



Alvin家鸡鸭鱼的小米米
码龄7年 暂无认证

27
原创

18万+
周排名

7万+
总排名

14万+
访问

等级

1339
积分

81
粉丝

75
获赞

63
评论

372
收藏

私信

关注

搜博文文章

Q

热门文章

Kafka史上最详细原理总结 48514


浅析Tomcat、JBoss、WebSphere、WebLogic、Apache 20134

Google Custom Search API使用详解 14442

每秒处理10万订单的支付架构 7012

关于高并发支付、秒杀的一些设计思路 5417

分类专栏

 druid

4篇

最新评论

Kafka史上最详细原理总结

Tisfy: 这让我想起了先贤的一句话: 尔曹身与名俱灭, 不废江河万古流。

Kafka史上最详细原理总结

weixin_39564277: 说说哪里错了呀?

Kafka史上最详细原理总结

楼下小鲤鱼: 我再更新一下, 不是小错误, 关键的好几个知识点都错了, ack和offset ...

Kafka史上最详细原理总结

楼下小鲤鱼: 写的不错, 不过我发现了几个小错误, 瑕不掩瑜

Kafka史上最详细原理总结

楼下小鲤鱼: nice

最新文章

Druid大数据实时处理的开源分布式系统——Coordinator

Druid大数据实时处理的开源分布式系统——Broker

Druid大数据实时处理的开源分布式系统——Historical Node

2017年 6篇

2015年 25篇

2014年 5篇

scala

Kafka史上最详细原理总结

原创 Alvin家鸡鸭鱼的小米米 2015-08-31 23:04:51 48540 收藏 331 版权

文章标签: kafka apache scala 分布式

Kafka

Kafka是最初由Linkedin公司开发, 是一个分布式、支持分区的 (partition)、多副本的 (replica), 基于zookeeper协调的分布式消息系统, 它的最大的特性就是可以实时的处理大量数据以满足各种需求场景: 比如基于hadoop的批处理系统、低延迟的实时系统、storm/spark流式处理引擎, web/nginx日志、访问日志, 消息服务等等, 用scala语言编写, Linkedln于2010年贡献给了Apache基金会并成为顶级开源 项目。

1.前言

消息队列的性能好坏, 其文件存储机制设计是衡量一个消息队列服务技术水平和最关键指标之一。下面将从Kafka文件存储机制和物理结构角度, 分析Kafka是如何实现高效文件存储, 及实际应用效果。

- 1.1 Kafka的特性:
- 高吞吐量、低延迟: kafka每秒可以处理几十万条消息, 它的延迟最低只有几毫秒, 每个topic可以分多个partition, consumer group 对partition进行consume操作。
 - 可扩展性: kafka集群支持热扩展
 - 持久性、可靠性: 消息被持久化到本地磁盘, 并且支持数据备份防止数据丢失
 - 容错性: 允许集群中节点失败 (若副本数量为n,则允许n-1个节点失败)
 - 高并发: 支持数千个客户端同时读写

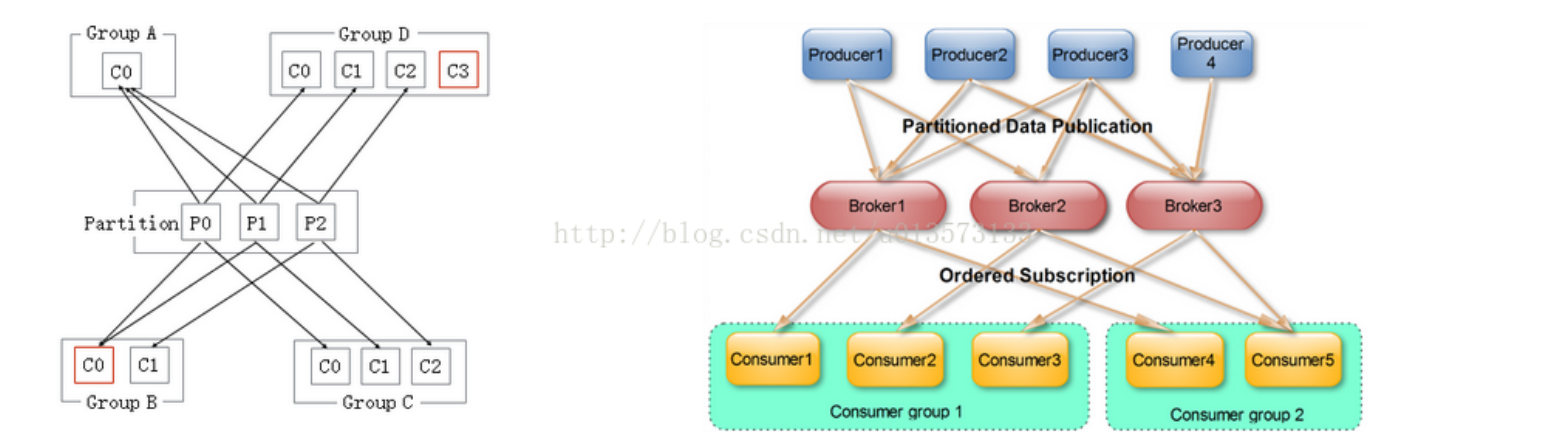
- 1.2 Kafka的使用场景:
- 日志收集: 一个公司可以用Kafka可以收集各种服务的log, 通过kafka以统一接口服务的方式开放给各种consumer, 例如hadoop、Hbase、Solr等。
 - 消息系统: 解耦和生产者和消费者、缓存消息等。
 - 用户活动跟踪: Kafka经常被用来记录web用户或者app用户的各种活动, 如浏览网页、搜索、点击等活动, 这些活动信息被各个服务器发布到kafka的topic中, 然后订阅者通过订阅这些topic来做实时的监控分析, 或者装载到hadoop、数据仓库中做离线分析和挖掘。
 - 运营指标: Kafka也经常用来记录运营监控数据。包括收集各种分布式应用的数据, 生产各种操作的集中反馈, 比如报警和报告。
 - 流式处理: 比如spark streaming和storm
 - 事件源

- 1.3 Kafka的设计思想
- **Kafka Broker Leader的选举:** Kafka Broker集群受Zookeeper管理。所有的Kafka Broker节点一起去Zookeeper上注册一个临时节点, 因为只有一个Kafka Broker会注册成功, 其他的都会失败, 所以这个成功在Zookeeper上注册临时节点的这个Kafka Broker会成为Kafka Broker Controller, 其他的Kafka broker叫Kafka Broker follower。 (这个过程叫Controller在ZooKeeper注册Watch)。这个Controller会监听其他的Kafka Broker的所有信息, 如果这个kafka broker controller宕机了, 在zookeeper上面的那个临时节点就会消失, 此时所有的kafka broker又会一起去 Zookeeper上注册一个临时节点, 因为只有一个Kafka Broker会注册成功, 其他的都会失败, 所以这个成功在Zookeeper上注册临时节点的这个Kafka Broker会成为Kafka Broker Controller, 其他的Kafka broker叫Kafka Broker follower。 例如: 一旦有一个broker宕机了, 这个kafka broker controller会读取该宕机broker上所有的partition在zookeeper上的状态, 并选取ISR列表中的一个replica作为partition leader (如果ISR列表中的replica全挂, 选一个幸存的replica作为leader; 如果该partition的所有的replica都宕机了, 则将新的leader设置为-1, 等待恢复, 等待ISR中的任一Replica"活"过来, 并且选它作为Leader; 或选择第一个"活"过来的Replica (不一定是ISR中的) 作为Leader), 这个broker宕机的事情, kafka controller也会通知zookeeper, zookeeper就会通知其他的kafka broker。**这里曾经发生过一个bug, TalkingData使用Kafka0.8.1的时候, kafka controller在Zookeeper上注册成功后, 它和Zookeeper通信的timeout时间是6s, 也就是如果kafka controller如果有6s中没有和Zookeeper做心跳, 那么Zookeeper就认为这个kafka controller已经死了, 就会在Zookeeper上把这个临时节点删掉, 那么其他Kafka就会认为controller已经没了, 就会再次抢着注册临时节点, 注册成功的那个kafka broker成为controller, 然后, 之前的那个kafka controller就需要各种shut down去关闭各种节点和事件的监听。但是当kafka的读写流量都非常巨大的时候, TalkingData的一个bug是, 由于网络等原因, kafka controller和Zookeeper有6s中没有通信, 于是重新选举出了一个新的kafka controller, 但是原来的controller在shut down的时候总是不成功, 这个时候producer进来的message由于Kafka集群中存在两个kafka controller而无法落地。导致数据淤积。**这里曾经还有一个bug, TalkingData使用Kafka0.8.1的时候, 当ack=0的时候, 表示producer发送出去message, 只要对应的kafka broker topic partition leader接收到的这条message, producer就返回成功, 不管partition leader 是否真的成功把message真正存到kafka, 当ack=1的时候, 表示producer发送出去message, 同步的把message存到对应topic的partition的leader上, 然后producer就返回成功, partition leader异步的把message同步到其他partition replica上。当ack=all或-1, 表示producer发送出去message, 同步的把message存到对应topic的partition的leader和对应的replica上之后, 才返回成功。但是如果某个kafka controller 切换的时候, 会导致partition leader的切换 (老的kafka controller上面的partition leader会选举到其他的kafka broker上) ,但是这样就会导致丢数据。

- **Consumer group:** 各个consumer (consumer 线程) 可以组成一个组 (Consumer group), partition中的每个message只能被 组 (Consumer group) 中的一个consumer (consumer 线程) 消费, 如果一个message可以被多个consumer (consumer 线程) 消费的话, 那么这些consumer必须要在不同的组。Kafka不支持一个partition中的message由两个或两个以上的同一个consumer group下的consumer thread来处理, 除非再启动一个新的consumer group, 所以如果想同时对一个topic做消费的话, 启动多个consumer group就可以了, 但是要注意的是, 这里的多个consumer的消费都必须顺序读取partition里面的message, 新启动的consumer默认从partition队列最头端最新的地方开始阻塞的读message, 它不能像AMQ那样可以多个BET作为consumer去互斥的 (for update悲观锁) 并发处理message, 这是因为多个BET去消费一个Queue中的数据的时候, 由于要保证不能多个线程拿同一条message, 所以就需要行级别悲观锁 (for update) ,这就导致了consume的性能下降, 吞吐量不够。而kafka为了保证吞吐量, 只允许同一个consumer group下的一个consumer线程去访问一个partition。如果觉得效率不高的时候, 可以加partition的数量来横向扩展, 那么再加新的consumer thread去消费。如果想多个不同的业务都需要这个topic的数据, 起多个consumer group就好了, 大家都是顺序的读取message, offsite的值互不影响。这样没有锁竞争, 充分发挥了横向的扩展性, 吞吐量极高。这也就形成了分布式消费的概念。
- 当启动一个consumer group去消费一个topic的时候, 无论topic里面有多少个partition, 无论我们consumer group里面配置了多少个consumer thread, 这个consumer group下面的所有consumer thread一定会消费全部的partition; 即便这个consumer group下只有一个consumer thread, 那么这个consumer thread也会去消费所有的partition。因此, 最优的设计就是, consumer group下的consumer thread的数量等于partition数量, 这样效率是最高的。
- 同一partition的一条message只能被同一个Consumer Group内的一个Consumer消费。不能够一个consumer group的多个consumer同时消费一个partition。
- 一个consumer group下, 无论有多少个consumer, 这个consumer group一定回去把这个topic下所有的partition都消费了。当consumer group里面的consumer数量小于这个topic下的partition数量的时候, 如下图groupA,groupB, 就会出现一个consumer thread消费多个partition的情况, 总之是这个topic下的partition都会被消费。如果 consumer group里面的consumer数量等于这个topic下的partition数量的时候, 如下图groupC, 此时效率是最高的, 每个partition都有一个consumer thread去消费。当 consumer group里面的consumer数量大于这个topic下的partition数量的时候, 如下图GroupD, 就会有有一个consumer thread空闲。因此, 我们在设定consumer group的时候, 只需要指明里面有几个consumer数量即可, 无需指定对应的消费partition序号, consumer会自动进行rebalance。
- 多个Consumer Group下的consumer可以消费同一条message, 但是这种消费也是以o (1) 的方式顺序的读取message去消费,, 所以一定会重复消费这批message的, 不能向AMQ那样多个BET作为consumer消费 (对message加锁, 消费的时候不能重复消费message)
- **Consumer Rebalance的触发条件:** (1) Consumer增加或删除会触发 Consumer Group的Rebalance (2) Broker的增加或者减少都会触发 Consumer Rebalance
- **Consumer:** Consumer处理partition里面的message的时候是o (1) 顺序读取的。所以必须维护着上一次读到哪里offsite信息。high level API,offset存于Zookeeper中, low level API的offset由自己维护。一般来说都是使用high level api的。Consumer的delivery gurantee, 默认是读完message先commit再处理message, autocommit默认是true, 这时候先commit就会更新offsite+1, 一旦处理失败, offsite已经+1, 这个时候就会丢message; 也可以配置成读完消息处理再commit, 这种情况下consumer端的响应就会比较慢的, 需要等处理完才行。
- 一般情况下, 一定是一个consumer group处理一个topic的message。Best Practice是这个consumer group里面consumer的数量等于topic里面partition的数量, 这样效率是最高的, 一个consumer thread处理一个partition。如果这个consumer group里面consumer的数量小于topic里面partition的数量, 就会有consumer thread同时处理多个partition (这个是kafka自动的机制, 我们不用指定), 但是总之这个topic里面的所有partition都会被处理到的。。如果这个consumer group里面consumer的数量大于topic里面partition的数量, 多出的consumer thread就会闲着啥也不干, 剩下的是一个consumer thread处理一个partition, 这就造成了资源的浪费, 因为一个partition不可能被两个consumer thread去处理。 **所以我们线上的分布式多个service服务, 每个service里面的kafka consumer数量都小于对应的topic的partition数量, 但是所有服务的consumer数量只和等于partition的数量, 这是因为分布式service服务的所有consumer都来自一个consumer group, 如果来自不同的consumer aroud就会处理重复的message了 (同一个consumer aroud下的consumer不能处理同一个**

partition，不同的consumer group可以处理同一个topic，那么都是顺序处理message，一定会处理重复的。一般这种情况都是两个不同的业务逻辑，才会启动两个consumer group来处理一个topic）。

如果producer的流量增大，当前的topic的partition数量=consumer数量，这时候的应对方式就是很想扩展：增加topic下的partition，同时增加这个consumer group下的consumer。



- **Delivery Mode**：Kafka producer 发送message不用维护message的offset信息，因为这个时候，offset就相当于一个自增id，producer就尽管发送message就好了。而且Kafka与AMQ不同，AMQ大都用在处理业务逻辑上，而Kafka大都是日志，所以Kafka的producer一般都是大批量的batch发送message，向这个topic一次性发送一大批message，load balance到一个partition上，一起插进去，offset作为自增id自己增加就好。但是Consumer端是需要维护这个partition当前消费到哪个message的offset信息的，这个offset信息，high level api是维护在Zookeeper上，low level api是自己的程序维护。（Kafka管理界面上只能显示high level api的consumer部分，因为low level api的partition offset信息是程序自己维护，kafka是不知道的，无法在管理界面上展示）当使用high level api的时候，先拿message处理，再定时自动commit offset+1（也可以改成手动），并且kafka处理message是没有锁操作的。因此如果处理message失败，此时还没有commit offset+1，当consumer thread重启后会重复消费这个message。但是作为高吞吐量高并发的实时处理系统，at least once的情况下，至少一次会被处理到，是可以容忍的。如果无法容忍，就得使用low level api来自己程序维护这个offset信息，那么想什么时候commit offset+1就自己搞定了。

- **Topic & Partition**：Topic相当于传统消息系统MQ中的一个队列queue，producer端发送的message必须指定是发送到哪个topic，但是不需要指定topic下的哪个partition，因为kafka会把收到的message进行load balance，均匀的分布在这个topic下的不同的partition上（hash(message) % [broker数量]）。物理上存储上，这个topic会分成一个或多个partition，每个partition相当于是一个子queue。在物理结构上，每个partition对应一个物理的目录（文件夹），文件夹命名是[topicname]_[partition]_[序号]，一个topic可以有无数多的partition，根据业务需求和数据量来设置。在kafka配置文件中可随时更高num.partitions参数来配置更改topic的partition数量，在 创建Topic时通过参数指定partition数量。Topic创建之后通过Kafka提供的工具也可以修改partition数量。

一般来说，（1）一个Topic的Partition数量大于等于Broker的数量，可以提高吞吐率。（2）同一个Partition的Replica尽量分散到不同的机器，高可用。

当add a new partition的时候，partition里面的message不会重新进行分配，原来的partition里面的message数据不会变，新加的这个partition刚开始是空的，随后进入这个topic的message就会重新参与所有partition的load balance

- **Partition Replica**：每个partition可以在其他的kafka broker节点上存副本，以便某个kafka broker节点宕机不会影响这个kafka集群。存replica副本的方式是按照kafka broker的顺序存。例如有5个kafka broker节点，某个topic有3个partition，每个partition存2个副本，那么partition1存broker1,broker2，partition2存broker2,broker3。。。以此类推（**replica副本数目不能大于kafka broker节点的数目，否则报错。这里的replica数其实就是partition的副本总数，其中包括一个leader，其他的就是copy副本**）。这样如果某个broker宕机，其实整个kafka内数据依然是完整的。但是，replica副本数越高，系统虽然越稳定，但是回来带资源和性能上的下降；replica副本少的话，也会造成系统丢数据的风险。

（1）怎样传送消息：producer先把message发送到partition leader，再由leader发送给其他partition follower。（如果让producer发送给每个replica那就太慢了）

（2）在向Producer发送ACK前需要保证有多少个Replica已经收到该消息：根据ack配的个数而定

（3）怎样处理某个Replica不工作的情况：如果这个部工作的partition replica不在ack列表中，就是producer在发送消息到partition leader上，partition leader向partition follower发送message没有响应而已，这个不会影响整个系统，也不会有什么问題。如果这个不工作的partition replica在ack列表中的话，producer发送的message的时候会等待这个不工作的partition replica写message成功，但是会等到time out，然后返回失败因为某个ack列表中的partition replica没有响应，此时kafka会自动的把这个部工作的partition replica从ack列表中移除，以后的producer发送message的时候就不会有这个ack列表下的这个部工作的partition replica了。

（4）怎样处理Failed Replica恢复回来的情况：如果这个partition replica之前不在ack列表中，那么启动后重新受Zookeeper管理即可，之后producer发送message的时候，partition leader会继续发送message到这个partition follower上。如果这个partition replica之前在ack列表中，此时重启后，需要把这个partition replica再手动加到ack列表中。（ack列表是手动添加的，出现某个部工作的partition replica的时候自动从ack列表中移除的）

- **Partition leader与follower**：partition也有leader和follower之分。leader是主partition，producer写kafka的时候先写partition leader，再由partition leader push给其他的partition follower。partition leader与follower的信息受Zookeeper控制，一旦partition leader所在的broker节点宕机，zookeeper会冲其他的broker的partition follower上选择follower变为partition leader。

- **Topic分配partition和partition replica的算法**：（1）将Broker（size=n）和待分配的Partition排序。（2）将第i个Partition分配到第（i%n）个Broker上。（3）将第i个Partition的第j个Replica分配到第（（i+j）% n）个Broker上

- 消息投递可靠性

一个消息如何算投递成功，Kafka提供了三种模式：

- 第一种是啥都不管，发送出去就当作成功，这种情况当然不能保证消息成功投递到broker；
- 第二种是Master-Slave模型，只有当Master和所有Slave都接收到消息时，才算投递成功，这种模型提供了最高的投递可靠性，但是损伤了性能；
- 第三种模型，即只要Master确认收到消息就算投递成功；实际使用时，根据应用特性选择，绝大多数情况下都会中和可靠性和性能选择第三种模型

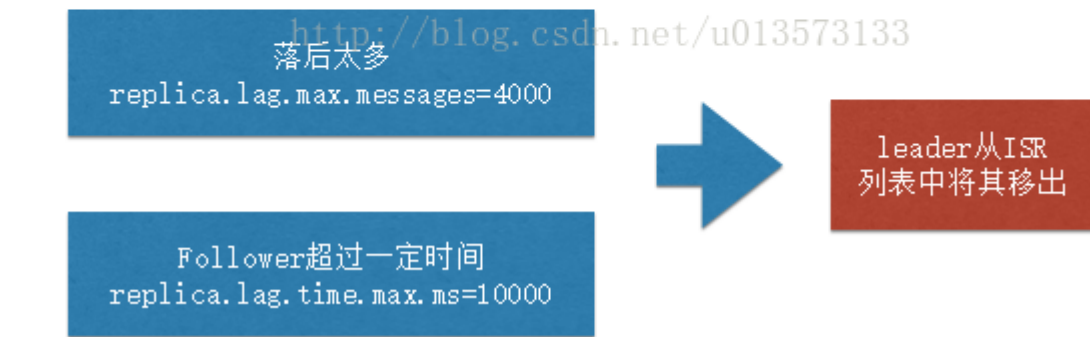
消息在broker上的可靠性，因为消息会持久化到磁盘上，所以如果正常stop一个broker，其上的数据不会丢失；但是如果不正常stop，可能会使存在页面缓存来不及写入磁盘的消息丢失，这可以通过配置flush页面缓存的周期、阈值缓解，但是同样会频繁的写磁盘会影响性能，又是一个选择题，根据实际情况配置。

消息消费的可靠性，Kafka提供的是“**At least once**”模型，因为消息的读取进度由offset提供，offset可以由消费者自己维护也可以维护在zookeeper里，但是当消息消费后consumer挂掉，offset没有即时写回，就有可能发生重复读的情况，这种情况同样可以通过调整commit offset周期、阈值缓解，甚至消费者自己把消费和commit offset做成一个事务解决，但是如果你的应用不在乎重复消费，那就干脆不要解决，以换取最大的性能。

- **Partition ack**：当ack=1，表示producer写partition leader成功后，broker就返回成功，无论其他的partition follower是否写成功。当ack=2，表示producer写partition leader和其他一个follower成功的时候，broker就返回成功，无论其他的partition follower是否写成功。当ack=-1 [partition的数量]的时候，表示只有producer全部写成功的时候，才算成功，kafka broker才返回成功信息。**这里需要注意的是，如果ack=1的时候，一旦有个broker宕机导致partition的follower和leader切换，会导致丢数据。**

ACK前需要保证有多少个备份

- **ISR(in-sync Replica)**: Leader中记录的与其保持同步的Replica列表



produce请求被认为完成时的确认值
request.required.acks=0

- **0**:不等待broker的确认信息 最小延迟
- **1**:leader已经接收了数据的确认信息,Replica异步拉取消息,比较折衷
- **-1**:ISR列表中的所有Replica都返回确认消息

min.insync.replicas=1至少有1个Replica返回成功,否则product异常.

- **message状态**：在Kafka中，消息的状态被保存在consumer中，broker不会关心哪个消息被消费了被谁消费了，只记录一个offset值（指向partition中下一个要被消费的消息位置），这就意味着如果consumer处理不好的话，broker上的一个消息可能会被消费多次。

- **message持久化**：Kafka中会把消息持久化到本地文件系统中，并且保持o(1)极高的效率。我们众所周知IO读取是非常耗资源的性能也是最慢的，这就是为了数据库的瓶颈经常在IO上，需要换SSD硬盘的原因。但是Kafka作为吞吐量极高的MQ，却可以非常高效的message持久化到文件。这是因为Kafka是顺序写入o（1）的时间复杂度，速度非常快。也是高吞吐量的原因。由于message的写入持久化是顺序写入的，因此message在被消费的时候也是按顺序被消费的，保证partition的message是顺序消费的。一般的机器,单机每秒100k条数据。

(1条消息) Kafka史上最详细原理总结_我是Alvin家鸡鸭鱼的小米米-CSDN博客_cwh代表什么

- **Producer** : Producer向Topic发送message，不需要指定partition，直接发送就好了。kafka通过partition ack来控制是否发送成功并把信息返回给producer，producer可以有任何多的thread，这些kafka服务器端是不care的。Producer端的delivery guarantee默认是At least once的。也可以设置Producer异步发送实现At most once。Producer可以用主键幂等性实现Exactly once
- **Kafka高吞吐量** : Kafka的高吞吐量体现在读写上，分布式并发的读和写都非常快，写的性能体现在以o(1)的时间复杂度进行顺序写入。读的性能 体现在以o(1)的时间复杂度进行顺序读取，对topic进行partition分区，consume group中的consume线程可以以很高性能进行顺序读。
- Kafka delivery guarantee(message传送保证): （1）At most once消息可能会丢，绝对不会重复传输；（2）At least once消息绝对不会丢，但是可能会重复传输；（3）Exactly once每条信息肯定会被传输一次且仅传输一次，这是用户想要的。
- **批量发送**: Kafka支持以消息集合为单位进行批量发送，以提高push效率。
- **push-and-pull** : Kafka中的Producer和consumer采用的是push-and-pull模式，即Producer只管向broker push消息，consumer只管从broker pull消息，两者对消息的生产和消费是异步的。
- **Kafka集群中broker之间的关系**: 不是主从关系，各个broker在集群中地位一样，我们可以随意的增加或删除任何一个broker节点。
- **负载均衡方面**: Kafka提供了一个 metadata API来管理broker之间的负载（对Kafka0.8.x而言，对于0.7.x主要靠zookeeper来实现负载均衡）。
- **同步异步**: Producer采用异步push方式，极大提高Kafka系统的吞吐率（可以通过参数控制是采用同步还是异步方式）。
- **分区机制partition**: Kafka的broker端支持消息分区partition，Producer可以决定把消息发到哪个partition，在一个 partition 中message的顺序就是Producer发送消息的顺序，一个topic可以有多个partition，具体partition的数量是可配置的。partition的概念使得kafka作为MQ可以横向扩展，吞吐量巨大。partition可以设置replica副本，replica副本存在不同的kafka broker节点上，第一个partition是leader,其他的是follower，message先写到partition leader上，再由partition leader push到partition follower上。所以说kafka可以水平扩展，也就是扩展partition。
- **离线数据装载**: Kafka由于对可扩展的数据持久化的支持，它也非常适合向Hadoop或者数据仓库中进行数据装载。
- **实时数据与离线数据**: kafka既支持离线数据也支持实时数据，因为kafka的message持久化到文件，并可以设置有效期，因此可以把kafka作为一个高效的存储来使用，可以作为离线数据供后面的分析。当然作为分布式实时消息系统，大多数情况下还是用于实时的数据处理的，但是当cosumer消费能力下降的时候可以通过message的持久化在淤积数据在kafka。
- **插件支持**: 现在不少活跃的社区已经开发出不少插件来拓展Kafka的功能，如用来配合Storm、Hadoop、flume相关的插件。
- **解耦**: 相当于一个MQ，使得Producer和Consumer之间异步的操作，系统之间解耦
- **冗余**: replica有多个副本，保证一个broker node宕机后不会影响整个服务
- **扩展性**: broker节点可以水平扩展，partition也可以水平增加，partition replica也可以水平增加
- **峰值**: 在访问量剧增的情况下，kafka水平扩展,应用仍然需要继续发挥作用
- **可恢复性**: 系统的一部分组件失效时，由于有partition的replica副本，不会影响到整个系统。
- **顺序保证性**: 由于kafka的producer的写message与consumer去读message都是顺序的读写，保证了高效的性能。
- **缓冲**: 由于producer那面可能业务很简单，而后端consumer业务会很复杂并有数据库的操作，因此肯定是producer会比consumer处理速度快，如果没有kafka，producer直接调用consumer，那么就会造成整个系统的处理速度慢，加一层kafka作为MQ，可以起到缓冲的作用。
- **异步通信**: 作为MQ，Producer与Consumer异步通信

2.Kafka文件存储机制

2.1 Kafka部分名词解释如下：

Kafka中发布订阅的对象是topic。我们可以为每类数据创建一个topic，把向topic发布消息的客户端称作producer，从topic订阅消息的客户端称作consumer。Producers和consumers可以同时从多个topic读写数据。一个kafka集群由一个或多个broker服务器组成，它负责持久化和备份具体的kafka消息。

- **Broker**: Kafka节点，一个Kafka节点就是一个broker，多个broker可以组成一个Kafka集群。
- **Topic**: 一类消息，消息存放的目录即主题，例如page view日志、click日志等都可以以topic的形式存在，Kafka集群能够同时负责多个topic的分发。
- **Partition**: topic物理上的分组，一个topic可以分为多个partition，每个partition是一个有序的队列
- **Segment**: partition物理上由多个segment组成，每个Segment存着message信息

- **Producer** : 生产message发送到topic

- **Consumer** : 订阅topic消费message，consumer作为一个线程来消费

- **Consumer Group**: 一个Consumer Group包含多个consumer, 这个是预先在配置文件中配置好的。各个consumer（consumer 线程）可以组成一个组（Consumer group），partition中的每个message只能被组（Consumer group）中的一个consumer（consumer 线程）消费，如果一个message可以被多个consumer（consumer 线程）消费的话，那么这些consumer必须要在不同的组。Kafka不支持一个partition中的message由两个或两个以上的consumer thread来处理。除非来自不同的consumer group。它不能像AMQ那样可以多个BET作为consumer去处理message，这是因为多个BET去消费一个Queue中的数据的时候，由于要保证不能多个线程拿同一条message，所以就需要行级悲观锁（for update）,这就导致了consume的性能下降，吞吐量不够。而kafka为了保证吞吐量，只允许一个consumer线程去访问一个partition。如果觉得效率不高的时候，可以加partition的数量来横向扩展，那么再加新的consumer thread去消费。这样没有锁竞争，充分发挥了横向的扩展性，吞吐量极高。这也就形成了分布式消费的概念。

• 2.2 kafka一些原理概念

1.持久化

kafka使用文件存储消息(append only log),这就直接决定kafka在性能上严重依赖文件系统的本身特性.且无论任何OS下,对文件系统本身的优化是非常艰难的.文件缓存/直接内存映射等是常用的手段.因为kafka是对日志文件进行append操作,因此磁盘检索的开支是较小的;同时为了减少磁盘写入的次数,broker会将消息暂时buffer起来,当消息的个数(或尺寸)达到一定阈值时,再flush到磁盘,这样减少了磁盘IO调用的次数.对于kafka而言,较高性能的磁盘,将会带来更加直接的性能提升.

2.性能

除磁盘IO之外,我们还需要考虑网络IO,这直接关系到kafka的吞吐量问题. kafka并没有提供太多高超的技巧;对于producer端,可以将消息buffer起来;当消息的条数达到一定阈值时,批量发送给broker;对于consumer端也是一样,批量fetch多条消息.不过消息量的大小可以通过配置文件来指定.对于kafka broker端,似乎有个sendfile系统调用可以潜在的提升网络IO的性能:将文件的数据映射到系统内存中,socket直接读取相应的内存区域即可,而无需进程再次copy和交换(这里涉及到"磁盘IO数据"/"内核内存"/"进程内存"/"网络缓冲区",多者之间的数据copy).其实对于producer/consumer/broker三者而言,CPU的开支应该都不大,因此启用消息压缩机制是一个良好的策略:压缩需要消耗少量的CPU资源,不过对于kafka而言,网络IO更应该需要考虑.可以将任何在网络上传输的消息都经过压缩,kafka支持gzip/snappy等多种压缩方式.

3.负载均衡

kafka集群中的任何一个broker,都可以向producer提供metadata信息,这些metadata中包含"集群中存活的servers列表"/"partitions leader列表"等信息(请参看zookeeper中的节点信息).当producer获取到metadata信息之后,producer将会和Topic下所有partition leader保持socket连接;消息由producer直接通过socket发送到broker,中间不会经过任何"路由层".
异步发送: 将多条消息暂且在客户端buffer起来,并将他们批量发送到broker;小数据IO太多,会拖慢整体的网络延迟.批量延迟发送事实上提升了网络效率;不过这也有一定的隐患.比如当producer失效时,那些尚未发送的消息将会丢失.

4.Topic模型

其他JMS实现,消息消费的位置是有prodiver保留,以便避免重复发送消息或者将没有消费成功的消息重发等,同时还要控制消息的状态.这就要求JMS broker需要太多额外的工作.在kafka中,partition中的消息只有一个consumer在消费,且不存在消息状态的控制,也没有复杂的消息确认机制,可见kafka broker端是相当轻量级的.当消息被consumer接收之后,consumer可以在本地保存最后消息的offset,并间歇性的向zookeeper注册offset.由此可见,consumer客户端也很轻量级.
kafka中consumer负责维护消息的消费记录,而broker则不关心这些.这种设计不仅提高了consumer端的灵活性,也适度的减轻了broker端设计的复杂度;这是和众多JMS prodiver的区别.此外,kafka中消息ACK的设计也和JMS有很大不同,kafka中的消息是批量(通常以消息的条数或者chunk的尺寸为单位)发送给consumer,当消息消费成功后,向zookeeper提交消息的offset,而不会向broker交付ACK.或许你已经意识到,这种"宽松"的设计,将会有"丢失"消息/"消息重发"的危险.

5.消息传输一致

Kafka提供3种消息传输一致性语义: 最多1次, 最少1次, 恰好1次。
最少1次: 可能会重传数据, 有可能出现数据被重复处理的情况;
最多1次: 可能会出现数据丢失情况;
恰好1次: 并不是指真正只传输1次, 只不过有一个机制.确保不会出现"数据被重复处理"和"数据丢失"的情况。

at most once: 消费者fetch消息,然后保存offset,然后处理消息;当client保存offset之后,但是在消息处理过程中consumer进程失效(crash),导致部分消息未能继续处理.那么此后可能其他consumer会接管,但是因为offset已经提前保存,那么新的consumer将不能fetch到offset之前的消息(尽管它们尚没有被处理),这就是"at most once".
at least once: 消费者fetch消息,然后处理消息,然后保存offset.如果消息处理成功之后,但是在保存offset阶段zookeeper异常或者consumer失效,导致保存offset操作未能执行成功,这就导致接下来再次fetch时可能获得上次已经处理过的消息,这就是"at least once".
"Kafka Cluster"到消费者的场景中可以采取以下方案来得到"恰好1次"的一致性语义:
最少1次 + 消费者的输出中额外增加已处理消息最大编号: 由于已处理消息最大编号的存在, 不会出现重复处理消息的情况。



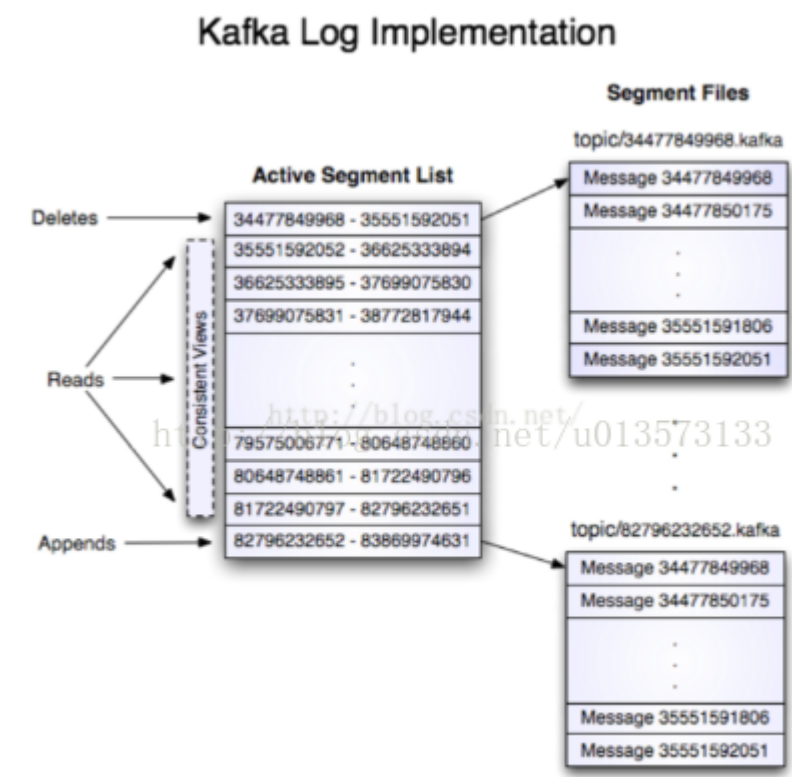
kafka中,replication策略是基于partition,而不是topic;kafka将每个partition数据复制到多个server上,任何一个partition有一个leader和多个follower(可以没有);备份的个数可以通过broker配置文件来设定。leader处理所有的read-write请求,follower需要和leader保持同步。Follower就像一个“consumer”,消费消息并保存在本地日志中;leader负责跟踪所有的follower状态,如果follower“落后”太多或者失效,leader将会把它从replicas同步列表中删除。当所有的follower都将一条消息保存成功,此消息才被认为是“committed”,那么此时consumer才能消费它。这种同步策略,就要求follower和leader之间必须具有良好的网络环境,即使只有一个replicas实例存活,仍然可以保证消息的正常发送和接收,只要zookeeper集群存活即可。

选择follower时需要兼顾一个问题,就是新leader server上所已经承载的partition leader的个数,如果一个server上有过多的partition leader,意味着此server将承受着更多的IO压力。在选举新leader,需要考虑到“负载均衡”,partition leader较少的broker将会更有可能成为新的leader。

7.log

每个log entry格式为"4个字节的数字N表示消息的长度" + "N个字节的消息内容";每个日志都有一个offset来唯一的标记一条消息,offset的值为8个字节的数字,表示此消息在此partition中所处的起始位置..每个partition在物理存储层面,有多个log file组成(称为segment),segment file的命名为"最小offset".kafka.例如"00000000000.kafka";其中"最小offset"表示此segment中起始消息的offset。

获取消息时,需要指定offset和最大chunk尺寸,offset用来表示消息的起始位置,chunk size用来表示最大获取消息的总长度(间接的表示消息的条数)。根据offset,可以找到此消息所在segment文件,然后根据segment的最小offset取差值,得到它在file中的相对位置,直接读取输出即可。



8.分布式

kafka使用zookeeper来存储一些meta信息,并使用了zookeeper watch机制来发现meta信息的变更并作出相应的动作(比如consumer失效,触发负载均衡等)

Broker node registry: 当一个kafka broker启动后,首先会向zookeeper注册自己的节点信息(临时znode),同时当broker和zookeeper断开连接时,此znode也会被删除。

Broker Topic Registry: 当一个broker启动时,会向zookeeper注册自己持有的topic和partitions信息,仍然是一个临时znode。

Consumer and Consumer group: 每个consumer客户端被创建时,会向zookeeper注册自己的信息,此作用主要是为了“负载均衡”,一个group中的多个consumer可以交错的消费一个topic的所有partitions;简而言之,保证此topic的所有partitions都能被此group所消费,且消费时为了性能考虑,让partition相对均衡的分散到每个consumer上。

Consumer id Registry: 每个consumer都有一个唯一的ID(host:uuid,可以通过配置文件指定,也可以由系统生成),此id用来标记消费者信息。

Consumer offset Tracking: 用来跟踪每个consumer目前所消费的partition中最大的offset.此znode为持久节点,可以看出offset跟group_id有关,以表明当group中一个消费者失效,其他consumer可以继续消费。

Partition Owner registry: 用来标记partition正在被哪个consumer消费.临时znode。此节点表达了“一个partition”只能被group下一个consumer消费,同时当group下某个consumer失效,那么将会触发负载均衡(即:让partitions在多个consumer间均衡消费,接管那些“游离”的partitions)

当consumer启动时,所触发的操作:

A) 首先进行“Consumer id Registry”;

B) 然后在“Consumer id Registry”节点下注册一个watch用来监听当前group中其他consumer的“leave”和“join”;只要此znode path下节点列表变更,都会触发此group下consumer的负载均衡。(比如一个consumer失效,那么其他consumer接管partitions)。

C) 在“Broker id registry”节点下,注册一个watch用来监听broker的存活情况;如果broker列表变更,将会触发所有的groups下的consumer重新balance。

总结:

- 1) Producer端使用zookeeper用来“发现”broker列表,以及和Topic下每个partition leader建立socket连接并发送消息。
- 2) Broker端使用zookeeper用来注册broker信息,已经监测partition leader存活性。
- 3) Consumer端使用zookeeper用来注册consumer信息,其中包括consumer消费的partition列表等,同时也用来发现broker列表,并和partition leader建立socket连接,并获取消息。

9.Leader的选择

Kafka的核心是日志文件，日志文件在集群中的同步是分布式数据系统最基础的要素。

如果leaders永远不会down的话我们就不需要followers了！一旦leader down掉了，需要在followers中选择一个新的leader.但是followers本身有可能延时太久或者crash，所以必须选择高质量的follower作为leader.必须保证，一旦一个消息被提交了，但是leader down掉了，新选出的leader必须可以提供这条消息。大部分的分布式系统采用了多数投票法则选择新的leader.对于多数投票法则，就是根据所有副本节点的状况动态的选择最适合的作为leader.Kafka并不是使用这种方法。

Kafka动态维护了一个同步状态的副本的集合（a set of in-sync replicas），简称ISR，在这个集合中的节点都是和leader保持高度一致的，任何一条消息必须被这个集合中的每个节点读取并追加到日志中了，才回通知外部这个消息已经被提交了。因此这个集合中的任何一个节点随时都可以被选为leader.ISR在ZooKeeper中维护，ISR中有f+1个节点，就可以允许在f个节点down掉的情况下不会丢失消息并正常提供服。ISR的成员是动态的，如果一个节点被淘汰了，当它重新达到“同步中”的状态时，他可以重新加入ISR.这种leader的选择方式是非常快速的，适合Kafka的应用场景。

一个邪恶的想法：如果所有节点都down掉了怎么办？Kafka对于数据不会丢失的保证，是基于至少一个节点是存活的，一旦所有节点都down了，这个就不能保证了。

实际应用中，当所有的副本都down掉时，必须及时作出反应。可以有以下两种选择:

1. 等待ISR中的任何一个节点恢复并担任leader。

2. 选择所有节点中（不只是ISR）第一个恢复的节点作为leader。

这是一个在可用性和连续性之间的权衡。如果等待ISR中的节点恢复，一旦ISR中的节点起不起来或者数据都是了，那集群就永远恢复不了了。如果等待ISR意外的节点恢复，这个节点的数据就会被作为线上数据，有可能和真实的数据有所出入，因为有些数据它可能还没同步到。Kafka目前选择了第二种策略，在未来的版本中将使这个策略的选择可配置，可以根据场景灵活的选择。

这种窘境不只Kafka会遇到，几乎所有的分布式数据系统都会遇到。

10.副本管理

以上仅仅以一个topic一个分区为例子进行了讨论，但实际上一个Kafka将会管理成千上万的topic分区 Kafka尽量的使所有分区均匀的分布到集群所有的节点上而不是集中在某些节点上，另外主从关系也尽量均衡这样每个几点都会担任一定比例的分区的leader。

优化leader的选择过程也是很重要的，它决定了系统发生故障时的空窗期有多久。Kafka选择一个节点作为“controller”,当发现有节点down掉的时候它负责在游泳分区的所有节点中选择新的leader.这使得Kafka可以批量的高效的管理所有分区节点的主从关系。如果controller down掉了，活着的节点中的一个会切换为新的controller。

11.Leader与副本同步

对于某个分区来说，保存正分区的“broker”为该分区的“leader”，保存备份分区的“broker”为该分区的“follower”。备份分区会完全复制正分区的消息，包括消息的编号等附加属性值。为了保持正分区和备份分区的内容一致，Kafka采取的方案是在保存备份分区的“broker”上开启一个消费者进程进行消费，从而使得正分区的内容与备份分区的内容保持一致。一般情况下，一个分区有一个“正分区”和零到多个“备份分区”。可以配置“正分区+备份分区”的总数量，关于这个配置，不同主题可以有不同的配置值。注意，生产者，消费者只与保存正分区的“leader”进行通信。

Kafka允许topic的分区拥有若干副本，这个数量是可以配置的，你可以为每个topic配置副本的数量。Kafka会自动在每个副本上备份数据，所以当 一个节点down掉时数据依然是可用的。

Kafka的副本功能不是必须的，你可以配置只有一个副本，这样其实就相当于只有一份数据。

创建副本的单位是topic的分区，每个分区都有一个leader和零或多个followers.所有的读写操作都由leader处理，一般分区的数量都比broker的数量多的多，各分区的leader均匀的分布在brokers中。所有的followers都复制leader的日志，日志中的消息和顺序都和leader中的一致。followers向普通的consumer那样从leader那里拉取消息并保存在自己的日志文件中。

许多分布式的信息系统自动的处理失败的请求，它们对一个节点是否着（alive）”有着清晰的定义。Kafka判断一个节点是否活着有两个条件：

1. 节点必须可以维护和ZooKeeper的连接，Zookeeper通过心跳机制检查每个节点的连接。

2. 如果节点是个follower,他必须能及时的同步leader的写操作，延时不能太久。

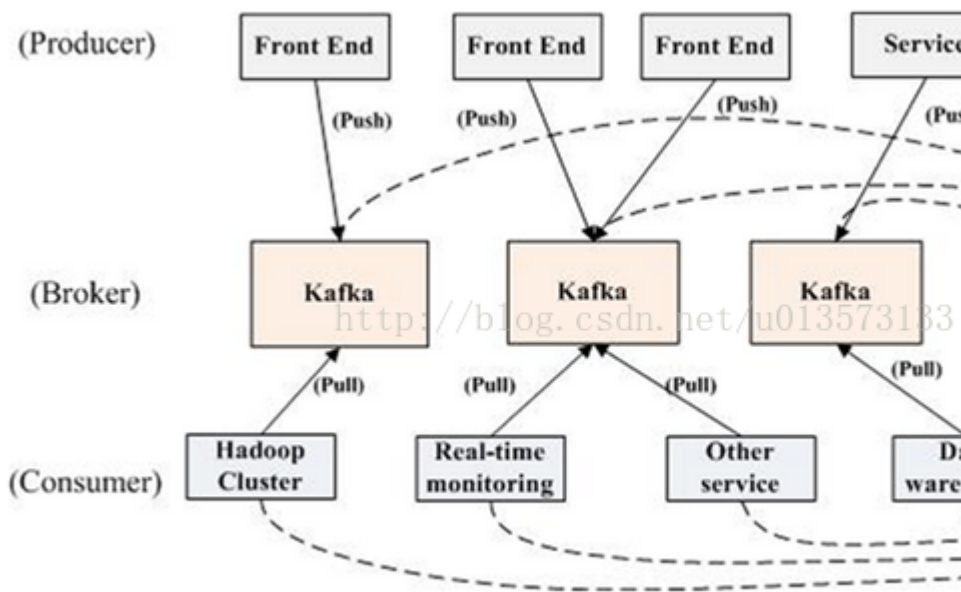
如果以上条件有任何一个不满足，那么该节点就被认为是“失败的”。Leader就会把它移除。至于延



(1条消息) Kafka史上最详细原理总结_我是Alvin家鸡鸭鱼的小米米-CSDN博客_cwh代表什么

时多久算是“太久”，是由参数replica.lag.max.messages决定的，怎样算是卡住了，怎是由参数replica.lag.time.max.ms决定的。
只有当消息被所有的副本加入到日志中时，才算是“committed”，只有committed的消息才会发送给consumer，这样就不用担心一旦leader down掉了消息会丢失。Producer也可以选择是否等待消息被提交的通知，这个是由参数acks决定的。
Kafka保证只要有一个“同步中”的节点，“committed”的消息就不会丢失。

2.3 kafka拓扑结构



一个典型的Kafka集群中包含若干Producer（可以是web前端FET，或者是服务器日志等），若干broker（Kafka支持水平扩展，一般broker数量越多，集群吞吐率越高），若干ConsumerGroup，以及一个Zookeeper集群。Kafka通过Zookeeper管理Kafka集群配置：选举Kafka broker的leader，以及在Consumer Group发生变化时进行rebalance，因为consumer消费kafka topic的partition的offset信息是存在Zookeeper的。Producer使用push模式将消息发布到broker，Consumer使用pull模式从broker订阅并消费消息。

分析过程分为以下4个步骤：

- topic中partition存储分布
- partition中文件存储方式 (partition在linux服务器上就是一个目录（文件夹）)
- partition中segment文件存储结构
- 在partition中如何通过offset查找message

通过上述4过程详细分析，我们就可以清楚认识到kafka文件存储机制的奥秘。

2.3 topic中partition存储分布

假设实验环境中Kafka集群只有一个broker，xxx/message-folder为数据文件存储根目录，在Kafka broker中server.properties文件配置(参数log.dirs=xxx/message-folder)，例如创建2个topic名 称分别为report_push、launch_info，partitions数量都为partitions=4

存储路径和目录规则为：

xxx/message-folder

```
|--report_push-0
|--report_push-1
|--report_push-2
|--report_push-3
|--launch_info-0
|--launch_info-1
|--launch_info-2
|--launch_info-3
```

在Kafka文件存储中，同一个topic下有多个不同partition，每个partition为一个目录，partiton命名规则为 **topic名称+有序序号**，第一个partition序号从0开始，序号最大值为partitions数量减1。

消息发送时都被发送到一个topic，其本质就是一个目录，而topic由是由一些Partition组成,其组织结构如下图所示：

我们可以看到，Partition是一个Queue的结构，每个Partition中的消息都是有序的，生产的消息被不断追加到Partition上，其中的每一个消息都被赋予了一个唯一的offset值。

Kafka集群会保存所有的消息，不管消息有没有被消费；**我们可以设定消息的过期时间，只有过期的数据才会被自动清除以释放磁盘空间。**比如我们设置消息过期时间为2天，那么这2天内的所有消息都会被保存到集群中，数据只有超过了两天才会被清除。

Kafka只维护在Partition中的offset值，因为这个offset标识着这个partition的message消费到哪条了。Consumer每消费一个消息，offset就会加1。其实消息的状态完全是由Consumer控制的，Consumer可以跟踪和重设这个offset值，这样的话Consumer就可以读取任意位置的消息。

把消息日志以Partition的形式存放有多重考虑，第一，方便在集群中扩展，每个Partition可以通过调整以适应它所在的机器，而一个topic又可以有多个Partition组成，因此整个集群就可以适应任意大小的数据了；第二就是可以提高并发，因为可以以Partition为单位读写了。

通过上面介绍的我们可以知道，kafka中的数据是持久化的并且能够容错的。Kafka允许用户为每个topic设置副本数量，副本数量决定了有几个broker来存放写入的数据。如果你的副本数量设置为3，那么一份数据就会被存放在3台不同的机器上，那么就允许有2个机器失败。一般推荐副本数量至少为2，这样就可以保证增减、重启机器时不会影响到数据消费。如果对数据持久化有更高的要求，可以把副本数量设置为3或者更多。

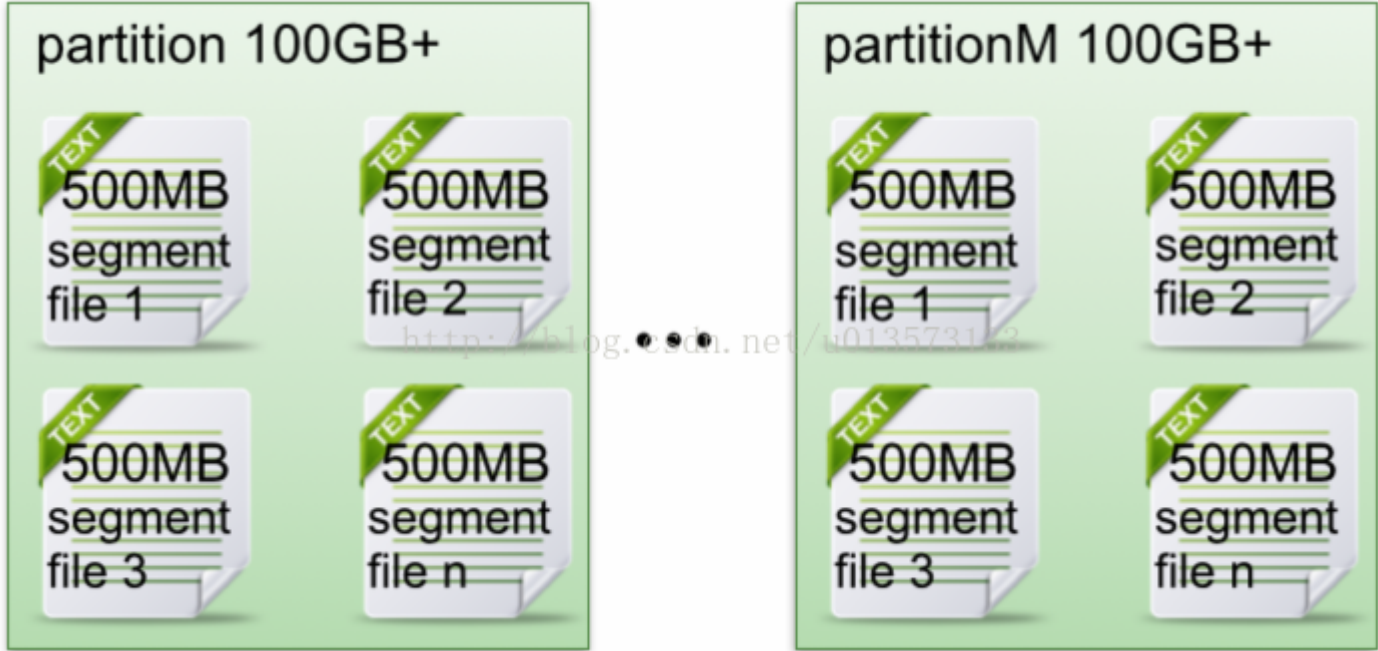
Kafka中的topic是以partition的形式存放的，每一个topic都可以设置它的partition数量，Partition的数量决定了组成topic的message的数量。Producer在生产数据时，会按照一定规则（这个规则是可以自定义的）把消息发布到topic的各个partition中。上面将的副本都是以partition为单位的，不过只有一个partition的副本会被选举成leader作为读写用。

关于如何设置partition值需要考虑的因素。**一个partition只能被一个消费者消费（一个消费者可以同时消费多个partition）**，因此，如果设置的partition的数量小于consumer的数量，就会有消费者消费不到数据。所以，推荐partition的数量一定要大于同时运行的consumer的数量。另外一方面，建议partition的数量大于集群broker的数量，这样leader partition就可以均匀的分布在各个broker中，最终使得集群负载均衡。在Cloudera,每个topic都有上百个partition。需要注意的是，kafka需要为每个partition分配一些内存来缓存消息数据，如果partition数量越大，就要为kafka分配更大的heap space。

2.4 partiton中文件存储方式

- 每个partion(目录)相当于一个巨型文件被平均分配到多个大小相等segment(段)数据文件中。但每个段segment file消息数量不一定相等，这种特性方便old segment file快速被删除。
- 每个partiton只需要支持顺序读写就行了，segment文件生命周期由服务端配置参数决定。

这样做的好处就是能快速删除无用文件，有效提高磁盘利用率。



2.5 partiton中segment文件存储结构

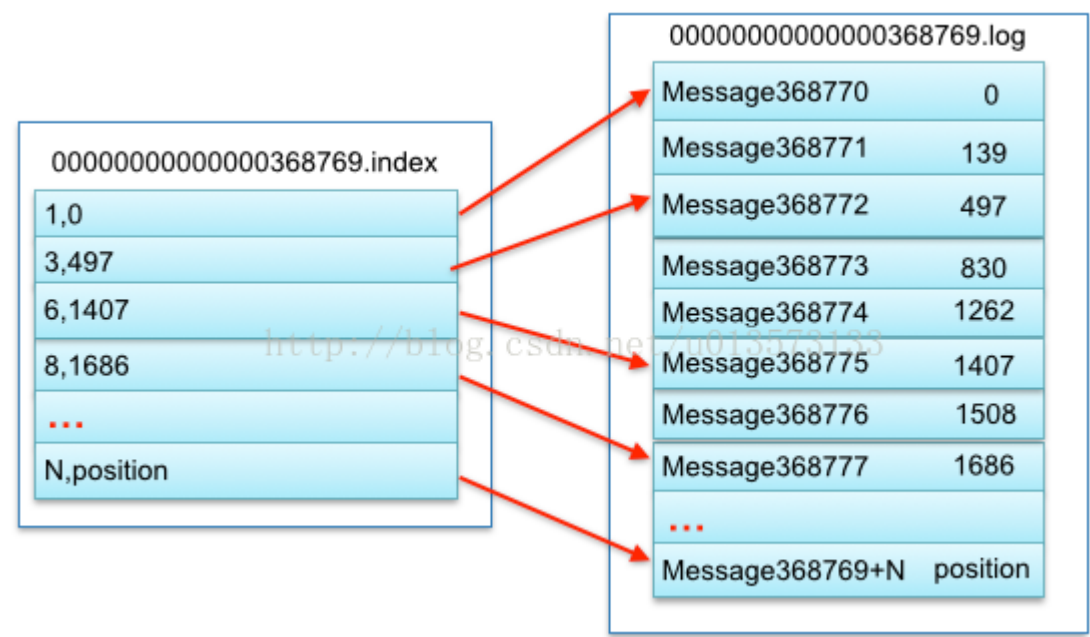
producer发message到某个topic，message会被均匀的分布到多个partition上（随机或根据用户指定的回调函数进行分布），kafka broker收到message往对应partition的最后一个segment上添加该消息，当某个segment上的消息条数达到配置值或消息发布时间超过阈值时，segment上的消息会被flush到磁盘，只有flush到磁盘上的消息consumer才能消费，segment达到一定的大小后将不会再往该segment写数据，broker会创建新的

每个part在内存中对应一个index，记录每个segment中的第一条消息偏移。

- segment file组成：由2大部分组成，分别为index file和data file，此2个文件一一对应，成对出现，后缀".index"和".log"分别表示为segment索引文件、数据文件。
- segment文件命名规则：partion全局的第一个segment从0开始，后续每个segment文件名为上一个全局partion的最大offset(偏移message数)。数值最大为64位long大小，19位数字字符长度，没有数字用0填充。

每个segment中存储很多条消息，消息id由其逻辑位置决定，即从消息id可直接定位到消息的存储位置，避免id到位置的额外映射。

下面文件列表是笔者在Kafka broker上做的一个实验，创建一个topicXXX包含1 partition，设置每个segment大小为500MB,并启动producer向Kafka broker写入大量数据,如下图2所示segment文件列表形象说明了上述2个规则：

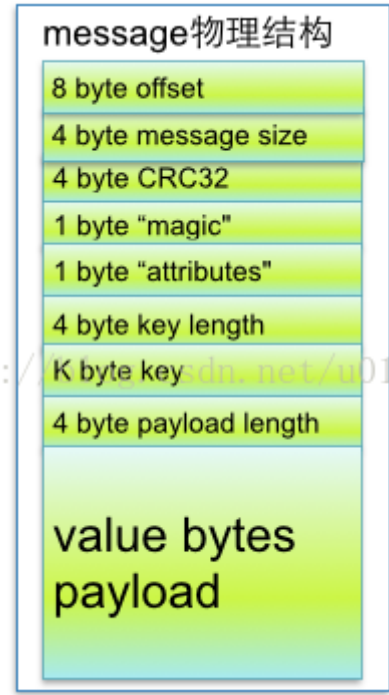


以上述图2中一对segment file文件为例，说明segment中index<—>data file对应关系物理结构如下：

```
0000000000000000000.index
0000000000000000000.log
00000000000000368769.index
00000000000000368769.log
0000000000000737337.index
0000000000000737337.log
0000000000001105814.index
0000000000001105814.log
.....
```

上述图3中索引文件存储大量元数据，数据文件存储大量消息，索引文件中元数据指向对应数据文件中message的物理偏移地址。其中以索引文件中 元数据3,497为例，依次在数据文件中表示第3个message(在全局partion表示第368772个message)、以及该消息的物理偏移 地址为497。

从上述图3了解到segment data file由许多message组成，下面详细说明message物理结构如下：



参数说明：

关键字	解释说明
8 byte offset	在partition(分区)内的每条消息都有一个有序 id号，这个id号被称为偏移(offset),它可以唯一确定每条消息在partition(分区)内的位置。即 offset表示partition的第多少message
4 byte message size	message大小
4 byte CRC32	用crc32校验message
1 byte "magic"	表示本次发布Kafka服务程序协议版本号
1 byte "attributes"	表示为独立版本、或标识压缩类型、或编码类型。
4 byte key length	表示key的长度,当key为-1时，K byte key字段不填
K byte key	可选
value bytes payload	表示实际消息数据。

2.6 在partition中如何通过offset查找message

例如读取offset=368776的message，需要通过下面2个步骤查找。

- 第一步查找segment file

上述图2为例，其中0000000000000000000.index表示最开始的文件，起始偏移量(offset)为0.第二个文件 00000000000000368769.index的消息量起始偏移量为368770 = 368769 + 1.同样，第三个文件 00000000000000737337.index的起始偏移量为737338=737337 + 1，其他后续文件依次类推，以起始偏移量命名并排序这些文件，只要根据offset **二分查找**文件列表，就可以快速定位到具体文件。

当offset=368776时定位到00000000000000368769.index|log

- 第二步通过segment file查找message通过第一步定位到segment file，当offset=368776时，依次定位到00000000000000368769.index的元数据物理位置和 00000000000000368769.log的物理偏移地址，然后再通过00000000000000368769.log顺序查找直到 offset=368776为止。

segment index file采取稀疏索引存储方式，它减少索引文件大小，通过mmap可以直接内存操作，稀疏索引为数据文件的每个对应message设置一个元数据指针,它 比稠密索引节省了更多的存储空间，但查找起来需要消耗更多的时间。

kafka会记录offset到zk中。但是，zk client api对zk的频繁写入是一个低效的操作。0.8.2 kafka引入了native offset storage，将offset管理从zk移出，并且可以做到水平扩展。其原理就是利用了kafka的compacted topic，offset以consumer group,topic与partion的组合作为key直接提交到compacted topic中。同时Kafka又在内存中维护了的三元组来维护最新的offset信息，consumer来取最新offset信息的时候直接内存里拿即可。当然，kafka允许你快速的checkpoint最新的offset信息到磁盘上。

3.Partition Replication原则

Kafka高效文件存储设计特点

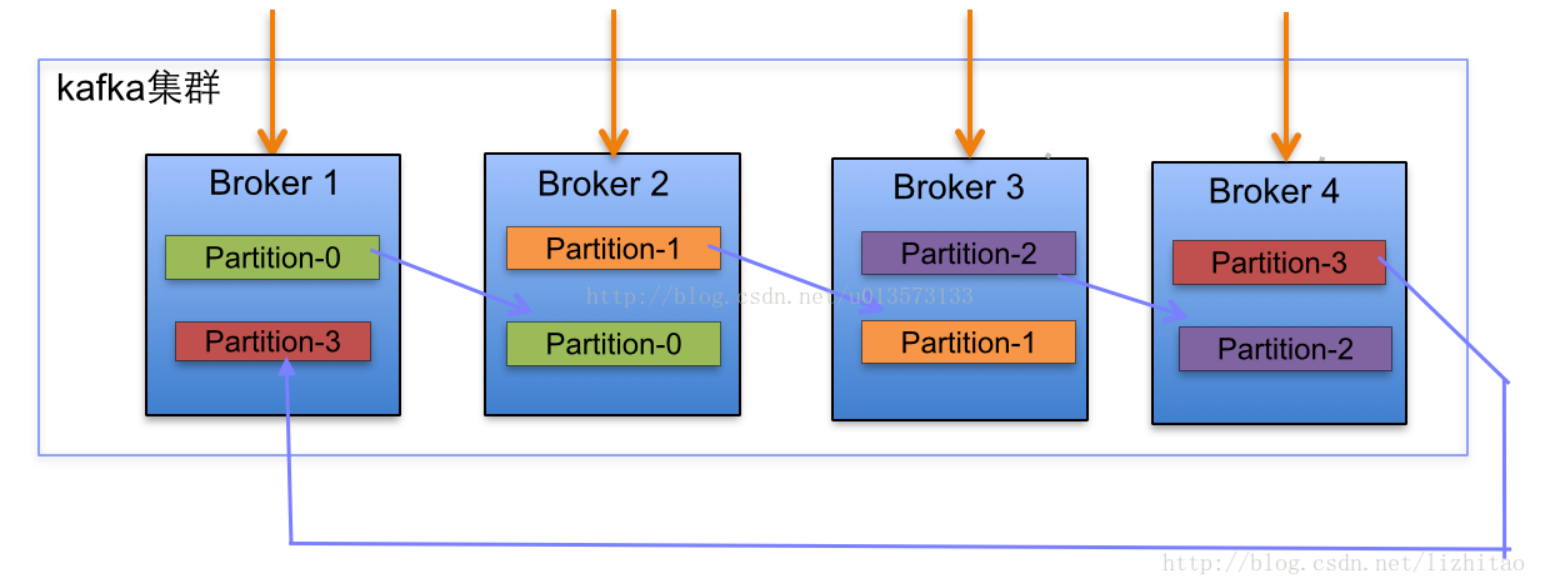
- Kafka把topic中一个partition大文件分成多个小文件段，通过多个小文件段，就容易定期清除或删除已经消费完文件，减少磁盘占用。
- 通过索引信息可以快速定位message和确定response的最大大小。
- 通过index元数据全部映射到memory，可以避免segment file的IO磁盘操作。
- 通过索引文件稀疏存储，可以大幅降低index文件元数据占用空间大小。

1. Kafka集群partition replication默认自动分配分析

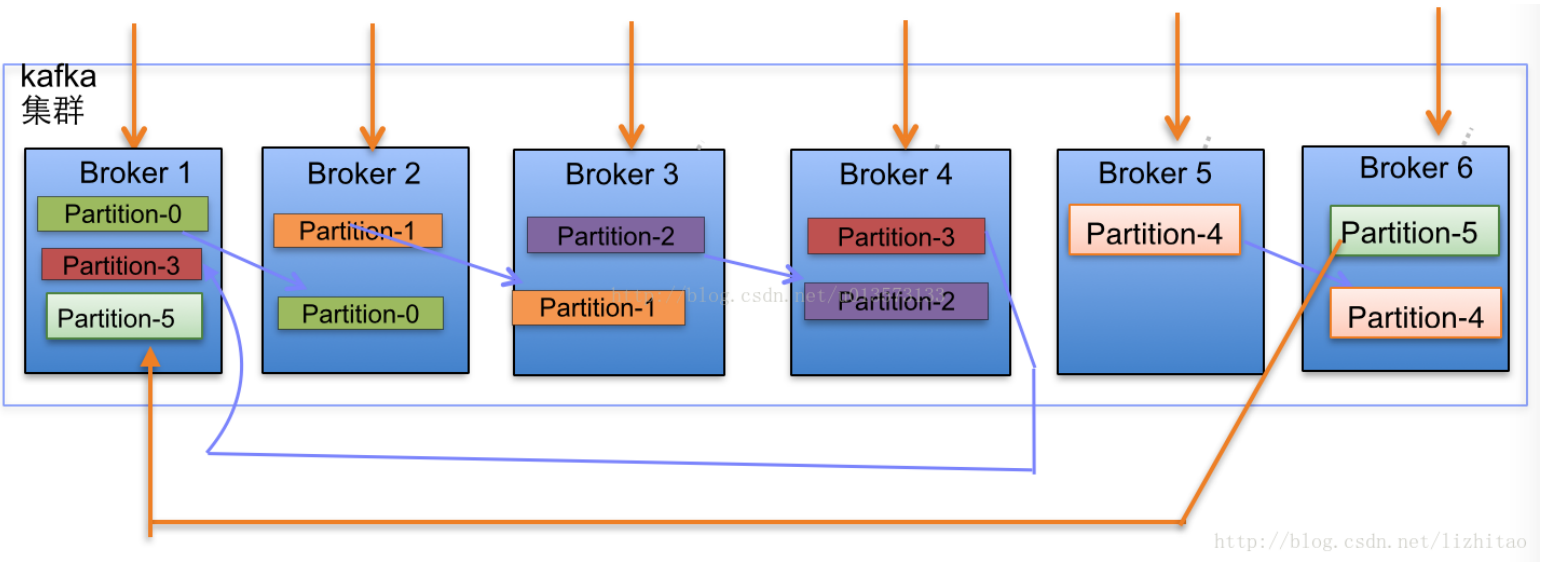
下面以一个Kafka集群中4个Broker举例，创建1个topic包含4个Partition，2 Replication；数据Producer流动如图所示：

(1)





(2)当集群中新增2节点，Partition增加到6个时分布情况如下：



副本分配逻辑规则如下：

- 在Kafka集群中，每个Broker都有均等分配Partition的Leader机会。
- 上述图Broker Partition中，箭头指向为副本，以Partition-0为例:broker1中partition-0为Leader，Broker2中Partition-0为副本。
- 上述图种每个Broker(按照BrokerId有序)依次分配主Partition,下一个Broker为副本，如此循环迭代分配，多副本都遵循此规则。

副本分配算法如下：

- 将所有N Broker和待分配的i个Partition排序。
- 将第i个Partition分配到第 $(i \bmod n)$ 个Broker上。
- 将第i个Partition的第j个副本分配到第 $((i + j) \bmod n)$ 个Broker上。

4.Kafka Broker一些特性

4.1 无状态的Kafka Broker：

- Broker没有副本机制，一旦broker宕机，该broker的消息将都不可用。
- Broker不保存订阅者的状态，由订阅者自己保存。
- 无状态导致消息的删除成为难题（可能删除的消息正在被订阅），kafka采用基于时间的SLA(服务水平保证)，消息保存一定时间（通常为7天）后会被删除。
- 消息订阅者可以rewind back到任意位置重新进行消费，当订阅者故障时，可以选择最小的offset进行重新读取消费消息。

4.2 message的交付与生命周期：

- 不是严格的JMS，因此kafka对消息的重复、丢失、错误以及顺序型没有严格的要求。（这是与AMQ最大的区别）
- kafka提供at-least-once delivery,即当consumer宕机后，有些消息可能会被重复delivery。
- 因每个partition只会被consumer group内的一个consumer消费，故kafka保证每个partition内的消息会被顺序的订阅。
- Kafka为每条消息为每条消息计算CRC校验，用于错误检测，crc校验不通过的消息会直接被丢弃掉。

4.3 压缩

Kafka支持以集合（batch）为单位发送消息，在此基础上，Kafka还支持对消息集合进行压缩，Producer端可以通过GZIP或Snappy格式对消息集合进行压缩。Producer端进行压缩之后，在Consumer端需进行解压。压缩的好处就是减少传输的数据量，减轻对网络传输的压力，在对大数据处理上，瓶颈往往体现在网络上而不是CPU。

那么如何区分消息是压缩的还是未压缩的呢，Kafka在消息头部添加了一个描述压缩属性字节，这个字节的后两位表示消息的压缩采用的编码，如果后两位为0，则表示消息未被压缩。

4.4 消息可靠性

在消息系统中，保证消息在生产 and 消费过程中的可靠性是十分重要的，在实际消息传递过程中，可能会出现如下三中情况：

- 一个消息发送失败
- 一个消息被发送多次
- 最理想的情况：exactly-once ,一个消息发送成功且仅发送了一次

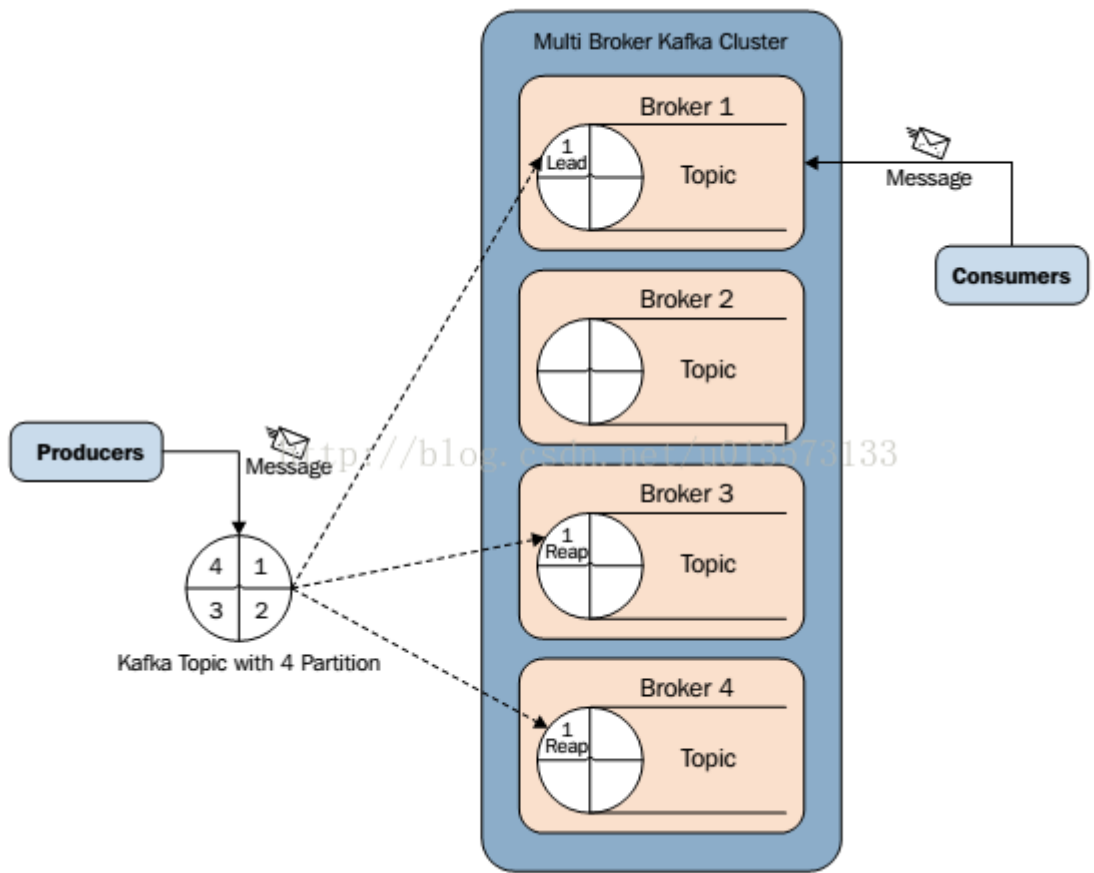
有许多系统声称它们实现了exactly-once，但是它们其实忽略了生产者或消费者在生产 and 消费过程中有可能失败的情况。比如虽然一个Producer成功发送一个消息，但是消息在发送途中丢失，或者成功发送到broker，也被consumer成功取走，但是这个consumer在处理取过来的消息时失败了。

从Producer端看：Kafka是这么处理的，当一个消息被发送后，Producer会等待broker成功接收到消息的反馈（可通过参数控制等待时间），如果消息在途中丢失或是其中一个broker挂掉，Producer会重新发送（我们知道Kafka有备份机制，可以通过参数控制是否等待所有备份节点都收到消息）。

从Consumer端看：前面讲到过partition，broker端记录了partition中的一个offset值，这个值指向Consumer下一个即将消费message。当Consumer收到了消息，但在处理过程中挂掉，此时Consumer可以通过这个offset值重新找到上一个消息再进行处理。Consumer还有权限控制这个offset值，对持久化到broker端的消息做任意处理。

4.5 备份机制

备份机制是Kafka0.8版本的新特性，备份机制的出现大大提高了Kafka集群的可靠性、稳定性。有了备份机制后，Kafka允许集群中的节点挂掉后而不影响整个集群工作。一个备份数量为n的集群允许n-1个节点失败。在所有备份节点中，有一个节点作为lead节点，这个节点保存了其它备份节点列表，并维持各个备份间的状态同步。下面这幅图解释了Kafka的备份机制：



4.6 Kafka高效性相关设计

4.6.1 消息的持久化

Kafka高度依赖文件系统来存储和缓存消息(AMQ的message是持久化到mysql数据库中的), 因为一般的人认为磁盘是缓慢的, 这导致人们对持久化结构具有竞争性持怀疑态度。其实, 磁盘的快或者慢, 这决定于我们如何使用磁盘。 因为磁盘线性写的速度远远大于随机写。线性读写在大多数应用场景下是可以预测的。

4.6.2 常数时间性能保证

每个Topic的Partition的是一个文件夹, 里面有无数个小文件夹segment, 但partition是一个队列, 队列中的元素是segment,消费的时候先从第0个segment开始消费, 新来message存在最后一个消息队列中。对于segment也是对队列, 队列元素是message,有对应的offset标识是哪个message。消费的时候先从这个segment的第一个message开始消费, 新来的message存在segment的最后。

消息系统的 持久化队列可以构建在对一个文件的读和追加, 就像一般情况下的日志解决方案。它有一个优点, 所有的操作都是常数时间, 并且读写之间不会相互阻塞。这种设计具有极大的性能优势: 最终系统性能和数据大小完全无关, 服务器可以充分利用廉价的硬盘来提供高效的信息服务。

事实上还有一点, 磁盘空间的无限增大而不影响性能这点, 意味着我们可以提供一般消息系统无法提供的特性。比如说, 消息被消费后不是立马被删除, 我们可以将这些消息保留一段相对比较长的时间 (比如一个星期)。

5.Kafka 生产者-消费者

消息系统通常都会由生产者, 消费者, Broker三大部分组成, 生产者会将消息写入到Broker, 消费者会从Broker中读取出消息, 不同的MQ实现的Broker实现会有所不同, 不过Broker的本质都是要负责将消息落地到服务端的存储系统中。具体步骤如下:

1. 生产者客户端应用程序产生消息:
1. 客户端连接对象将消息包装到请求中发送到服务端
2. 服务端的入口也有一个连接对象负责接收请求, 并将消息以文件的形式存储起来
3. 服务端返回响应结果给生产者客户端
2. 消费者客户端应用程序消费消息:
1. 客户端连接对象将消费信息也包装到请求中发送给服务端
2. 服务端从文件存储系统中取出消息
3. 服务端返回响应结果给消费者客户端
4. 客户端将响应结果还原成消息并开始处理消息

图4-1 客户端和服务端交互

5.1 Producers

Producers直接发送消息到broker上的leader partition, 不需要经过任何中介或其他路由转发。为了实现这个特性, kafka集群中的每个broker都可以响应producer的请求, 并返回topic的一些元信息, 这些元信息包括哪些机器是存活的, topic的leader partition都在哪, 现阶段哪些leader partition是可以直接被访问的。

Producer客户端自己控制着消息被推送到哪些partition。实现的方式可以是随机分配。实现一类随机负载均衡算法, 或者指定一些分区算法。Kafka提供了接口供用户实现自定义的partition, 用户可以为每个消息指定一个partitionKey, 通过这个key来实现一些hash分区算法。比如, 把userid作为partitionkey的话, 相同userid的消息将会被推送到同一个partition。

以Batch的方式推送数据可以极大的提高处理效率, kafka Producer 可以将消息在内存中累计到一定数量后作为一个batch发送请求。Batch的数量大小可以通过Producer的参数控制, 参数值可以设置为累计的消息的数量(如500条)、累计的时间间隔(如100ms)或者累计的数据大小(64KB)。通过增加batch的大小, 可以减少网络请求和磁盘IO的次数, 当然具体参数设置需要在效率和时效性方面做一个权衡。

Producers可以异步的并行的向kafka发送消息, 但是通常producer在发送完消息之后会得到一个future响应, 返回的是offset值或者发送过程中遇到的错误。这其中有个非常重要的参数“acks”, 这个参数决定了producer要求leader partition 收到确认的副本个数, 如果acks设置数量为0, 表示producer不会等待broker的响应, 所以, producer无法知道消息是否发送成功, 这样有可能会导数据丢失, 但同时, acks值为0会得到最大的系统吞吐量。

若acks设置为1, 表示producer会在leader partition收到消息时得到broker的一个确认, 这样会有更好的可靠性, 因为客户端会等待直到broker确认收到消息。若设置为-1, producer会在所有备份的partition收到消息时得到broker的确认, 这个设置可以得到最高的可靠性保证。

Kafka 消息有一个定长的header和变长的字节数组组成。因为kafka消息支持字节数组, 也使得kafka可以支持任何用户自定义的序列号格式或者其它已有的格式如Apache Avro、protobuf等。Kafka没有限定单个消息的大小, 但我们推荐消息大小不要超过1MB,通常一般消息大小都在1~10kB之前。

发布消息时, kafka client先构造一条消息, 将消息加入到消息集set中(kafka支持批量发布, 可以往消息集合中添加多条消息, 一次行发布), send消息时, producer client需指定消息所属的topic。

5.2 Consumers

Kafka提供了两套consumer api, 分为high-level api和sample-api。Sample-api 是一个底层的API, 它维持了一个和单一broker的连接, 并且这个API是完全无状态的, 每次请求都需要指定offset值, 因此, 这套API也是最灵活的。

在kafka中, 当前读到哪条消息的offset值是由consumer来维护的, 因此, consumer可以自己决定如何读取kafka中的数据。**比如, consumer可以通过重设offset值来重新消费已消费过的数据。不管有没有被消费, kafka会保存数据一段时间, 这个时间周期是可配置的, 只有到了过期时间, kafka才会删除这些数据。(这一点与AMQ不一样, AMQ的message一般来说都是持久化到mysql中的, 消费完的message会被delete掉)**

High-level API封装了对集群中一系列broker的访问, 可以透明的消费一个topic。它自己维持了已消费消息的状态, 即每次消费的都是下一个消息。

High-level API还支持以组的形式消费topic, 如果consumers有同一个组名, 那么kafka就相当于一个队列消息服务, 而各个consumer均衡的消费相应partition中的数据。若consumers有不同的组名, 那么此时kafka就相当与一个广播服务, 会把topic中的所有消息广播到每个consumer。

High level api和Low level api是针对consumer而言的, 和producer无关。

High level api是consumer读的partition的offset是存在zookeeper上。High level api 会启动另外一个线程去每隔一段时间, offset自动同步到zookeeper上。换句话说, 如果使用了High level api, 每个message只能被读一次, 一旦读了这条message之后, 无论我consumer的处理是否ok。High level api的另外一个线程会自动的把offset+1同步到zookeeper上。如果consumer读取数据出了问题, offset也会在zookeeper上同步。因此, 如果consumer处理失败了, 会继续执行下一条。这往往是不对的行为。因此, Best Practice是一旦consumer处理失败, 直接让整个consumer group抛Exception终止, 但是最后读的这一条数据是丢失了, 因为在zookeeper里面的offset已经+1了。等再次启动consumer group的时候, 已经从下一条开始读取处理了。

Low level api 是consumer读的partition的offset在consumer自己的程序中维护。不会同步到zookeeper上。但是为了kafka manager能够方便的监控, 一般也会手动的同步到zookeeper上。这样的好处是一旦读取某个message的consumer失败了, 这条message的offset我们自己维护, 我们不会+1。下次再启动的时候, 还会从这个offset开始读。这样可以做到exactly once对于数据的准确性有保证。

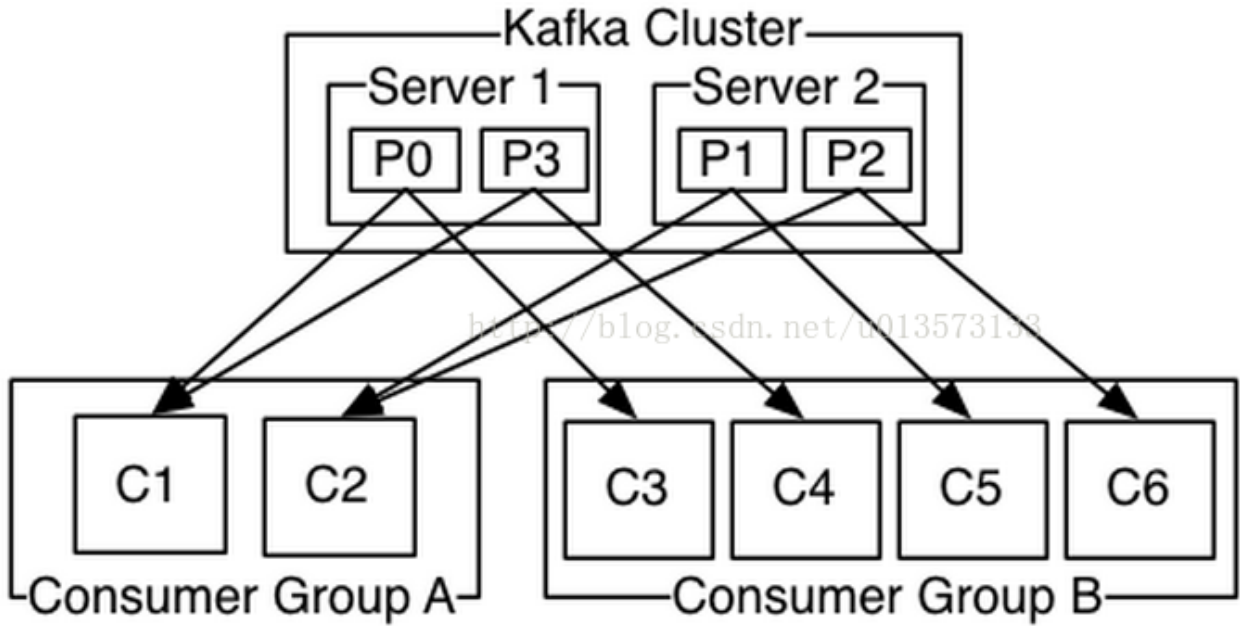
对于Consumer group:

1. 允许consumer group (包含多个consumer, 如一个集群同时消费) 对一个topic进行消费, 不同的consumer group之间独立消费。

2. 为了对减小一个consumer group中不同consumer之间的分布式协调开销, 指定partition为最小的并行消费单位, 即一个group内的consumer只能消费不同的partition。



举报



- Consumer与Partition的关系:
- 如果consumer比partition多, 是浪费, 因为kafka的设计是在一个partition上是不允许并发的, 所以consumer数不要大于partition数
 - 如果consumer比partition少, 一个consumer会对应于多个partitions, 这里主要合理分配consumer数和partition数, 否则会导致partition里面的数据被取的不均匀
 - 如果consumer从多个partition读到数据, 不保证数据间的顺序性, kafka只保证在一个partition上数据是有序的, 但多个partition, 根据你读的顺序会有不同
 - 增减consumer, broker, partition会导致rebalance, 所以rebalance后consumer对应的partition会发生变化
 - High-level接口中获取不到数据的时候是会block的

负载低的情况下可以每个线程消费多个partition。但负载高的情况下, Consumer 线程数最好和Partition数量保持一致。如果还是消费不过来, 应该再开 Consumer 进程, 进程内线程数同样和分区数一致。

消费消息时, kafka client需指定topic以及partition number (每个partition对应一个逻辑日志流, 如topic代表某个产品线, partition代表产品线的日志按天切分的结果), consumer client订阅后, 就可迭代读取消息, 如果没有消息, consumer client会阻塞直到有新的消息发布。consumer可以累积确认接收到的消息, 当其确认了某个offset的消息, 意味着之前的消息也都已成功接收到, 此时broker会更新zookeeper上地offset registry。

5.3 高效的数据传输

1. 发布者每次可发布多条消息 (将消息加到一个消息集中发布), consumer每次迭代消费一条消息。
创建单独的cache, 使用系统的page cache。发布者顺序发布, 订阅者通常比发布者滞后一点点, 直接使用linux的page cache效果也比较后, 同时减少了cache管理及垃圾收集的开销。
3. 使用sendfile优化网络传输, 减少一次内存拷贝。

6.Kafka 与 Zookeeper

6.1 Zookeeper 协调控制

1. 管理broker与consumer的动态加入与离开。(Producer不需要管理, 随便一台计算机都可以作为Producer向Kakfa Broker发消息)
2. 触发负载均衡, 当broker或consumer加入或离开时会触发负载均衡算法, 使得一个consumer group内的多个consumer的消费负载均衡。(因为一个consumer消费一个或多个partition, 一个partition只能被一个consumer消费)
3. 维护消费关系及每个partition的消费信息。

6.2 Zookeeper上的细节:

- 每个broker启动后会在zookeeper上注册一个临时的broker registry, 包含broker的ip地址和端口号, 所存储的topics和partitions信息。
- 每个consumer启动后会在zookeeper上注册一个临时的consumer registry: 包含consumer所属的consumer group以及订阅的topics。
- 每个consumer group关联一个临时的owner registry和一个持久的offset registry。对于被订阅的每个partition包含一个owner registry, 内容为订阅这个partition的consumer id; 同时包含一个offset registry, 内容为上一次订阅的offset。

kafka工作原理介绍 Saint 7万+
两张图读懂kafka应用: Kafka 中的术语 broker: 中间的kafka cluster, 存储消息, 是由多个server组成的集群。 topic: kafk...

Kafka常见面试题 qq_28900249的博客 11万+
1 什么是kafka Kafka是分布式发布-订阅消息系统, 它最初是由LinkedIn公司开发的, 之后成为Apache项目的一部分, Kafka...

优质评论可以帮助作者获得更高权重

评论

飞飞范: 我看到两篇一模一样。。。 7月前 回复 ...

爱码士 * Tisfy: 这让我想起了先贤的一句话: 尔曹身与名俱灭, 不废江河万古流。 19天前 回复 ...

稷下小鲤鱼: 我再更新一下, 不是小错误, 关键的好几个知识点都错了, ack和offset high level, low level都错了 5月前 回复 ...

码哥 * weixin_39564277 回复 : 说说哪里错了呀? 4月前 回复 ...

稷下小鲤鱼: 写的不错, 不过我发现了几个小错误, 瑕不掩瑜 5月前 回复 ...

稷下小鲤鱼: nice 5月前 回复 ...

码农 * Eclipse小乖: 楼主想问下, kafka 假设broker有三个0、1、2,分区有三个,副本均匀分布, isr都能跟上, 那么按照2、1、0的顺序关闭broker时, 是不是在关闭broker0之前, 所有leader都在broker0上。另外当broker0也关闭后, 再启动broker1.那么是不是leader都在broker1上呢 6月前 回复 ...

dab-hand: 应该是follower向leader同步, 而不是leader给follower发送 7月前 回复 ...

魔力小猪 回复 Alvin家鸡鸭鱼的小小米: Followers 节点就像普通的 consumer 那样从 leader 节点那里拉取消息并保存在自己的日志文件中 官网的问答 7月前 回复 ...

Alvin家鸡鸭鱼的小小米 回复 : 不是的, 你再看一下kafka官网的介绍 7月前 回复 ...

君宝逍遥: 多谢大佬分享 8月前 回复 ...

一只艾涕喵: 可以转载吗, 注明出处和作者 8月前 回复 ...

Alvin家鸡鸭鱼的小小米 回复 : 可以的, 能学到东西就好 7月前 回复 ...

码哥 * weixin_39564277: 有个问题一直不懂, 生产者生产消息和消费者消费消息是异步的, 那生产者生产消息后, 消费者怎么监听到新增了个信息然后去消费这个新增的信息的呢。有知道的大佬帮忙答疑下, 谢谢! 9月前 回复 ...

Alvin家鸡鸭鱼的小小米 回复 : Producer生产消息, 会把数据通过tcp长连接发送发kafka broker服务端, kafka broker会把数据用append-only的方式顺序写到磁盘中, 并且更新latest offset。Consumer消费会从当前的offset开始消费, 直到消费到latest offset 7月前 回复 ...

< 1 2 >

相关推荐

名词解释:cWh_mainbanp_cwh代表什么

3-19

电池容量mAh大家都知道.那么cWh 是什么呢?举例解释: 设计电压是7.2V,设计容量是3000mAh,那么设计能量cWh是多少? ...

tomcat+https协议的接口编写及客户端访问_菜头笔记_cwh...

3-6

a.利用jdk自带的证书生成工具来生成一个key,keytool -genkey -alias cwh -keyalg RSA -keystore e:/keys/cwhkey 主意一下!':...

kafka之二: Kafka 设计与原理详解

weixin_34279246的博客 1738

一、Kafka简介 本文综合了我之前写的kafka相关文章, 可作为一个全面了解学习kafka的培训学习资料。转载请注明出处: ...

点赞51 评论30 分享 收藏331 打赏 举报 关注 一键三连

alXueLing专栏 2万+

https://blog.csdn.net/U013573133/article/details/48142677?utm_medium=distribute.pc_relevant.none-task-blog-2-default-BlogCommendFromMachineLearnPaI2~default-12.control&dist_request_id=&depth_1-utm_source=distribute.pc_relevant.none-task-blog-2-default-BlogCommendFromMachineLearnPaI2~default-12.control

9/10

(1条消息) Kafka史上最详细原理总结_我是Alvin家鸡鸭鱼的小小米-CSDN博客_cwh代表什么		
一、前提条件 1、部署Kafka集群搭建需要服务器至少3台，奇数台 2、Kafka的安装需要java环境，jdk1.8 3、Kafka安装包...		
...c]转为Tensor的[c, w, h]_莫邪莫急的博客_cwh代表什么	3-31	
在神经网络中,图像被表示成[c, h, w]格式或者[n, c, h, w]格式,但如果想要将图像以np.ndarray形式输入,因np.ndarray默认将...		
判断一个链表是否有环采用快慢指针_cwh_Blog的博客	3-8	
你能给出O(1)空间复杂度的解法么? //快慢指针能够相遇则说明有环#include<stdbool.h>boolhasCycle(structListNode*head)...		
Kafka实战	weixin_30614109的博客	2226
1. kafka介绍 1.1. 主要功能 根据官网的介绍，ApacheKafka®是一个分布式流媒体平台，它主要有3种功能： 1: It lets ...		
Kafka安装教程（详细过程）	Poppy_Evan的博客	4万+
安装前期准备： 1，准备三个节点（根据自己需求决定） 2，三个节点上安装好zookeeper（也可以使用kafka自带的zookee...		
...分别代表什么意义?在try块中可以抛出异常吗?_CWH615...	3-21	
finally块代表异常处理流程中总会执行的代码块。 对于一个完整的异常处理流程而言,try块是必须的,try块后可以紧跟一个或...		
关于机器人的常用术语_CWH的博客_机器人专业术语	3-30	
关于机器人的一些常用术语: S:即时定位与地图构建 (SLAM:Simultaneous Localization and Mapping)。是行走机器人在运动...		
dubbo的工作原理	聆听梦飞扬的博客	1万+
转载地址：https://blog.csdn.net/A_BlackMoon/article/details/85609328 dubbo的工作原理 1、面试题 说一下的dubbo的工...		
Kafka最全面试题整理	uncle_胡的博客	1614
一、开篇 我最近在看kafka的面试题，有我自己在网上找的题目，但是有的面试题挺好的，但是没有答案，发现有一篇很好...		
关键字:throws,throw,try,catch,finally分别代表什么意...	3-24	
关键字:throws,throw,try,catch,finally分别代表什么意义? throws是获取异常 throw是抛出异常 try是将会发生异常的语句括起...		
Docker到底是什么?为什么它这么火!_CWH615的博客	3-17	
Docker到底是什么?为什么它这么火! Docker到底是什么?Docker为什么它这么火! 09-30 为什么要使用Docker Just for fun 7...		
Dubbo原理简单分析	逍遥飞鹤的专栏	3万+
alibaba有好几个分布式框架，主要有： 进行远程调用(类似于RMI的这种远程调用)的(dubbo、hsf)，jms消息服务(napoli、n...		
Kafka底层原理架构	qichangjian的博客	6233
1、Kafka 核心组件概述 Kafka 是 LinkedIn 用于日志处理的分布式消息队列，同时支持离线和在线日志处理。 Kafka 对消息...		
kafka原理及面试套路	weixin_35720385的博客	3752
kafka原理及面试套路一、 面试： 1、列举kafka的使用场景2、Kafka消息是采用Pull模式，还是Push模式? 3、Kafka 与传...		
kafka的工作原理分析（一）	嘎嘎的博客	1万+
一、kafka中的topic与partition分区 首先需要了解kafka中基本的组成部分。在 kafka 中， topic 是一个存储消息的逻辑概念...		
Kafka的ACK机制有三种，是哪三种	linux下安装配置 MySQL5.6	1万+
Kafka producer有三种ack机制 初始化producer时在config中进行配置 0 意味着producer不等待broker同步完成的确认，继...		
Kafka使用入门教程	下雨天_的专栏	7346
介绍 Kafka是一个分布式的、可分区的、可复制的消息系统。它提供了普通消息系统的功能，但具有自己独特的设计。这个...		
Kafka集群原理讲解及分区机制	weixin_43866709的博客	7962
首先我们要理解Kafka和zookeeper的关系： Kafka的一些基本概念： Broker： 安装Kafka服务的那台集群就是一个broker...		
kafka原理及常见问题总结	hw120219的博客	969
broker：（kafka的节点，也就是服务器） 1.接受来自生产者的消息，为消息设置偏移量，并提交消息到磁盘保存。2.为消费...		
大白话 kafka 架构原理	Felix	590
大数据时代来临，如果你还不知道Kafka那就真的out了！据统计，有三分之一的世界财富500强企业正在使用Kafka，包括...		
©2020 CSDN 皮肤主题: 大白 设计师: CSDN官方博客 返回首页		

关于我们 招贤纳士 广告服务 开发助手 400-660-0108 kefu@csdn.net 在线客服 工作时间 8:30-22:00
公安备案号11010502030143 京ICP备19004658号 京网文〔2020〕1039-165号 经营性网站备案信息 北京互联网违法和不良信息举报中心
网络110报警服务 中国互联网举报中心 家长监护 Chrome商店下载 ©1999-2021北京创新乐知网络技术有限公司 版权与免责声明 版权申诉



举报