

# 基于 MySQL 的 Spring 应用的读写效率

## 2-6 小组

**摘要：**信息技术的发展，互联网的普及，使企业信息系统架构方式由传统的 C/S 架构模式已经逐步转换为 B/S 架构模式。随之而来的是系统用户量以及数据量的增加，而且其增速远远大于数据库系统访问速度的提升，导致数据库系统读写处理速度成为影响系统性能和系统规模亟待解决的难题。因而本次实验基于 MySQL，利用负载功能测试和性能测试开源工具软件 Apache JMeter 去测试 Spring 应用的读写效率，并引入线程池技术实现读、写数据库间的数据同步更新。

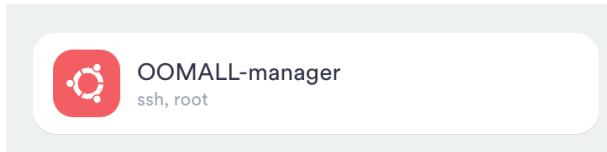
**关键词：** MySQL JMeter Spring 应用 读写分离 线程

### 1 实验目的

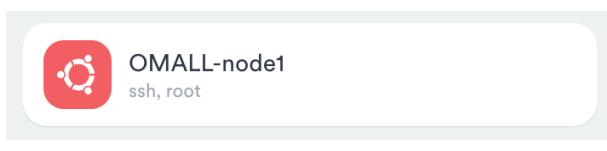
- 掌握如何在 SpringBoot 应用中通过 MyBatis 读写 MySQL 数据库。
- 掌握如何使用 SpringMVC 实现 RESTful API。
- 掌握如何使用 JMeter 对 RESTful API 进行性能测试。
- 验证通过 MyBatis 读写数据库的效率。

### 2 实验环境

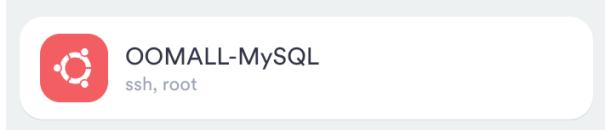
- 服务器 A:** Ubuntu 18.04 2 核 1G 虚拟机，安装了 Docker、Maven、git，作为管理机，用于编译和部署 productdemoap 应用。



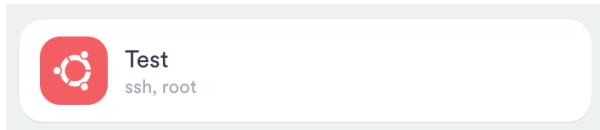
- 服务器 B:** Ubuntu 18.04 2 核 1G 虚拟机，安装了 Docker，用于部署 productdemoap Docker 容器。



- 服务器 C:** Ubuntu 18.04 2 核 1G 虚拟机，安装了 Docker，用于部署 MySQL Docker 容器。



- 服务器 D:** Ubuntu 18.04 2 核 1G 虚拟机，安装了 JMeter 5.6.3，用于测试。



### 3 实验内容

本次实验旨在对比 MyBatis 读写 MySQL 数据库的效率。使用 productdemoap 应用中的两个 RESTful API 进行测试：

- **查询产品信息:** GET /admin/products/{id}
- **新建产品:** POST /admin/products

实验中，分别针对数据库的查询和写入操作进行性能测试，并利用 JMeter 生成负载，模拟实际的多用户并发访问场景。

## 4、实验设计

### 4. 1 数据库设计:

根据 ER 图，数据库设计了产品表和相关的关系结构，确保可以支持 GET 和 POST 操作。



图 1 数据的 ER 图

### 4. 2 API 实现:

使用 SpringBoot 和 MyBatis 实现 RESTful API，GET 请求用于查询产品信息，POST 请求用于添加新的产品记录。

### 4. 3 性能测试:

4. 3. 1 在服务器 D 上使用 JMeter 模拟多用户并发请求，分别对两个 API 进行压力测试。

4. 3. 2 设置 100、200、500、1000 用户的并发请求数量，分别测试查询和写入操作的响应时间和成功率。

4. 3. 3 我们的判定标准是 80%以上的反应时间都低于 50ms

### 4. 4 测试准备

在每次对数据库进行 POST 请求时，都需要对数据库进行初始化操作，避免数据库数据增多造成对后续

写入操作造成影响。

## 5 实验结果分析

### 5.1 GET 查询产品信息：

我们在这里实验了 6 种不同的场景，分别是：

#### 5.1.1 1000 线程在 10 秒内循环 1 次

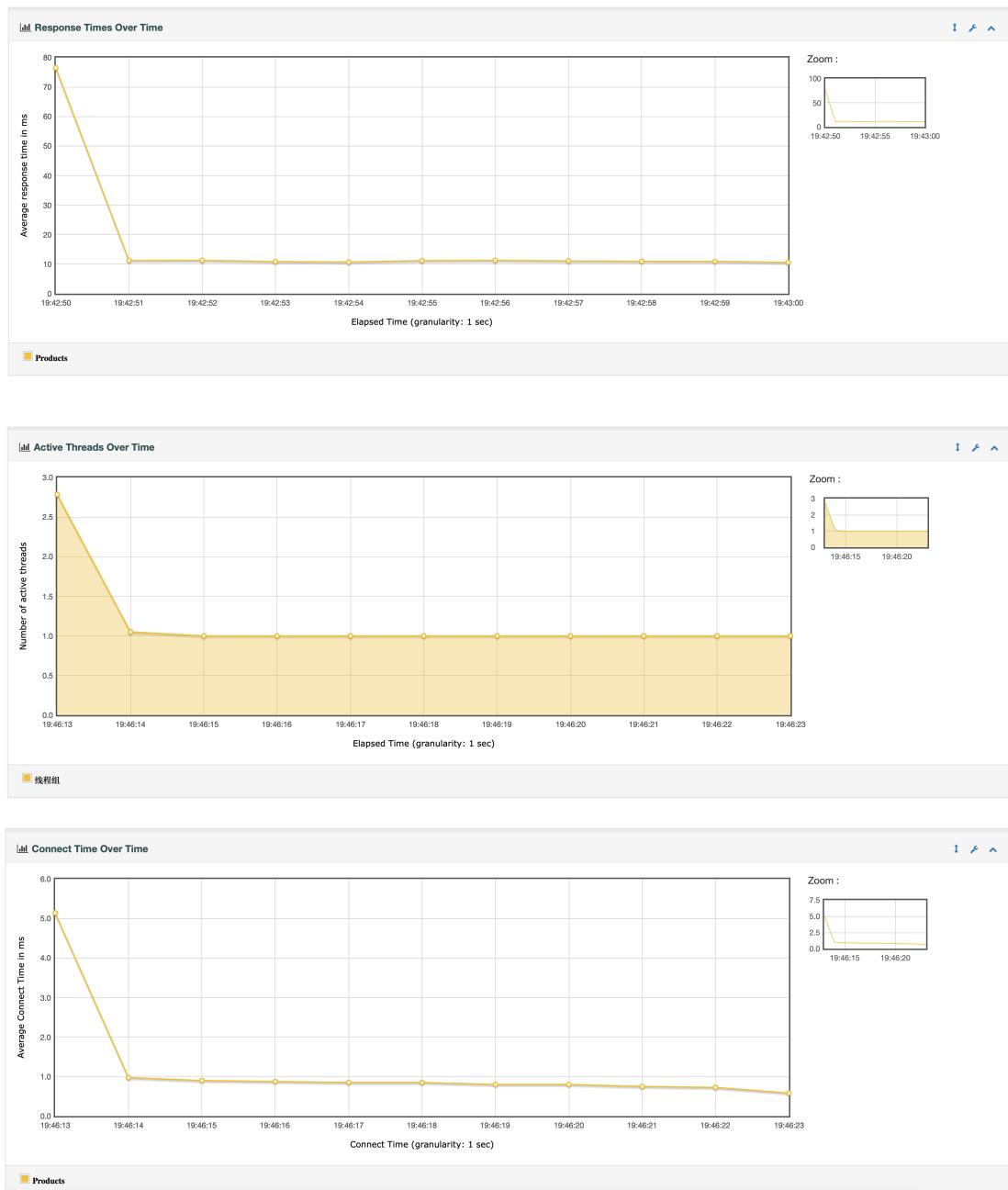


图 2 1000 线程在 10 秒内循环 1 次响应时间和活动线程数

- **响应时间：**从图表可以看出，系统的平均响应时间较为稳定，维持在 10ms 左右。在 10 秒的测试时间段内，几乎没有明显的波动。

- **活动线程数:** 活动线程数随着时间逐渐减少, 表明请求处理较为迅速。
- **连接时间:** 除开头 HTTP 协议网络连接外, 平均连接时间很短, 大约在 1ms 左右, 说明服务器能够快速建立连接并响应请求。

### 5. 1. 2 2000 线程在 10 秒内循环 1 次

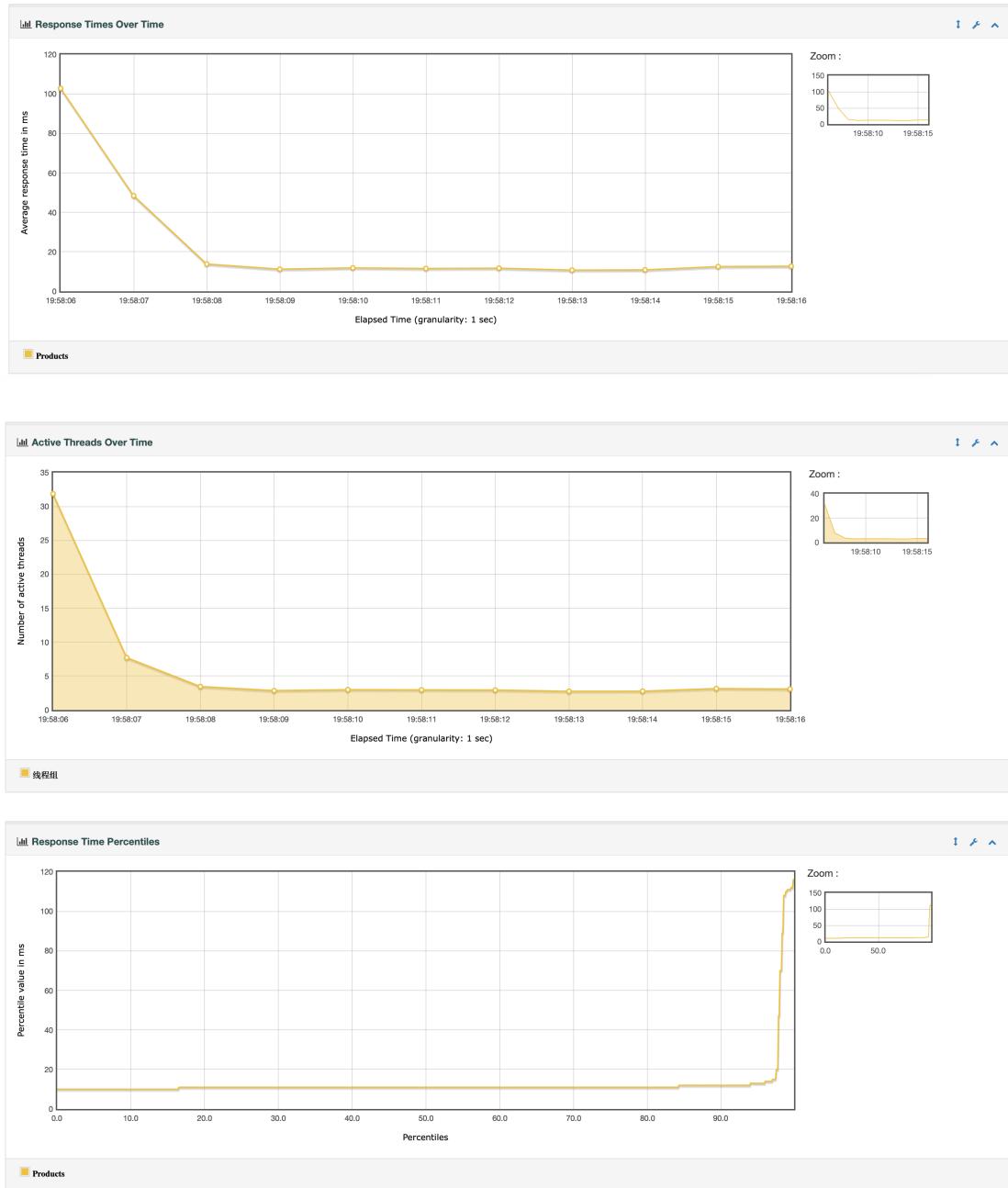


图 3 2000 线程在 10 秒内循环 1 次响应时间和活动线程数

- **响应时间:** 随着并发线程数增加, 响应时间显著增加, 后期平均响应时间略超过 10ms, 表明系统在处理并发请求时有一定的延迟, 但恢复较快。
- **活动线程数:** 活动线程数开始时急剧上升, 但很快趋于稳定。

- **响应时间百分位:** 绝大部分响应时间都低于 20%。

### 5.1.3 2000 线程在 10 秒内循环 2 次

```
root@test:~/test# jmeter -n -t ReadProduct.jmx -l read-2000-10-2.jtl -e -o read/read-2000-10-2
WARN StatusConsoleListener The use of package scanning to locate plugins is deprecated and will be removed in a future release
WARN StatusConsoleListener The use of package scanning to locate plugins is deprecated and will be removed in a future release
WARN StatusConsoleListener The use of package scanning to locate plugins is deprecated and will be removed in a future release
Creating summariser <summary>
Created the tree successfully using ReadProduct.jmx
Starting standalone test @ 2024 Oct 12 20:06:15 CST (1728734775665)
Waiting for possible Shutdown/StopTestNow/HeapDump/ThreadDump message on port 4445
[3.288s][warning][os,thread] Failed to start thread "Unknown thread" - pthread_create failed (EAGAIN) for attributes: stacksize: 1024K, guardsize: 0K, detached.
[3.288s][warning][os,thread] Failed to start the native thread for java.lang.Thread "线程组 1-1922"
summary + 2115 in 0:00:14 = 148.6/s Avg: 3730 Min: 27 Max: 10621 Err: 0 (0.0%) Active: 1239 Started: 1921 Finished: 682
root@test:~/test#
```

图 4 2000 线程在 10 秒内循环 2 次报错代码

这里服务器崩溃，仅有 jtl 文件而没有可视化结果生成了。因为我们的服务器为 2 核 1G，总内存为 2G，单条线程所占的内存为 1M，所以最大的线程数大概在 2000 左右，多次循环由于随机性导致同时运行的线程数可能超过了 2000，因此锁定性能上限不会超过 4000。

### 5.1.4 1000 线程在 10 秒内循环 3 次

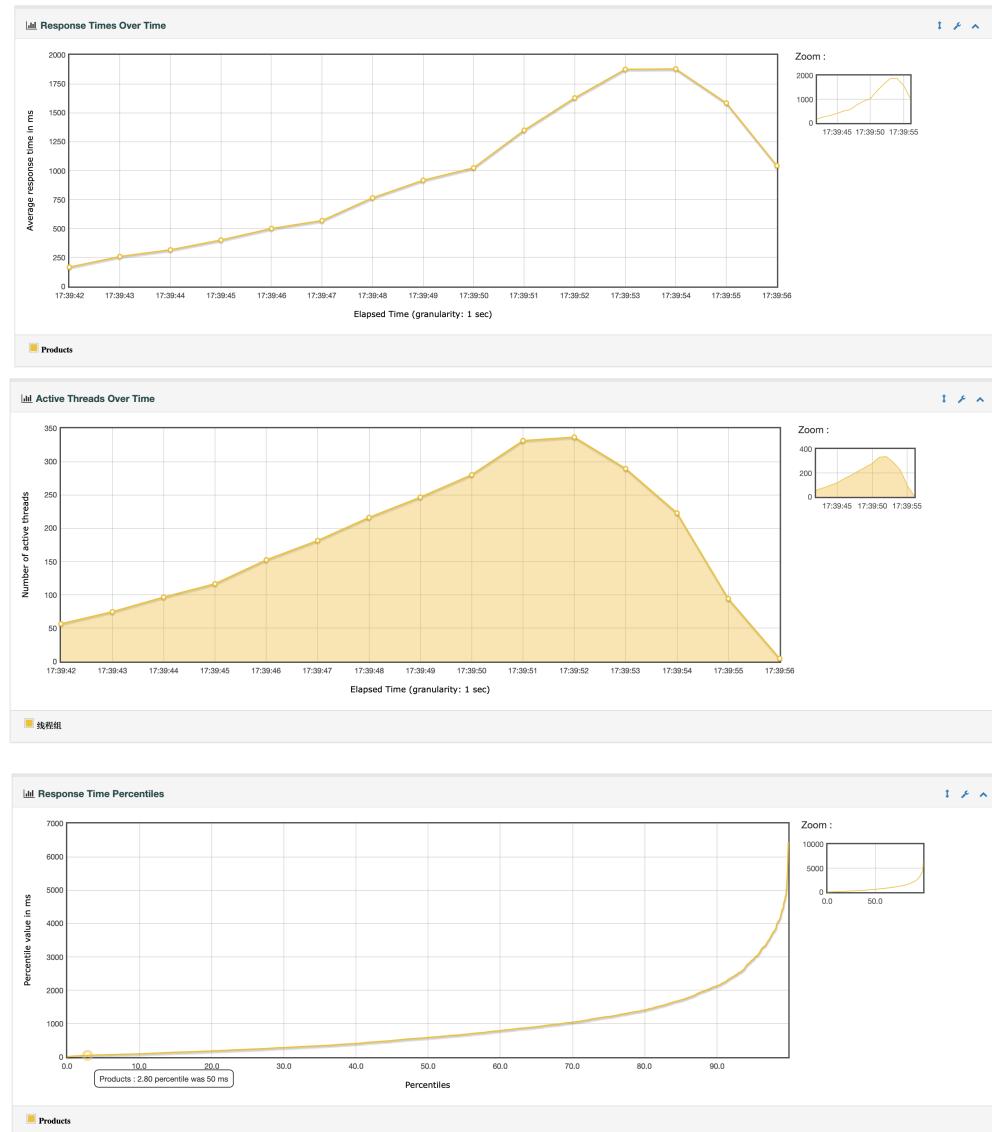


图 5 1000 线程在 10 秒内循环 3 次响应时间和活动线程数

仅有 2.80% 达到 50ms 的标准，因此锁定性能上限不会超过 3000。

### 5.1.5 500 线程在 10 秒内循环 5 次



图 6 500 线程在 10 秒内循环 5 次响应时间和活动线程数

仅有 24.20% 达到 50ms 的标准，因此锁定性能上限不会超过 2500。

### 5.1.6 700 线程在 10 秒内循环 3 次

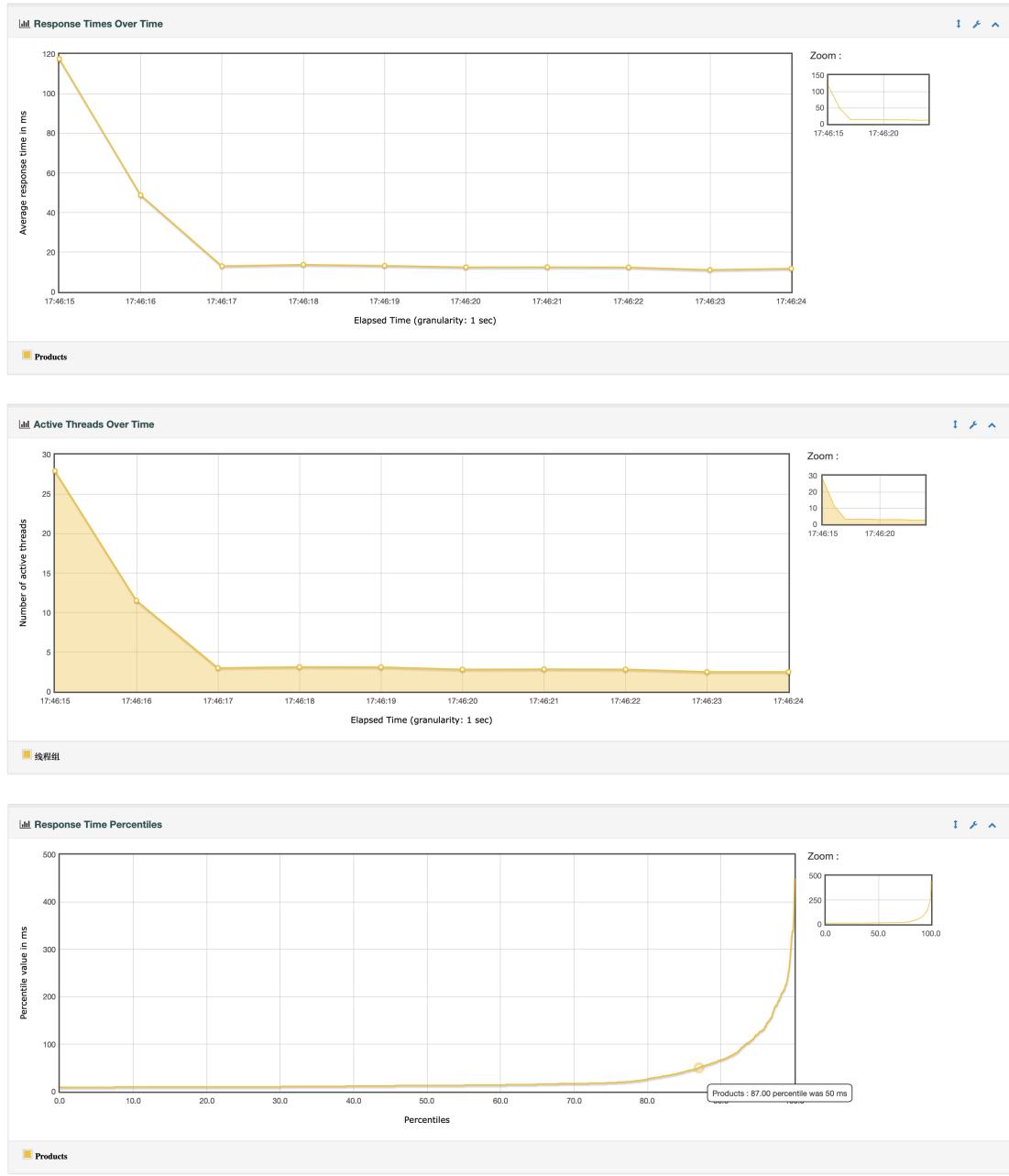


图 7 700 线程在 10 秒内循环 3 次响应时间和活动线程数

有 87.00% 达到 50ms 的标准，因此锁定性能下限肯定比 2100 高，因此现在已经知道性能瓶颈在 2100–2500 之间。

#### 5.1.7 570 线程在 10 秒内循环 4 次

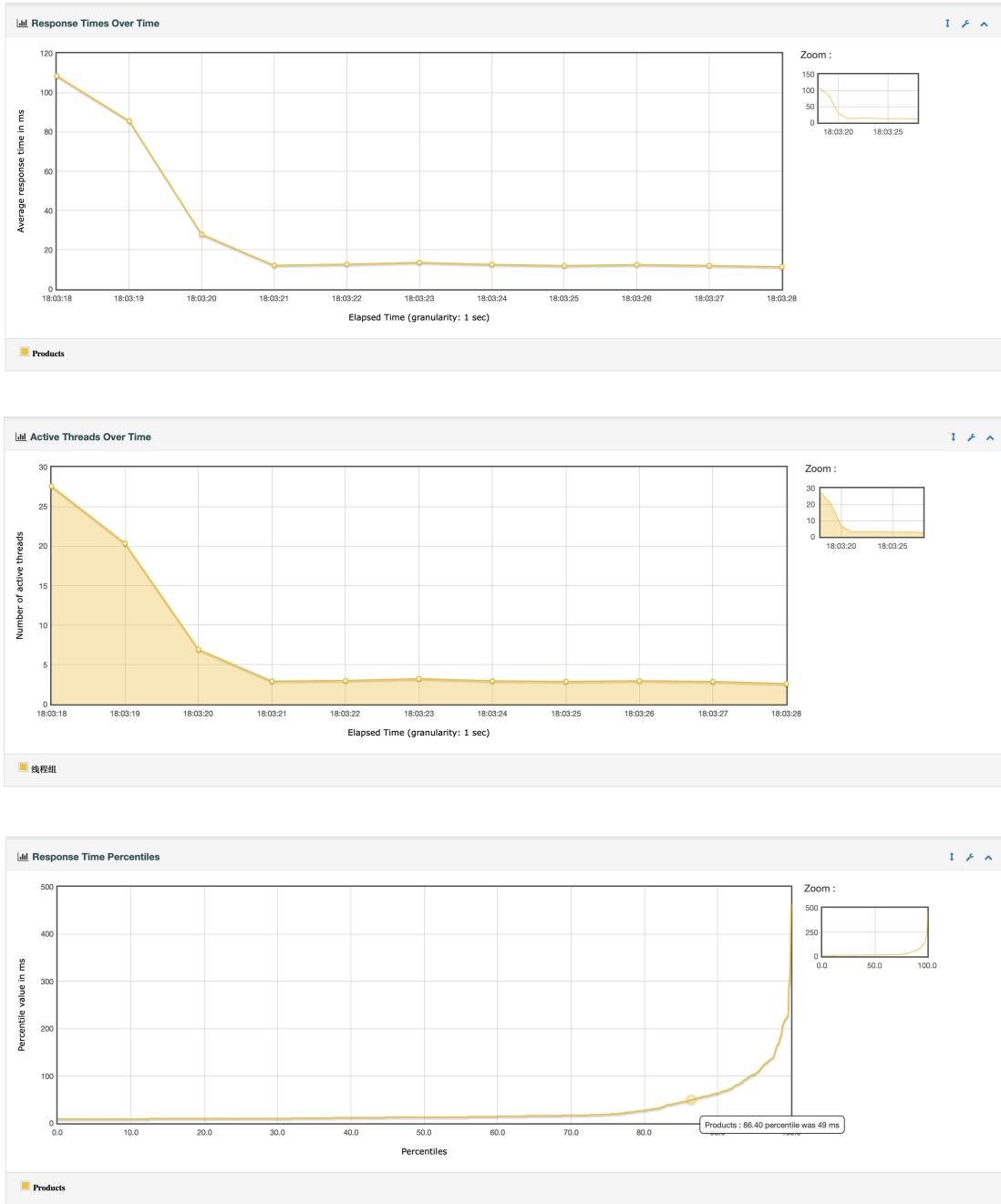


图 8 570 线程在 10 秒内循环 4 次响应时间和活动线程数

有 84.40% 达到 50ms 的标准，因此锁定性能下限比 2280 高，因此现在已经知道性能瓶颈在 2280–2500 之间。

#### 5.1.8 800 线程在 10 秒内循环 3 次

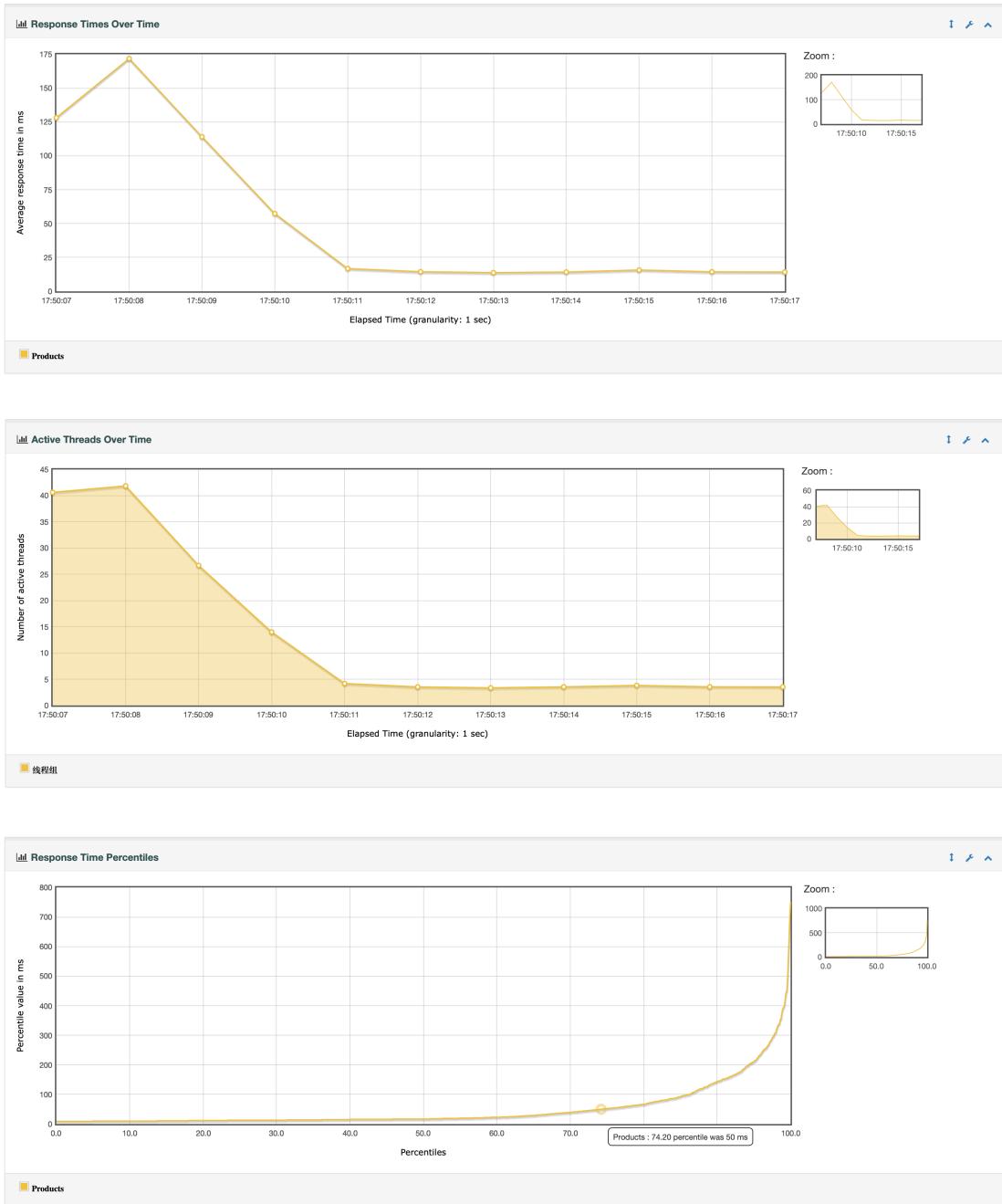


图 8 800 线程在 10 秒内循环 3 次响应时间和活动线程数

有 74.20% 达到 50ms 的标准，因此锁定性能上限肯定比 2400 低，因此现在已经知道性能瓶颈在 2280–2400 之间。

### 5.1.9 575 线程在 10 秒内循环 4 次

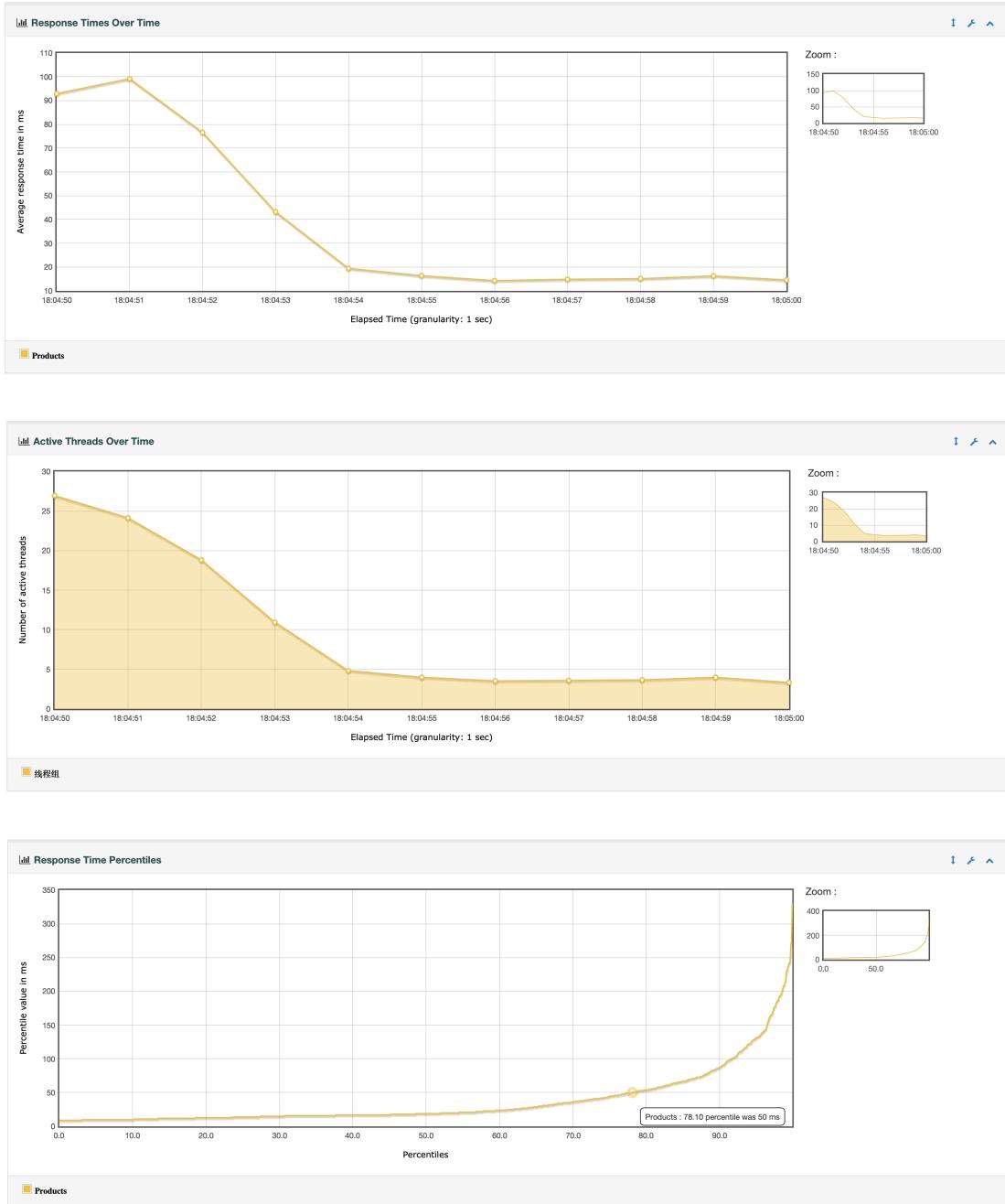


图 9 575 线程在 10 秒内循环 4 次响应时间和活动线程数

有 84.40% 达到 50ms 的标准，因此锁定性能上限比 2280 高，因此现在已经知道性能瓶颈在 2280–2300 之间。

**【结论】**因此通过上面的实验，可以估计该系统每秒能够稳定处理的线程数的上限大概接近  $2280/10=228$  条/s。

## 5.2 POST 新建产品：

我们在这里实验了 4 种不同的场景，分别是：

### 5.2.1 400 线程在 10 秒内循环 1 次

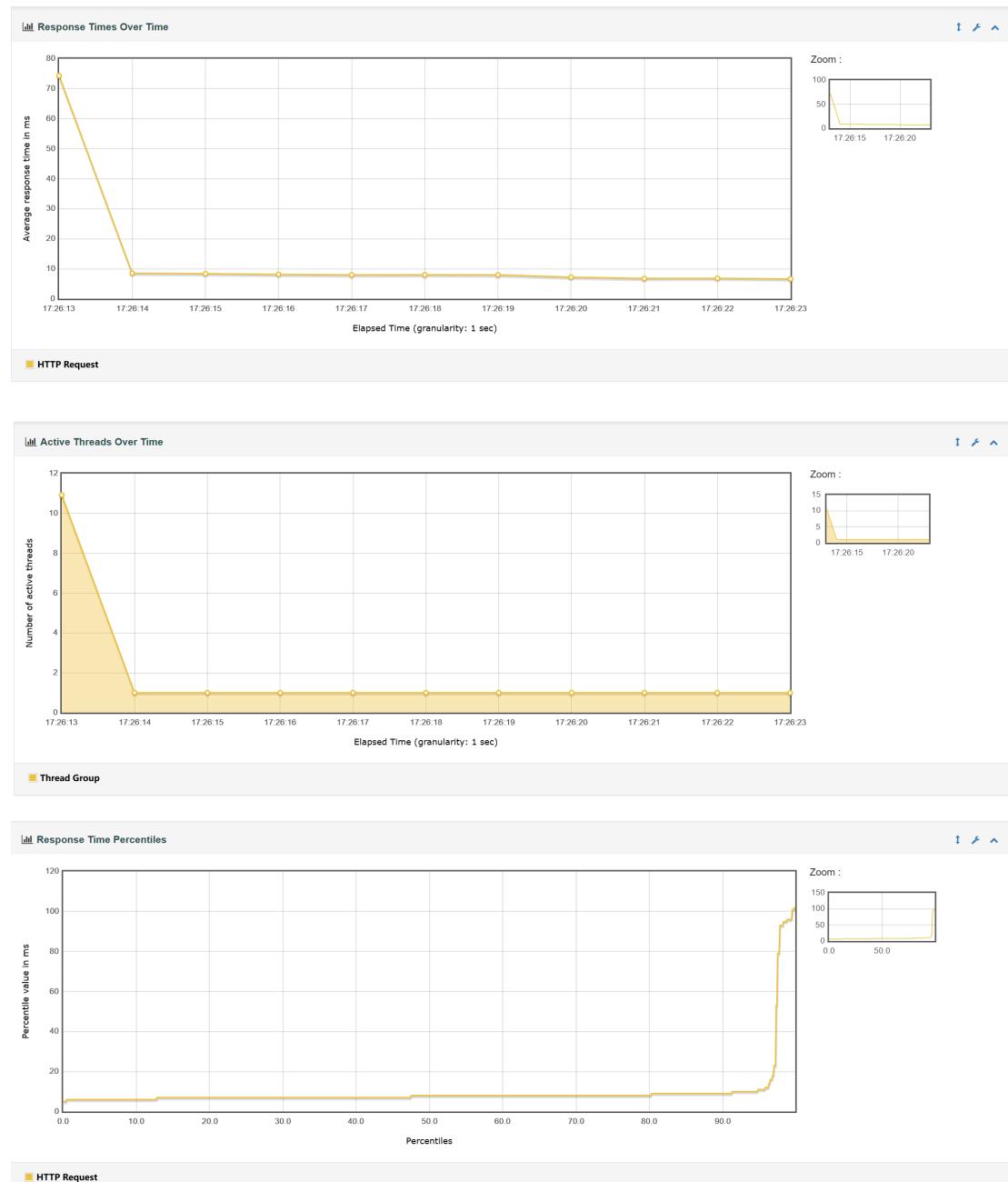


图 10 400 线程在 10 秒内循环 1 次响应时间和活动线程数

- **响应时间:** 从图表中可以看出，系统的平均响应时间在第 1s 内是 75ms 左右，随后维持在 8ms 左右。在 10 秒的测试时间段内，几乎没有明显的波动。
- **活动线程数:** 可以看出活动线程数与平均响应时间走势几乎一致，系统同时处理的线程数在第 2 秒后都为 1 个。
- **响应时间百分位:** 在百分位图中可以看出，仅有 94% 左右的请求在 10ms 内完成，表明 400 线程在 10 秒内循环 1 次是很顺利的。

### 5.2.2 2000 线程在 10 秒内循环 1 次

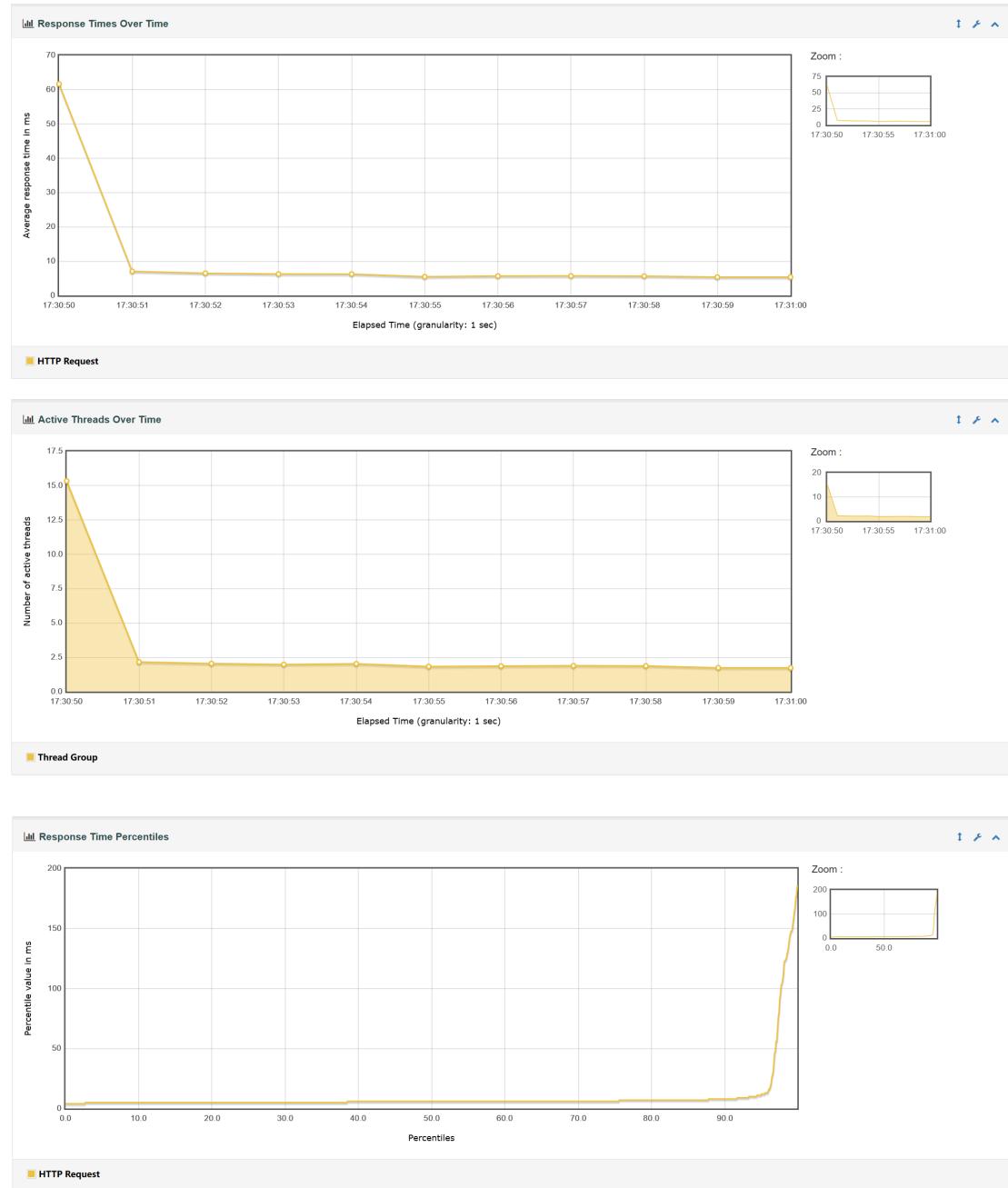


图 11 2000 线程在 10 秒内循环 1 次响应时间和活动线程数

- **响应时间:** 随着并发线程数的增加，响应时间有所增加，初始响应时间在 63ms 左右，随后都维持在 10ms 以下，表现较为平稳。
- **活动线程数:** 活动线程数迅速减少，说明系统能够较快地处理并发请求，系统同时处理的线程数在第 2 秒后都为 2 个。

- **响应时间百分位:** 在百分位图中可以看出, 92%左右的请求都在 10ms 内完成, 表明 2000 线程在 10 秒内循环 1 次是顺利的。

### 5. 2. 3 2000 线程在 10 秒内循环 5 次



图 12 2000 线程在 10 秒内循环 5 次响应时间和活动线程数

2000 线程在 10 秒内循环 5 次的平均响应时间呈现陡坡状上升态, 相应峰值达到了 300ms, 每秒并发进 程峰值也达到 290 左右, 仅有 4%左右的请求在 50ms 内完成, 说明当前服务器负载很大, 并发性能达到较

强的状态，也说明服务器将要接近系统瓶颈，因此后续测试将总线程数限制在 10000 以内。

#### 5.2.4 1500 线程 10s 内循环 5 次

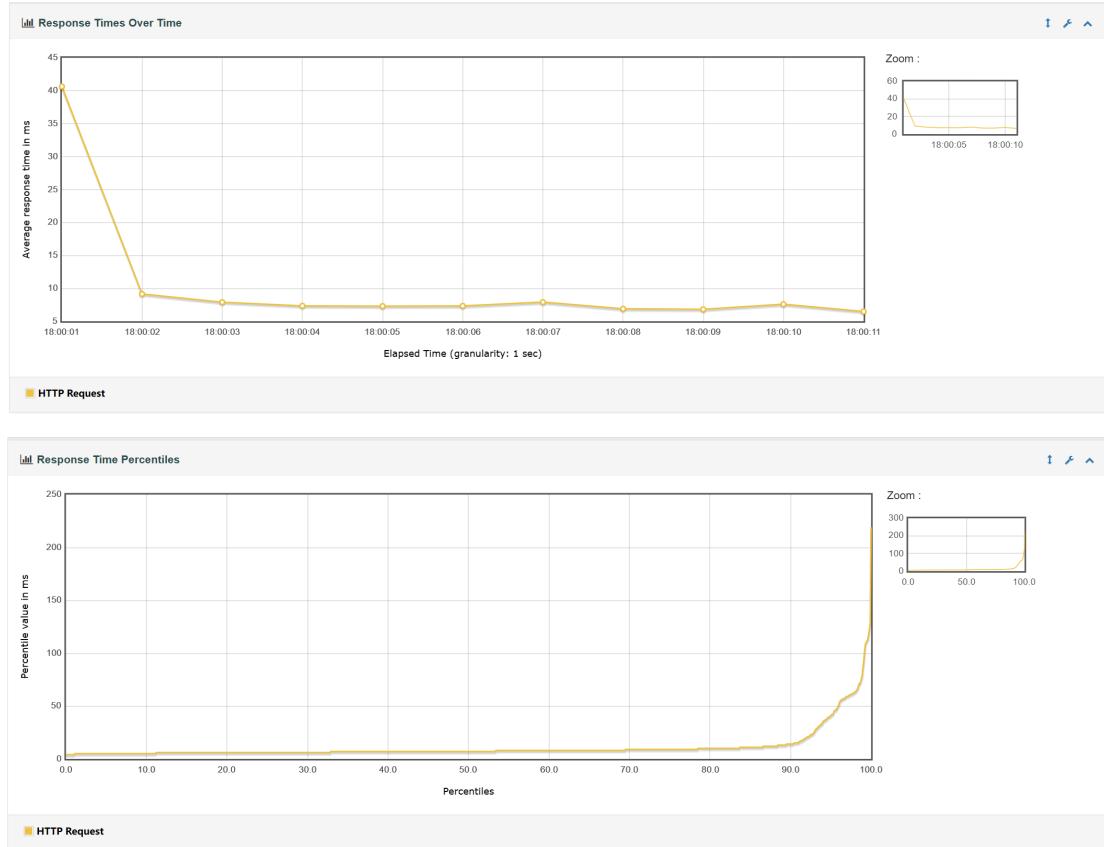
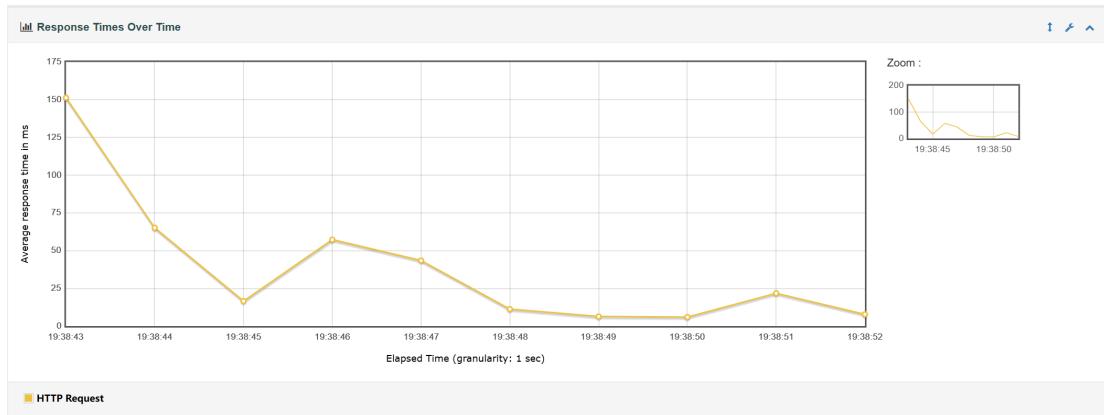


图 13 1500 线程在 10 秒内循环 5 次响应时间

本次测试响应时间稳定在 10ms 内，且有 85% 左右的请求在 10ms 内完成，说明服务器此时负载较小，没有什么压力，后续在此线程数上稍作增加进行测试。

#### 5.2.5 1550 线程 10s 内循环 5 次



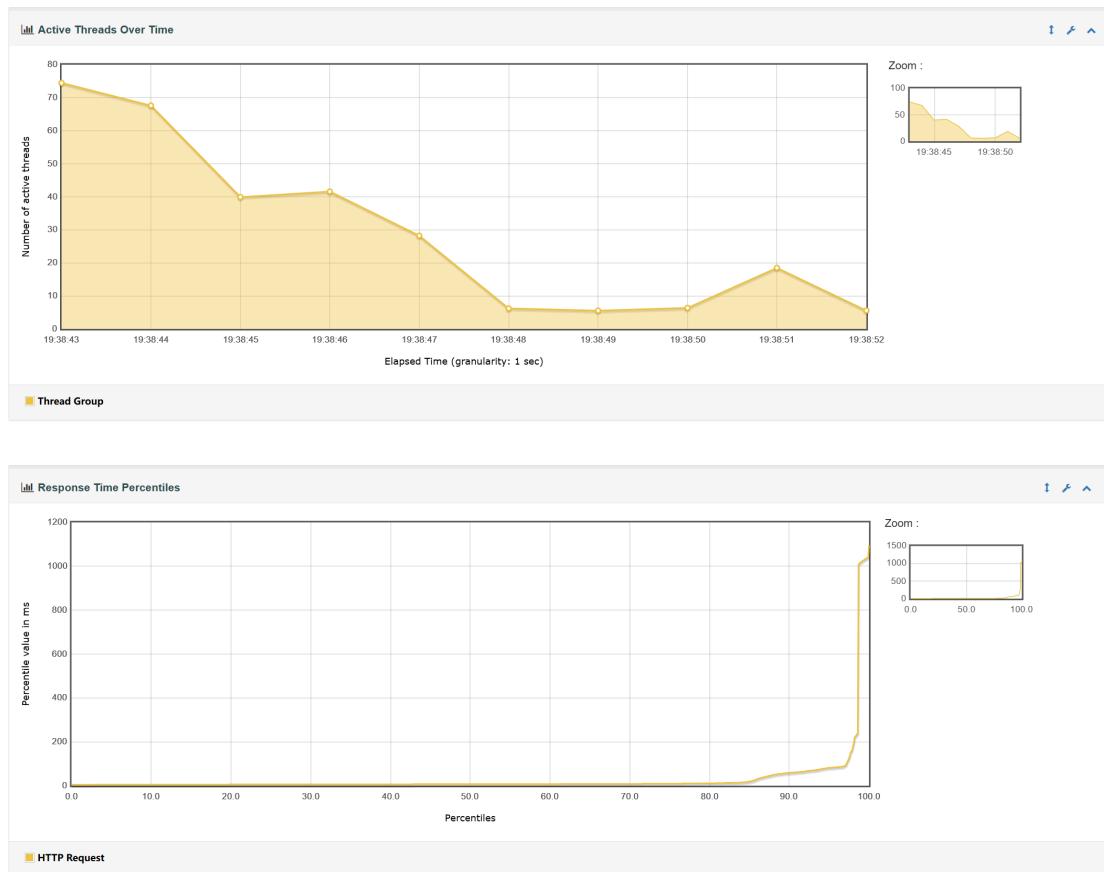
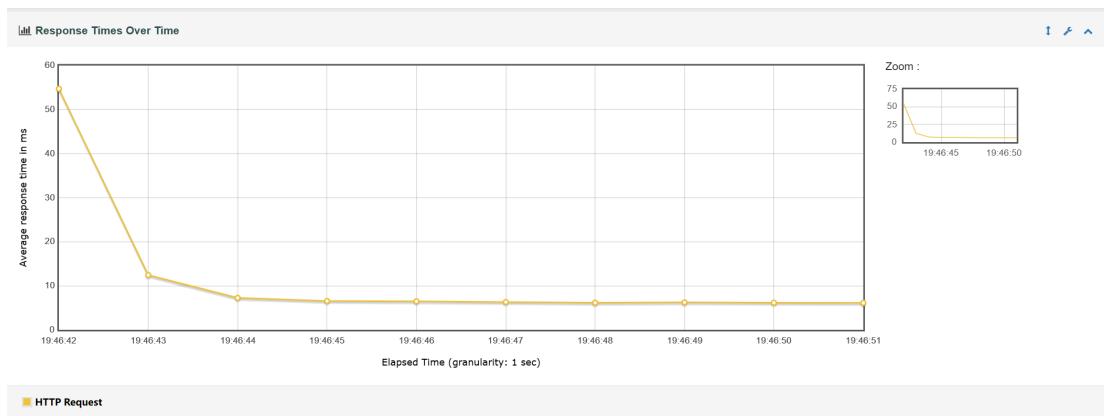


图 14 1550 线程在 10 秒内循环 5 次响应时间和活动线程数

每秒并发线程数起伏不平，每秒最高并发线程数为 74，每秒最低并发线程数为 6，可能由于线程在各个时间分布不均匀，从而导致各节点上的负载时轻时重，使得响应时间也起伏不平。总的看来，有 80% 左右的请求在 10ms 内处理完成，相比 1500 线程 10 秒内循环 5 次有所下降，此后测试增量较小。

#### 5.2.6 1580 线程 10s 内循环 5 次



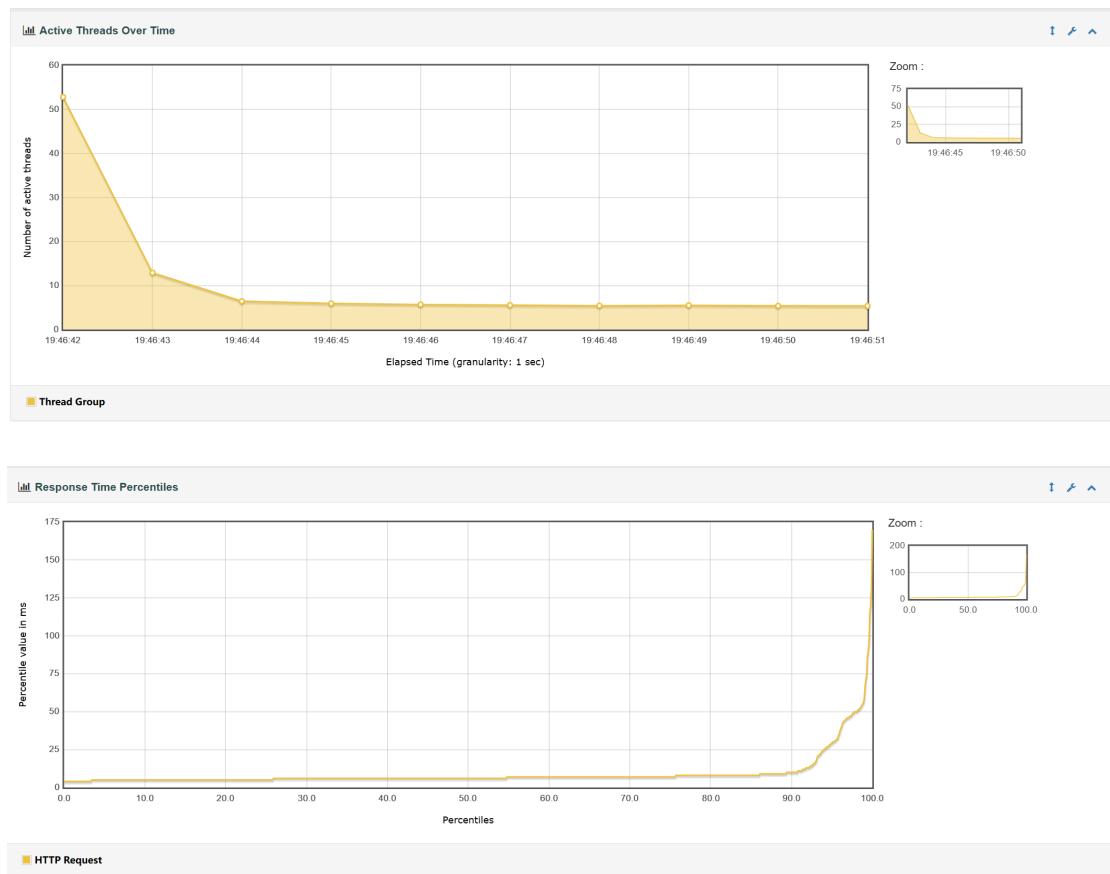
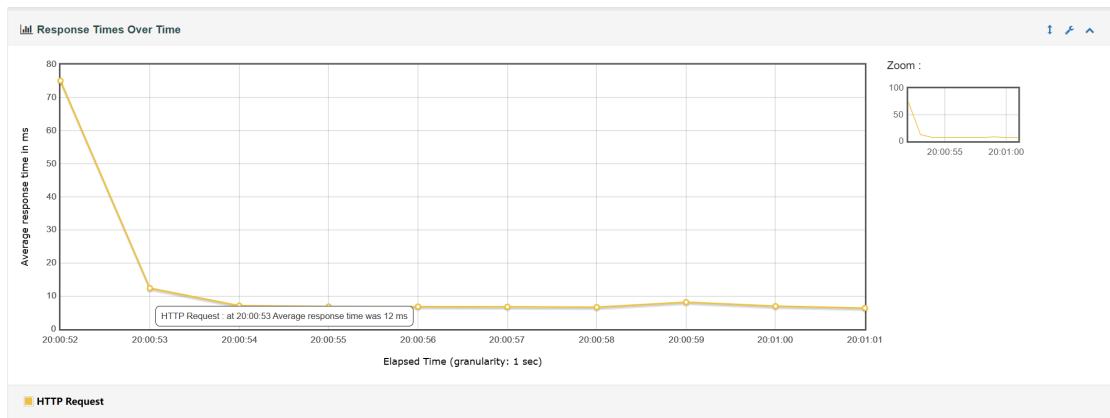


图 15 1580 线程在 10 秒内循环 5 次响应时间和活动线程数

本次测试响应时间在最初的 2 秒内相对较长，随后便稳定在了 10ms 以下，有 92% 左右的请求在 10ms 内处理完成，相比 1550 线程 10 秒内循环 5 次响应处理有所上升，这可能是由于 1580 线程在 10 秒内循环 5 次在各个结点上分布比较均匀，减少了单结点的负载从而提高了整体相应时间。

### 5.2.7 1700 线程 10s 内循环 5 次



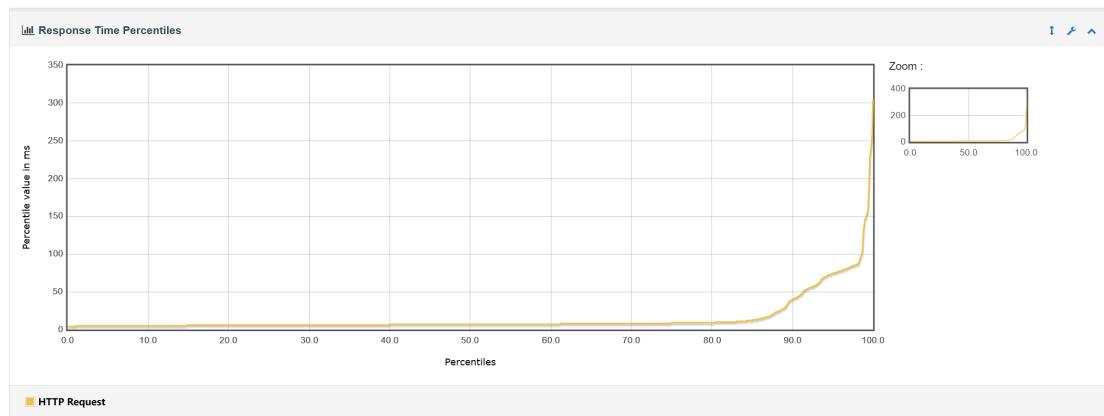


图 16 1700 线程在 10 秒内循环 5 次响应时间

本次测试响应时间和 1580 线程在 10 秒内循环 5 次的相似，但第 1s 内响应时间相对较长，总体有 81% 左右的请求在 10ms 内完成，相对于 1580 线程在 10 秒内循环 5 次有所下降。

#### 5.2.8 1850 线程 10s 内循环 5 次

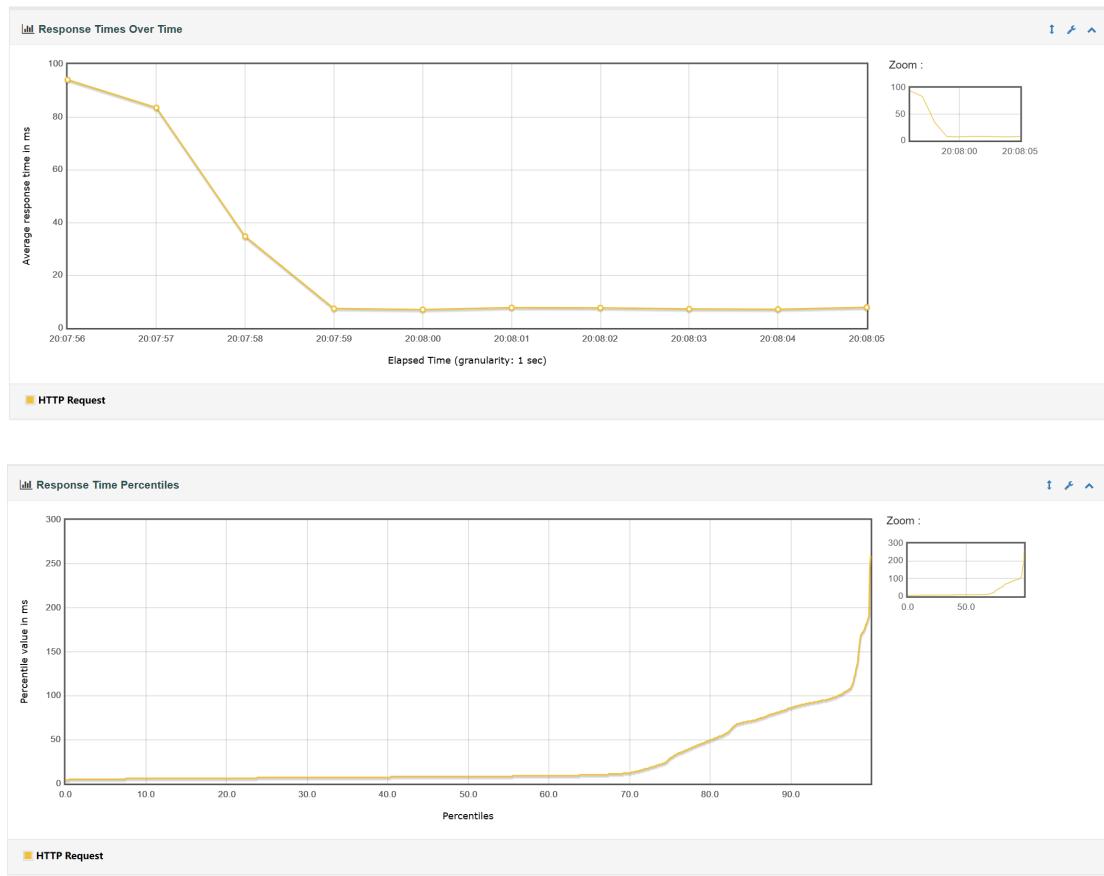


图 17 1850 线程在 10 秒内循环 5 次响应时间

本次测试中，响应时间在 3 秒后才趋于稳定，但也稳定在 10ms 以内，仅有 65% 左右的请求在 10ms 以内完成，性能相较于前面也有所下降。

**【结论】**通过上面的基于 Mysql 的 Spring 应用的写实验，可以估计该系统 10 秒内能够稳定处理的线程数

的上限大概接近 9250~1000 之间，每秒能够稳定处理的线程数在 9250/10=925 条左右。

## 6 结论

本实验验证了基于 MyBatis 的读写效率，尤其是在高并发情况下的性能表现。每次测试结果中，响应时间都在前几秒内花费时间较长，这是由于应用需要初始化建立连接以及 CPU、内存等分配会消耗资源，但随着系统趋于稳定，响应时间也趋于稳定；在请求数较多时，响应时间会出现较大的波动。

这次实验中，对数据库的写操作要比读操作快很多，可能是因为系统异步机制，数据先写入内存的缓冲区，再异步写入磁盘；也可能因为索引开销，写操作可能设计更新索引，但如果在没有复杂索引的情况下，可能写操作的开销比较低，读操作在查找数据时需要遍历索引，也可能涉及复杂的查询，如多表联接、排序和过滤等，导致较高的延迟。

在本次实验中，一个线程数所占内存为 1024kb，为了处理更多线程数可以降低线程数的大小或增大服务器内存。从而对数据库访问的优化，服务器资源的扩展，以及负载均衡与缓存策略的引入，将是提升系统性能的重要措施。

## 参考文献

- [1] <https://github.com/LeonSam1026/JMeterTest.git>
- [2] 滕欢. 基于 Spring3 的数据读写分离技术研究[D]. 哈尔滨工程大学,  
2016. [https://kns.cnki.net/kcms2/article/abstract?v=ZoyQtr8ALTdj4Vp0PniS\\_EUXFJ36gpDZy2JE\\_CaoIhkUer-S0g8TfipS-1uLanMH-Q4Z4C3h3-TFM3s5kA\\_MqSFP60\\_ctIxD8UpI-a3BIeJ1007UWunIFHmwS53AEBlhiXstIQT4972JgsYCVVH8WsAQiXcNoPiU6mfcrH1P-trHFluGkt19YUNX1eVTFc7vrfwM88nWbeI=&uniplatform=NZKPT&language=CHS](https://kns.cnki.net/kcms2/article/abstract?v=ZoyQtr8ALTdj4Vp0PniS_EUXFJ36gpDZy2JE_CaoIhkUer-S0g8TfipS-1uLanMH-Q4Z4C3h3-TFM3s5kA_MqSFP60_ctIxD8UpI-a3BIeJ1007UWunIFHmwS53AEBlhiXstIQT4972JgsYCVVH8WsAQiXcNoPiU6mfcrH1P-trHFluGkt19YUNX1eVTFc7vrfwM88nWbeI=&uniplatform=NZKPT&language=CHS)
- [3] 崔冬. 利用 Apache Jmeter 进行并发测试[J]. 网络安全和信息化, 2023, (07):94-  
96. [https://kns.cnki.net/kcms2/article/abstract?v=ZoyQtr8ALTf2CabXnh\\_Lm\\_xj0GJC40wCcK2FUADxSzyE5mokBu1NamDiS\\_dbNSvtiArzauQ1\\_2Cvkvu01h-nJigp98v6NgPby9ywiMTSy\\_CWTGoiinKFMLmScs1Knmu2PeTZ2V\\_aJttZj1PiD1xT4FscotN3vNiWJWWFsCONE=&uniplatform=NZKPT](https://kns.cnki.net/kcms2/article/abstract?v=ZoyQtr8ALTf2CabXnh_Lm_xj0GJC40wCcK2FUADxSzyE5mokBu1NamDiS_dbNSvtiArzauQ1_2Cvkvu01h-nJigp98v6NgPby9ywiMTSy_CWTGoiinKFMLmScs1Knmu2PeTZ2V_aJttZj1PiD1xT4FscotN3vNiWJWWFsCONE=&uniplatform=NZKPT)