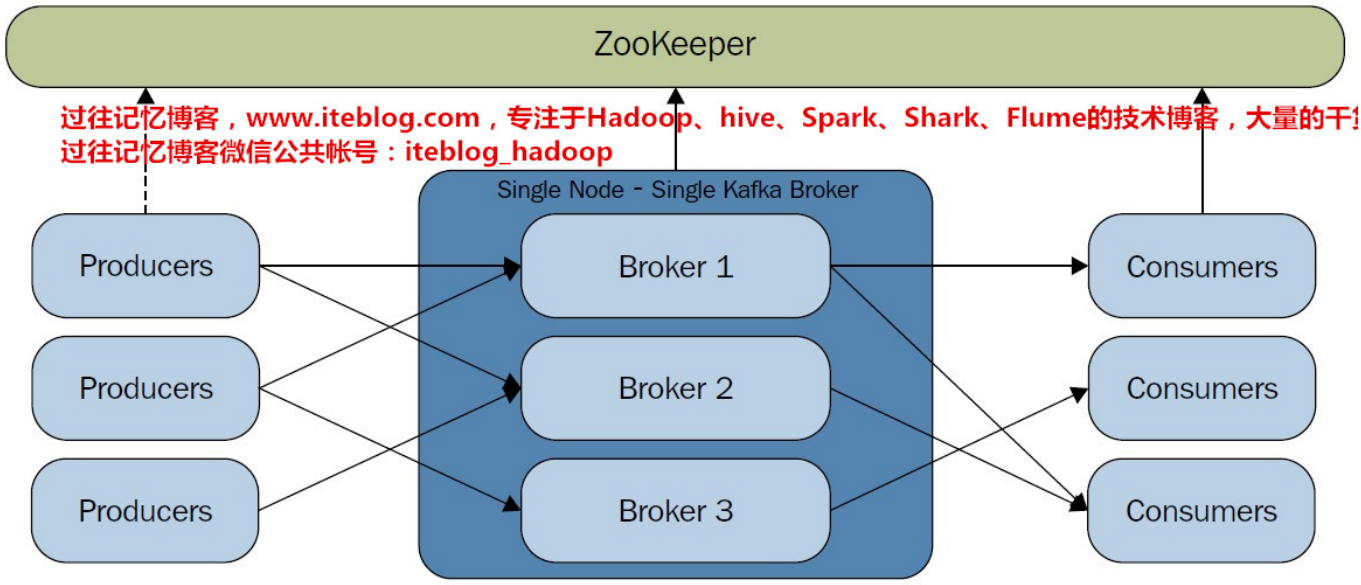
Kafka集群调优

Kafka Cluster模式最大的优点：可扩展性和容错性，下图是关于Kafka集群的结构图：



Kafka Broker个数决定因素

磁盘容量：首先考虑的是所需保存的消息所占用的总磁盘容量和每个broker所能提供的磁盘空间。如果Kafka集群需要保留 10 TB数据，单个broker能存储 2 TB，那么我们需要的最小Kafka集群大小 5 个broker。此外，如果启用副本参数，则对应的存储空间需至少增加一倍（取决于副本参数）。这意味着对应的Kafka集群至少需要 10 个broker。

请求量：另外一个要考虑的是Kafka集群处理请求的能力。这主要取决于对Kafka client请求的网络处理能力，特别是，有多个consumer或者网路流量不稳定。如果，高峰时刻，单个broker的网络流量达到80%，这时是撑不住两个consumer的，除非有两个broker。再者，如果启用了副本参数，则需要考虑副本这个额外的consumer。也可以扩展多个broker来减少磁盘的吞吐量和系统内存。

操作系统优化

大部分Linux发布版本默认的内核参数配置能让大部分应用工作的相当好。但对于实际的Kafka broker场景来说，做稍些改变会提升broker性能。主要涉及的配置：虚拟内存、网络和磁盘挂载（用来存储log segment），一般在 /etc/sysctl.conf (CentOS系统)。

Virtual Memory

一般来说，Linux的虚拟内存会根据系统负载自动调整。内存页（page）swap到磁盘会显著

的影响Kafka的性能，并且Kafka重度使用page cache，如果VM系统swap到磁盘，那说明没有足够的内存来分配page cache。

避免swap的一种方式是设置swap空间为0。但是，swap会在系统崩溃时提供安全机制，或者会在out of memory的情况下阻止操作系统 kill 掉进程。由于这个原因，推荐 vm.swappiness 参数设置为一个非常低的值：1 。这个参数表示 VM系统中的多少百分比用来作为swap空间。

另外一种方式是通过内核调节“脏页”（注：“脏页”会被刷到磁盘上）。Kafka依赖磁盘I/O性能来提高producer的响应时间。这也是为什么通常优先把log segment功能放在可以快速响应的磁盘中（比如，SSD）。这样使得flush进程把“脏数据”写入磁盘前，“脏页”数目就减少了，可以设置 vm.dirty_background_ratio（表示占用系统内存的百分比）参数的值为 10 以下。大部分应用场景下，vm.dirty_background_ratio 设置为 5 就够用了，要注意了：这个参数值不能设置为 0 ，因为设置为 0 后会引起内核持续刷“脏页”，使得内核的buffer write功能没法施展。

“脏页”的总量可以通过 vm.dirty_ratio 来改变，默认值是 20（此处也是百分比），这个值的设置范围较大，一般建议设置 60 到 80 为合理的值。但是 vm.dirty_ratio 参数也引来了不小的风险，会造成大量unflush的数据在硬刷到磁盘时产生较长的I/O停顿。如果vm.dirty_ratio 值设置的较大时，强烈建议Kafka开启备份功能，以备系统崩溃。

在设置了这些参数后，需要监控Kafka集群运行时“脏页”的数量，当前“脏页”数量可由如下方式查看（/proc/vmstat文件）：

```
# cat /proc/vmstat | egrep "dirty|writeback" nr_dirty 3875
nr_writeback 29
nr_writeback_temp 0
```

磁盘

除了考虑磁盘硬件本身和RAID配置外，磁盘的filesystem对Kafka集群的影响最大。虽然有许多filesystem，但最常用的是EXT4或者XFS。在这里XFS文件系统比EXT4稍好，具体原因Google下。

另外一点是，建议开启mount的noatime mount选项。文件系统在文件被访问、创建、修改等的时候会记录文件的一些时间戳，比如：文件创建时间（ctime）、最近一次修改时间（mtime）和最近一次访问时间（atime）。默认情况下，atime的更新会有一次读操作，这会产生大量的磁盘读写，然而atime对Kafka完全没用。

网络

Linux发布版本的网络参数对高网络流量不适用。对于Kafka集群，推荐更改每个socket发送和接收buffer的最大内存：net.core.wmem_default 和 net.core.rmem_default 为128 kb，net.core.wmem_max 和 net.core.rmem_max 为 2 Mb。另外一个socket参数是TCP

socket的发送和接收buffer： net.ipv4.tcp_wmem 和 net.ipv4.tcp_rmem。

Kafka集群稳定

GC调优

调GC是门手艺活，幸亏Java 7引进了G1垃圾回收，使得GC调优变的没那么难。G1主要有两个配置选项来调优：MaxGCPauseMillis 和 InitiatingHeapOccupancyPercent，具体参数设置可以参考Google，这里不赘述。

Kafka broker能够有效的利用堆内存和对象回收，所以这些值可以调小点。对于64Gb内存，Kafka运行堆内存5Gb，MaxGCPauseMillis 和 InitiatingHeapOccupancyPercent 分别设置为 20毫秒和 35。Kafka的启动脚本使用的不是 G1回收，需要在环境变量中加入：

```
# export JAVA_HOME=/usr/java/jdk1.8.0_51
# export KAFKA_JVM_PERFORMANCE_OPTS="-server -XX:+UseG1GC -XX:MaxGCPauseMillis=20 -XX:InitiatingHeapOccupancyPercent=35 -XX:+DisableExplicitGC -Djava.awt.headless=true"
# /usr/local/kafka/bin/kafka-server-start.sh -daemon /usr/local/kafka/config/server.properties
```

数据中心布局

原则上Kafka broker不建议都在一个机架上，为了容灾，但现实情况大部分公司做不到，此处略去。

Zookeeper

Kafka集群利用ZK来存储broker、topic和partition的元数据信息。在Kafka 0.9.0之前，consumer利用ZK来直接存储consumer group的信息，包括topic的消费情况、每个partition消费的周期性commit。在0.9.0版本，提供新的consumer接口利用Kafka broker来管理。

Consumer可以选择使用Zk或者Kafka来提交 offset和 提交间隔。如果consumer使用ZK管理offset，那每个consumer在每个partition的每个时间间隔写入ZK。合理的offset提交间隔是1分钟，但如果一个Kafka集群有大量的consumer消费时，这个ZK流量将是巨大的。所以如果ZK不能处理大流量，那只能把offset提交间隔设大，但同时也带来丢数据的风险。最保险的建议是使用Kafka来提交offset。

另外，建议Kafka集群不要和其他应用使用同一组ZK，因为Kafka对于ZK的延迟和超时是相当敏感的，ZK的不通将会导致Kafka的不可预测性。