Table of Contents

Kafka中文文档

This file file serves as your book's preface, a great place to describe your book's content and ideas.

版本

Kafka 0.10.0

说明

此文档是对照Apache Kafka的英文文档翻译过来，网址为kafka

第一章 开始

1.1 介绍

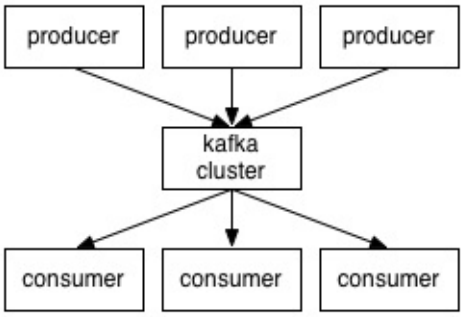
Kafka是一个分布式,分区,重复提交的日志服务。它提供了消息系统的功能,但是采用了独特的设计。

这意味着什么呢?

首先，让我们回顾一些基本的消息术语：

- Kafka掌管的类别消息叫做主题(topics)
- 我们将发布消息到Kafka主题中的处理叫做生产者(producers)
- 我们将订阅主题以及处理发布的消息叫做消费者(consumers)
- Kafka以集群运行在一个或者多个服务器上,每个服务器叫做broker

在一个高水平上,生产者通过网络发送消息到Kafka集群中,之后这些消息服务于消费者,如下图所示：

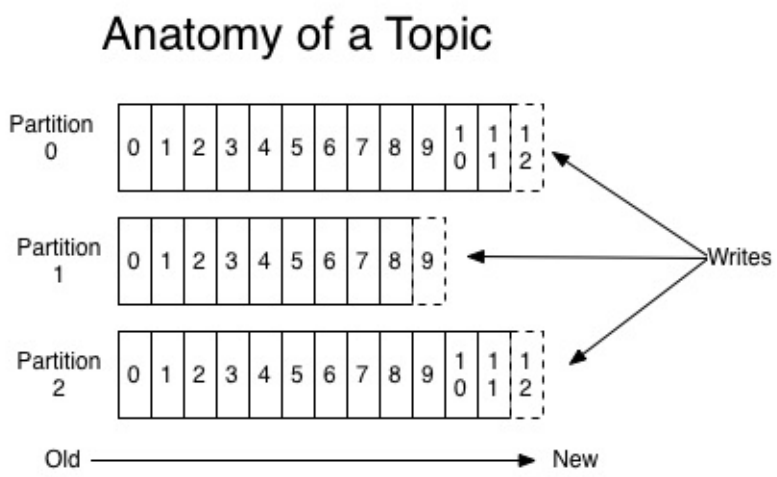


客户端与服务端的通信采用简单,高性能且与语言无关的TCP协议。我们提供了一个Java客户端,但是客户端也可以采用其他语言。

主题和日志(Topics and Logs)

首先我们先深入了解Kafka提供的高水平抽象概念-主题

主题是一个类别或者被发布消息的名称。对于每个主题,Kafka集群采用分区日志的方式来管理，如下图所示：



每个分区都是一个按顺序，不变的消息序列被持续添加的提交日志。分区中的每个消息都被分配给了一个序列数字，即偏移量(offset),偏移量在分区中唯一的标识了一个消息。

Kafka集群根据配置文件中的时间来保留所有发布的消息，而不是根据它们是否被消费。例如：日志保留时间被设置为两天，那么一个消息被发布两天之内都是可以被用来消费的，过了时间后Kafka就会释放对应的空间。即使对于大数据量，Kafka的性能仍然可以保证为常数，所以保留大量的数据不是一个问题。

实际上，每个消费者保留的元数据是这个消费者在日志中的位置，即“偏移量”(offset).偏移量是被消费者控制的：通常情况下，一个消费者在读取消息时按照线性来读取，但是实际上，消费者可以控制位置，从而可以以它想要的顺序来消费数据。比如一个消费者可以将偏移量重置为一个老的偏移量，从而可以重新处理。

这些特点的组合意味着Kafka消费者是非常低廉(cheap)的，可以在不影响集群或者其他消费者的情况下随意的去除或者添加消费者。例如，你可以使用我们的命令行工具"tail"显示任意主题下的内容，同时不改变这个内容，即使这个内容已经被存在的消费者在消费。

日志上的分区有几个目的。首先，它们允许日志扩展的大小超过一个服务器的大小。每个单独的分区一定要被服务器承载，但是一个主题可以有很多分区，所以它能够处理任意数量的数据。其次，它们是作为并行运算的单位，后面对这部分会有更加详细的介绍。

分布式

日志的分区分布在Kafka集群上的服务器中，每个服务器处理数据以及共享分区上的内容。每个分区都可以被复制到一定数量的服务器上，这个数量可以进行配置，从而提高了容错能力。

每个分区都有一个服务器作为leader，零个或者多个其他服务器作为followers。leader处理针对这个分区所有的读以及写请求，同时followers被动的复制leader。如果leader宕机了，其中的一个followers将会自动成为新的leader。每个服务器作为一些分区的leader，同时也作为其他分区的follower，所以Kafka集群中的负载平衡的很好。

生产者

生产者发布数据到它们选定的主题上。生产者负责选择哪个消息发送到指定主题的分区上。可以采用轮询的方式选择分区来简单的均衡负载或者根据一些语义函数（比如基于消息中的某些关键词）来选择分区。稍后介绍分区的更多用法。

消费者

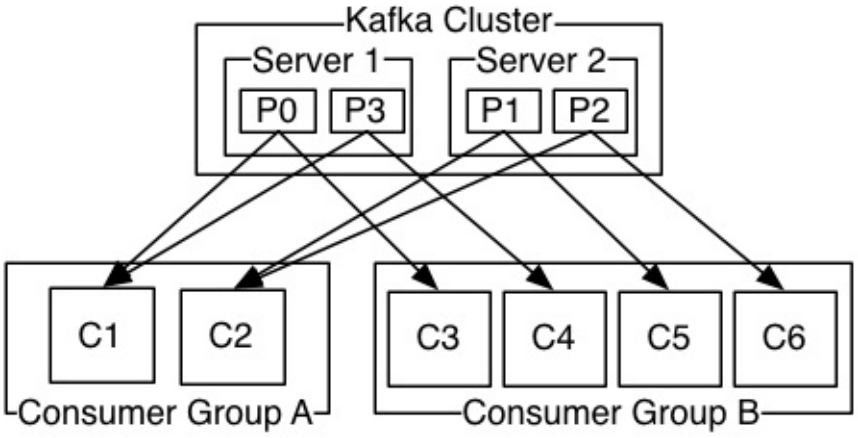
消息处理通常有两种模式：队列和发布-订阅。在队列模式中，一组消费者可能从服务器读取消息，每个消息被其中一个消费者消费。在发布-订阅模式中，消息是广播到全部的消费者中。Kafka提供了一个单独的消费者抽象用来概括这两种模式-消费组

消费者用一个消费组名称来标识自己，一个主题中的消息被发送到一个订阅此主题消费组中的一个消费组实例上。消费者实例可以在单独的进程或者单独的机器上。

如果全部的消费者实例都同一个组中，那么这种工作方式就变成了传统的队列均衡负载模式。

如果所有的消费者存在不同的组中，那么这种方式就是发布-订阅模式，即所有的消息被广播给所有的消费者。

更一般的情况，我们建立了拥有少量数量消费组的主题，每一个组都是一个“逻辑订阅者”。为了伸展性和容错能力，每个组都是由很多消费者实例组成。这无非是发布-订阅模式，其中的订阅者是一个消费者簇而不是单一的进程。



A two server Kafka cluster hosting four partitions (P0-P3) with two consumer groups. Consumer group A has two consumer instances and group B has four.

同样，Kafka相较于传统的消息系统有很强的顺序保证。

传统的队列模式保持了消息在服务器上的顺序，如果有大量的消费者从队列中消费，那么服务器将会按照它们存储的顺序来处理消息。尽管服务器是按序处理消息，但是消息是异步的传递到消费者中，所以它们可能不是按序到达不同的消费者中。这实际上意味着在并行处理

中，消息的顺序不能保证。消息系统可以仅仅让一个进程来消费队列中的消息来保证顺序，但是这也就意味着处理过程不是并行的。

Kafka做的比较好。依靠并行的概念-分区以及主题，Kafka能够提供顺序保证并且在一组消费者处理过程中均衡负载。将主题中的分区分给一个消费者组中的消费者，每个分区是被组中的一个准确的消费者进行消费，从而能够达到这一目的。我们要确保消费者仅仅是读取分区以及按顺序的消费数据。因为有很多分区，所以这仍然能够在存在大量消费者实例的情况下均衡负载。但是请注意，在一个消费者组中的消费者实例数量不能超过分区的数量。

Kafka仅仅提供在一个分区内消息的顺序，而不是在一个主题的不同分区上。分区上的顺序保证以及依靠键来分区数据的能力，这两者的结合可以满足大多数的应用。然而，如果你保证消息的顺序，你可以指定一个主题只有一个分区，但是这也就意味着在每个消费者组中仅仅只能有一个消费者过程。

保证

高水平的Kafka提供了以下的保证：

- 生产者发送到指定主题分区中的消息将会被按照它们被发送的顺序进行添加。也就是说，如果一个消费者发送了两个消息，消息M1先被发送，M2后被发送，那么M1的偏移量将会小于M2的偏移量并且在日志中会出现的更靠前
- 一个消费者实例是以消息被存储在日志中的顺序来看待消息的。
- 如果一个主题有复制系数N，即使有N-1个服务器宕机了，我们也可以在不丢失任何已经被提交到日志中的消息。

有关于这些保证的更多细节在文档的设计部分会给出。

1.2 应用场景

这里有一些使用Apache Kafka的流行的例子描述。若想对大量实际领域中的应用有个概览，可以看看这篇博客

消息处理

Kafka可以替换传统的消息代理。由于各种各样的原因（与数据生成这解耦处理过程，缓存没有处理的消息等等）消息代理有广泛的用处。对比其他的消息系统，由于Kafka有更好的吞吐量，内建的分区，复制以及容错能力，所以Kafka对于大规模的消息处理应用都是很好的解决方案。

根据我们的经验，消息系统的使用中常常有较低的吞吐量，但是需要端到端有较低的延迟并且常常依靠很强的耐用性，而这些Kafka都能够保证。

在这一领域，Kafka可以媲美传统的消息系统例如ActiveMQ和RabbitMQ。

网站活动跟踪

Kafka最初是用于构建一个用户活动跟踪管道，作为一组实时发布-订阅饲料(feeds).这意味着站点的活动(页面浏览，搜索或者用户其他的动作)被发布到一个中心主题，每个活动类型有一个主题。这些消息可以一系列的用户订阅使用，比如实时处理，实时监测，Hadoop加载或者离线数据仓库系统来提供离线处理以及报告。

由于每个用户浏览的界面所产生的活动消息数量较大，用户跟踪常常是大规模的数据量。

度量

Kafka通常用于操作监测数据的处理。包括从分布式应用的聚类统计到集中操作数据。

日志聚合

很多人使用Kafka作为日志聚合的一种解决方案。典型的日志聚合是收集服务器上的物理日志文件，然后将其放置到一个中心位置（一个文件服务器或者是HDFS）用来进行处理。Kafka抽象了文件的详细信息，给出了一个日志或者事件数据的更清晰的抽象来作为消息流。这样就允许低延迟处理并且针对分布式数据消费和多种数据源有更好的支持。相比较中心系统（log-centric systems）如Scribe或者Flume，Kafka在保证同样出色性能的前提下，对于复制提供了更强的耐用性并且大大的降低了端到端的延迟。

流处理

许多人使用Kafka处理数据，这个处理管道是由多个阶段组成的。原始的输入数据从Kafka主题中被消费之后经过整合，修饰或者其他的装换后存入新的主题中，可以进一步的进行消费或者后续的处理。例如，针对推荐新闻文章的处理管道可能首先从RSS中爬取文章内容并把它发布到一个“articles”主题中；之后的处理可能是规范化或者复制内容，之后发布处理过的文章内容到一个新的主题中；最后的处理阶段可能是尝试推荐内容给用户。这种处理管道基于独立的主题创建了实时的数据流图。从Kafka的0.10.0.0版本开始，一个轻量级且有用的流处理库（Kafka Stream）可以用来进行类似上面描述的这种数据处理方式。除了Kafka Streams，其他可用的开源流处理工具还有Apache Storm和Apache Samza。

事件来源（**Event Sourcing**）

Event Sourcing是一种应用程序的设计样式。在这种样式中，状态的改变作为一种按照时间排序的记录。由于Kafka支持大数据量的日志数据存储，这使得它成为建立这种样式的应用程序的一个极好的后端。

日志提交（**Commit Log**）

对于分布式系统，Kafka用于一种外部的日志提交。日志帮助复制节点之间的数据并且对于失效的节点是一种同步机制来复原它们的数据。Kafka中的日志压缩特性支持了这一用途。在这一用途上，Kafka跟Apache BookKeeper工程是很相似的。

1.3 快速开始(Quick Start)

接下来的教程假定你是第一次使用Kafka并且没有任何的Kafka或者ZooKeeper数据。

第一步：下载代码

下载0.10.0.0版本然后进行解压。

```
tar -xzf kafka_2.11-0.10.0.0.tgz
cd kafka_2.11-0.10.0.0
```

第二步：启动服务器

Kafka使用ZooKeeper，所以如果你还没有开启ZooKeeper服务器，你应该首先开启ZooKeeper服务器。你可以使用这个方便的脚本包启动一个单节点的ZooKeeper实例。

```
bin/zookeeper-server-start.sh config/zookeeper.properties
[2013-04-22 15:01:37,495] INFO Reading configuration from: config/zookeeper.properties
 (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
...
```

现在开启Kafka服务器：

```
> bin/kafka-server-start.sh config/server.properties

[2013-04-22 15:01:47,028] INFO Verifying properties (kafka.utils.VerifiableProperties)
[2013-04-22 15:01:47,051] INFO Property socket.send.buffer.bytes is overridden to 1048
576 (kafka.utils.VerifiableProperties)
...
```

第三步：创建一个主题：

我们首先创建一个名为“test”的主题，它只有一个单独的分区并且只有一个副本：

```
> bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --par
titions 1 --topic test
```

如果我们运行列出主题的命令，那么我们可以看到这个主题：

```
> bin/kafka-topics.sh --list --zookeeper localhost:2181
test
```

另外一种选择是，你可以不用人工创建主题，你可以配置你的brokers当没有存在的主题用来发布时来自动地创建主题。

第四步：发送消息（send some message）

Kafka用命令行客户端可以将从文件或者标准输入的消息输出到Kafka集群中。默认情况是每一行都被作为一个单独的消息进行发送。

运行生成者，然后输入一些消息到控制台就可以将消息发送到服务器。

```
> bin/kafka-console-producer.sh --broker-list localhost:9092 --topic test
This is a message
This is another message
```

第五步：开启消费者（Start a consumer）

Kafka也有一个消费者的命令行，它可以将消息输出到标准输出上。

```
> bin/kafka-console-consumer.sh --zookeeper localhost:2181 --topic test --from-beginni
ng
This is a message
This is another message
```

如果你在不同的终端上运行以上的命令，那么你在生产者终端上输入的消息都可以在消费者终端上看到。

所有的命令行工具都有额外的选项；不带参数的运行命令将会更加详细的显示命令的文档信息。

第六步：Setting up a multi-broker cluster

直到现在我们仍然是在一个broker上运行这些，这不是有趣的事情。对于Kafka，一个broker仅仅是一个大小为1的集群，如果不开启其他的broker实例是没有什么改变的。为了学习这个，让我们在集群上扩展三个节点（仍然在本地的机器上）

首先我们对于每个brokers生成一个配置文件：

```
> cp config/server.properties config/server-1.properties
> cp config/server.properties config/server-2.properties
```

现在编辑这些新的文件并且设置下面的属性：

```
config/server-1.properties:
    broker.id=1
    listeners=PLAINTEXT://:9093
    log.dir=/tmp/kafka-logs-1

config/server-2.properties:
    broker.id=2
    listeners=PLAINTEXT://:9094
    log.dir=/tmp/kafka-logs-2
```

broker.id是节点在集群中唯一且永久的名称。因为我们运行这些节点是在同一个机器上并且我们想要阻止brokers注册在同一个端口或者覆盖其他节点的数据，所以我们指定了不同了端口和日志目录。

我们已经开启了ZooKeeper以及单个节点，现在我们需要开启另外两个新的节点：

```
> bin/kafka-server-start.sh config/server-1.properties &
...
> bin/kafka-server-start.sh config/server-2.properties &
...
```

现在我们创建一个复制因子是3的主题。

```
> bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 3 --par
titions 1 --topic my-replicated-topic
```

现在我们才能知道哪个broker正在干什么呢？我们可以运行“describe topics”命令来看一看：

```
> bin/kafka-topics.sh --describe --zookeeper localhost:2181 --topic my-replicated-topi
c
Topic:my-replicated-topic    PartitionCount:1    ReplicationFactor:3    Configs:
    Topic: my-replicated-topic    Partition: 0    Leader: 1    Replicas: 1,2,0    Isr:
 1,2,0
```

下面是针对这一输出的解释。第一行给出了所有分区的摘要，其他的每行给出了每个分区的信息。因为我们对于这个主题只有一个分区所以这里只有一行。

- “leader”是负责针对给定分区的所有读以及写的节点。对于随机选定的部分分区，每个节点都可以成为leader

- “replicas”列出了针对这个分区进行复制日志的所有节点，不管它们是leader或者甚至它们是否还存活。
- “isr”是同步副本集。它是现在存活的以及caught-up leader的集合，是replicas的子集。

注意到在我们例子代码中，对于这个主题的分区，1是leader

我们运行同样的命令在我们第一次创建的主题上，以下是输出结果：

```
> bin/kafka-topics.sh --describe --zookeeper localhost:2181 --topic test
Topic:test    PartitionCount:1    ReplicationFactor:1    Configs:
    Topic: test    Partition: 0    Leader: 0    Replicas: 0    Isr: 0
```

这个主题没有副本并且没有服务器，当我们创建它的时候才会存在服务器。

让我们发布一些消息到这个新的主题上：

```
> bin/kafka-console-producer.sh --broker-list localhost:9092 --topic my-replicated-top
ic
...
my test message 1
my test message 2
^C
```

现在消费这些消息：

```
> bin/kafka-console-consumer.sh --zookeeper localhost:2181 --from-beginning --topic my
-replicated-topic
...
my test message 1
my test message 2
^C
```

现在让我们测试一下系统的容错能力。我们杀死扮演leader的broker1节点。

```
> ps | grep server-1.properties
7564 ttys002    0:15.91 /System/Library/Frameworks/JavaVM.framework/Versions/1.8/Home/
bin/java...
> kill -9 7564
```

其他的一个broker自动变成了leader，节点1不在同步副本集合中了：

```
> bin/kafka-topics.sh --describe --zookeeper localhost:2181 --topic my-replicated-topi
c
Topic:my-replicated-topic    PartitionCount:1    ReplicationFactor:3    Configs:
    Topic: my-replicated-topic    Partition: 0    Leader: 2    Replicas: 1,2,0    Isr:
 2,0
```

即使将最初的那个leader去除，这些消息对于消费来说仍然是可以使用的。

```
> bin/kafka-console-consumer.sh --zookeeper localhost:2181 --from-beginning --topic my
-replicated-topic
...
my test message 1
my test message 2
^C
```

第七步：使用Kafka导出/导入数据

从控制台读取数据或者将数据写到控制台是一个很方便的开始，但是你可能想要将数据从Kafka输出到其他系统中。在Kafka中，你可以使用Kafka Connect导入/导出数据，而不是像在其他的系统中=编写用户集成代码。Kafka包括Kafka Connect工具，它可以导入或者导出数据。它是运行connectors的扩展工具，针对和外部系统的交互，它完成了用户逻辑。在这个简略的教程中，我们将学会运行Kafka Connect将数据从文件导入到Kafka的主题中，将数据从Kafka主题中导出到文件。首相，我们创建一些种子数据：

```
> echo -e "foo\nbar" > test.txt
```