消息队列业界调研

C-2 创建:王军飞, 最后修改: 王军飞 2021-05-25 17:35

目录

1.概况

消息队列是现代互联网企业技术中不可缺少的一个中间件，主要用来做组件解耦、流量削峰、异步处理，支持订阅发布(pub/sub)模式，在业务或数据的上下游中，起到一个链接作用。最早在2000年左右，J2EE 2.0时代，消息队列有Sun主导的JMS标准，实现这个标准的产品比较多，比如IBM websphere MQ，Apache Active MQ，RabbitMQ等。到2012年后，随着linkedin开源自建的消息队列Kafka，Apache Kafka逐渐成为引领开源社区消息队列的潮流。2016年，阿里巴巴将RocketMQ贡献给Apache，2018年后，Yahoo开始开源自己的消息队列产品Pulsar并贡献给Apache，两者都变成了apache的顶级项目。

2.主要产品

从整个互联网行业来看，Amazon云，阿里云，腾讯云，京东云以及各大知名互联网公司，都有自己的消息队列服务，产品非常多，可以参照附录A，但是开源的比较少。本文为了更准确和客观，仅分析公布了源代码的产品，一方面是因为根据源代码来做出的调研和分析得出的结论更有客观依据，另一方面是因为这些开源的产品基本上覆盖了整个行业的绝大部分用户。这些开源的产品主要包含Kafka、RocketMQ、Pulsar。下边从产品功能和技术架构两个方面，分别简单介绍下这些产品。

2.1 功能

功能分为几类，基本功能，高级功能，个性化功能和前沿功能。所谓的基本和高级功能，是作为消息队列这个产品本质来说，必须具有的基础功能点。个性化功能是指各个产品开发的具有本产品特色的功能，这些功能并不是消息队列产品必须具备的。前沿功能是指行业的领头羊正在做的活研究的功能点，他不是消息队列必须的发展方向，仅代表个体厂商的动向。

功能分类	功能项	Apache RocketMQ	Apache Pulsar	Apache Kafka	美团 Mafka
前沿功能	去除zookeeper			✅	
	Tiered storage		✅	✅	
	Stream SQL		✅	✅	
	流式计算消息事务		✅	✅	
	轻量流式计算、事件驱动架构(EDA)		✅	✅	
个性化功能	存储计算分离		✅		
	消息过滤	✅			
	多租户隔离		✅		✅
	主题鉴权				✅
	storm/flink接入SDK			✅	✅
	全链路压测				✅
	故障注入				✅
	Set化/灰度链路				✅
	环境/泳道隔离				✅
	多集群生产、消费自动熔断				✅
	Push代消费(RPC方式集中消费)				✅
高级功能	生产、消费跨集群、地域调度				✅
	优先级消息				✅
	事务消息	✅			✅

	消息轨迹	✅			✅
基本功能	定时消息	☑️	☑️		✅
	消息回溯(按时间、offset)	✅	☑️	✅	✅
	死信	✅	✅		✅
	顺序生产和消费	✅	✅	✅	✅
功能分类	功能项	Apache RocketMQ	Apache Pulsar	Apache Kafka	美团 Mafka

☑️ : 弱支持

✅ : 支持

基本功能: 四个产品都支持最基础的顺序消息生产和消费功能，Kafka不支持定时消息和死信。消息回溯方面，Pulsar只支持根据MessageId做回溯，不支持按时间和offset。定时消息方面，RocketMQ只支持按固定时间的延迟，比如1m,5m,10m等有限数量的延迟，Pulsar只支持单机内存大小的定时消息，功能弱一些，Kafka不支持定时消息，Mafka支持任意时间的的定时消息。

高级功能: RocketMQ和Mafka都支持事务消息和消息轨迹，除此之外，Mafka还支持优先级消息和跨集群、地域的生产者、消费者客户端调度。

个性化功能: Mafka支持比较多的美团公司特有的功能需求。

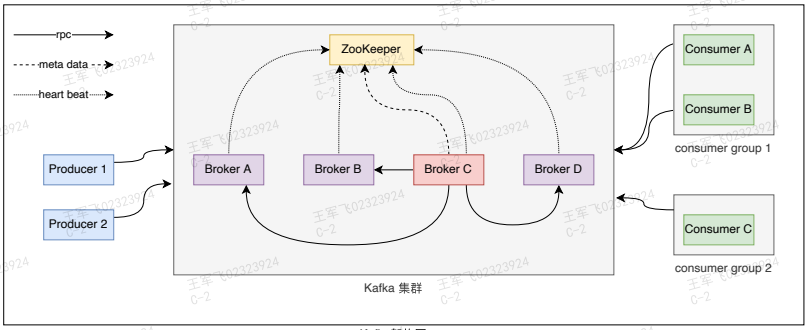
前沿功能: Pulsar和Kafka都在往轻量流式计算、云原生方面演进，特别是Kafka，在轻量流式计算方面已经发展了超过3年，功能相对成熟，Pulsar在对标追赶。云原生方面，Kafka和Pulsar都有自己的商业运营平台在做相关的扩展，比如快速扩容，按需付费，无需devops。开源社区方面，两个组件都在做分层存储，支持在廉价的存储介质上做长期存储，比如HDFS、S3、云存储等。Kafka单独在演进"KIP-500"---去除zookeeper，方便Kafka的部署和维护。

2.2 架构

2.2.1 Apache Kafka

2.2.1.1 Kafka的整体架构

Kafka集群主要包含Kafka Broker本身，以及zookeeper组件，如下图所示。



Kafka架构图

Kafka Broker: 主要存储消息数据，同时提供RPC接口，供客户端发送、拉取消息。

Kafka 集群: 上图中，BrokerA、BrokerB、BrokerC、BrokerD四台机器组成一个集群，其中Broker C被选为controller。集群controller主要用于管理集群内节点、副本、分区的上下线，以及队列的创建、删除、扩容分区等。

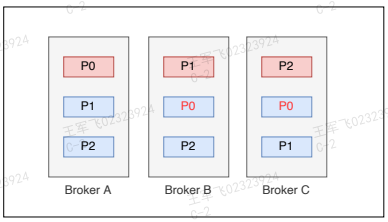
ZooKeeper: 在Kafka集群中，主要用来做集群选主、副本选主，同时存储集群元数据，如Topic、Partition、Replica等。

Consumer Group: 主题的消费组

Prodcuer：主题的生产者

2.2.1.2 分区和副本，高可用和高可靠

Kafka Broker内的分区和副本，如下图所示:



Kafka的分区和副本

1. 整个集群有3台机器组成
2. 集群内有一个topic，这个topic有3个分区，分别是P0/P1/P2
3. 每个分区有3个副本，比如分区P0的第一个副本在Broker A上，另外两个副本在Broker B和Broker C上。

分区是为了方便队列能并发的生产和消费，一个队列可以拆分成多个分区，分布在相同或不同的机器上。

副本是为了保持队列的容灾和高可用，所以每个分区可以设置一个或多个副本，每个副本必须分布在不同的机器上，以保持N-1的机器宕机可用性，但客户端在生产和消费消息时，只会在主副本上进行。

相应的消息可靠性，体现在一条消费发送给服务端的时候，需要有几个副本收到才能算生产成功。如果可靠性要求很高，那么可以设置队列的ack为-1，要求所有副本收到后才算生产成功，或设置为1，表示只要主副本收到后就算成功。

集群维度的可靠性，也可以通过设置最小同步副本(min InSync replica count)，表示必须有这么多个副本处于同步状态时，集群才可以使用，否则拒绝写入，以防止在副本不同步的情况下，发生几起宕机后，消息丢失。

2.2.1.3 存储模型

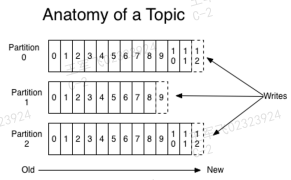
kafka的消息存储大量使用文件，所有的消息都存储在文件中，文件使用二进制编码。每个partition有一组相应的文件，在每个文件内，消息按顺序追加到文件末尾。每个消息有标准的存储格式，以字节方式写入。每个partition有多个副本，分布在不同的机器上，副本之间依靠复制来保持消息高可用。副本在拉取主本消息后，写入本机磁盘作为备份。一般每个partition会有两到三个副本，副本之间依靠zookeeper来选主，只有主副本会接受客户端的读写。

如下图所示，有三个partition，每个partition的消息都是顺序追加，老数据在文件头，新数据在文件末尾，这样做得好处是，如果生产和消费都能跟的上，相差时延不多，那么新数据在被落盘之前，已经在pagecache里被消费者拿走消费了，效率非常高。

此外，在消息写入和读取时，因为使用顺序读写文件，效率也很高，基本都是disk IO bound型操作，cpu和内存使用非常少，不会遇到java gc带来的问题。

除了消息文件之外，还有一些index索引文件，各种checkpoint文件，leader epoch等帮助文件。除了zookeeper中存储的队列元数据信息外，其他数据全部存储在磁盘上。

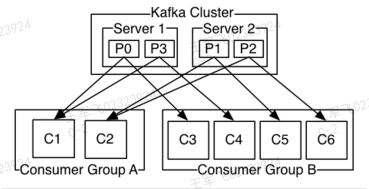
*下图引用自kafka官方文档



2.2.1.4 生产和消费模型

如上文所介绍，所有的消息都是按照partition来存储的，一个队列可以有多个partition，在partition内，消息是按顺序来写入的，partition之间的顺序依据用户的hash策略不同而不一样。多个partition有助于消息并发生产和写入，消息在消费时，也是按照partition来分配的，一个消费者可以分一个或多个partition，但是一个partition只能分给一个消费者。消费是按照组的粒度来区分的，每个消费组都能完整消费到一个队列的所有消息。每个消费组内，各个消费者之间，按照抢占方式来获取所消费的partition，抢占分配策略不同。因为partition数量和消费者数量不一定完全匹配，前者大于后者时，需要一个消费者承担多个partition的消费，相反，如果后者大于前者时，会有消费者抢占不到partition而处于闲置状态，如下图所示：

*下图引用自kafka官方文档



2.2.1.5 Kafka架构的优势和劣势

从技术架构角度来讲，kafka具有以下优势和劣势

优势:

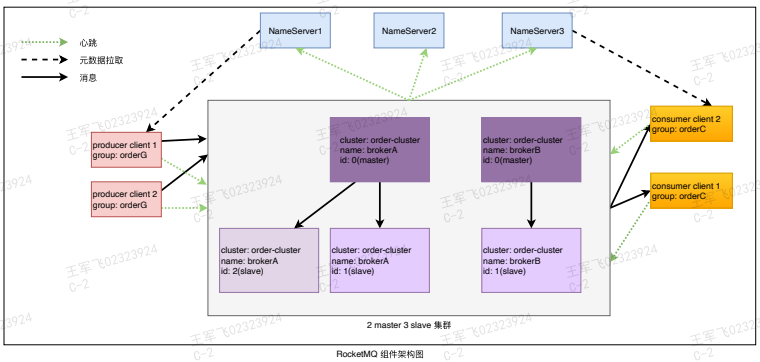
1. 吞吐高，延迟低：因为消息的读写采用了顺序文件读写，效率高，速度快，kafka的吞吐非常大，延迟也很低。
2. 可用性高，可靠性高：因为使用了分布式的方式，partition有自己的多副本，副本所在节点宕机后，依靠zookeeper一致性来选主，很快有副本顶上来作为主副本，在系统内有机器节点宕机时，可以维持n-1(n为副本数量)的可用性。如果设置为ack=-1和minInsyncIRS >=2，只有所有副本写入后才算发送成功，而且集群内一直保持有足够的副本同步，消息的可靠性很高。
3. 扩缩容维护简单：一个集群内所有机器等位，扩缩容方法简单，加入机器、或减少机器，然后开启数据再平衡即可实现。
4. 无java gc相关问题： 因为消息数据最终都是写入磁盘，在内存中没有存储和替换，对java内存使用少，没有java gc相关的问题。

劣势:

1. 系统复杂：因为采用了分布式系统多活机制，集群内主节点和其他节点的通信，集群内选主，partition leader选主，主题的上下线，消息清除等，这些节点间的rpc调用以及集群内选主等带来了复杂本地数据维护，特别是在多节点数据一致性方面，很容易产生bug。
2. 消费这和分区关系强耦合，不够灵活：一个分区只能分给一个消费者消费，要增加消费能力，增加新的消费者，就必须扩容分区数量，消费者数量决定分区多少，间接会增大集群内分区数量的消耗。
3. 集群必须附带一个zookeeper，增加维护成本：kafka将集群内一致性选主问题委托给zookeeper来处理，导致部署kafka时，必须部署一个zookeeper，多了一个组件，增加了一定的运营维护成本。

2.2.2 Apache RocketMQ

RockeMQ 主要包含两个模块 Broker和Name Server，如下图所示:



RocketMQ 组件架构图

2.2.2.1 NameServer

*以下引用自rocketmq github 文档

> NameServer是一个非常简单的Topic路由注册中心，其角色类似Dubbo中的zookeeper，支持Broker的动态注册与发现。
>
> 主要包括两个功能：
>
> 1. Broker管理，NameServer接受Broker集群的注册信息并且保存下来作为路由信息的基本数据。然后提供心跳检测机制，检查Broker是否还存活；
> 2. 路由信息管理，每个NameServer将保存关于Broker集群的整个路由信息和用于客户端查询的队列信息。
>
> 然后Producer和Consumer通过NameServer就可以知道整个Broker集群的路由信息，从而进行消息的投递和消费。
>
> NameServer通常也是集群的方式部署，各实例间相互不进行信息通讯。Broker是向每一台NameServer注册自己的路由信息，所以每一个NameServer实例上面都保存一份完整的路由信息。
>
> 当某个NameServer因某种原因下线了，Broker仍然可以向其它NameServer同步其路由信息，Producer,Consumer仍然可以动态感知Broker的路由的信息。

2.2.2.2 Broker

Broker主要负责消息的存储、投递和查询以及服务高可用保证。broker会每隔30s向集群中的所有nameserver发送一个心跳包，nameserver会每隔10s扫描自己保存的broker列表，看broker最后一次发送的心跳包是否是120s前的，如果是就删除这个broker，关闭链接。

2.2.2.3 生产端和生产端集群

Producer与NameServer集群中的其中一个节点（随机选择）建立长连接，定期从NameServer获取Topic路由信息，并向提供Topic 服务的Master建立长连接，且定时向Master发送心跳。Producer完全无状态，可集群部署。

group name相同的一组生产端，称之为生产端集群。集群内每个生产者都会给master发送心跳，所以master是掌握所有生产者信息的，在事务消息回查时，broker端可选择生产端集群中的一个，来执行回查逻辑。

2.2.2.4 消费端和消费端集群

Consumer与NameServer集群中的其中一个节点（随机选择）建立长连接，定期从NameServer获取Topic路由信息，并向提供Topic服务的Master、Slave建立长连接，且定时向Master、Slave发送心跳。Consumer既可以从Master订阅消息，也可以从Slave订阅消息，消费者在向Master拉取消息时，Master服务器会根据拉取偏移量与最大偏移量的距离（判断是否读老消息，产生读I/O），以及从服务器是否可读等

因素建议下一次是从Master还是Slave拉取。

groupname相同的消费端，称之为一个集群。集群内每个消费者都会给broker发送心跳，所以broker端也掌握了所有消费者的信息，每个消费者上线、或下线时都会来查阅这个信息，进行队列重分配。

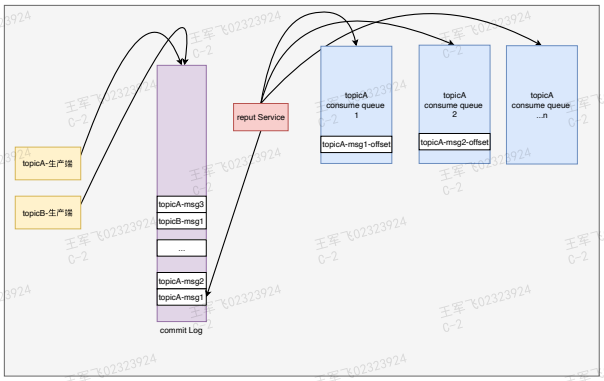
2.2.2.5 Broker集群

RocketMQ的集群比较特殊，是多个单元组成的一个集群。如上图所示，整个集群包含5台broker，两个单元，第一个单元是3台broker，一主两从，第二个单元是一主一从。集群的划分是以cluster name名称为准备，命名相同的机器都属于一个集群。如上图，所有broker的cluster name属性都叫order-cluster，他们都属于一个集群。

name相同的一组broker是一个单元，同一单元内，id属性为0的broker是master，id属性为1的为第一slave，其他都是slave.

2.2.2.6 队列存储模型

Kafka里每个topic各自的partition消息，都会写入自己的文件里。RocketMQ不一样，它把所有的topic数据全部写入一个文件里，称之为commit log。Broker接收到消息后，统一都写入一个消息日志(commit log)文件，由转发服务(reput Service)再转发生成消费队列(consume queue)文件，如下图所示:



RocketMQ commit log & consume queue

上图可以看到有两个文件：

*以下引用自rocketmq github 文档

> (1) CommitLog：消息主体以及元数据的存储主体，存储Producer端写入的消息主体内容,消息内容不是定长的。单个文件大小默认1G ，文件名长度为20位，左边补零，剩余为起始偏移量，比如00000000000000000000代表了第一个文件，起始偏移量为0，文件大小为1G=1073741824；当第一个文件写满了，第二个文件为00000000001073741824，起始偏移量为1073741824，以此类推。消息主要是顺序写入日志文件，当文件满了，写入下一个文件；
>
> (2) ConsumeQueue：消息消费队列，引入的目的主要是提高消息消费的性能，类似于kafka中的partition概念，由于RocketMQ是基于主题topic的订阅模式，消息消费是针对主题进行的，如果要遍历commitlog文件中根据topic检索消息是非常低效的。Consumer即可根据ConsumeQueue来查找待消费的消息。其中，ConsumeQueue（逻辑消费队列）作为消费消息的索引，保存了指定Topic下的队列消息在CommitLog中的起始物理偏移量offset，消息大小size和消息Tag的HashCode值。consumequeue文件可以看成是基于topic的commitlog索引文件。同样consumequeue文件采取定长设计，每一个条目共20个字节，分别为8字节的commitlog物理偏移量、4字节的消息长度、8字节tag hashcode，单个文件由30W个条目组成，可以像数组一样随机访问每一个条目，每个ConsumeQueue文件大小约5.72M；

还有一个文件是:

> IndexFile：IndexFile（索引文件）提供了一种可以通过key或时间区间来查询消息的方法。Index文件名fileName是以创建时的时间戳命名的，固定的单个IndexFile文件大小约为400M，一个IndexFile可以保存2000W个索引，IndexFile的底层存储设计为在文件系统中实现HashMap结构，故rocketmq的索引文件其底层实现为hash索引。

2.2.2.7 RocketMQ架构的优势和劣势

从架构上来看，RocketMQ具有以下优势和劣势

优势:

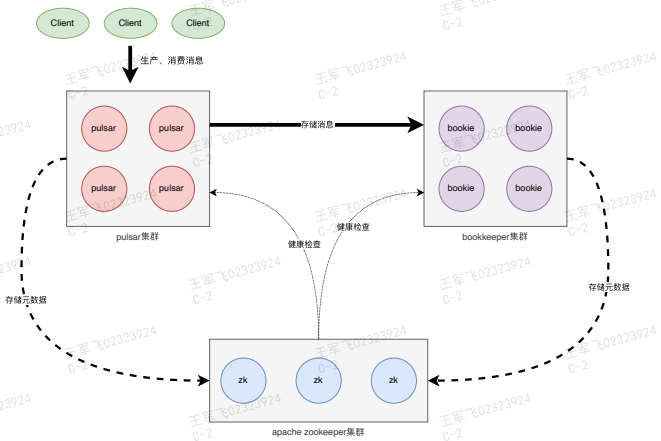
1. 模型简单：架构比kafka要简化很多，kafka是多节点组成的集群，RocketMQ简化为多组两台机器组成的主从结构集群。存储模型上，Kafka有复杂的节点controller控制节点来负责切换集群leader，分区leader，创建删除partition，replica等，容灾依靠多副本复制机制，以及高水位控制消费和副本拉取，rocketmq简化为一个commitLog和多个consume queue来实现，容灾依靠简单一对一或对多复制。
2. 结构松散，模块之间无紧耦合的关系: NameServer是无状态的，可以多台部署，每台之间角色等位，单台宕机无影响；Broker服务发现依靠自行、定期上报到NameServer上去，NameServer对broker的简况检查也是定期巡检(默认10s心跳间隔，120s剔除)，来实现添加、删除Broker实例。NameServer和Broker之间网络抖动基本无影响，相互影响力弱。这点要优于kafka，Kafka和zookeeper之间的网络抖动、broker或zookeeper发生OOM无法响应心跳时，broker的状态会发生抖动。
3. 组件单一，无依赖第三方组件: 集群内主从是固定死的，在部署都已经定义好，不需要选主操作，在部署上不需要额外部署zookeeper这样的一致性组件。

劣势:

1. 由于消息数据全部落在一个commit log文件上，消费端检索消息时不能批量获取，需要逐个检索，在队列的qps增大时，对CPU的损耗比较大。
2. 集群是有多组两台主从结构组成，在队列量数量变多、或是队列消息量变大时，因为所有机器并不像kafka一样角色等位，产生的集群扩缩容运维会比较繁琐。
3. 存储模型上，一个JVM实例仅有一个数据文件，资源利用率不高。单机部署多个实例时，会产生额外的JVM、OS资源占用。
4. 数据文件采用mmap读写，虽然效率很高，但mmap本身有诸多缺陷，比如mmap在jvm内无法显式unmap，必须等jvm内存gc或通过hack方法来回收，但是mmap在进程内文件句柄数又是有限的，如果不及时回收，可能会耗尽，mmap最大文件大小不能超过2G。

2.2.3 Apache Pulsar

Pulsar实际上是两个开源组件的组合，Pulsar集群+BookKeeper集群，下图是Pulsar的架构概览：



2.2.3.1 Pulsar架构简述:

1. Pulsar依赖一个开源项目Apache BookKeeper，使用它来做消息存储，而pulsar本身是一个无状态服务。
2. Apache BookKeeper是一个分布式的日志条目(log entry)存储服务。

3. Pulsar和bookeeper都使用zookeeper来存储自己的元数据，并在启动时往zookkeeper上注册节点，来供其他节点或客户端发现自己。
4. zookeeper同时负责监控pulsar和bookkeeper的健康状态。

由此可见，Pulsar是一个典型的“计算+存储”类型的消息队列，Pulsar本身只做消息队列层的概念抽象逻辑，真正的消息数据落地在BookKeeper中。这种架构类似于美团很早之前自研消息队列Swallow，后者使用MongoDB作为存储，前端也是做简单的消息队列抽象逻辑。

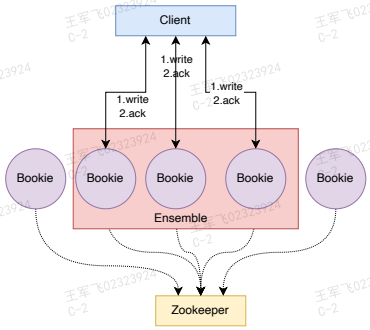
在“计算层”，Pulsar抽象出了topic(主题)，subscription(订阅)和cursor(游标)的概念。topic是消息队列，是一系列连续的消息实体，日志结构的数据，每个消息一个偏移量(offset)；subscription是消费者的订阅关系，定义了消费者消费哪个主题，是独享消费，还是和其他消费者共享消费；cursor消费的位点信息，表示消费者消费到了topic里哪个位置。

在“存储层”，BookKeeper也是Yahoo开发的，之前是作为Apache Zookeeper的一个子项目，而后在2015年孵化为Apache的正式项目，bookkeeper人和zookeeper的是开发者是同一批人。

BookKeeper是一个通用的日志(log)存储方案，它定义了几个实体，entry是日志中的最小实体，类似于消息队列中的"消息"，一部分连续的entry组成了fragement(片段)，若干个fragement组成一个ledger(账本)。pulsar在抽象消息队列时，将entry抽象为自己的消息，ledger抽象为自己的topic。BookKeeper是以集群形式工作的，集群中每台机器称之为Bookie。

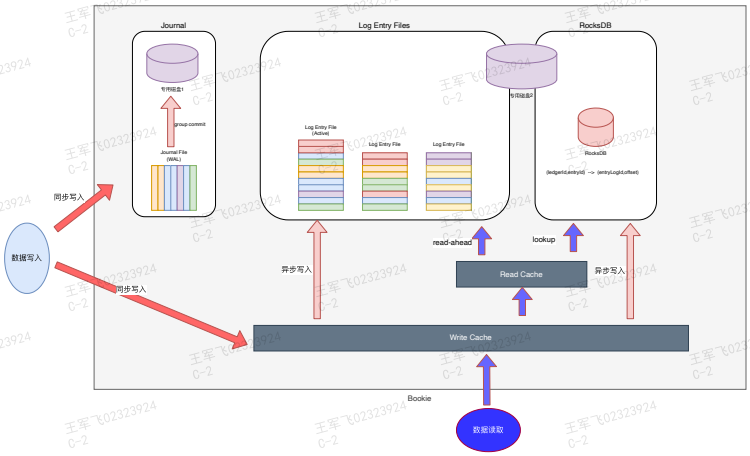
BookKeeper内没有leader或controller的概念，客户端在写入时，需要设置E(Ensemble)，E实际就是几个Bookie集合，从现有的集群中选取几个。Qw(Write Quorum)是在E这个集合内，客户端在写入数据时，需要在多少个机器上写入，为的是将数据做备份和冗余，当某台bookie宕机后保持高可用。Qa(Ack Quorum)是在Qw内，客户端需要等待多少个Bookie机器确认存储完数据后返回写入成功。如下图所示，E是3台Bookie机器，Qw是3，Qa也是3，当然为了加快写入速度，可以将Qa设置为2或1，但也会相应增加机器宕机后数据丢失的风险，因为确认写入数据的机器数变少了。

如果某个bookie宕机了，client可以迅速形成新的E集合，并且在可用的E内选取新的Qa，对于增量数据来说，恢复速度比较快。



每台Bookie在接收数据时，需要将数据真实flush到磁盘上才算写入成功，为了加快写入速度，BookKeeper也使用了类似于Zookeeper和Mysql的group commit机制，由此可见Bookeeper的高可靠是依靠数据写入多个磁盘来保障的，不同于Kafka的依靠复制来保障。

真实数据在Bookkeeper内部的保存方式要复杂很多，如下图所示：



2.2.3.2 BooKKeeper存储架构简述:

1. bookie内部使用两种类型的文件journal file和log EntryFile，还有一个RocksDB组件来存储数据。
2. 官方建议使用两块磁盘来做存储，一块磁盘专用于写journal file，另一块专门用来写log entryFile和作为rocksDB存储盘来使用。
3. RocksDB也是一个开源的KV存储组件
4. 写入数据时，需要同步先将数据写入到journal file中，然后异步线程再将数据写入log entryFile和 rocksDB，这种存储方式是一个典型WAL(write ahead logging)案例应用。
5. 在同步写入journal file时，也会同步写入一份到Write Cache里，这个cache是一个内存数据结构。在write cache内，数据会按topic分类来排序，以便以后在读取时能提升读取效率。
6. 写入write Cache后，异步线程会将真实数据写入log EntryFile(一个树形存储结构)里，同时将消息(entry)条目在log EntryFile里的位置索引记录在rocksDB里，以方便后续读取。
7. write Cache会缓存最近的写入，所以最近写入的消息读取效率会很高，如果读取稍早的数据，或数据已经不在cache里的，需要到Log EntryFile磁盘文件里去溯源。

2.2.3.3 Pulsar架构的优势和劣势

从架构来看，pulsar有以下优势和劣势

优势:

1. 因为消息可以选择ack最快的两个节点来存储，可以避免慢节点写入带来的延时影响。
2. 集群可以快速扩容，新加入的bookie节点可以很快作为Qw的一员来接收消息。
3. 由于整个集群没有leader的概念，所以不存在脑裂的风险。而kafka则会由于网络分区，形成脑裂。

劣势:

1. 架构比较复杂，整个架构包含pulsar，bookkeeper，rocksDB三个组件，数据分散在这三个组件内。
2. 数据存储模型复杂，最小存储单元fragement的Qa可以在多个bookie上，整个集群内fragement数量会巨大，而且分布琐碎和零散，维护复杂度高。而且读取时需要跳跃在多台机器上读取，效率会比较低下。
3. 强依赖zookeeper。因为ledger、fragement对应关系，存储位置等信息都存放在zookeeper上，一旦一台bookie连接不上zookeeper，为了保持一致性，bookie停止接受服务，并自动重启，这意味着一旦集群内几个节点、或全部链接不上zookeeper，或者zookkeeper挂掉，整个集群都无法再继续服务。
4. 没有顺序写入和读取的优势，在写入时需要做group commit，强制刷盘，读取时则需要根据索引在文件内随机读取，整体磁盘使用效率低。
5. bookie单点宕机后，仍然需要批量移动大量数据作为容灾副本恢复。 这点跟kafka类似，但比kafka好的是，增量数据不会有可靠性威胁，因为bookie在单点宕机后可以快速形成新的副本组，而kafka则需要通过打散来补齐副本，在此之前，无论增量数据还是历史数据都少一个副本。

2.2.3.3 Pulsar的演进规划

pulsar的商业支持公司streamnative.io，已经将pulsar搬到云上，提供消息和事件流式计算服务。规划中的发展方向包含以下几个方面：

1. Pulsar Function: 类似于kafka stream的轻量流式计算，在pulsar内部做流式计算，不需要将数据再搬运到storm/flink一类大数据组件上。

2. Pulsar IO: 类似于kafka connect，使用pulsar桥接两个系统的数据流，比如从database到应用，从database到hBase大数据存储。

3. 分层存储：类似于kafka 分层存储，将老数据搬运到低廉的存储组件上，例如S3，hadoop一类，而保持新数据在pulsar本地，降低pulsar的机器成本。

4. Helm: 类似于confluent商业公司的运营平台，通过平台来运维管理pulsar集群。

附录

附录A: 业界消息队列产品概要

公司	消息队列产品	简介	特点
Pivotal Software	RabbitMQ	比较早的一个消息队列产品，在2000年左右的J2EE时代实现了JMS消息队列规范。 背后商业公司 Pivotal Software。	• github热度： star–8.3K fork–3.1K • 使用erlang语言编写，比较偏门，组件可访问性，可控性差。 • 在国外有一定的用户量，比如T–mobile。
Apache 开源基金会	Apache Kafka	最早由Linkedink开发，在2012年捐献给Apache基金会. 背后商业公司confluent，提供商业的kafka集群服务。	• github热度： star–18.6K fork–9.9K • 吞吐大，低延时，常用于消息队列，大数据离线计算内部数据桥，轻量流式在线计算框架等。 • 社区很大，比较流行。最初只专注于消息队列领域，后来发展为流式计算平台(distributed event streaming platform)。 • 用户量比较大，有数千家，号称世界财富100强的80%公司都在使用，客户领域比较广，包括银行，金融，IT，零食，保险，制造，通信，运输和能源等。 • 使用scala 和 java语言编写，有一定的学习成本。
	Apache RocketMQ	2016年，阿里巴巴将RocketMQ贡献给Apache 背后商业公司阿里云，提供商业RocketMQ云服务。	• github热度： star–14K fork–7.7K • 阿里巴巴参照kafka模型，根据自身公司特点改写的消息队列。 • 国内社区比较活跃，阿里巴巴经常举办线下活动布道，给与中小型公司予以支持，并引申到自己的阿里云消息队列上去。 • 国内用户量大，热度高 • 使用纯java语言编写
	Apache Pulsar	Yahoo在2018年贡献给Apache基金会。 背后商业公司streamnative.io，提供商业的pulsar集群服务。	• github热度： star–7.6K fork–1.9K • Purlsar实际上是两个开源产品的组合，Purlsar和Apache Bookkeeper。Bookkeeper是一个流式日志数据存储，Purlsar在它之上建立了消息队列产品，将消息内容存储在其中，自身则是一个无状态服务，只负责管理生产和消费的路由。 • 相对于kafka和rocketmq，社区比较小。有一定的用户量，主要是国内用户，腾讯计费平台、Yahoo日本、智联招聘等。
阿里云	RocketMQ	引用自阿里云： > 消息队列 RocketMQ 版（原ONS）是阿里云基于 Apache RocketMQ 构建的低延迟、高并发、高可用、高可靠的分布式消息中间件。该产品最初由阿里巴巴自研并捐赠给 Apache 基金会，服务于阿里集团 13 年，覆盖全集团所有业务。作为双十一交易核心链路的官方指定产品，支撑千万级并发、万亿级数据洪峰，历年刷新全球最大的交易消息流转记录。	• 同Apache RocketMQ
	Kafka	引用自阿里云： > 消息队列 Kafka 版是阿里云基于 Apache Kafka 构建的高吞吐量、高可扩展性的分布式消息队列服务，广泛用于日志收集、监控数据聚合、流式数据处理、在线和离线分析等，是大数据生态中不可或缺的产品之一，阿里云提供全托管服务，用户无需部署运维，更专业、更可靠、更安全。	• 同Apache Kafka
	RabbitMQ	引用自阿里云： > 消息队列 RabbitMQ 版支持 AMQP 协议，完全兼容 RabbitMQ 开源生态以及多语言客户端，打造分布式、高吞吐、低延迟、高可扩展的云消息服务。开箱即用，用户无需部署免运维，轻松实现快速上云，阿里云提供全托管服务，更专业、更可靠、更安全。	• 同RabbitMQ
	MQTT	引用自阿里云： > 微消息队列 MQTT 版是专为移动互联网(MI)、物联网(IoT)领域设计的消息产品，覆盖互动直播、金融支付、智能餐饮、即时聊天、移动 Apps、智能设备、车联网等多种应用场景；通过对 MQTT、WebSocket 等协议的全面支持，连接端和云之间的双向通信，实现 C2C、C2B、B2C 等业务场景之间的消息通信，可支撑千万级设备与消息并发。	• 未开源，社区热度、技术架构和细节不祥
	MNS	引用自阿里云： > 阿里云消息服务 MNS 是一种高效、可靠、安全、便捷、可弹性扩展的分布式消息服务。MNS能够帮助应用开发者在他们应用的分布式组件上自由的传递数据、通知消息，构建松耦合系统。	• 未开源，社区热度、技术架构和细节不祥
腾讯云	TDMQ	引用自腾讯云： > 消息队列 TDMQ（Tencent Distributed Message Queue，简称 TDMQ）基于 Apache 开源项目 Pulsar 自研的金融级分布式消息中间件，是一款具备高一致、高可靠、高并发的消息队列。TDMQ 拥有原生 Java 、 C++、Python、GO 多种 API， 同时支持 Kafka 协议以及 HTTP Proxy 方式接入，可为分布式应用	• 未开源，社区热度、技术架构和细节不祥

		系统提供异步解耦和削峰填谷的能力，同时也具备互联网应用所需的海量消息堆积、高吞吐、可靠重试等特性。详细说明请参考产品概述。	
	CMQ	引用自腾讯云： 消息队列（Cloud Message Queue，CMQ）是一种分布式消息队列服务，它能够提供可靠的基于消息的异步通信机制，能够将分布式部署的不同应用（或同一应用的不同组件）之间的收发消息，存储在可靠有效的 CMQ 队列中，防止消息丢失。 CMQ 支持多进程同时读写，收发互不干扰，无需各应用或组件始终处于运行状态。	• 未开源，社区热度、技术架构和细节不祥
	CKafka	引用自腾讯云： 消息队列 CKafka（Cloud Kafka）是一个分布式、高吞吐量、高可扩展性的消息系统，100%兼容开源 Kafka API 0.9.0至2.4.2版本。CKafka 基于发布/订阅模式，通过消息解耦，使生产者和消费者异步交互，无需彼此等待。CKafka 具有数据压缩、同时支持离线和实时数据处理等优点，适用于日志压缩收集、监控数据聚合等场景。	• 未开源，社区热度、技术架构和细节不祥
京东	JCQ	引用自京东云： 京东云消息队列（JD Cloud Message Queue，简称JCQ）是京东云自主研发的分布式消息队列服务。产品能够提供消息发布订阅、消息查询和死信队列等一系列高可靠、高可用、高处理性能的消息云服务，是云架构中不可或缺的核心产品。	• 未开源，社区热度、技术架构和细节不祥
	JMQ	引用自网络*[1]： JMQ 是京东自主研发的一款消息中间件系统，具有高可用、数据高可靠等特性。广泛应用于公司内部系统，包括订单、支付、库房等场景。	• 存储模型: WAL文件+索引文件 • 消费模型: 按分区顺序消费，共用一个索引文件存储消费位点
	消息队列Kafka	引用自京东云： 消息队列Kafka版是基于Apache Kafka的分布式发布订阅消息队列服务，广泛应用于日志收集、流式数据处理、在线和离线分析等场景，致力于为用户提供分布式、高吞吐、可扩展的全托管服务	• 未开源，社区热度、技术架构和细节不祥
Amazon	SQS	亚马逊云标准的消息队列服务	• 未开源，社区热度、技术架构和细节不祥

*[1] 2016年，丁俊，《京东消息中间件 JMQ：架构，与 Kafka 的对比，主要特性和应用场景》