基于 MySQL 的 Spring 应用的读写效率

摘　要:　本实验旨在评估基于 MySQL 的 Spring 应用中，使用 MyBatis 进行数据库读写操作的效率。实验通过构建一个名为 productdemoaop 的 SpringBoot 应用，实现了两个 RESTful API：一个用于管理员查询产品信息（GET /admin/products/{id}），另一个用于管理员新建产品（POST /admin/products）。实验环境包括四台配置相同的 Ubuntu 18.04 虚拟机，分别用于管理、部署应用、数据库服务以及性能测试。实验中使用了 JMeter 5.6.3 进行性能测试，以获取 API 的读写效率数据。实验结果通过对比不同 API 的响应时间和吞吐量，验证了 MyBatis 在读写数据库操作中的性能表现。实验数据通过 JMeter 的 jtl 文件记录，并在实验报告中进行了详细分析。报告还包含了实验的设计、过程、原理。

关键词:　SpringBoot, MyBatis, RESTful API, 性能测试，JMeter, MySQL

Read and Write Efficiency of Spring Applications Based on MySQL

Abstract:　This experiment aims to evaluate the efficiency of database read and write operations using MyBatis in a Spring application based on MySQL. The experiment involved the construction of a SpringBoot application named productdemoaop, which implemented two RESTful APIs: one for administrators to query product information (GET /admin/products/{id}), and another for administrators to create new products (POST /admin/products). The experimental environment consisted of four identically configured Ubuntu 18.04 virtual machines, each used for management, application deployment, database services, and performance testing. JMeter 5.6.3 was utilized for performance testing to obtain data on the read and write efficiency of the APIs. The results of the experiment, which compared the response times and throughput of different APIs, validated the performance of MyBatis in database read and write operations. The experimental data was recorded through JMeter's jtl files and was analyzed in detail in the experimental report. The report also included the design, process, and principles of the experiment.

Key words:　SpringBoot, MyBatis, RESTful API, Performance testing，JMeter, MySQL

在当前的信息技术领域，数据库的读写效率对于基于数据库的应用程序性能至关重要。Spring 框架结合 MyBatis 持久层框架，为开发高效、可维护的数据库交互提供了强大的支持。然而，不同的数据库操作和 API 设计可能会对读写效率产生显著影响。本实验旨在通过实证研究，评估和比较在基于 MySQL 数据库的 Spring 应用中，使用 MyBatis 进行数据库读写操作的效率。

实验环境搭建在四台配置相同的 Ubuntu 18.04 虚拟机上，分别承担管理、应用部署、数据库服务和性能测试的角色。实验的核心是对比分析两个 RESTful API——一个用于管理员查询产品信息（GET /admin/products/{id}），另一个用于管理员新建产品（POST /admin/products）——在 productdemoaop 应用中的性能表现。为了实现这一目标，我们采用了 JMeter 5.6.3 作为性能测试工具，以收集和分析 API 的响应时间和吞吐量等关键性能指标。

实验的主要问题包括：

1.MyBatis 在读取数据库操作中的效率如何

2.MyBatis 在写入数据库操作中的效率如何

3.读与写的效率上的差异

1　读取状态下的最大负载

1.1　　测试条件：

固定时间 10 秒，优先逐级增加线程数，测试服务器最大可分的线程数；超出最大线程后，选取合适的线程数，循环次数递增以测得服务器的最大负载。

1.2 测试指标：

Response Times Over Times、Active Threads Over Times、Response Time Percentiles。

1.3 关键指标：

Active Threads Over Times，用于判断系统在不同线程数下的处理能力

1.4 Active Threads Over Times判断示例

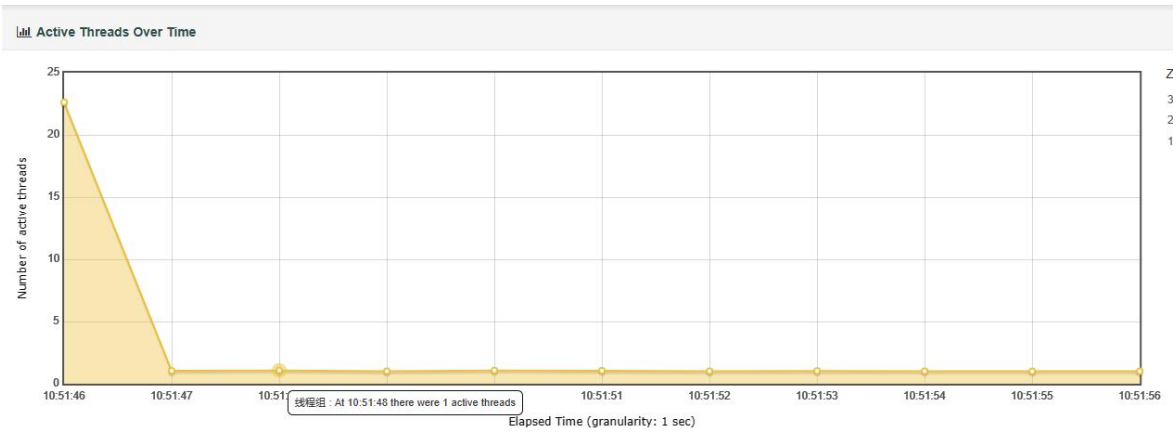


图 1：可负载示意图

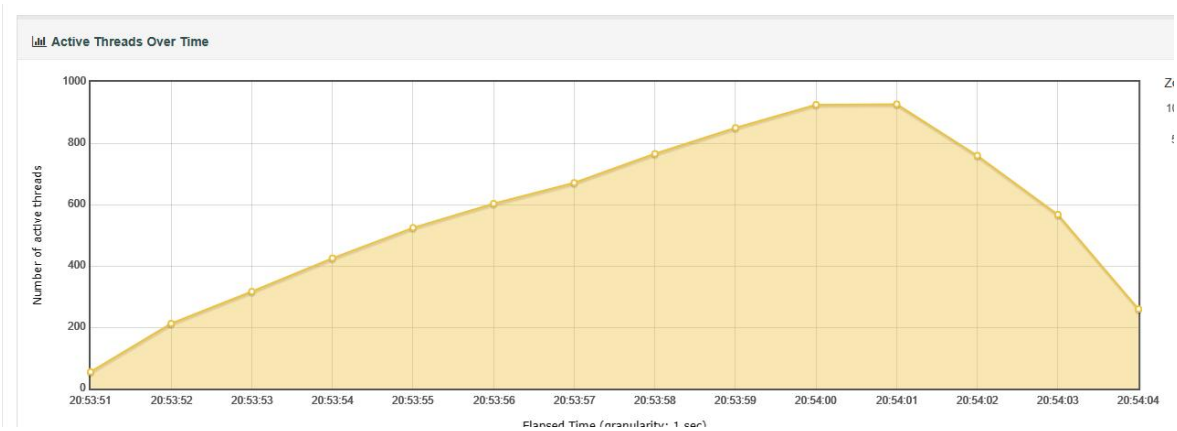


图 2：无法负载示意图

2 写入状态下的最大负载

2.1 测试条件：

固定时间 10 秒，优先逐级增加线程数，测试服务器最大可分的线程数；超出最大线程后，选取合适的线程数，循环次数递增以测得服务器的最大负载。

2.2 测试指标：

Response Times Over Times、Active Threads Over Times、Response Time Percentiles。

2.3 关键指标：

Active Threads Over Times，用于判断系统在不同线程数下的处理能力

2.4 Active Threads Over Times判断示例

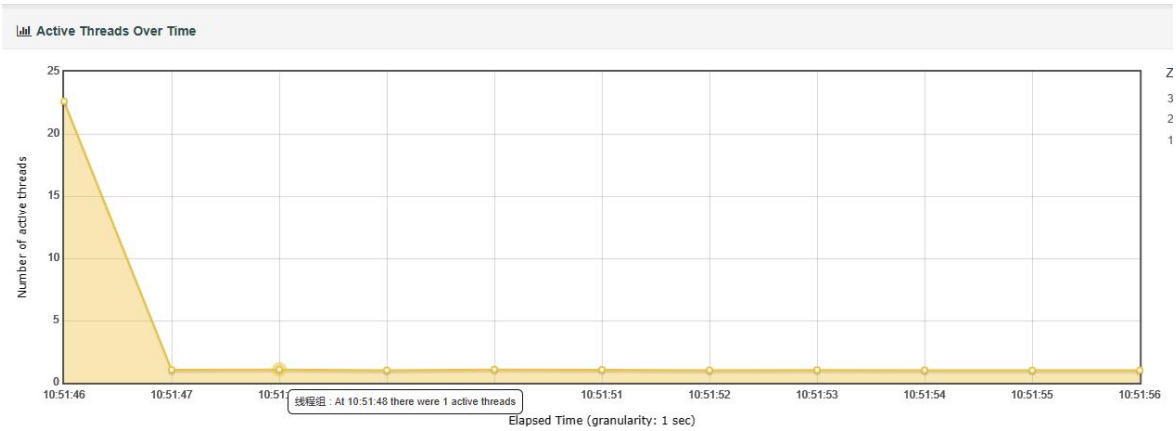


图 1：可负载示意图

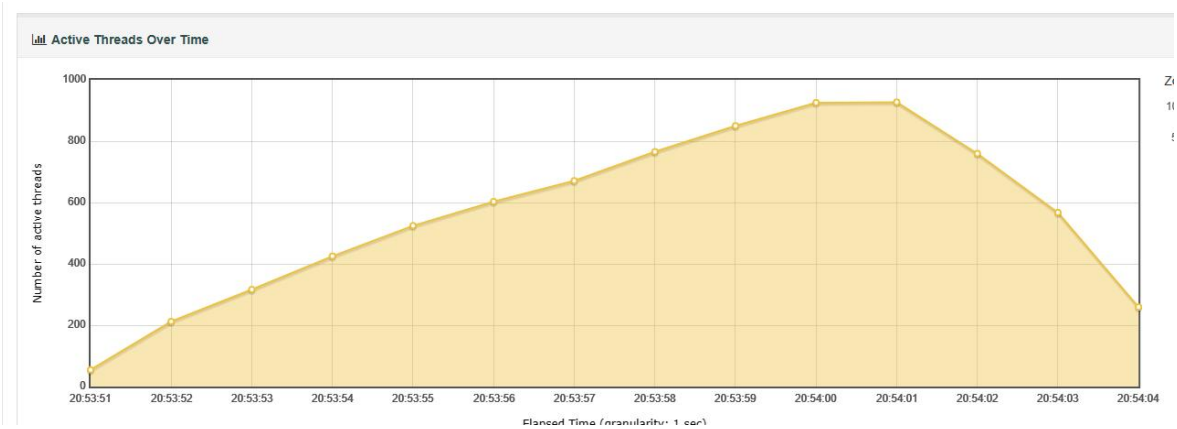
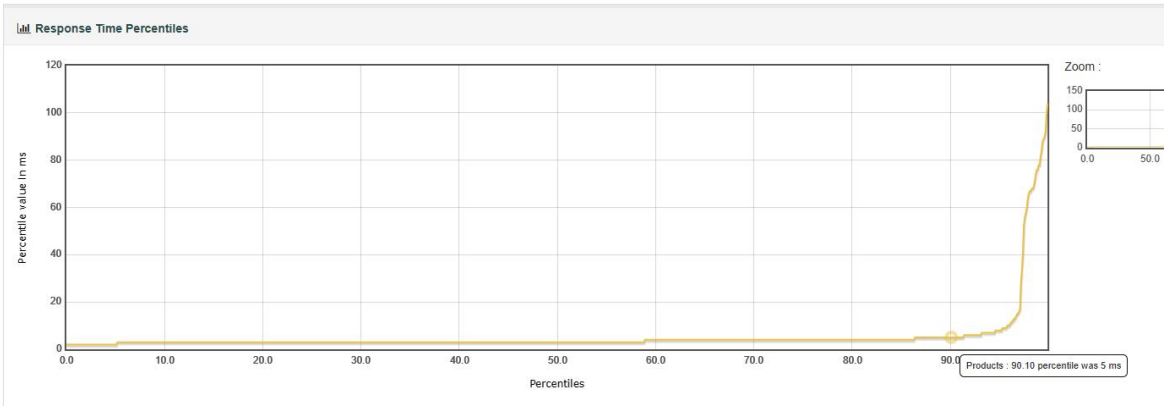


图 2：无法负载示意图

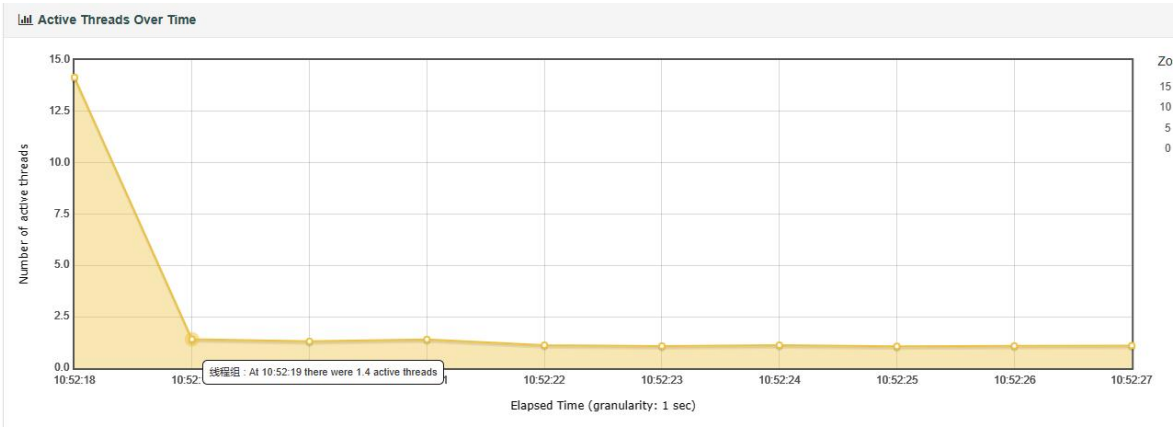
3 结果分析与讨论

3.1 R-2000-10-1

在线程数为400（每隔25ms均匀发出一个请求）和线程数为1000（每隔10ms均匀发出一个请求）时，服务器都十分稳定（原始数据见文件夹r-400-10-1与r-1000-10-1）。下面是2000线程循环1次的数据。



超过 90%的请求都在 5ms 内完成



除刚开始的一秒，所有的 active threads over time 都在 1-2ms 内完成

Statistics

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	2000	0	0.00%	5.57	2	117	3.00	5.00	8.00	77.99	203.67	116.05	36.80
Products	2000	0	0.00%	5.57	2	117	3.00	5.00	8.00	77.99	203.67	116.05	36.80

2000 个请求全部无误，此时服务器负载为 200，当前服务器的负载极低，响应时间稳定

3.2 R-3000-1

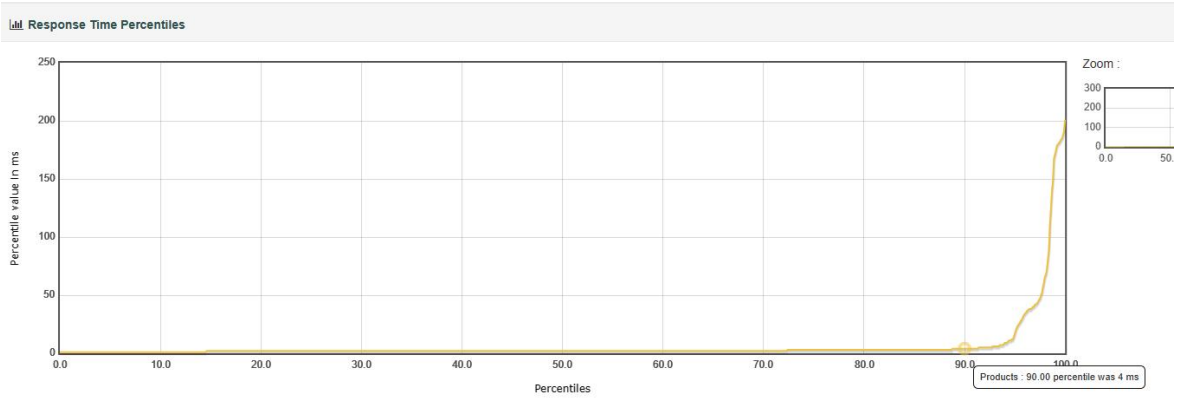
```
root@Test:~/test# vim ReadProduct.jmx
root@Test:~/test# jmeter -n -t ReadProduct.jmx -l read_jtl/read-3000-10-1.jtl -e -o read/read-3000-10-1
WARN StatusConsoleListener The use of package scanning to locate plugins is deprecated and will be removed in a future relea
e
WARN StatusConsoleListener The use of package scanning to locate plugins is deprecated and will be removed in a future relea
e
WARN StatusConsoleListener The use of package scanning to locate plugins is deprecated and will be removed in a future relea
e
WARN StatusConsoleListener The use of package scanning to locate plugins is deprecated and will be removed in a future relea
e
Creating summariser <summary>
Created the tree successfully using ReadProduct.jmx
Starting standalone test @ 2024 Oct 19 10:52:50 CST (1729306370795)
Waiting for possible Shutdown/StopTestNow/HeapDump/ThreadDump message on port 4445
[3.750s][warning][os,thread] Failed to start thread "Unknown thread" - pthread_create failed (EAGAIN) for attributes: stacks
ze: 1024k, guardsize: 0k, detached.
[3.750s][warning][os,thread] Failed to start the native thread for java.lang.Thread "线程组 1-2688"
```

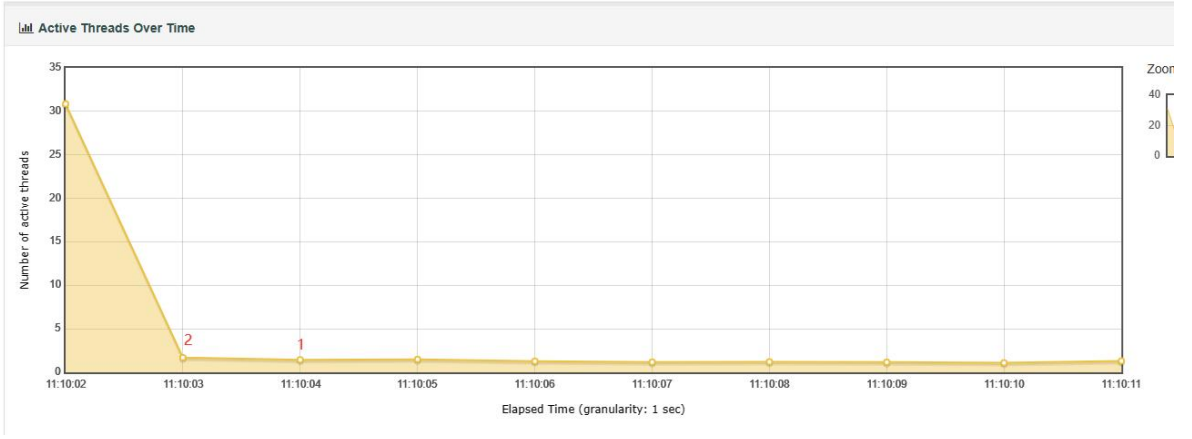
服务器无法分出 3000 个线程

```
root@Test:~/test# ulimit -u
2904
```

服务器线程数最大应为 2904，但是，经测试，实际最大应为 2300（详见 r-2300-10-1 成功运行，而 r-2400-10-1 运行失败）

3.3 R-2000-10-2



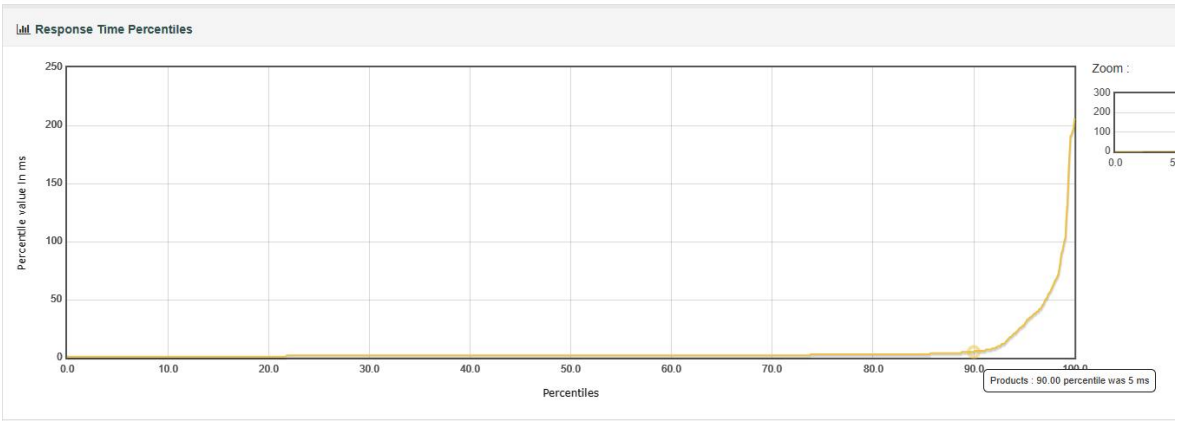


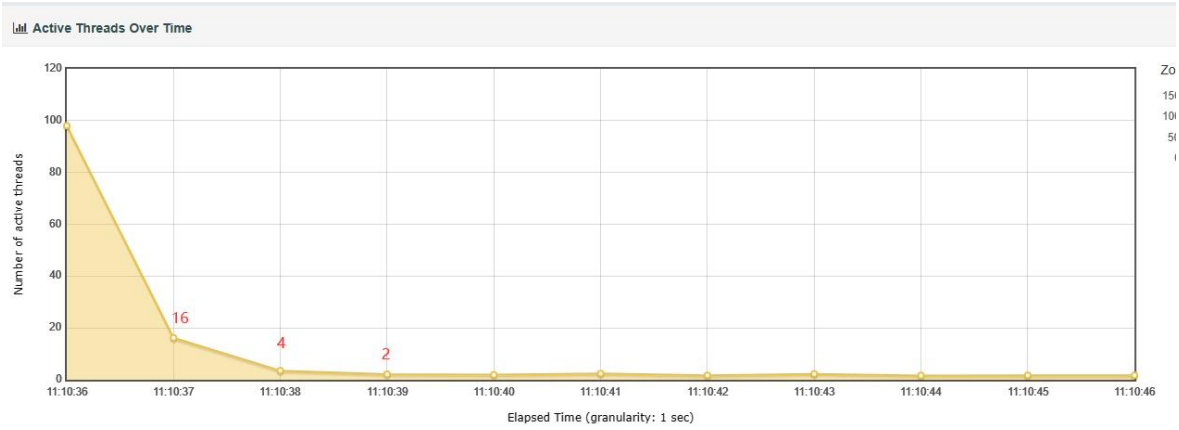
Statistics

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**4000**	**0**	**0.00%**	**6.28**	**1**	**201**	**2.00**	**4.00**	**18.90**	**171.00**	**408.58**	**232.77**	**73.82**
Products	4000	0	0.00%	6.28	1	201	2.00	4.00	18.90	171.00	408.58	232.77	73.82

负载为 400，服务器依然可以正常响应

3.4 R-2000-10-4



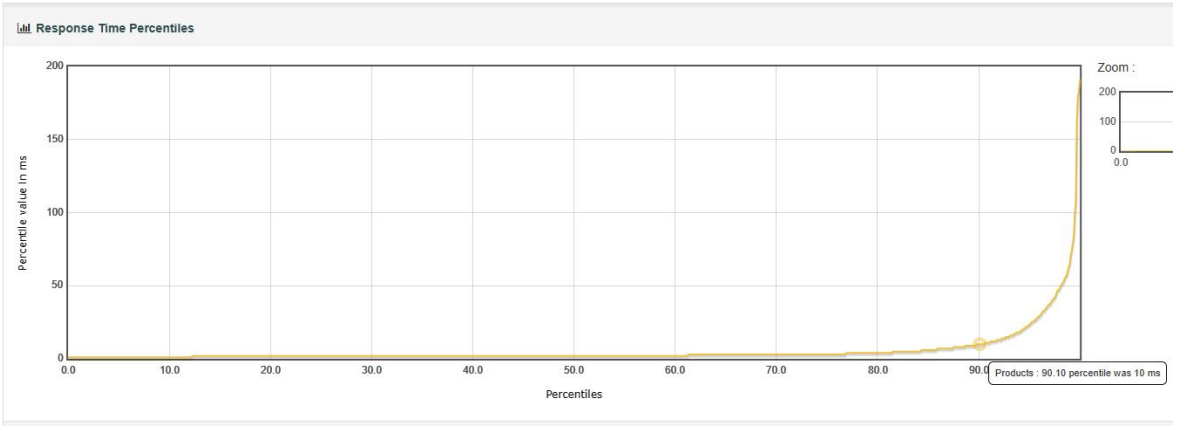


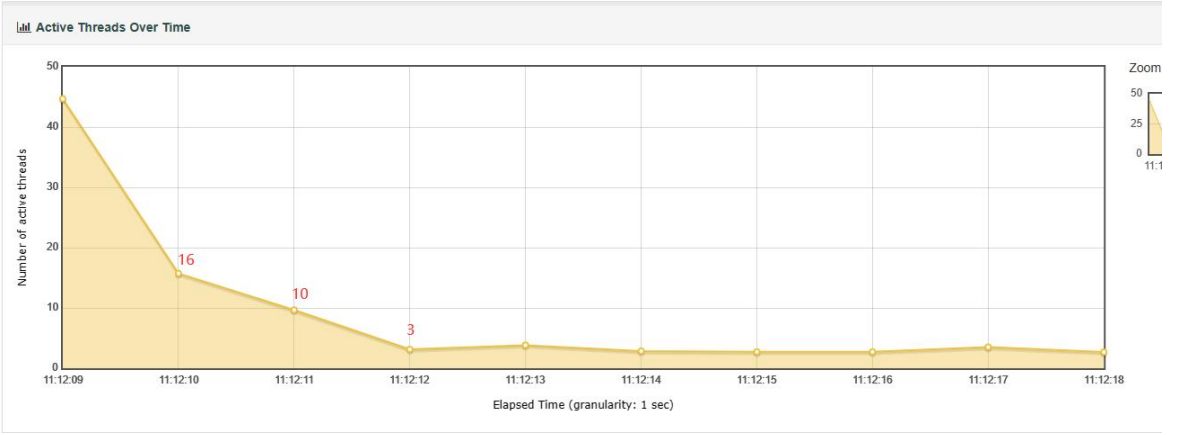
Statistics

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	8000	0	0.00%	6.18	1	206	2.00	5.00	28.95	101.00	814.58	464.13	147.17
Products	8000	0	0.00%	6.18	1	206	2.00	5.00	28.95	101.00	814.58	464.13	147.17

负载为 800，服务器依然可以正常响应

3.5 R-2000-10-6



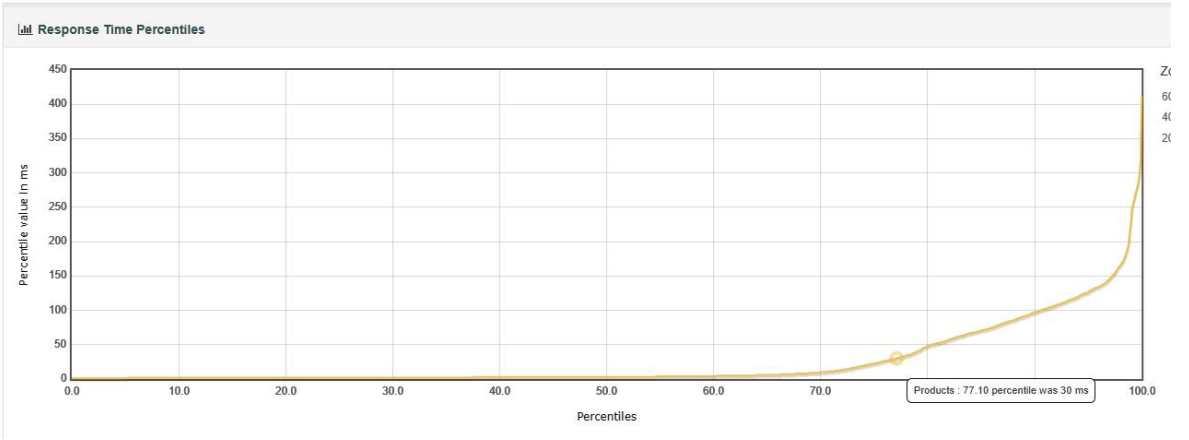


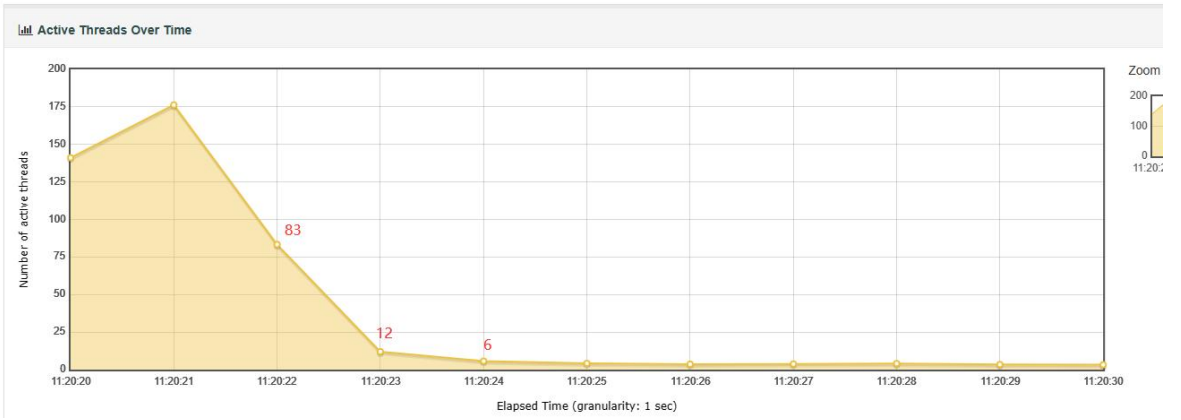
Statistics

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	12000	0	0.00%	5.81	1	191	2.00	10.00	23.00	65.99	1220.88	695.61	220.57
Products	12000	0	0.00%	5.81	1	191	2.00	10.00	23.00	65.99	1220.88	695.61	220.57

负载 1200，服务器任然可以应对

3.6 R-2000-10-7

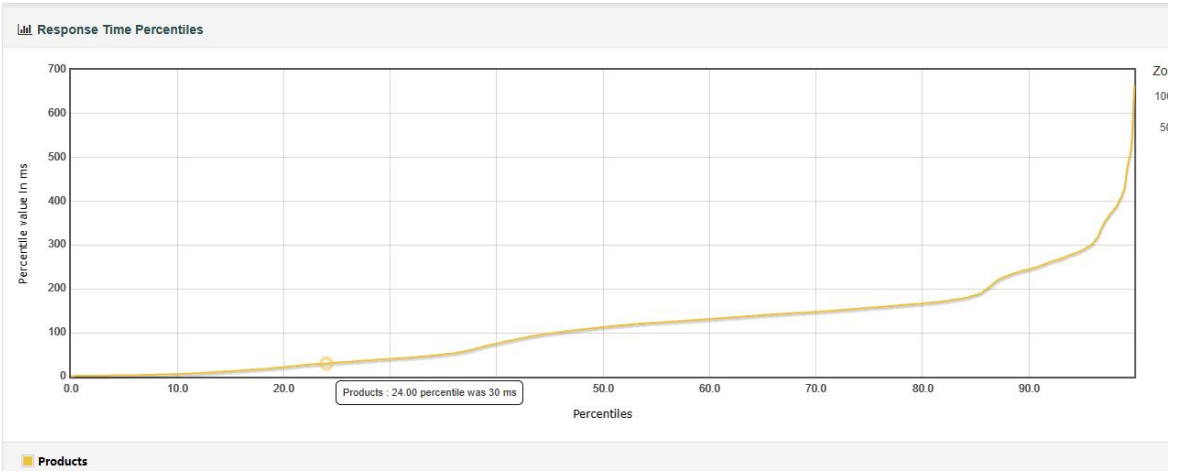


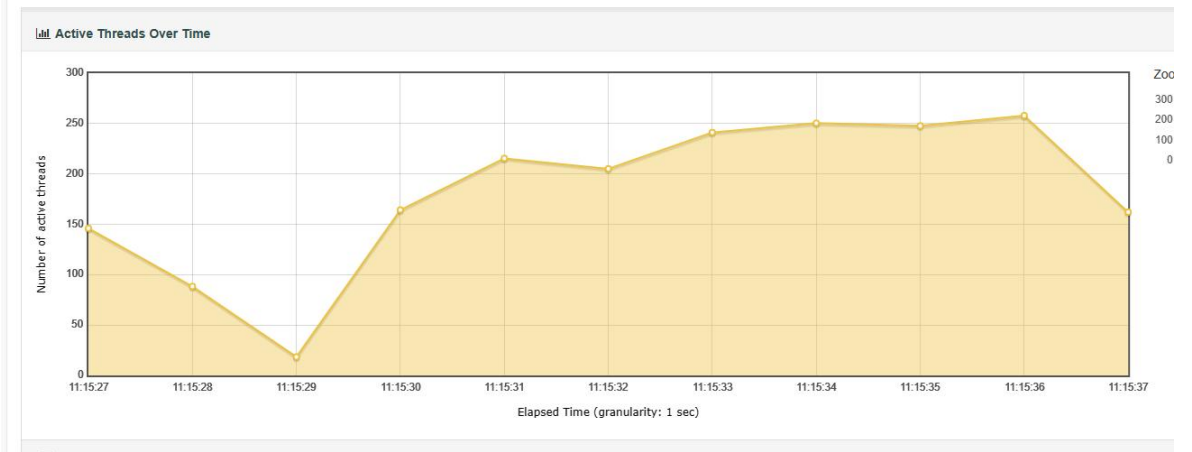


Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	14000	0	0.00%	26.04	1	411	3.00	97.00	126.95	228.00	1431.49	15.52	258.62
Products	14000	0	0.00%	26.04	1	411	3.00	97.00	126.95	228.00	1431.49	15.52	258.62

负载 1400，服务器的响应速度有所降低，但还能响应。其中第二秒时 active threads over time 的升高推测可能与网络波动有关

3.7 R-2000-10-8



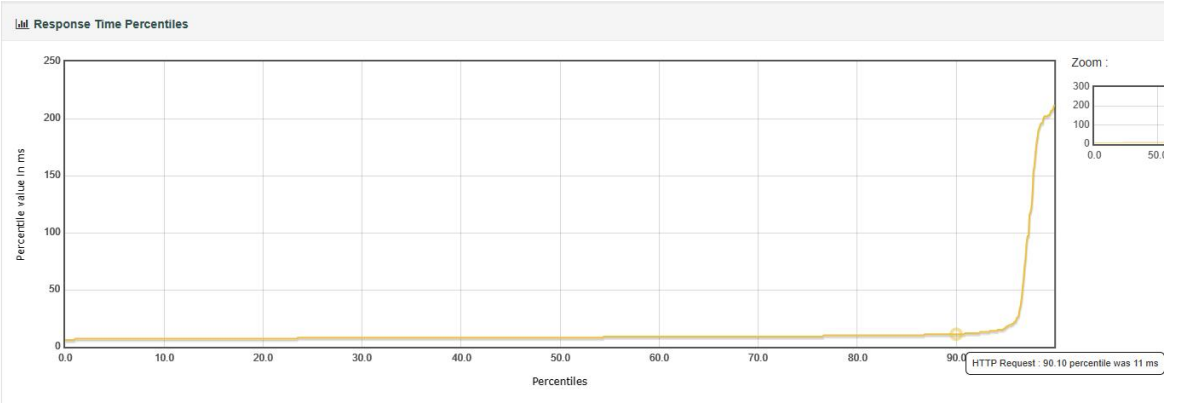


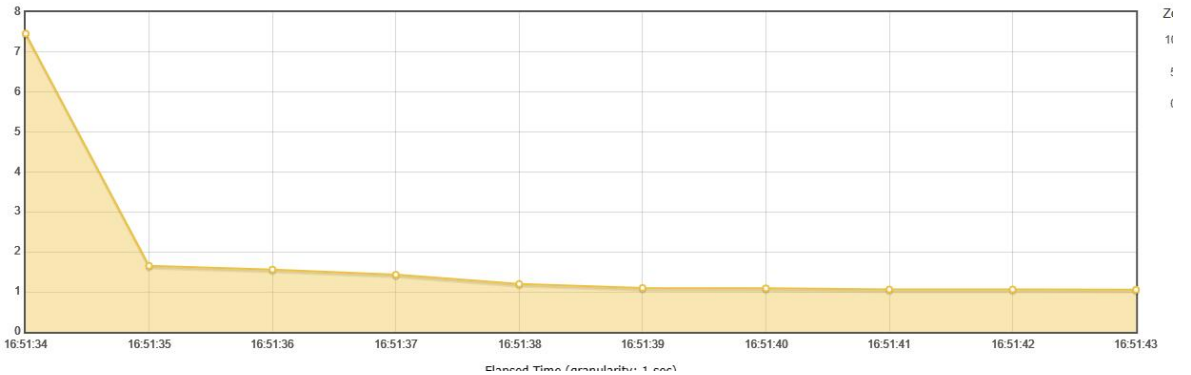
Statistics

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**16000**	**0**	**0.00%**	**114.07**	**1**	**919**	**113.00**	**245.00**	**288.95**	**432.00**	**1525.84**	**869.46**	**275.67**
Products	16000	0	0.00%	114.07	1	919	113.00	245.00	288.95	432.00	1525.84	869.46	275.67

服务器发生阻塞，已经无法正常处理这些请求

3.8 W-1000-10-1



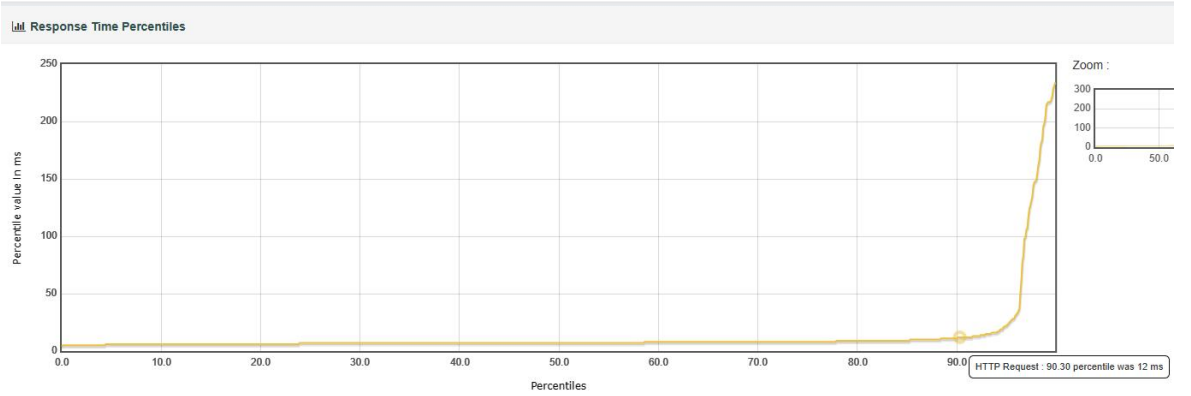


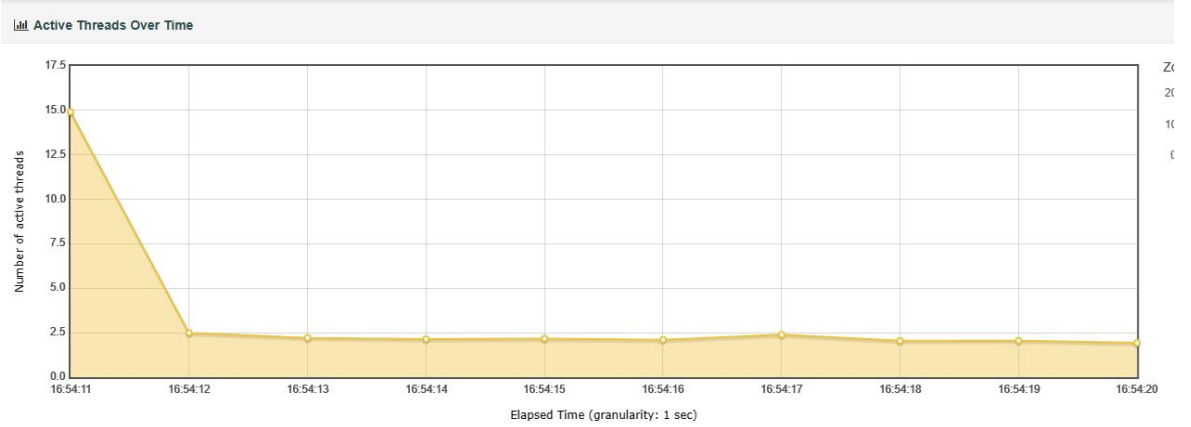
Statistics

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**1000**	**0**	**0.00%**	**13.82**	**6**	**212**	**8.00**	**11.00**	**16.95**	**202.00**	**101.28**	**46.26**	**31.92**
HTTP Request	1000	0	0.00%	13.82	6	212	8.00	11.00	16.95	202.00	101.28	46.26	31.92

当前服务器的负载低，响应时间稳定

3.9 W-2000-10-1





Statistics

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	2000	0	0.00%	13.45	4	237	7.00	11.90	22.95	213.90	202.41	92.46	63.80
HTTP Request	2000	0	0.00%	13.45	4	237	7.00	11.90	22.95	213.90	202.41	92.46	63.80

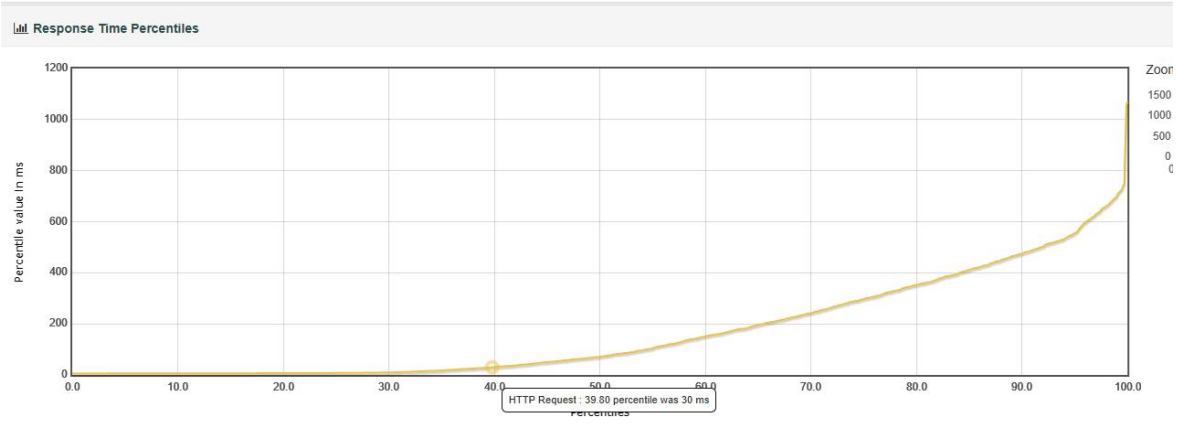
当前服务器的负载低，响应时间稳定

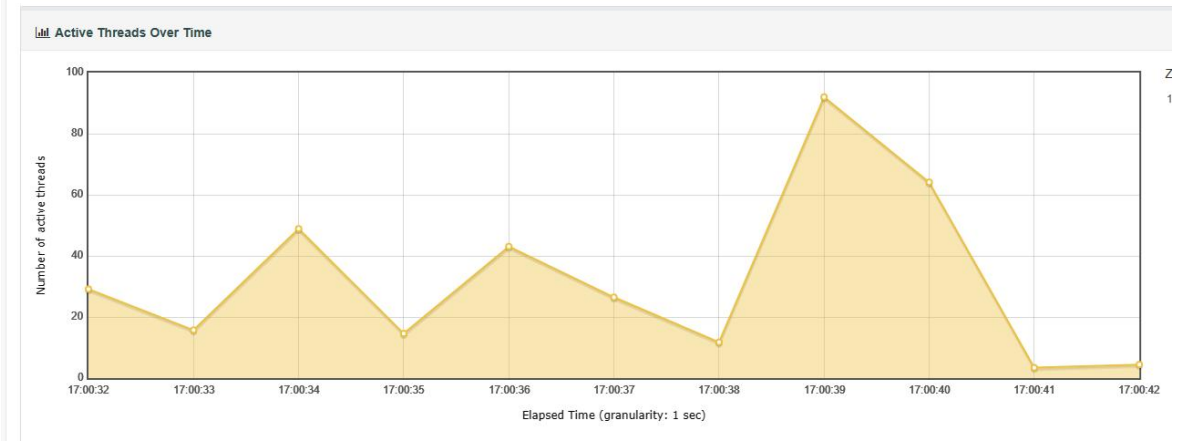
3.10 W-3000-10-1

```
root@Test:~/test# vim WriteProduct.jmx
root@Test:~/test# jmeter -n -t WriteProduct.jmx -l write_jtl/write-3000-10-1.jtl -e -o write/write-3000-10-1
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
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 WriteProduct.jmx
Starting standalone test @ 2024 Oct 19 16:57:13 CST (1729328233709)
Waiting for possible Shutdown/StopTestNow/HeapDump/ThreadDump message on port 4445
[3.135s][warning][os,thread] Failed to start thread "Unknown thread" - pthread_create failed (EAGAIN) for attributes: stacksize: 1024k, guardsize: 0k, deta
ched.
[3.135s][warning][os,thread] Failed to start the native thread for java.lang.Thread "Thread Group 1-2466"
root@Test:~/test#
```

服务器无法分出 3000 个线程

3.11 W-2200-10-1



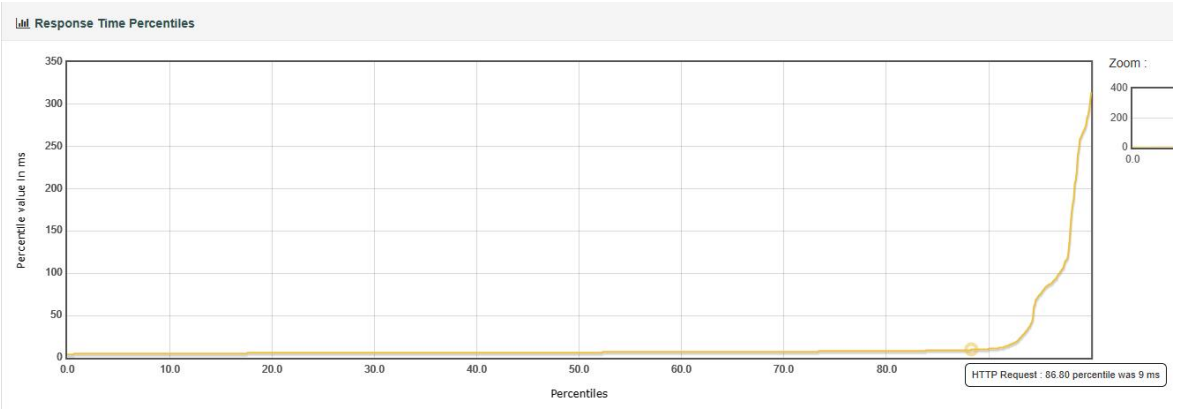


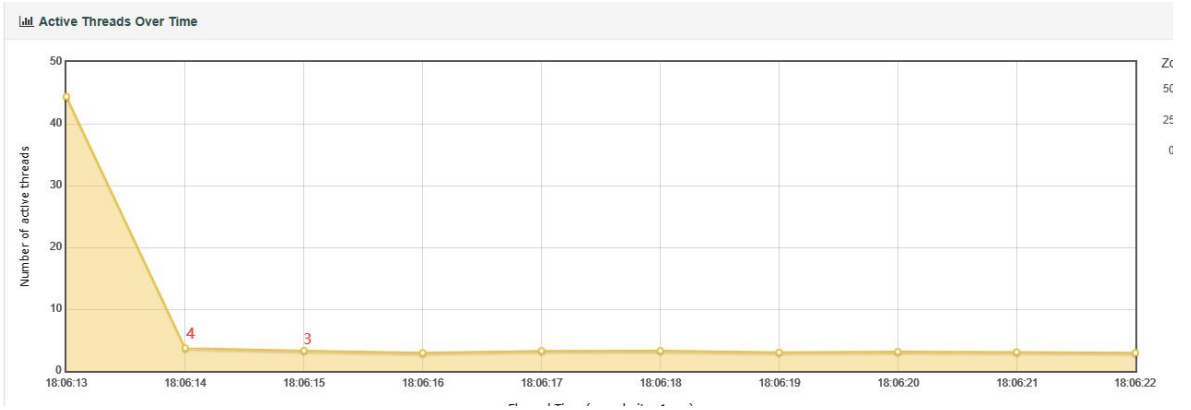
Statistics

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**2200**	**0**	**0.00%**	**168.69**	**5**	**1071**	**71.50**	**475.00**	**554.00**	**699.99**	**201.28**	**91.94**	**63.44**
HTTP Request	2200	0	0.00%	168.69	5	1071	71.50	475.00	554.00	699.99	201.28	91.94	63.44

经测试，服务器在写入操作时最多分出的线程约为 2200

3.12 W-2000-10-2



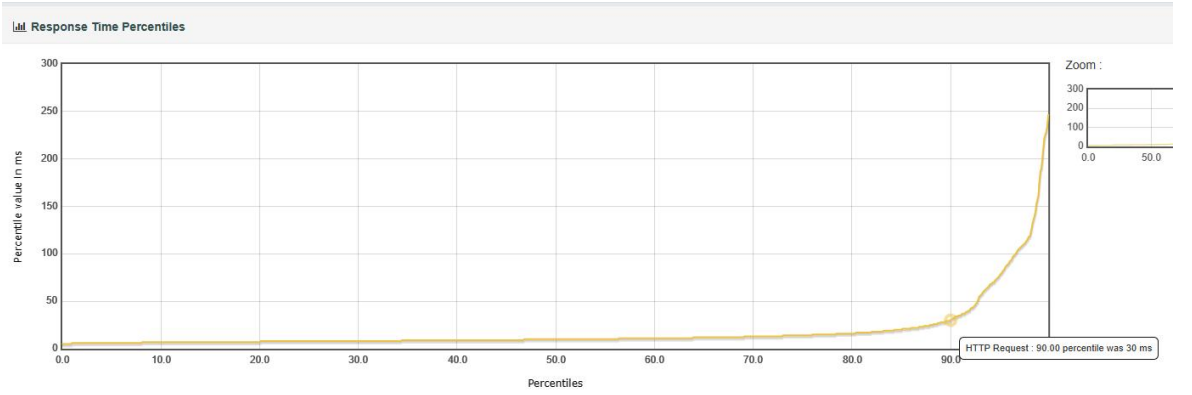


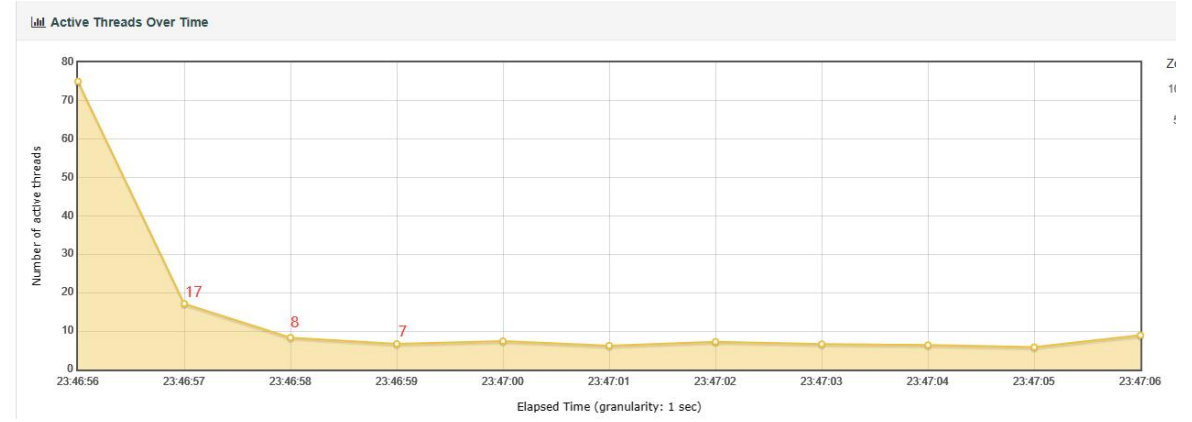
Statistics

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	4000	0	0.00%	15.05		314	6.00	11.00	75.00	262.98	408.12	86.44	128.65
HTTP Request	4000	0	0.00%	15.05		314	6.00	11.00	75.00	262.98	408.12	86.44	128.65

负载为 400，服务器依然可以正常响应

3.13 W-2000-10-3



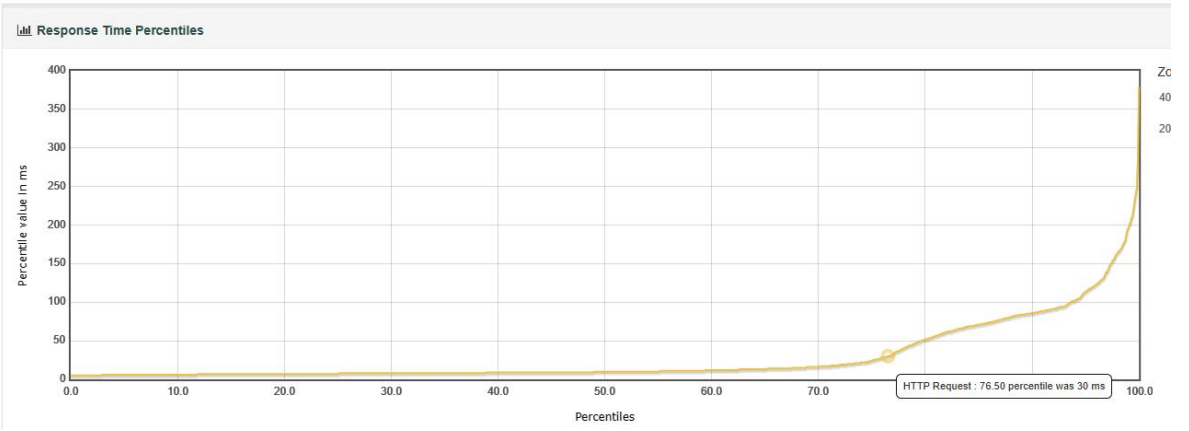


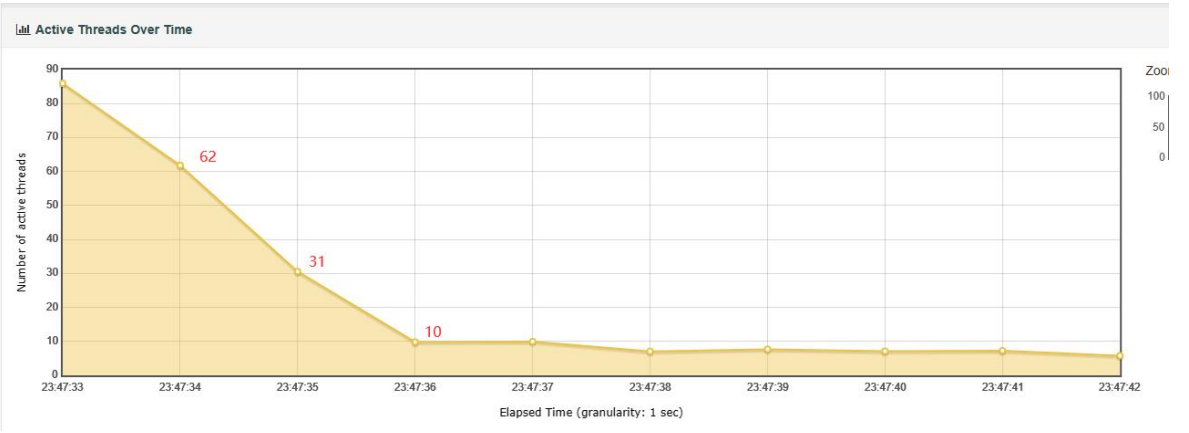
Statistics

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	6000	0	0.00%	18.74	4	358	10.00	30.00	77.95	176.00	604.47	276.77	190.58
HTTP Request	6000	0	0.00%	18.74	4	358	10.00	30.00	77.95	176.00	604.47	276.77	190.58

负载为 600，服务器依然可以正常响应

3.14 W-2000-10-4



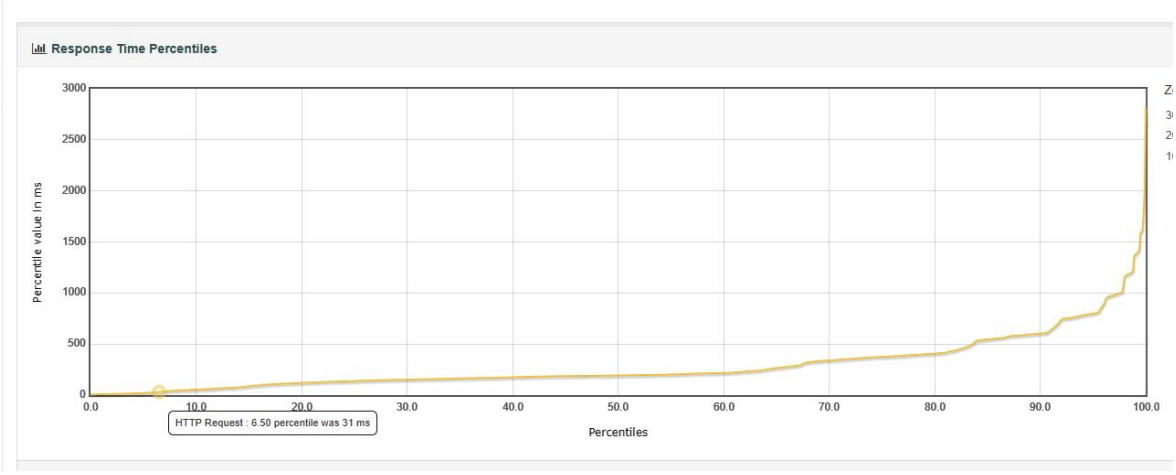


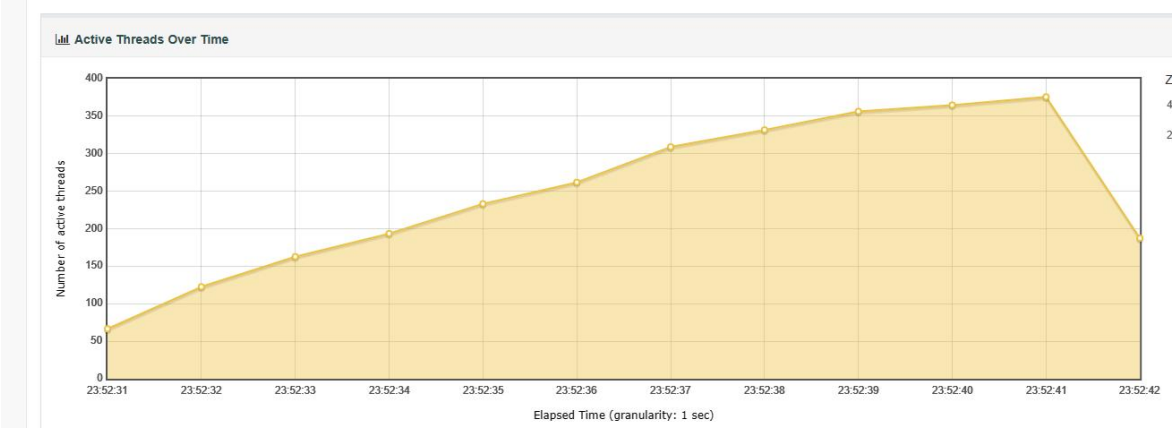
Statistics

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	8000	0	0.00%	28.86	4	379	10.00	86.00	113.00	196.00	811.19	371.39	255.73
HTTP Request	8000	0	0.00%	28.86	4	379	10.00	86.00	113.00	196.00	811.19	371.39	255.73

负载为 800，服务器依然可以正常响应，但响应时间明显变慢

3.15 W-2000-10-5





Statistics

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	10000	0	0.00%	288.93	5	2819	193.00	603.00	799.00	1374.00	900.58	412.29	283.88
HTTP Request	10000	0	0.00%	288.93	5	2819	193.00	603.00	799.00	1374.00	900.58	412.29	283.88

服务器发生阻塞，已经无法正常处理这些请求

4 总 结

4.1 MyBatis在读写数据库操作中的效率

性能瓶颈：

随着线程数的增加，系统性能逐渐下降

最大线程数：

根据测试结果，系统在 1000 线程下表现最佳，2000 线程下仍可接受，但超过 2000 线程后性能急剧下降，实际最大线程数约为 2200-2300。

最大负载：

由于多次测试后数据库性能不稳定，跑崩需要的请求频率难以测出，测试能够无错误运行的最大负载：读取 1400 线程响应/秒，写入 800 线程响应/秒

表一：Mybatis 读写数据库性能分析

操作类型	无错误运行的最大每秒线程数	每秒 100 线程时平均响应时间(0)s
读取	1400	5.57
写入	800	13.45

由此得出，使用 Mybatis 读取数据库操作比写入数据库操作快。

提升数据库操作效率的技巧与最佳实践[1]：

1.合理的数据库设计和索引优化：通过合理的数据库表设计和索引优化，可以提高查询性能。

2.使用简单而高效的查询方式：尽量避免使用复杂的子查询和联合查询，采用简单而高效的查询方式。

3.充分利用动态 SQL：使用 MyBatis 的动态 SQL 功能，根据不同的条件动态生成 SQL 语句，避免无谓的数据库查询。

4.优化数据库连接管理：合理配置数据库连接池和连接的超时时间，管理和复用数据库连接资源，减少连接的创建和销毁开销。

5.配置缓存机制：通过配置 MyBatis 的缓存机制，减少数据库查询的次数，提高查询效率。

4.2 读与写的效率上的差异

一般来说，读操作比写操作更快，这是因为读操作不需要修改数据，只需要从数据库中读取数据即可，而写操作需要修改数据库中的数据，这会涉及到磁盘IO、锁等操作，因此会比读操作慢一些。 但是，不同类型的数据库在读写操作上的表现也有所不同，这取决于数据库的实现方式和优化策略。[2]

参考文献:

[1] 深入解析 MyBatis：提升数据库操作效率的技术指南. https://blog.csdn.net/weixin_57094599/article/details/131799410

[2] 各个数据库的读和写那个更快一些. https://blog.csdn.net/weixin_59383491/article/details/130266887