了相当大的原因吧

有铭
2019-09-26

消息队列的性能是个很重要的需求，既然存储计算分离了也就等于加了中间层。我们都知道加了中间层那是会损失性能的