银河麒麟服务器操作系统 V4

kafka 软件适配手册

天津麒麟信息技术有限公司

2019 年 5 月

目 录

1 概述

1.1 系统概述

银河麒麟服务器操作系统主要面向军队综合电子信息系统、金融系统以及电力系统等国家关键行业的服务器应用领域，突出高安全性、高可用性、高效数据处理、虚拟化等关键技术优势，针对关键业务构建的丰富高效、安全可靠的功能特性，兼容适配长城、联想、浪潮、华为、曙光等国内主流厂商的服务器整机产品，以及达梦、金仓、神通等主要国产数据库和中创、金蝶、东方通等国产中间件，满足虚拟化、云计算和大数据时代，服务器业务对操作系统在性能、安全性及可扩展性等方面的需求，是一款具有高安全、高可用、高可靠、高性能的自主可控服务器操作系统。

1.2 环境概述

服务器型号	长城信安擎天 DF720 服务器
CPU 类型	飞腾 2000+处理器
操作系统版本	Kylin-4.0.2-server-sp2-2000-19050910.Z1
内核版本	4.4.131
kafka 版本	2.11

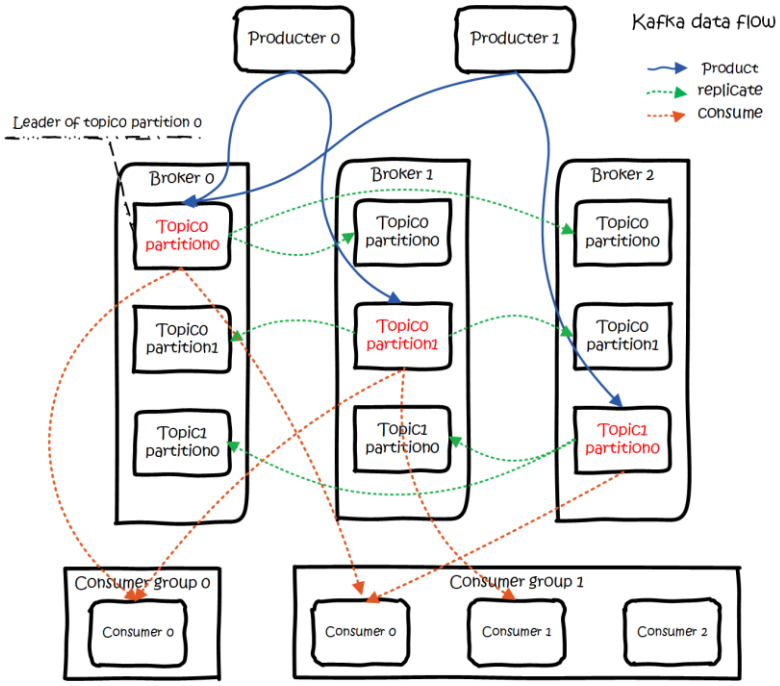
1.3 Kafka 软件简介

kafka 是一个分布式的消息队列。它提供了类似于 JMS 的特性，但是在设计实现上完全不同，此外它并不是 JMS 规范的实现。kafka 对消息保存时根据 Topic 进行归类，发送消息者成为 Producer,消息接受者成为 Consumer,此外 kafka 集群有多个 kafka 实例组成，每个实例(server)成为 broker。无论是 kafka 集群，还是 producer 和 consumer 都依赖于 zookeeper 来保证系统可用性集群保存一些 meta 信息。

kafka 对外使用 topic 的概念，生产者往 topic 里写消息，消费者从读消息。为了做到水平扩展，一个 topic 实际是由多个 partition 组成的，遇到瓶颈时，可以通过增加 partition 的数量来进行横向扩容。单个 parition 内是保证消息有序。每新写一条消息，kafka 就是在对应的文件 append 写，所以性能非常高。

1.4 Kafka 数据流

kafka 总体数据流如下：



Producers 往 Brokers 里面的指定 Topic 中写消息，Consumers 从 Brokers 里面拉去指定 Topic 的消息，然后进行业务处理。图中有两个 topic，topic 0 有两个 partition，topic 1 有一个 partition，三副本备份。可以看到 consumer gourp 1 中的 consumer 2 没有分到 partition 处理，这是有可能出现的。

关于 broker、topics、partitions 的一些元信息用 zk 来存，监控和路由啥的也都会用到 zk。

1.5 Kafka 使用场景

1. 消息队列

对于一些常规的消息系统，kafka 是个不错的选择；partitons/replication 和容错，可以使 kafka 具有良好的扩展性和性能优势。不过到目前为止，我们应该很清楚认识到，kafka 并没有提供 JMS 中的"事务性""消息传输担保(消息确认机制)""消息分组"等企业级特性；kafka 只能使用作为"常规"的消息系统，在一定程度上尚未确保消息的发送与接收绝对可靠(比如消息重发、消息发送丢失等)。

2. 网站活动跟踪

kafka 可以作为"网站活性跟踪"的最佳工具；可以将网页/用户操作等信息发送到 kafka 中，并实时监控或者进行离线统计分析。

3. 日志聚合

kafka 的特性决定它非常适合作为"日志收集中心"；application 可以将操作日志"批量""异步"的发送到 kafka 集群中，而不是保存在本地或者 DB 中；kafka 可以批量提交消息/压缩消息等，这对 producer 端而言，几乎感觉不到性能的开支。此时 consumer 端可以使 hadoop 等其他系统化的存储和分析系统。

1.6 Kafka 设计原理

kafka 的设计初衷是希望作为一个统一的信息收集平台，能够实时的收集反馈信息，并需要能够支撑较大的数据量,且具备良好的容错能力。

1. 持久性

kafka 使用文件存储消息，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几乎没有可能。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)达到一定阀值时，再 flush 到磁盘。这样减少了磁盘 IO 调用的次数。

2. 性能

需要考虑的影响性能点很多，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阀值时，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。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. 生产者

负载均衡: 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，有 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。其中 partition leader 的位置(host:port)注册在 zookeeper 中，producer 作为 zookeeper client，已经注册了 watch 用来监听 partition leader 的变更事件。

异步发送：将多条消息暂且在客户端 buffer 起来，并将他们批量的发送到 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，比如说当 producer 失效时，那些尚未发送的消息将会丢失。

4. 消费者

consumer 端向 broker 发送"fetch"请求，并告知其获取消息的 offset；此后 consumer 将会获得一定条数的消息；consumer 端也可以重置 offset 来重新消费消息。

在 JMS 实现中，Topic 模型基于 push 方式，即 broker 将消息推送给 consumer 端。不过在 kafka 中，采用了 pull 方式，即 consumer 在和 broker 建立连接之后，主动去 pull(或者说 fetch)消息；这中模式有些优点，首先 consumer 端可以根据自己的消费能力适时的去 fetch 消息并处理，且可以控制消息消费的进度(offset)；此外，消费者可以良好的控制消息消费的数量。

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。

5. 复制备份

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 和 consumer 一样，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。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。

6. 日志

如果一个 topic 的名称为"my_topic"，它有 2 个 partitions，那么日志将会保存在 my_topic_0 和 my_topic_1 两个目录中；日志文件中保存了一序列"log entries"(日志条目)。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).segmentfile 的命名为"最小 offset，其中每个 partiton 中所持有的 segments 列表信息会存储在 zookeeper 中。

当 segment 文件尺寸达到一定阀值时(可以通过配置文件设定,默认 1G)，将会

创建一个新的文件；当 buffer 中消息的条数达到阀值时将会触发日志信息 flush 到日志文件中，同时如果"距离最近一次 flush 的时间差"达到阀值时，也会触发 flush 到日志文件。如果 broker 失效，极有可能会丢失那些尚未 flush 到文件的消息。

2 Kafka 软件适配

2.1 下载软件并解压

```
$ wget https://archive.apache.org/dist/kafka/0.10.0.0/kafka_2.11-0.10.0.0.tgz
$ tar -zxvf kafka_2.11-0.10.0.0.tgz -C /usr/local/
```

2.2 配置 kafka

```
$ cd /usr/local/kafka_2.11-0.10.0.0/
$ vim config/server.properties
```

修改如下：

```
broker.id=0
host.name=localhost
log.dirs=/usr/local/kafka_2.11-0.10.0.0/logs

message.max.byte=5242880
default.replication.factor=1
replica.fetch.max.bytes=5242880

zookeeper.connect=localhost:2181
```

2.3 启动 kafka

```
$ bin/kafka-server-start.sh -daemon config/server.properties
```

3 Kafka 测试

3.1 创建 kafka topic

创建 topic：

```
$ bin/kafka-topics.sh --zookeeper localhost:2181 --create --topic test --partitions 3
--replication-factor 1
```

查看所创建的 topic：

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

查看更详细的信息：

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

3.2 单 broker 测试

在启动 kafka-server 之后启动，运行 producer：

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

在另一个终端运行 consumer：

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

在 producer 端输入字符串并回车，查看 consumer 端是否显示。