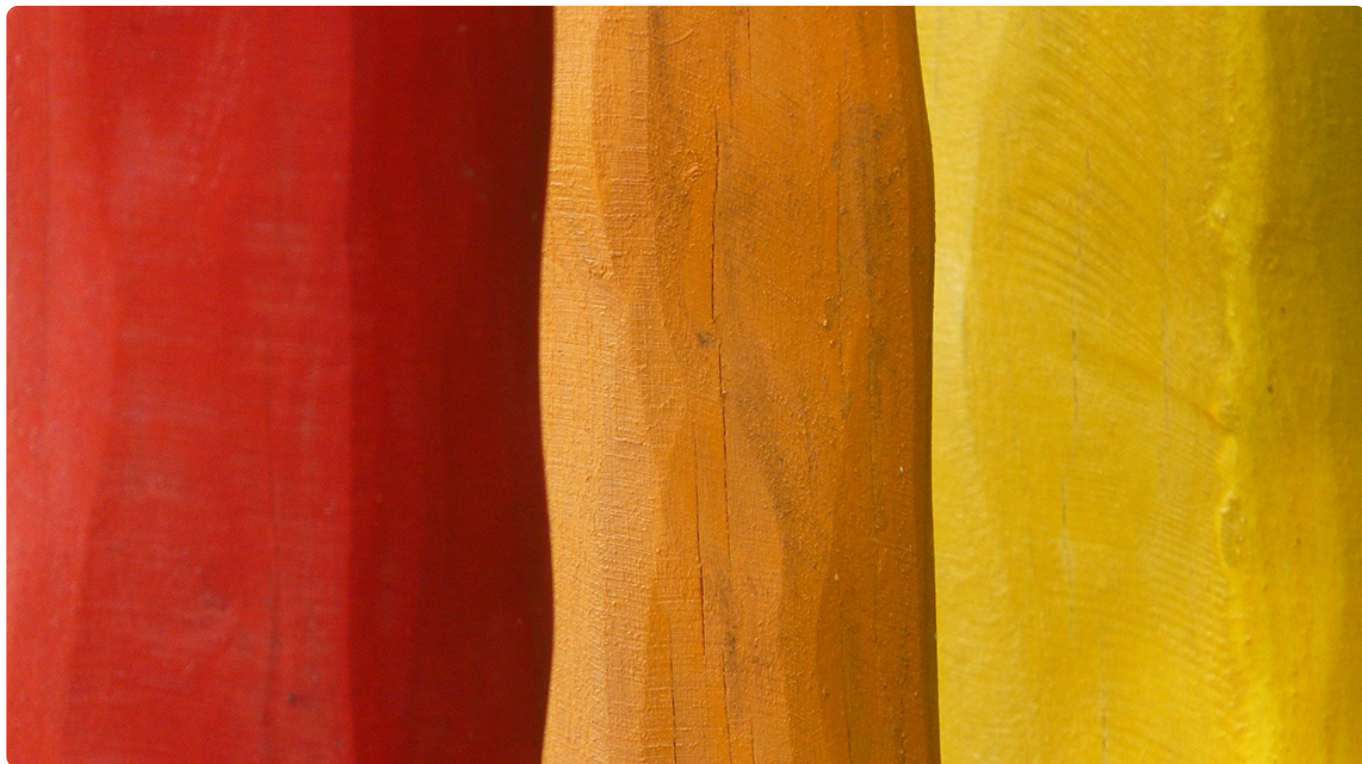


24 | Kafka：性能监控工具之队列级监控及常用计数器解析

2020-02-14 高楼

性能测试实战30讲

[进入课程 >](#)




讲述：高楼

时长 23:55 大小 21.91M



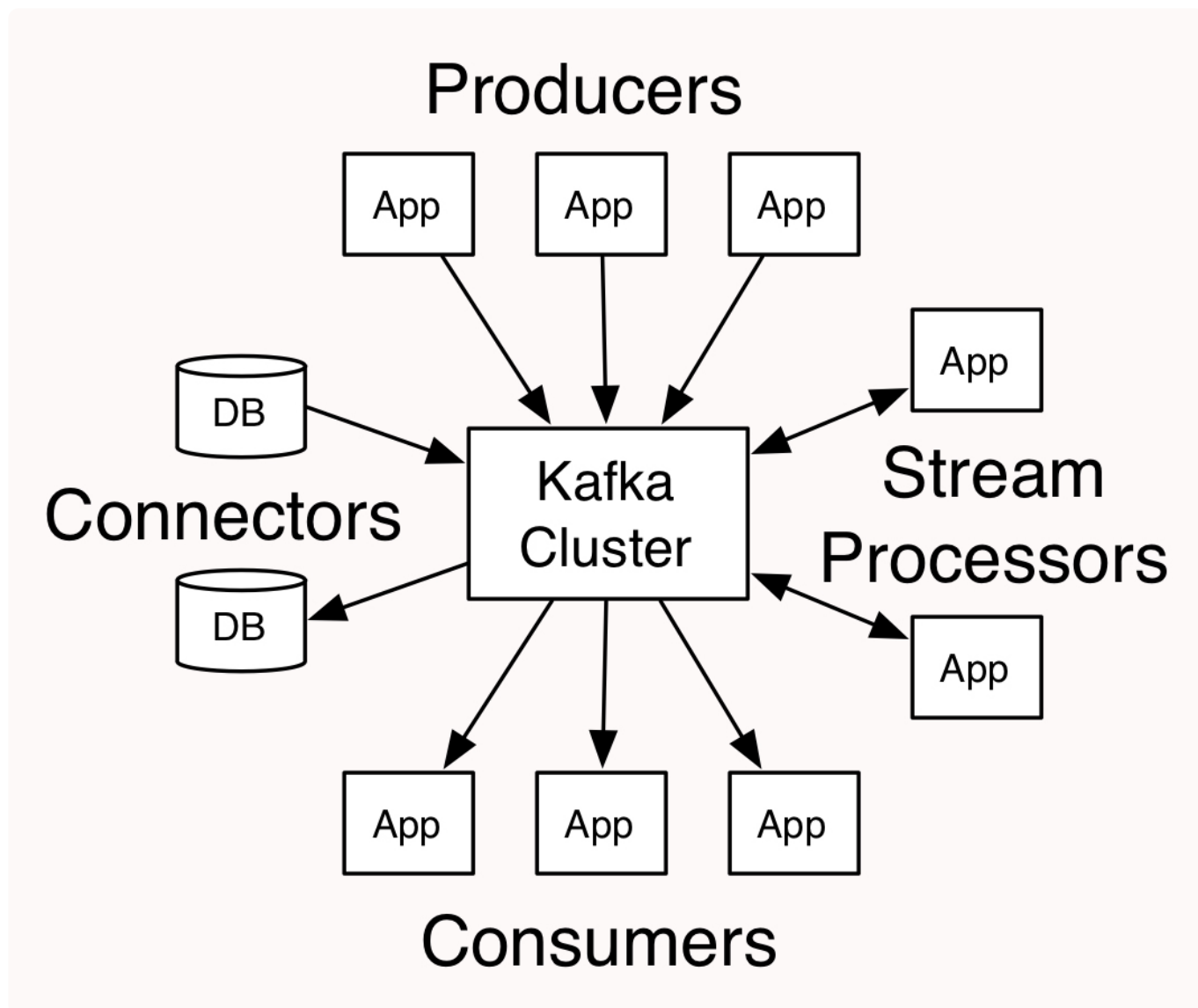
在我看来队列服务器是最简单的一种组件了。因为队列给我们下手的机会实在是并不多。我们只是用它，如果想改变它就只能去改代码，其他的都只是配置问题。

在当前的市场中，Kafka 算是用得非常火的一个队列服务器了，所以今天，我选择它来做一些解读。

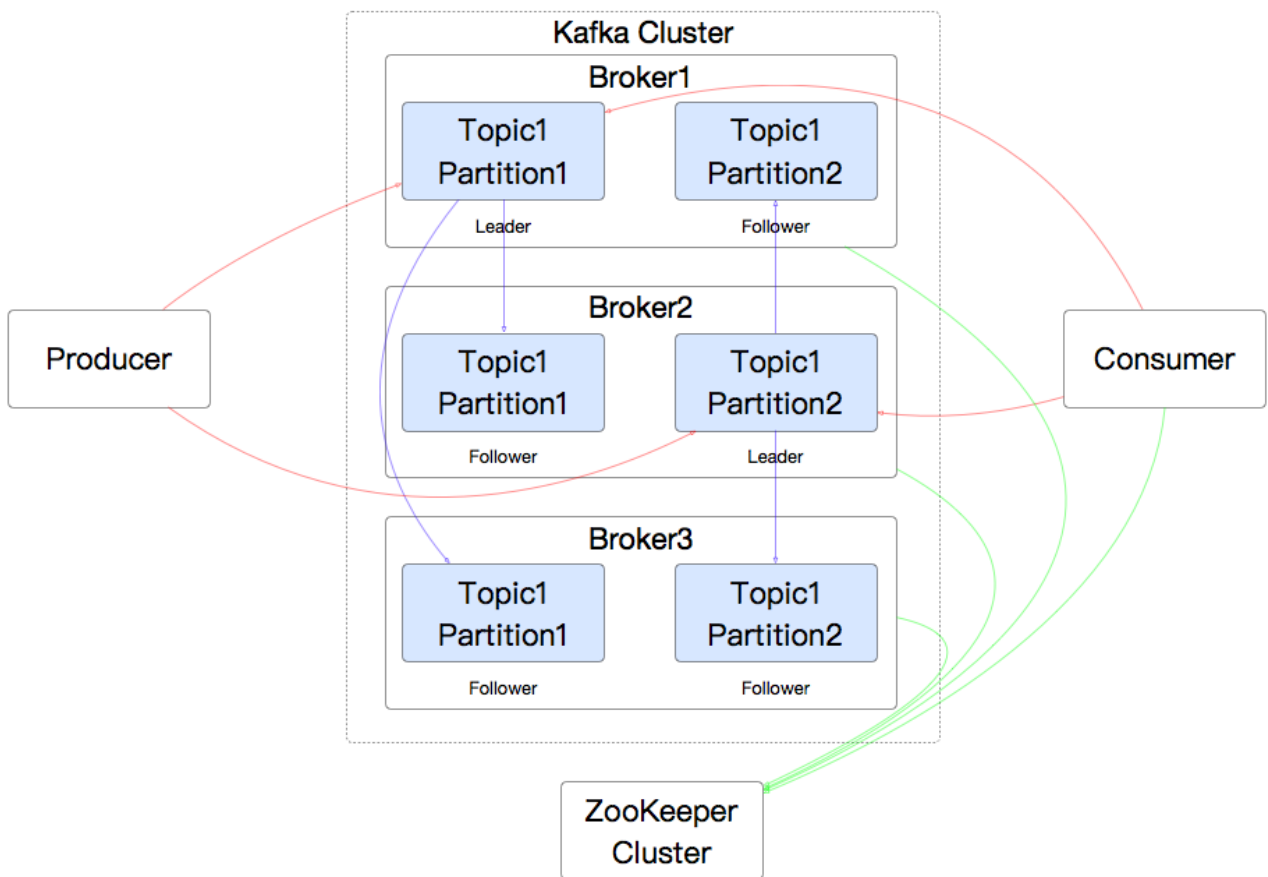
虽然我在前面一直在强调分析的思路，但在这一篇中，我打算换个思路，不是像以前那样，直接给你一个结论型的思维导图，而是一起来分析一个组件，让我们看看从哪里下手， 观察一个被分析对象的相关配置。

了解 Kafka 的基本知识

我们先看一下这张图，以便更好地了解一个队列服务器。



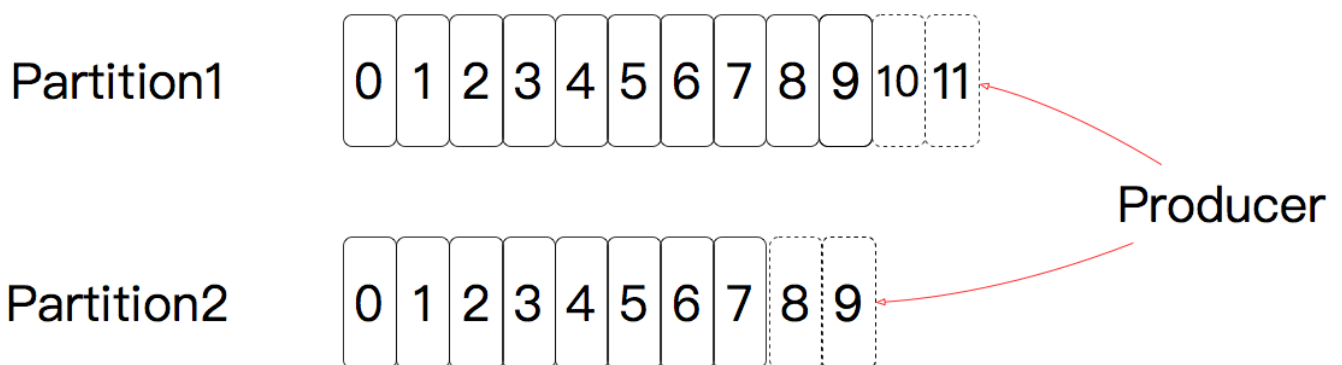
这是 Kafka 官网上的一张图。从这个图中可以看到，对 Kafka 来说，这就像一个典型的集线器。那它里面的结构是什么样子的呢？根据我的理解，我画了一个如下的示意图：



在这个图中，有三个 Broker，也就是三个集群节点。每个消息有一个 leader partition，还有两个 follower partition。我没有画更多的 Producer 和 Consumer、Consumer Group，是觉得线太多了容易乱。

因为 Producer 和 Consumer 肯定会从 leader partition 中读写数据，而 Kafka 也保证了 leader 在不同 broker 上的均衡，所以 Kafka 的集群能力很好。


我们再看一下消息是如何在 Kafka 中被存储的。



上图是 Kafka 数据的存储方式，也就是每个分区都是一直往后面加的。

我们再来看一下它的数据存储方式。


首先是目录：

 复制代码

```
1 drwxr-xr-x 2 root root 4096 Feb  7 23:39 test-0
2 drwxr-xr-x 2 root root 4096 Feb  7 01:34 test_perf-1
3 drwxr-xr-x 2 root root 4096 Feb  7 01:34 test_perf-4
```

Kafka 的目录是根据 topic 创建的，每个目录名中也包括一个 partition。比如上面名字中的 test_perf-1 就是 topic 名是 test_perf，partition 就是 1。

接着再来看下文件：

 复制代码

```
1 [root@node-2 test-2]# ll
2 total 10850656
3 -rw-r--r-- 1 root root      493128 Feb  9 14:14 00000000000000000000.index
4 -rw-r--r-- 1 root root 1073739646 Feb  9 14:14 00000000000000000000.log
5 -rw-r--r-- 1 root root      630504 Feb  9 14:14 00000000000000000000.timeindex
6 -rw-r--r-- 1 root root      443520 Feb  9 14:16 00000000000000240212.index
7 -rw-r--r-- 1 root root 1073727327 Feb  9 14:16 00000000000000240212.log
8 -rw-r--r-- 1 root root      551052 Feb  9 14:16 00000000000000240212.timeindex
9 -rw-r--r-- 1 root root      448840 Feb  9 14:18 00000000000000453584.index
10 -rw-r--r-- 1 root root 1073729759 Feb  9 14:18 00000000000000453584.log
11 -rw-r--r-- 1 root root      556920 Feb  9 14:18 00000000000000453584.timeindex
12 .....
13 -rw-r--r-- 1 root root          12 Feb  9 13:14 leader-epoch-checkpoint
14 [root@node-2 test-2]#
```

有索引文件，有数据文件，也有时间索引文件，非常明显的三个后缀名。索引文件显然就是指向 message 在数据文件中的什么位置，而这些数据文件就是一个一个的 Segment，也就是一段一段的。这些文件的大小受 server.properties 文件中的 log.segment.bytes 参数限制，默认为 1G。

要查到相应的 message 就要先查索引文件，找到 message 的位置；然后从 log 文件中找到具体的 message。

在这个逻辑中，Segment 的大小就很有讲究了，太细就会导致索引文件过大，查找索引费时间；太粗了就会导致查找得不够精准。那么该如何配置呢？也要通过性能测试才能知道。

有了这些信息之后，我们再看下 Kafka 高效的原因：

1. Kafka 直接使用 Linux 文件系统的 Cache 来高效缓存数据。
2. Kafka 采用 Linux Zero-Copy 技术提高发送性能（不懂 Linux Zero-copy 的请自行补课）。
3. Kafka 服务端采用的是 selector 多线程模式（从逻辑上理解，它和 Tomcat 的 NIO 类似，我就不单独画图了，以免占篇幅）。
4. Kafka 采用二分法找数据。

总体来说，就是一个 Java 的应用，直接使用文件系统和操作系统的特性实现了队列的高效应用场景。

配置文件

我们先来查看一下 Kafka 的配置文件中都有什么，为了简洁，在这里，我把一些注释以及和性能无关的配置删除了。当然如果你有兴趣的话，可以到 Kafka 的 config 目录中找到 server.properties 中，以查看这些内容。

 复制代码

```
1 ##### Socket Server Settings #####
2 num.network.threads=3
3 num.io.threads=8
4 socket.send.buffer.bytes=102400
5 socket.receive.buffer.bytes=102400
6 socket.request.max.bytes=104857600
7
8
9 ##### Log Basics #####
10 num.partitions=10
11 num.recovery.threads.per.data.dir=1
12
13
14 ##### Internal Topic Settings #####
15 offsets.topic.replication.factor=1
16 transaction.state.log.replication.factor=1
17 transaction.state.log.min.isr=1
18
```

```

19 ##### Log Flush Policy #####
20 log.flush.interval.messages=10000
21 log.flush.interval.ms=1000
22
23
24 ##### Log Retention Policy #####
25 log.retention.check.interval.ms=300000
26
27
28 ##### Zookeeper #####
29 zookeeper.connection.timeout.ms=6000
30
31
32 ##### Group Coordinator Settings #####
33 group.initial.rebalance.delay
34
35

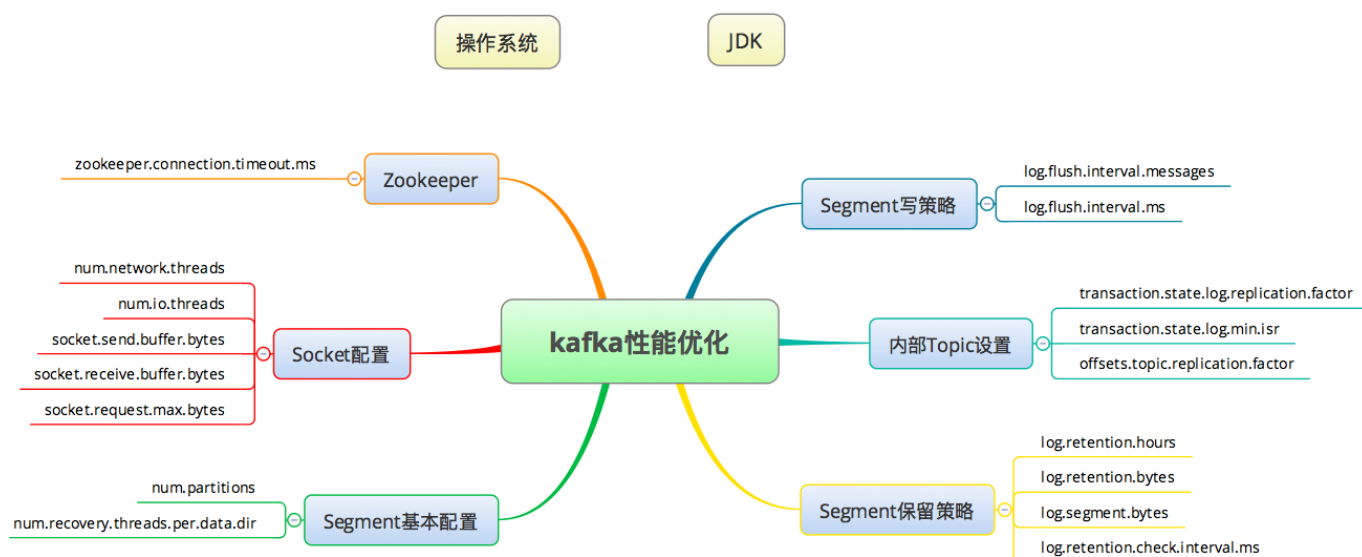
```

其实配置文件并不多，对不对？从配置名称上也很容易知道它们和什么相关。这里比较重要的参数就是 Socket Server 相关的，以及和 log 相关的。

我觉得到了这里，这个逻辑就基本清楚了，对 Kafka 的性能优化也就有了大体的判断。

构建 Kafka 的性能优化思维导图

我们可以根据以上的知识画出如下所示的，Kafka 的基本优化点：



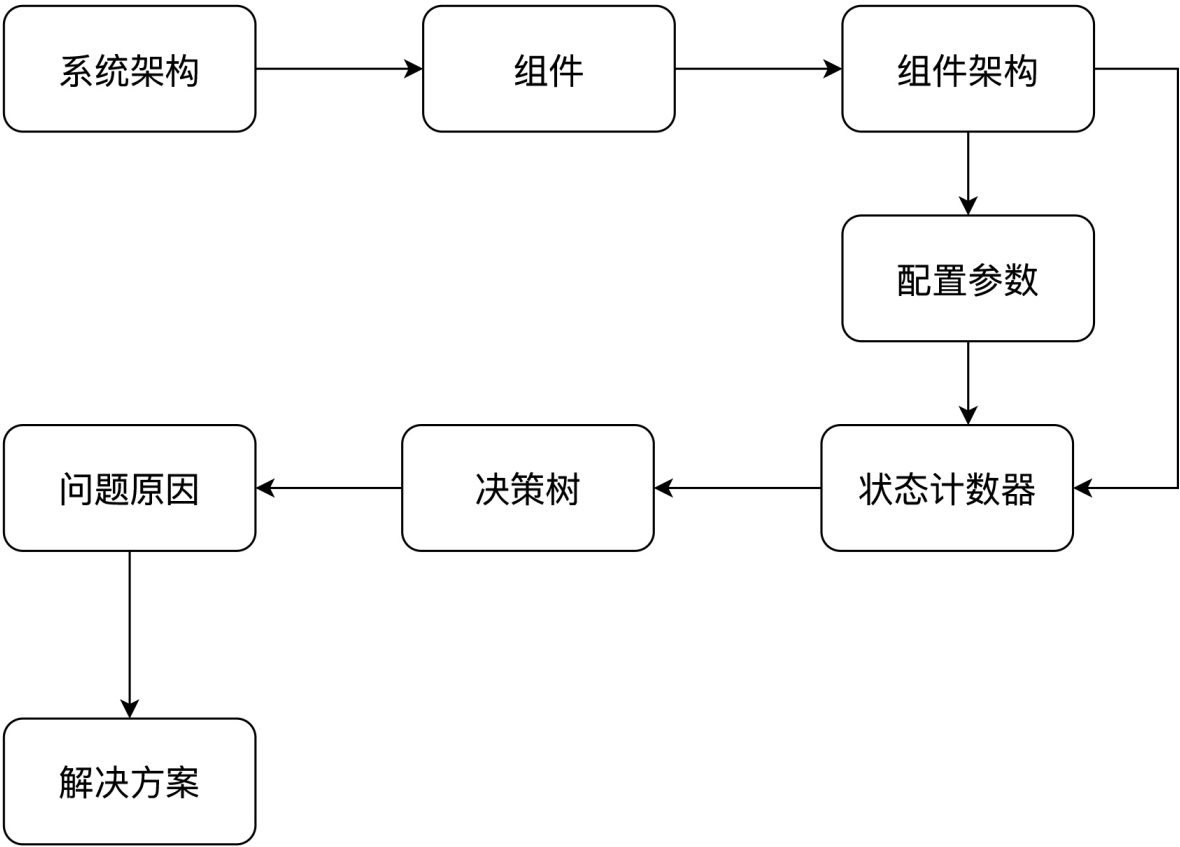
同样的，我把操作系统和 JDK 方面的优化当成独立的部分，在上图中只把 Kafka 相关的内容列出来。

有了上面的知识，也有了这个思维逻辑，那么就可以理出针对一个 Kafka 应用要干的事情：

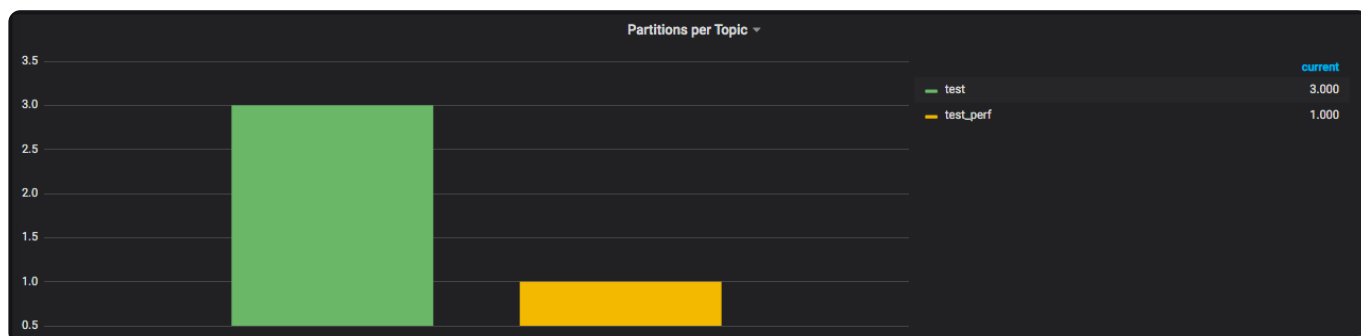
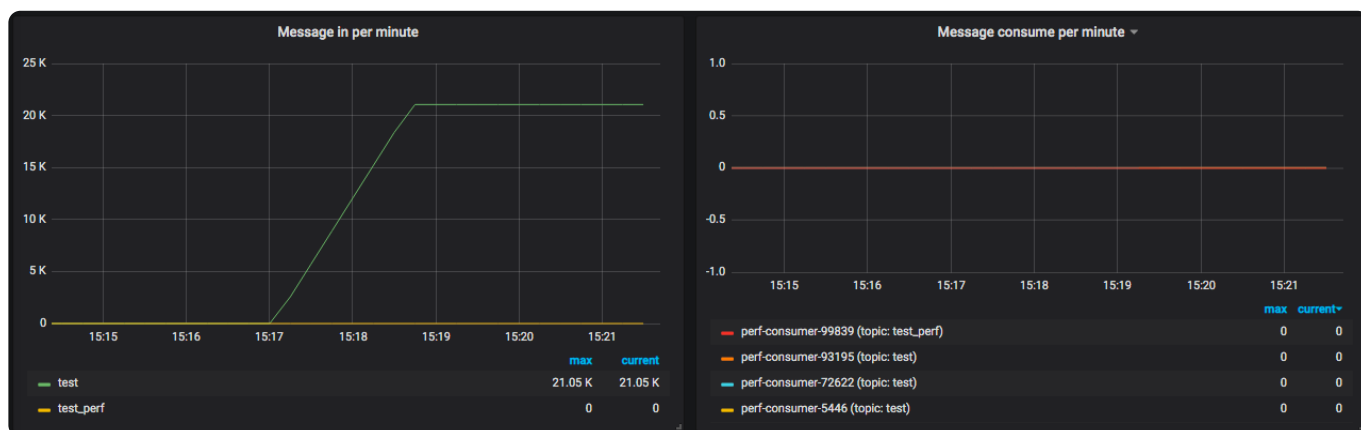
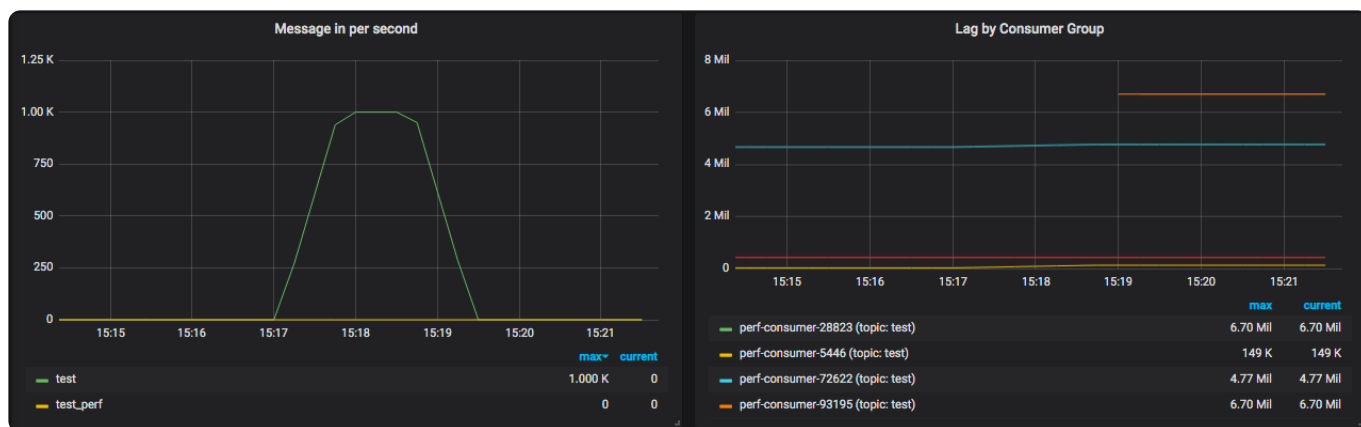
- 1. 先分析一下具体的应用场景，关键是 topic、partition 数量、message 大小。
- 2. 确定要支撑的业务容量和时间长度。
- 3. 分析架构中需要的 broker 量级、partition、Segment 等配置。这些配置应该是架构师给出的准确预估，如果不能给出，那只能靠我们，也就是做性能测试的人给出具体的结论了。

对组件的性能分析思路

我想告诉你的是对一个组件的性能分析思路。如果你有了下面这张图所示的思路，那至少可以覆盖大部分的性能问题了。这个思路就是：



对于 Kafka 这样的队列服务器来说，状态计数器是啥子呢？让我们看一下 Kafka 的一个 Grafana Dashboard。



从这几个图就能看得出来，最重要的是每秒产生了多少 message，以及消费时间间隔。这两个对我们来说是最重要的队列计数器了。

但是它们能不能告诉我们现在的队列服务器有没有瓶颈呢？显然是不能的。

对于队列来说，消息都是异步被消费者取走的。所以队列中要有保存消息的能力，但是保存多久呢？永远保存吗？显然不现实。但是如果保存得太短了，正常的业务都可能做不下去，所以，我们要制定策略，哪些 topic 是实时处理的，处理不完怎么办？内存多大，能保存多少消息，积压了怎么办？

所以对于队列服务器，只看上面的那几个计数器，我觉得过于片面。

我们前面提到的 grafana+prometheus 监控操作系统、MySQL 的 DashBoard 都有非常完整的数据，但是 Kafka 的 DashBoard 显然信息不够，不能判断它自己有没有问题。

操作系统的监控指标对 Kafka 来说，也是异常的重要。就像之前我说过的那样，操作系统是不可绕过的分析节点。所以所有要做性能测试和性能分析的人，首先要学的就是操作系统方面的知识。

示例

下面我们来看一个简单测试示例。

生产 10W 消息

在这个示例中，共生产 10W 的消息，每个消息大小是 2000 字节，每秒产生 5000 个消息。

 复制代码

```
1 [root@node-1 Kafka_2.13-2.4.0]# /home/zee/Kafka/Kafka_2.13-2.4.0/bin/Kafka-prod
2 24997 records sent, 4999.4 records/sec (9.54 MB/sec), 15.8 ms avg latency, 398
3 25010 records sent, 5001.0 records/sec (9.54 MB/sec), 26.0 ms avg latency, 514
4 25000 records sent, 5000.0 records/sec (9.54 MB/sec), 1.1 ms avg latency, 24.0
5 100000 records sent, 4998.000800 records/sec (9.53 MB/sec), 11.03 ms avg laten
```

可以看到每秒有 9.53MB 的消息产生，平均响应时延是 11.03ms，最大时延是 514ms。

生产 100W 消息

在这个示例中，共生产 100W 的消息，每个消息大小是 2000 字节，每秒产生 5000 个消息。


 复制代码

```
1 [root@node-4 bin]# /home/zee/Kafka/Kafka_2.13-2.4.0/bin/Kafka-producer-perf-te
2 24992 records sent, 4996.4 records/sec (9.53 MB/sec), 21.7 ms avg latency, 482
3 25025 records sent, 5004.0 records/sec (9.54 MB/sec), 0.9 ms avg latency, 16.0
4 .....
5 25000 records sent, 5000.0 records/sec (9.54 MB/sec), 0.6 ms avg latency, 9.0
6 25005 records sent, 5001.0 records/sec (9.54 MB/sec), 0.7 ms avg latency, 30.0
7 1000000 records sent, 4999.625028 records/sec (9.54 MB/sec), 2.05 ms avg laten
```

可以看到每秒有 9.54MB 的消息产生，平均响应时延是 2.05ms，最大时延是 482ms。

生产 1000W 消息

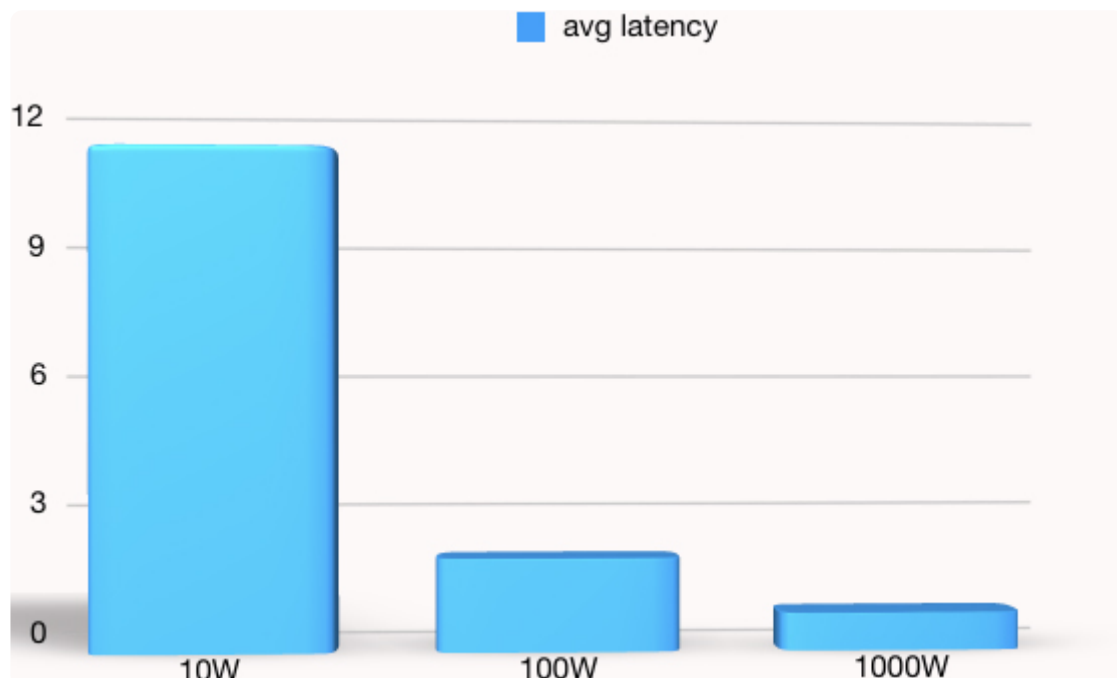
在这个示例中，生产 1000W 消息，其他参数不变：

 复制代码

```
1 [root@node-4 bin]# /home/zee/Kafka/Kafka_2.13-2.4.0/bin/Kafka-producer-perf-te:
2 24992 records sent, 4998.4 records/sec (9.53 MB/sec), 22.7 ms avg latency, 480
3 25015 records sent, 5002.0 records/sec (9.54 MB/sec), 0.8 ms avg latency, 13.0
4 25005 records sent, 5000.0 records/sec (9.54 MB/sec), 0.7 ms avg latency, 21.0
5 .....
6 25000 records sent, 5000.0 records/sec (9.54 MB/sec), 0.7 ms avg latency, 26.0
7 25010 records sent, 5001.0 records/sec (9.54 MB/sec), 0.7 ms avg latency, 24.0
8 10000000 records sent, 4999.900002 records/sec (9.54 MB/sec), 0.83 ms avg latei
```

从结果可以看到，每秒还是 9.54MB 大小的消息，平均时延 0.83ms，最大时延是 532ms。

来做一个图比对一下：



从这个图就明显看出生产的消息越少，平均响应时间越长。可见顺序写得越多，那每次写的平均时间就会越小，所以 Kafka 在大数据量的读写中会表现得非常好。

总结

严格来说，这一篇文章是为了告诉你一个逻辑，那就是对一个组件不了解的时候，如何用你的基础技术知识把对组件的性能优化方向整理出来，以及如何通过自己的基础知识来做一个非常合理的分析。

这个逻辑就是：

1. 先了解这个组件的基本知识：包括架构、实现原理等信息。
2. 再整理出这个组件的配置参数。
3. 找到合适的全局监控工具。
4. 做压力测试时给出明显的判断。

这是个大体的逻辑，当然这个逻辑还有一个前提，那就是你得有相应的基础知识，在 Kafka 的这个分析中，要有操作系统和 Java 的基础知识，在实操中还需要多找几个不懂的组件做些练习才能理解这个逻辑的真谛。

就我自己来说，我会找一个完全没有接触过的组件，从安装部署开始直到性能测试、瓶颈判断、优化分析，看看需要多长时间，我才能理解得了这个组件。

这种思维方式，给了我很多的安全感，就是遇到了没接触过的内容，也不至心慌气短。

思考题

最后给你留两道思考题吧，你觉得如何分析一个未知组件呢？Kafka 的分析逻辑又是什么？

欢迎你用自己的理解思考一下这两个问题，也欢迎把这篇文章分享给你的朋友或者同事，一起交流一下。

关注极客时间服务号 每日学习签到

月领 25+ 极客币

【点击】保存图片，打开【微信】扫码>>>



© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 23 | MySQL：数据库级监控及常用计数器解析（下）

下一篇 25 | SkyWalking：性能监控工具之链路级监控及常用计数器解析

精选留言 (2)

写留言

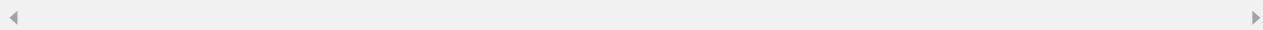


夜空中最亮的星（华仔...
2020-02-16

听了这节课 思路清晰了

展开

作者回复: 多谢支持。能理顺思路就是最重要的。



董飞
2020-02-16

老师，请教下jmeter测性能时，聚合报告中有一个错误率，具体怎样的请求会被统计成错误？

作者回复: 默认情况下，http的标准错误就会统计进去。

