参考：<http://blog.csdn.net/lizhitao/article/details/23743821>

**apache Kafka概要介绍**

apache kafka参考

[http://kafka.apache.org/documentation.html](http://kafka.apache.org/documentation.html#brokerconfigs)

消息队列分类：

 点对点：

消息生产者生产消息发送到queue中，然后消息消费者从queue中取出并且消费消息。这里要注意：

* 消息被消费以后，queue中不再有存储，所以消息消费者不可能消费到已经被消费的消息。
* Queue支持存在多个消费者，但是对一个消息而言，只会有一个消费者可以消费。

发布/订阅

消息生产者（发布）将消息发布到topic中，同时有多个消息消费者（订阅）消费该消息。和点对点方式不同，发布到topic的消息会被所有订阅者消费。

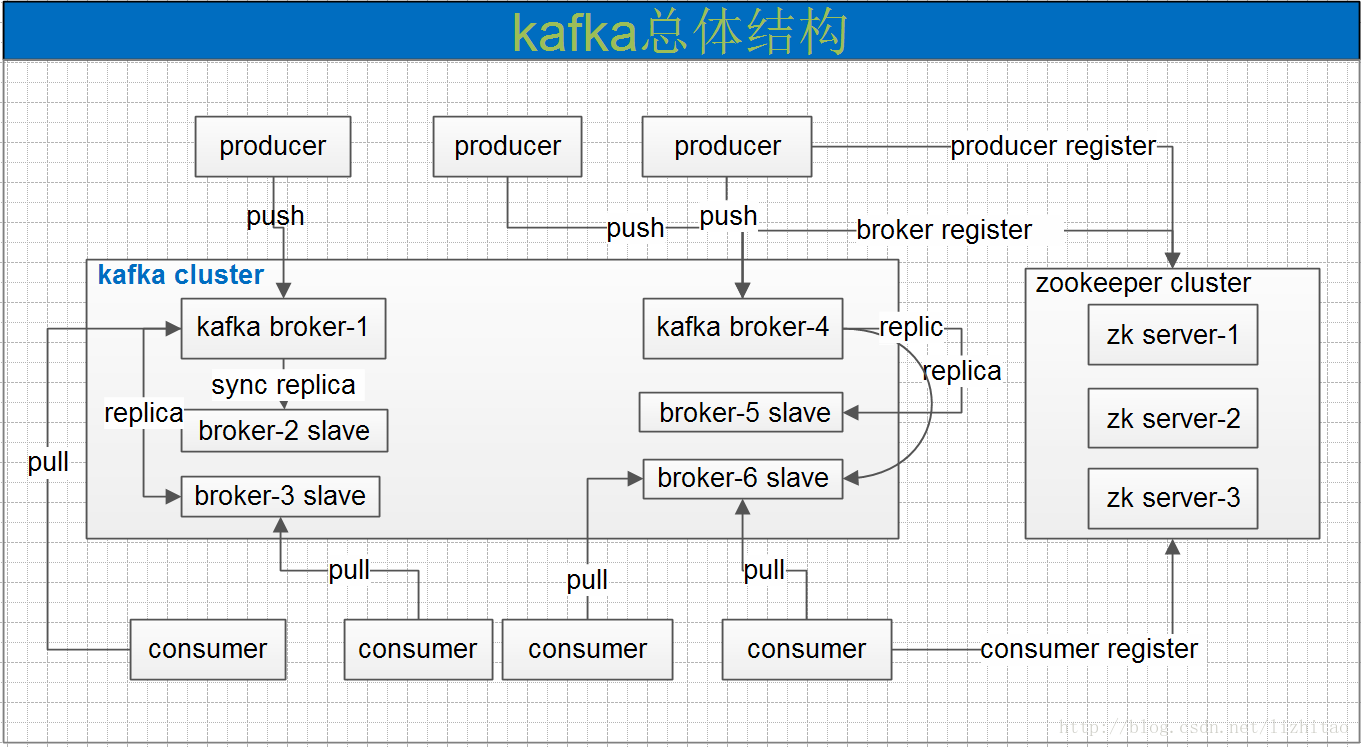
kafka消息队列调研

背景介绍

kafka是最初由Linkedin公司开发，使用Scala语言编写，Kafka是一个分布式、分区的、多副本的、多订阅者的日志系统(分布式MQ系统)，可以用于web/nginx日志，搜索日志，监控日志，访问日志等等。

kafka目前支持多种客户端语言：java，python，c++，php等等。

总体结构：



kafka名词解释和工作方式：

* Producer ：消息生产者，就是向kafka broker发消息的客户端。
* Consumer ：消息消费者，向kafka broker取消息的客户端
* Topic ：咋们可以理解为一个队列。
* Consumer Group （CG）：这是kafka用来实现一个topic消息的广播（发给所有的consumer）和单播（发给任意一个consumer）的手段。一个topic可以有多个CG。topic的消息会复制（不是真的复制，是概念上的）到所有的CG，但每个CG只会把消息发给该CG中的一个consumer。如果需要实现广播，只要每个consumer有一个独立的CG就可以了。要实现单播只要所有的consumer在同一个CG。用CG还可以将consumer进行自由的分组而不需要多次发送消息到不同的topic。
* Broker ：一台kafka服务器就是一个broker。一个集群由多个broker组成。一个broker可以容纳多个topic。
* Partition：为了实现扩展性，一个非常大的topic可以分布到多个broker（即服务器）上，在每个broker上，一个topic可以分为多个partition，每个partition是一个有序的队列。partition中的每条消息都会被分配一个有序的id（offset）。kafka只保证按一个partition中的顺序将消息发给consumer，不保证一个topic的整体（多个partition间）的顺序。
* Offset：kafka的存储文件都是按照offset.kafka来命名，用offset做名字的好处是方便查找。例如你想找位于2049的位置，只要找到2048.kafka的文件即可。当然the first offset就是00000000000.kafka

 kafka特性：

* 通过O(1)的磁盘数据结构提供消息的持久化，这种结构对于即使数以TB的消息存储也能够保持长时间的稳定性能。
* 高吞吐量：即使是非常普通的硬件kafka也可以支持每秒数十万的消息。
* 支持同步和异步复制两种HA
* Consumer客户端pull，随机读,利用sendfile系统调用，zero-copy ,批量拉数据
* 消费状态保存在客户端
* 消息存储顺序写
* 数据迁移、扩容对用户透明
* 支持Hadoop并行数据加载。
* 支持online和offline的场景。
* 持久化：通过将数据持久化到硬盘以及replication防止数据丢失。
* scale out：无需停机即可扩展机器，自动热扩展。
* 定期删除机制，支持设定partitions的segment file保留时间。

可靠性（一致性)

kafka(MQ)要实现从producer到consumer之间的可靠的消息传送和分发。传统的MQ系统通常都是通过broker和consumer间的确认（ack）机制实现的，并在broker保存消息分发的状态。

即使这样一致性也是很难保证的（参考原文）。kafka的做法是由consumer自己保存状态，也不要任何确认。这样虽然consumer负担更重，但其实更灵活了。

因为不管consumer上任何原因导致需要重新处理消息，都可以再次从broker获得。

kafak系统扩展性

kafka使用zookeeper来实现动态的集群扩展，不需要更改客户端（producer和consumer）的配置。broker会在zookeeper注册并保持相关的元数据（topic，partition信息等）更新。

连接到zookeeper查看节点：

[zk: localhost:2181(CONNECTED) 1] ls /

[cluster, controller\_epoch, controller, brokers, zookeeper, admin, isr\_change\_notification, consumers, config]

而客户端会在zookeeper上注册相关的watcher。一旦zookeeper发生变化，客户端能及时感知并作出相应调整。这样就保证了添加或去除broker时，各broker间仍能自动实现负载均衡。

kafka设计目标

高吞吐量是其核心设计之一。

* 数据磁盘持久化：消息不在内存中cache，直接写入到磁盘，充分利用磁盘的顺序读写性能。
* zero-copy：减少IO操作步骤。
* 支持数据批量发送和拉取。
* 支持数据压缩。
* Topic划分为多个partition，提高并行处理能力。

Producer负载均衡和HA机制

* producer根据用户指定的算法，将消息发送到指定的partition。
* 存在多个partiiton，每个partition有自己的replica（me:即每个partition有自己的副本集，以便于负载和保持可靠性），每个replica分布在不同的Broker节点上以便于负载。
* 多个partition需要选取出lead partition，lead partition负责读写，并由zookeeper负责fail over。
* 通过zookeeper管理broker与consumer的动态加入与离开。

Consumer的pull机制

由于kafka broker会持久化数据，broker没有cache压力，因此，consumer比较适合采取pull的方式消费数据，具体特点如下：

* 简化kafka设计，降低了难度。
* Consumer根据消费能力自主控制消息拉取速度。
* consumer根据自身情况自主选择消费模式，例如批量，重复消费，从制定partition或位置(offset)开始消费等.

Consumer与topic关系以及机制

本质上kafka只支持Topic.每个consumer属于一个consumer group;反过来说,每个group中可以有多个consumer.对于Topic中的一条特定的消息,  
只会被订阅此Topic的每个group中的一个consumer消费,此消息不会发送给一个group的多个consumer;那么一个group中所有的consumer将会交错的消费整个Topic.  
如果所有的consumer都具有相同的group,这种情况和JMS queue模式很像;消息将会在consumers之间负载均衡.  
如果所有的consumer都具有不同的group,那这就是"发布-订阅";消息将会广播给所有的消费者.

在kafka中,一个partition中的消息只会被group中的一个consumer消费(同一时刻);每个group中consumer消息消费互相独立;我们可以认为一个group是一个"订阅"者,

一个Topic中的每个partions,只会被一个"订阅者"中的一个consumer消费,不过一个consumer可以同时消费多个partitions中的消息.

kafka只能保证一个partition中的消息被某个consumer消费时是顺序的.事实上,从Topic角度来说,当有多个partitions时,消息仍不是全局有序的.

通常情况下,一个group中会包含多个consumer,这样不仅可以提高topic中消息的并发消费能力,而且还能提高"故障容错"性,如果group中的某个consumer失效,

那么其消费的partitions将会有其他consumer自动接管.kafka的设计原理决定,对于一个topic,同一个group中不能有多于partitions个数的consumer同时消费,

否则将意味着某些consumer将无法得到消息.

Producer均衡算法

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".事实上,消息被路由到哪个partition上由producer客户端决定.  
比如可以采用"random""key-hash""轮询"等,如果一个topic中有多个partitions,那么在producer端实现"消息均衡分发"是必要的.  
在producer端的配置文件中,开发者可以指定partition路由的方式.

**Consumer均衡算法**

当一个group中,有consumer加入或者离开时,会触发partitions均衡.均衡的最终目的,是提升topic的并发消费能力.  
1) 假如topic1,具有如下partitions: P0,P1,P2,P3  
2) 加入group中,有如下consumer: C0,C1  
3) 首先根据partition索引号对partitions排序: P0,P1,P2,P3  
4) 根据consumer.id排序: C0,C1  
5) 计算倍数: M = [P0,P1,P2,P3].size / [C0,C1].size,本例值M=2(向上取整)  
6) 然后依次分配partitions: C0 = [P0,P1],C1=[P2,P3],即Ci = [P(i \* M),P((i + 1) \* M -1)]

kafka broker集群内broker之间replica机制

kafka中,replication策略是基于partition,而不是topic，即partition有多个副本集，当然这也决定了topic有多个副本集。;kafka将每个partition数据复制到多个server上（以便容错和负载）,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.(备注:不同于其他分布式存储,比如hbase需要"多数派"存活才行)

k**afka判定一个follower存活与否的条件有2个**:

1) follower需要和zookeeper保持良好的链接

2) 它必须能够及时的跟进leader,不能落后太多.

如果同时满足上述2个条件,那么leader就认为此follower是"活跃的".如果一个follower失效(server失效)或者落后太多,

leader将会把它从同步列表中移除[备注:如果此replicas落后太多,它将会继续从leader中fetch数据,直到足够up-to-date,

然后再次加入到同步列表中;kafka不会更换replicas宿主!因为"同步列表"中replicas需要足够快,这样才能保证producer发布消息时接受到ACK的延迟较小。

**当leader失效时**,需在followers中选取出新的leader,可能此时follower落后于leader,因此需要选择一个"up-to-date"的follower.kafka中leader选举并没有采用"投票多数派"的算法,

因为这种算法对于"网络稳定性"/"投票参与者数量"等条件有较高的要求,而且kafka集群的设计,还需要容忍N-1个replicas失效.对于kafka而言,

每个partition中所有的replicas信息都可以在zookeeper中获得,那么选举leader将是一件非常简单的事情.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.

在整个集群中,只要有一个replicas存活,那么此partition都可以继续接受读写操作.

*总结:*

    1) Producer端直接连接broker.list列表,从列表中返回TopicMetadataResponse,该Metadata包含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,以及监控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.

性能测试

目前我已经在虚拟机上做了性能测试。

测试环境：cpu: 双核   内存 ：2GB   硬盘：60GB

| 测试指标 | 性能相关说明 | 结论 |
| --- | --- | --- |
| 消息堆积压力测试 | 单个kafka broker节点测试，启动一个kafka broker和Producer，Producer不断向broker发送数据，  直到broker堆积数据为18GB为止(停止Producer运行)。启动Consumer，不间断从broker获取数据，  直到全部数据读取完成为止，最后查看Producer==Consumer数据，没有出现卡死或broker不响应现象 | 数据大量堆积不会出现broker卡死  或不响应现象 |
| 生产者速率 | 1.200byte/msg,4w/s左右。2.1KB/msg,1w/s左右 | 性能上是完全满足要求，其性能主要由磁盘决定 |
| 消费者速率 | 1.200byte/msg,4w/s左右。2.1KB/msg,1w/s左右 | 性能上是完全满足要求，其性能主要由磁盘决定 |