**Kafka简介**



Kafka是一种高吞吐量的[分布式](http://baike.baidu.com/view/402382.htm)发布订阅消息系统，使用Scala编写。

对于熟悉JMS（Java Message Service）规范的同学来说，消息系统已经不是什么新概念了(例如ActiveMQ，RabbitMQ等)。

Kafka拥有作为一个消息系统应该具备的功能，但是确有着独特的设计。可以这样来说，Kafka借鉴了JMS规范的思想，但是确并没有完全遵循JMS规范。

kafka是一个分布式的，分区的消息(官方称之为commit log)服务。它提供一个消息系统应该具备的功能，但是确有着独特的设计。

首先，让我们来看一下基础的消息(Message)相关术语：

    **Topic:**Kafka按照Topic分类来维护消息

    **Producer：** 我们将发布(publish)消息到Topic的进程称之为生产者(producer)

    **Consumer：** 我们将订阅(subscribe)Topic并且处理Topic中消息的进程称之为消费者(consumer)

   **Broker：** Kafka以集群的方式运行，集群中的每一台服务器称之为一个代理(broker)。

因此，从一个较高的层面上来看，producers通过网络发送消息到Kafka集群，然后consumers来进行消费，如下图：

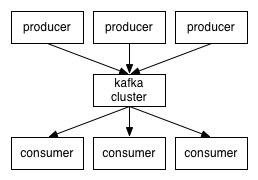


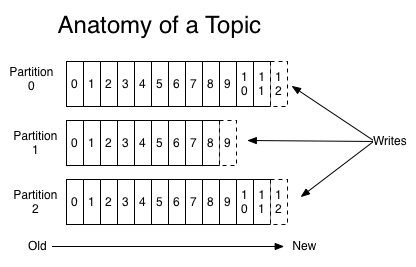
Image.png

服务端(brokers)和客户端(producer、consumer)之间通信通过TCP协议来完成。我们为Kafka提供了一个Java客户端，但是也可以使用其他语言编写的客户端。

**Topic和Log**

让我们首先深入理解Kafka提出一个高层次的抽象概念-Topic。

可以理解Topic是一个类别的名称，所有的message发送到Topic下面。对于每一个Topic，kafka集群按照如下方式维护一个分区(Partition,可以将消息就理解为一个队列Queue)日志文件:



partition是一个有序的message序列，这些message按顺序添加到一个叫做commit log的文件中。每个partition中的消息都有一个唯一的编号，称之为offset，用来唯一标示某个分区中的message。

提示：每个partition，都对应一个commit-log。一个partition中的message的offset都是唯一的，但是不同的partition中的message的offset可能是相同的。

kafka集群，在配置的时间范围内，维护所有的由producer生成的消息，而不管这些消息有没有被消费。例如日志保留( log retention )时间被设置为2天。kafka会维护最近2天生产的所有消息，而2天前的消息会被丢弃。kafka的性能与保留的数据量的大小没有关系，因此保存大量的数据(日志信息)不会有什么影响。

每个consumer是基于自己在commit log中的消费进度(offset)来进行工作的。在kafka中，offset由consumer来维护：一般情况下我们按照顺序逐条消费commit log中的消息，当然我可以通过指定offset来重复消费某些消息，或者跳过某些消息。

这意味kafka中的consumer对集群的影响是非常小的，添加一个或者减少一个consumer，对于集群或者其他consumer来说，都是没有影响的，因此每个consumer维护各自的offset。

对log进行分区（partitioned），有以下目的。首先，当log文件大小超过系统文件系统的限制时，可以自动拆分。每个partition对应的log都受到所在机器的文件系统大小的限制，但是一个Topic中是可以有很多分区的，因此可以处理任意数量的数据。另一个方面，是为了提高并行度。

**Distribution**

log的partitions分布在kafka集群中不同的broker上，每个broker可以请求备份其他broker上partition上的数据。kafka集群支持配置一个partition备份的数量。

针对每个partition，都有一个broker起到“leader”的作用，0个多个其他的broker作为“follwers”的作用。leader处理所有的针对这个partition的读写请求，而followers被动复制leader的结果。如果这个leader失效了，其中的一个follower将会自动的变成新的leader。**每个broker都是自己所管理的partition的leader，同时又是其他broker所管理partitions的followers，kafka通过这种方式来达到负载均衡**。

**Producers**

生产者将消息发送到topic中去，同时负责选择将message发送到topic的哪一个partition中。通过round-robin做简单的负载均衡。也可以根据消息中的某一个关键字来进行区分。通常第二种方式使用的更多。

**Consumers**

传统的消息传递模式有2种：队列( [queuing](http://en.wikipedia.org/wiki/Message_queue))和（ [publish-subscribe](http://en.wikipedia.org/wiki/Publish%E2%80%93subscribe_pattern)）。

在queuing模式中，多个consumer从服务器中读取数据，消息只会到达一个consumer。在 publish-subscribe 模型中，消息会被广播给所有的consumer。Kafka基于这2种模式提供了一种consumer的抽象概念：consumer group。

每个consumer都要标记自己属于哪一个consumer group。发布到topic中的message中message会被传递到consumer group中的一个consumer 实例。consumer实例可以运行在不同的进程上，也可以在不同的物理机器上。

如果所有的consumer都位于同一个consumer group 下，这就类似于传统的queue模式，并在众多的consumer instance之间进行负载均衡。

如果所有的consumer都有着自己唯一的consumer group，这就类似于传统的publish-subscribe模型。

更一般的情况是，通常一个topic会有几个consumer group，每个consumer group都是一个逻辑上的订阅者（ logical subscriber ）。每个consumer group由多个consumer instance组成，从而达到可扩展和容灾的功能。这并没有什么特殊的地方，仅仅是将publish-subscribe模型中的运行在单个进程上的consumers中的consumer替换成一个consumer group。如下图所示：

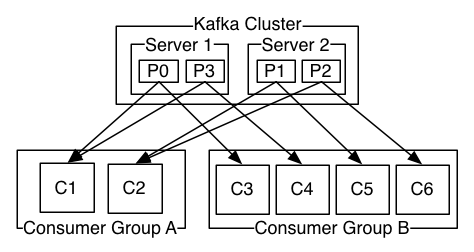


Image.png

说明：由2个broker组成的kafka集群，总共有4个Parition(P0-P3)。这个集群由2个Consumer Group， A有2个 consumer instances ，而B有四个.

**消费顺序**

Kafka比传统的消息系统有着更强的**顺序保证**。在传统的情况下，服务器按照顺序保留消息到队列，如果有多个consumer来消费队列中的消息，服务器 会接受消息的顺序向外提供消息。但是，尽管服务器是按照顺序提供消息，但是消息传递到每一个consumer是异步的，这可能会导致先消费的 consumer获取到消息时间可能比后消费的consumer获取到消息的时间长，导致不能保证顺序性。这表明，当进行并行的消费的时候，消息在多个 consumer之间可能会失去顺序性。消息系统通常会采取一种“ exclusive consumer”的概念，来确保同一时间内只有一个consumer能够从队列中进行消费，但是这实际上意味着在消息处理的过程中是不支持并行的。

Kafka在这方面做的更好。通过Topic中并行度的概念，即partition，Kafka可以同时提供顺序性保证和多个consumer同时消费时的负载均衡。实现的原理是通过将一个topic中的partition分配给一个consumer group中的不同consumer instance。通过这种方式，我们可以保证一个partition在同一个时刻只有一个consumer instance在消息，从而保证顺序。虽然一个topic中有多个partition，但是一个consumer group中同时也有多个consumer instance，通过合理的分配依然能够保证负载均衡。需要注意的是，一个consumer group中的consumer instance的数量不能比一个Topic中的partition的数量多。

Kafka只在partition的范围内保证消息消费的局部顺序性，不能在同一个topic中的多个partition中保证总的消费顺序性。通常来说，这已经可以满足大部分应用的需求。但是，如果的确有在总体上保证消费的顺序的需求的话，那么我们可以通过将topic的partition数量设置为1，将consumer group中的consumer instance数量也设置为1.

**Guarantees**

从较高的层面上来说的话，Kafka提供了以下的保证：

发送到一个Topic中的message会按照发送的顺序添加到commit log中。意思是，如果消息 M1，M2由同一个producer发送，M1比M2发送的早的话，那么在commit log中，M1的offset就会commit 2的offset小。

一个consumer在commit log中可以按照发送顺序来消费message

如果一个topic的备份因子( replication factor )设置为N，那么Kafka可以容忍N-1一个服务器的失败，而存储在commit log中的消息不会丢失。

**kafka集群搭建与使用**

**安装前的环境准备**

由于Kafka是用Scala语言开发的，运行在JVM上，因此在安装Kafka之前需要先安装JDK。

# yum install java-1.8.0-openjdk\* -y

kafka依赖zookeeper，所以需要先安装zookeeper

# wget <http://mirror.bit.edu.cn/apache/zookeeper/stable/zookeeper-3.4.12.tar.gz>

# tar -zxvf [zookeeper-3.4.12.tar.gz](http://mirror.bit.edu.cn/apache/zookeeper/stable/zookeeper-3.4.12.tar.gz)

# cd zookeeper-3.4.12

# cp conf/zoo\_sample.cfg conf/zoo.cfg

启动zookeeper

# bin/zkServer.sh start conf/zoo.cfg &

# bin/zkCli.sh

# ls / #查看zk的根目录相关节点

**第一步：下载安装包**

下载1.1.0 release版本，并解压：

# wget <http://mirrors.tuna.tsinghua.edu.cn/apache/kafka/1.1.0/kafka_2.11-1.1.0.tgz>

# tar -xzf kafka\_2.11-1.1.0.tgz

# cd kafka\_2.11-1.1.0

**第二步：启动服务**

现在来启动kafka服务：

启动脚本语法：kafka-server-start.sh [-daemon] server.properties

可以看到，server.properties的配置路径是一个强制的参数，-daemon表示以后台进程运行，否则ssh客户端退出后，就会停止服务。(注意，在启动kafka时会使用linux主机名关联的ip地址，所以需要把主机名配置到本地host里，用vim /etc/hosts)

# bin/kafka-server-start.sh -daemon config/server.properties

我们进入zookeeper目录通过zookeeper客户端查看下zookeeper的目录树

# bin/zkCli.sh

# ls / #查看zk的根目录kafka相关节点

# ls /brokers/ids #查看kafka节点

**第三步：创建主题**

现在我们来创建一个名字为“test”的Topic，这个topic只有一个partition，并且备份因子也设置为1：

# bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic test

现在我们可以通过以下命令来查看kafka中目前存在的topic

# bin/kafka-topics.sh --list --zookeeper localhost:2181

除了我们通过手工的方式创建Topic，我们可以配置broker，当producer发布一个消息某个指定的Topic，但是这个Topic并不存在时，就自动创建。

**第四步：发送消息**

kafka自带了一个producer命令客户端，可以从本地文件中读取内容，或者我们也可以以命令行中直接输入内容，并将这些内容以消息的形式发送到kafka集群中。在默认情况下，每一个行会被当做成一个独立的消息。

首先我们要运行发布消息的脚本，然后在命令中输入要发送的消息的内容：

# bin/kafka-console-producer.sh --broker-list localhost:9092 --topic test

>this is a msg

>this is a another msg

**第五步：消费消息**

对于consumer，kafka同样也携带了一个命令行客户端，会将获取到内容在命令中进行输出：

# bin/kafka-console-consumer.sh --zookeeper localhost:2181 --topic test --from-beginning

如果你是通过不同的终端窗口来运行以上的命令，你将会看到在producer终端输入的内容，很快就会在consumer的终端窗口上显示出来。

以上所有的命令都有一些附加的选项；当我们不携带任何参数运行命令的时候，将会显示出这个命令的详细用法。

**第六步：kafka集群配置**

到目前为止，我们都是在一个单节点上运行broker，这并没有什么意思。对于kafka来说，一个单独的broker意味着kafka集群中只有一个接点。要想增加kafka集群中的节点数量，只需要多启动几个broker实例即可。为了有更好的理解，现在我们在一台机器上同时启动三个broker实例。

首先，我们需要建立好其他2个broker的配置文件：

# cp config/server.properties config/server-1.properties

# cp config/server.properties config/server-2.properties

配置文件的内容分别如下：

config/server-1.properties:

    **broker.id=1**

**listeners=**[**PLAINTEXT://:9093**](plaintext://:9093)

**log.dir=/tmp/kafka-logs-1**

config/server-2.properties:

    **broker.id=2**

**listeners=**[**PLAINTEXT://:9094**](plaintext://:9094)

**log.dir=/tmp/kafka-logs-2**

broker.id属性在kafka集群中必须要是唯一的。我们需要重新指定port和log目录，因为我们是在同一台机器上运行多个实例。如果不进行修改的话，consumer只能获取到一个instance实例的信息，或者是相互之间的数据会被影响。

目前我们已经有一个zookeeper实例和一个broker实例在运行了，现在我们只需要在启动2个broker实例即可：

# bin/kafka-server-start.sh config/server-1.properties &

# bin/kafka-server-start.sh config/server-2.properties &

现在我们创建一个新的topic，备份因子设置为3：

# bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 3 --partitions 1 --topic my-replicated-topic

现在我们已经有了集群，并且创建了一个3个备份因子的topic，但是到底是哪一个broker在为这个topic提供服务呢(因为我们只有一个分区，所以肯定同时只有一个broker在处理这个topic)？

# bin/kafka-topics.sh --describe --zookeeper localhost:2181 --topic my-replicated-topic

clipboard.png

以下是输出内容的解释，第一行是所有分区的概要信息，之后的每一行表示每一个partition的信息。因为目前我们只有一个partition，因此关于partition的信息只有一行。

leader节点负责给定partition的所有读写请求。如果一个topic有多个partitions，那么每个节点都会其中一部分partition的leader。

replicas 表示某个partition在哪几个broker上存在备份。不管这个几点是不是”leader“，甚至这个节点挂了，也会列出。

isr 是replicas的一个子集，它只列出当前还存活着的，并且备份了该partition的节点。

现在我们的案例中，0号节点是leader，即使用server.properties启动的那个进程。

我们可以运行相同的命令查看之前创建的名称为”test“的topic

# bin/kafka-topics.sh --describe --zookeeper localhost:2181 --topic test

clipboard.png

没有什么值得惊讶的地方，我们之前设置了topic的partition数量为1，备份因子为1，因此显示就如上所示了。

现在我们向新建的topic中发送一些message：

# bin/kafka-console-producer.sh --broker-list localhost:9092 --topic my-replicated-topic

>my test msg 1

>my test msg 2

现在开始消费：

# bin/kafka-console-consumer.sh --zookeeper localhost:2181 --from-beginning --topic my-replicated-topic

my test msg 1

my test msg 2

现在我们来测试我们容错性，因为broker0目前是leader，所以我们要将其kill

# ps -ef | grep server.properties

# kill -9 1177

现在再执行命令：

# bin/kafka-topics.sh --describe --zookeeper localhost:2181 --topic my-replicated-topic

clipboard.png

我们可以看到，leader节点已经变成了broker 2.要注意的是，在Isr中，已经没有了0号节点。leader的选举也是从ISR中进行的。

此时，我们依然可以 消费新消息：

# bin/kafka-console-consumer.sh --zookeeper localhost:2181 --from-beginning --topic my-replicated-topic

my test msg 1

my test msg 2