Mongo 数据库的读写效率

摘　要:　本实验旨在探究 Mongo 数据库的读写效率，并与关系数据库进行比较。实验在 Mongo 集群环境下进行，使用 SpringBoot 应用访问 Mongo 数据库，并利用 JMeter 进行性能测试。通过分析 JMeter 的测试结果和主机监控数据，我们得出了 Mongo 数据库在不同负载下的读写效率。

关键词:　Mongo 数据库，读写效率，SpringBoot，JMeter，性能测试。

Based on Mongo exp Plan

Abstract:　This experiment aims to explore the read and write efficiency of MongoDB and compare it with relational databases. Conducted in a MongoDB cluster environment, the experiment accesses the MongoDB using a SpringBoot application and utilizes JMeter for performance testing. By analyzing the results from JMeter and host monitoring data, we derived the read and write efficiency of MongoDB under various loads.

Key words:　MongoDB, Read/Write Efficiency, SpringBoot, JMeter, Performance Testing.

Mongo 数据库以其高性能的读写能力而闻名，尤其是在处理大量数据时。然而，对于 Mongo 数据库在不同负载下的读写效率，尤其是与关系数据库相比，缺乏详细的实验数据和分析。本实验旨在通过构建 Mongo 集群环境，使用 SpringBoot 应用进行数据访问，并运用 JMeter 工具进行性能测试，来量化 Mongo 数据库的读写效率，并分析其在高并发情况下的表现。

主要问题：

1.Mongo 数据库读写方案的效率差距

2.Mongo 数据库读写方案的 CPU 和内存使用情况

1　使用 Jmeter 测试方案的速度差异

1.1　测试条件：

固定时间，保证相同线程与循环状态，进行比较

1.2　测试指标：

Response Times Over Times、Active Threads Over Times、Response Time Percentiles、平均响应时间

1.3　关键指标：

Response Time Percentiles，响应速度前 80%的请求所用的时间

2　使用华为云云监控服务监控服务器 B 和 C 的负载情况

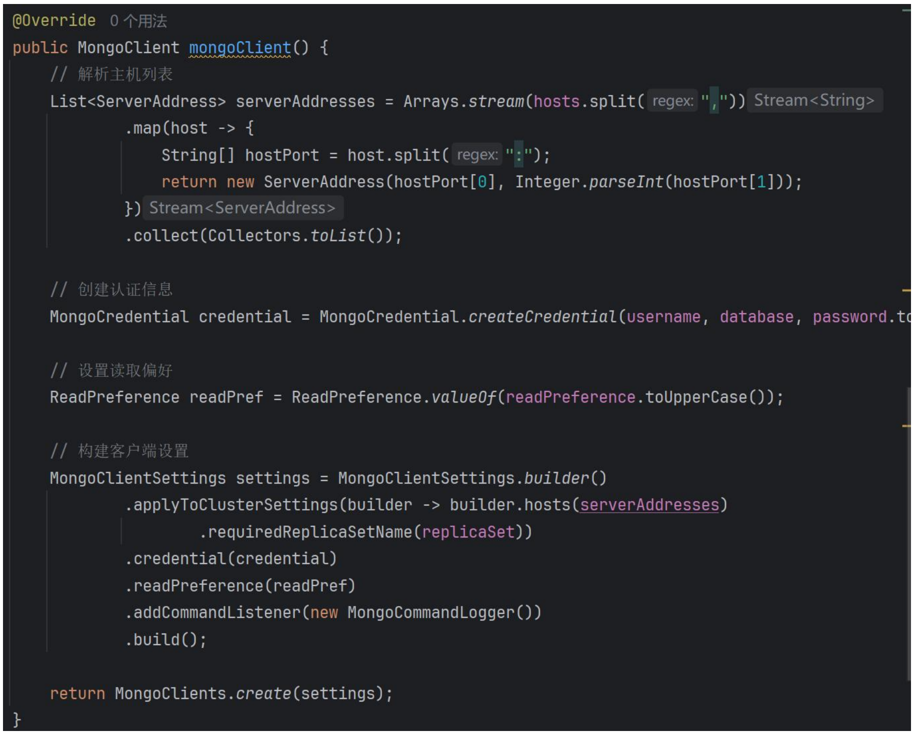
2.1　测试条件：

固定实验时间，通过华为云的云监控服务，比较相同线程与循环状态下 Mongo 数据库读写方案的 node1 和 Mongo 服务器的负载情况

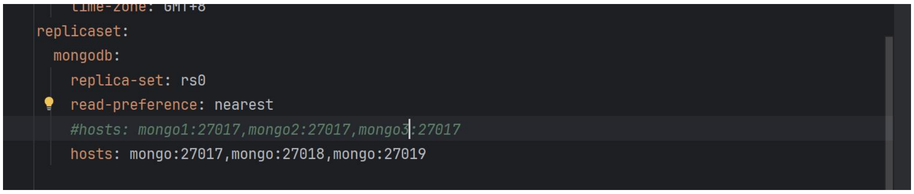
Github仓库链接：https://github.com/X-Believer/JavaEE-2-4/tree/master/Exp7

2.2　　方案设计

（1）Mongo 方案配置



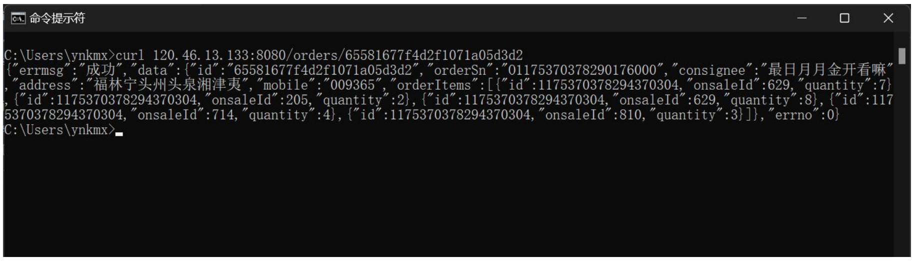
```
@Override  0 个用法
public MongoClient mongoClient() {
    // 解析主机列表
    List<ServerAddress> serverAddresses = Arrays.stream(hosts.split( regex: ",")) Stream<String>
            .map(host -> {
                String[] hostPort = host.split( regex: ":");
                return new ServerAddress(hostPort[0], Integer.parseInt(hostPort[1]));
            }) Stream<ServerAddress>
            .collect(Collectors.toList());

    // 创建认证信息
    MongoCredential credential = MongoCredential.createCredential(username, database, password.tc

    // 设置读取偏好
    ReadPreference readPref = ReadPreference.valueOf(readPreference.toUpperCase());

    // 构建客户端设置
    MongoClientSettings settings = MongoClientSettings.builder()
            .applyToClusterSettings(builder -> builder.hosts(serverAddresses)
                    .requiredReplicaSetName(replicaSet))
            .credential(credential)
            .readPreference(readPref)
            .addCommandListener(new MongoCommandLogger())
            .build();

    return MongoClients.create(settings);
}
```



```
  time-zone: GMT+8
replicaset:
  mongodb:
    replica-set: rs0
    read-preference: nearest
    #hosts: mongo1:27017,mongo2:27017,mongo3:27017
    hosts: mongo:27017,mongo:27018,mongo:27019
```

修改原代码，配置 Mongo 集群连接

（2）运行效果

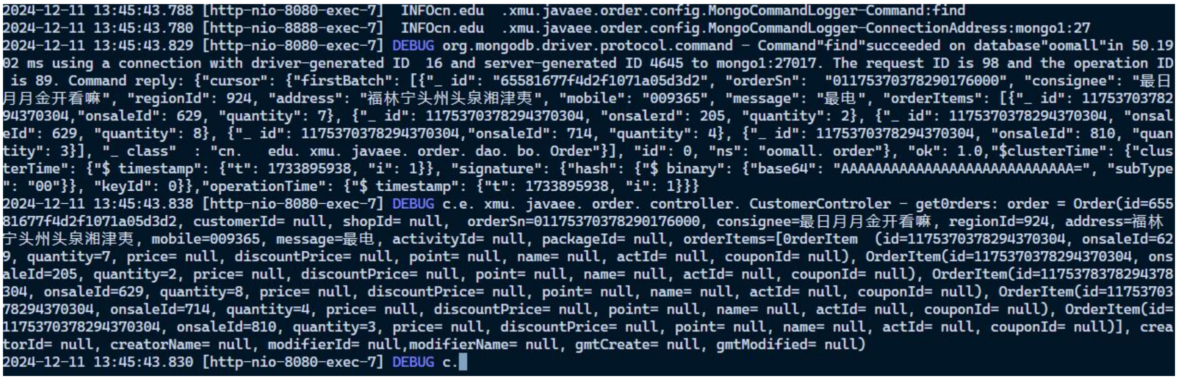


```
C:\Users\ynkmx>curl 120.46.13.133:8080/orders/65581677f4d2f1071a05d3d2
{"errmsg":"成功","data":{"id":"65581677f4d2f1071a05d3d2","orderSn":"01175370378290176000","consignee":"最日月月金开看嘛"
,"address":"福林宁头州头泉湘津夷","mobile":"009365","orderItems":[{"id":1175370378294370304,"onsaleId":629,"quantity":7}
,{"id":1175370378294370304,"onsaleId":205,"quantity":2},{"id":1175370378294370304,"onsaleId":629,"quantity":8},{"id":117
5370378294370304,"onsaleId":714,"quantity":4},{"id":1175370378294370304,"onsaleId":810,"quantity":3}]},"errno":0}
C:\Users\ynkmx>
```

在本机运行，正确完成了读订单请求

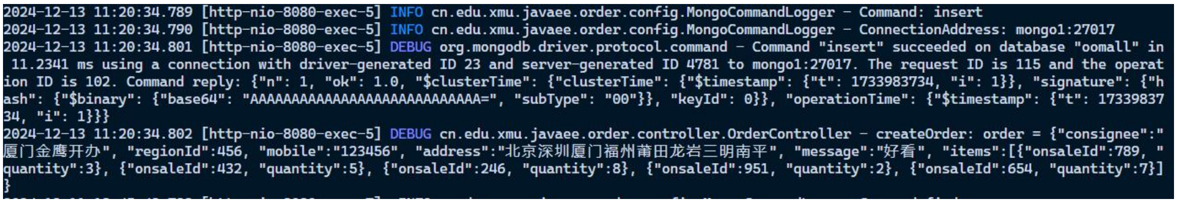
（3）日志输出

读订单日志



```
2024-12-11 13:45:43.788 [http-nio-8080-exec-7]  INFOcn.edu  .xmu.javaee.order.config.MongoCommandLogger-Command:find
2024-12-11 13:45:43.780 [http-nio-8888-exec-7]  INFOcn.edu  .xmu.javaee.order.config.MongoCommandLogger-ConnectionAddress:mongo1:27
2024-12-11 13:45:43.829 [http-nio-8080-exec-7] DEBUG org.mongodb.driver.protocol.command - Command"find"succeeded on database"oomall"in 50.19
02 ms using a connection with driver-generated ID  16 and server-generated ID 4645 to mongo1:27017. The request ID is 98 and the operation ID
 is 89. Command reply: {"cursor": {"firstBatch": [{"_ id": "65581677f4d2f1071a05d3d2", "orderSn":  "01175370378290176000", "consignee": "最日
月月金开看嘛", "regionId": 924, "address": "福林宁头州头泉湘津夷", "mobile": "009365", "message": "最电", "orderItems": [{"_ id": 11753703782
94370304,"onsaleId": 629, "quantity": 7}, {"_ id": 1175370378294370304, "onsalerd": 205, "quantity": 2}, {"_ id": 1175370378294370304, "onsal
eId": 629, "quantity": 8}, {"_ id": 1175370378294370304,"onsaleId": 714, "quantity": 4}, {"_ id": 1175370378294370304, "onsaleId": 810, "quan
tity": 3}], "_ class"  : "cn.   edu. xmu. javaee. order. dao. bo. Order"}], "id": 0, "ns": "oomall. order"}, "ok": 1.0,"$clusterTime": {"clus
terTime": {"$ timestamp": {"t": 1733895938, "i": 1}}, "signature": {"hash": {"$ binary": {"base64": "AAAAAAAAAAAAAAAAAAAAAAAAAAA=", "subType
": "00"}}, "keyId": 0}},"operationTime": {"$ timestamp": {"t": 1733895938, "i": 1}}}
2024-12-11 13:45:43.838 [http-nio-8080-exec-7] DEBUG c.e. xmu. javaee. order. controller. CustomerControler - getOrders: order = Order(id=655
81677f4d2f1071a05d3d2, customerId= null, shopId= null,  orderSn=01175370378290176000, consignee=最日月月金开看嘛, regionId=924, address=福林
宁头州头泉湘津夷, mobile=009365, message=最电, activityId= null, packageId= null, orderItems=[OrderItem  (id=1175370378294370304, onsaleId=62
9, quantity=7, price= null, discountPrice= null, point= null, name= null, actId= null, couponId= null), OrderItem(id=1175370378294370304, ons
aleId=205, quantity=2, price= null, discountPrice= null, point= null, name= null, actId= null, couponId= null), OrderItem(id=1175378378294378
304, onsaleId=629, quantity=8, price= null, discountPrice= null, point= null, name= null, actId= null, couponId= null), OrderItem(id=11753703
78294370304, onsaleId=714, quantity=4, price= null, discountPrice= null, point= null, name= null, actId= null, couponId= null), OrderItem(id=
1175370378294370304, onsaleId=810, quantity=3, price= null, discountPrice= null, point= null, name= null, actId= null, couponId= null)], crea
torId= null, creatorName= null, modifierId= null,modifierName= null, gmtCreate= null, gmtModified= null)
2024-12-11 13:45:43.830 [http-nio-8080-exec-7] DEBUG c.
```

写订单日志



```
2024-12-13 11:20:34.789 [http-nio-8080-exec-5] INFO cn.edu.xmu.javaee.order.config.MongoCommandLogger - Command: insert
2024-12-13 11:20:34.790 [http-nio-8080-exec-5] INFO cn.edu.xmu.javaee.order.config.MongoCommandLogger - ConnectionAddress: mongo1:27017
2024-12-13 11:20:34.801 [http-nio-8080-exec-5] DEBUG org.mongodb.driver.protocol.command - Command "insert" succeeded on database "oomall" in
 11.2341 ms using a connection with driver-generated ID 23 and server-generated ID 4781 to mongo1:27017. The request ID is 115 and the operat
ion ID is 102. Command reply: {"n": 1, "ok": 1.0, "$clusterTime": {"clusterTime": {"$timestamp": {"t": 1733983734, "i": 1}}, "signature": {"h
ash": {"$binary": {"base64": "AAAAAAAAAAAAAAAAAAAAAAAAAAA=", "subType": "00"}}, "keyId": 0}}, "operationTime": {"$timestamp": {"t": 17339837
34, "i": 1}}}
2024-12-13 11:20:34.802 [http-nio-8080-exec-5] DEBUG cn.edu.xmu.javaee.order.controller.OrderController - createOrder: order = {"consignee":"
厦门金鹰开办", "regionId":456, "mobile":"123456", "address":"北京深圳厦门福州莆田龙岩三明南平", "message":"好看", "items":[{"onsaleId":789, "
quantity":3}, {"onsaleId":432, "quantity":5}, {"onsaleId":246, "quantity":8}, {"onsaleId":951, "quantity":2}, {"onsaleId":654, "quantity":7}]
}
```

3　结果分析与讨论

3.1　readOrder-1000-60-126（60sMongo读订单达到的峰值）

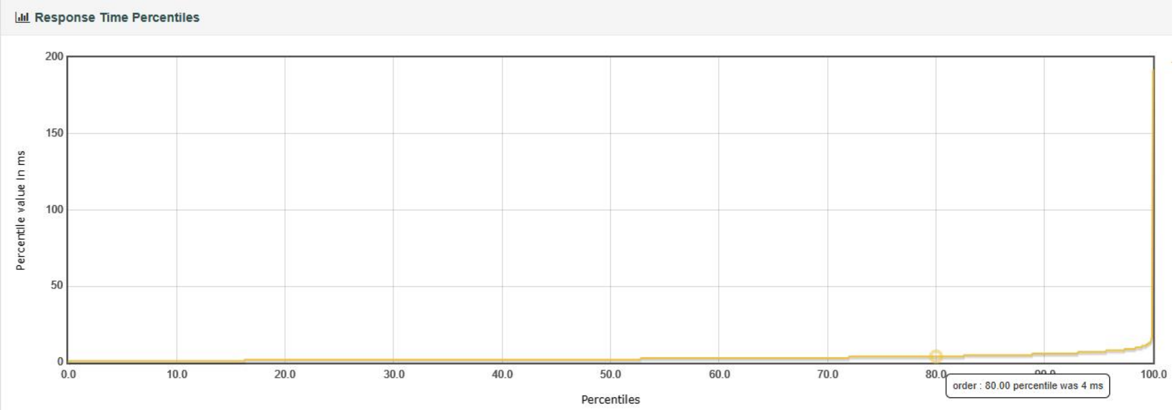


图　1readOrder-1000-60-126 Response Time Percentiles

80%的请求在 4ms 内响应

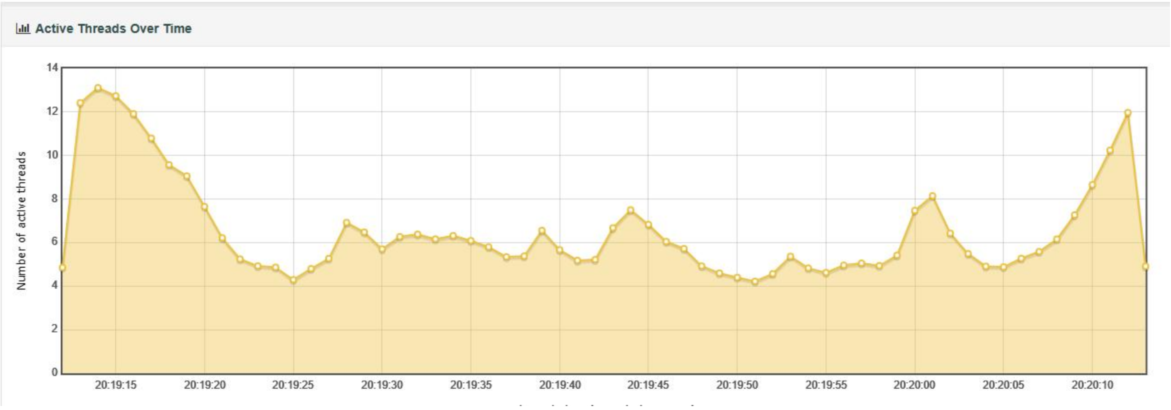


图 2readOrder-1000-60-126 Active Threads Over Time

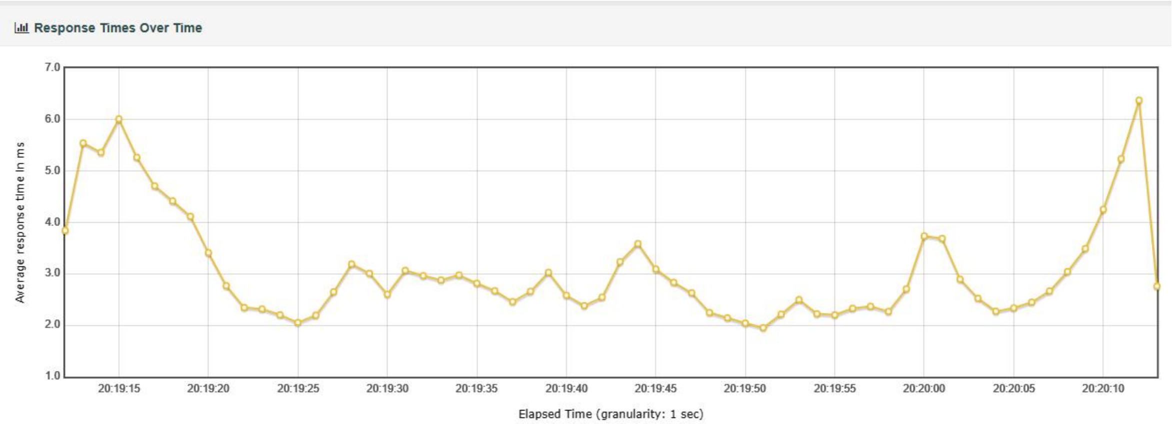
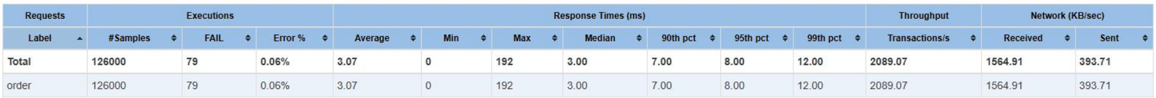


图 3readOrder-1000-60-126 Response Times Over Time



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	**126000**	**79**	**0.06%**	**3.07**	**0**	**192**	**3.00**	**7.00**	**8.00**	**12.00**	**2089.07**	**1564.91**	**393.71**
order	126000	79	0.06%	3.07	0	192	3.00	7.00	8.00	12.00	2089.07	1564.91	393.71

图 4readOrder-1000-60-126 Statistics

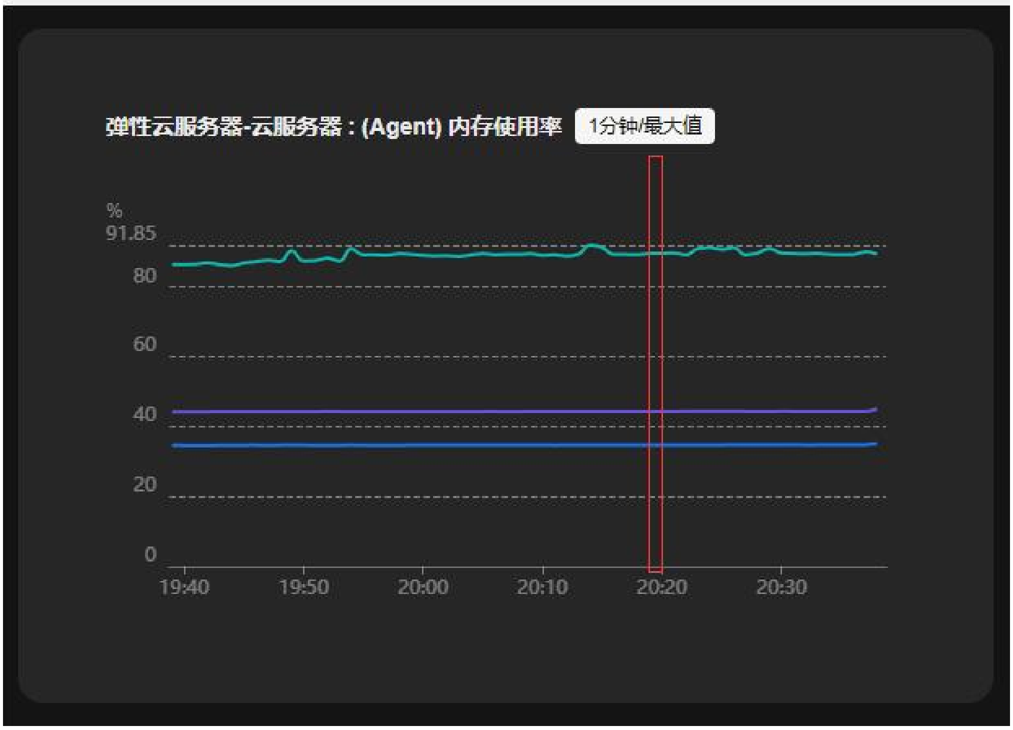


图 5readOrder-1000-60-126 内存使用率（绿 mongo2,紫 mongo1,蓝 node1）

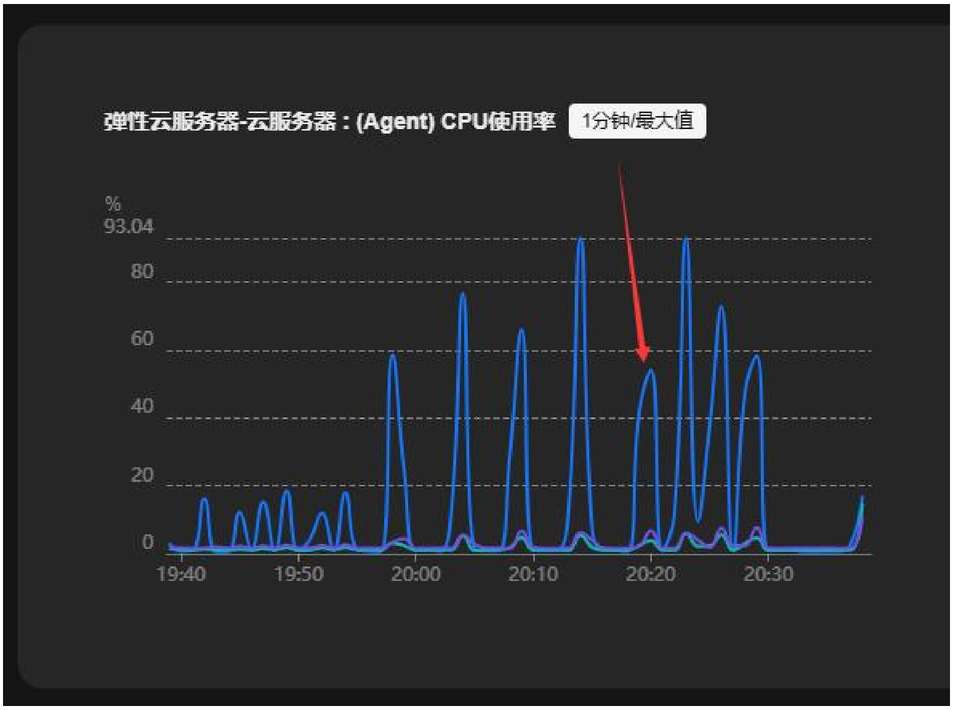


图 6readOrder-1000-60-126 CPU 使用率（绿 mongo2,紫 mongo1,蓝 node1）

3.2 readOrder-1000-60-127（60sMongo读订单阻塞）

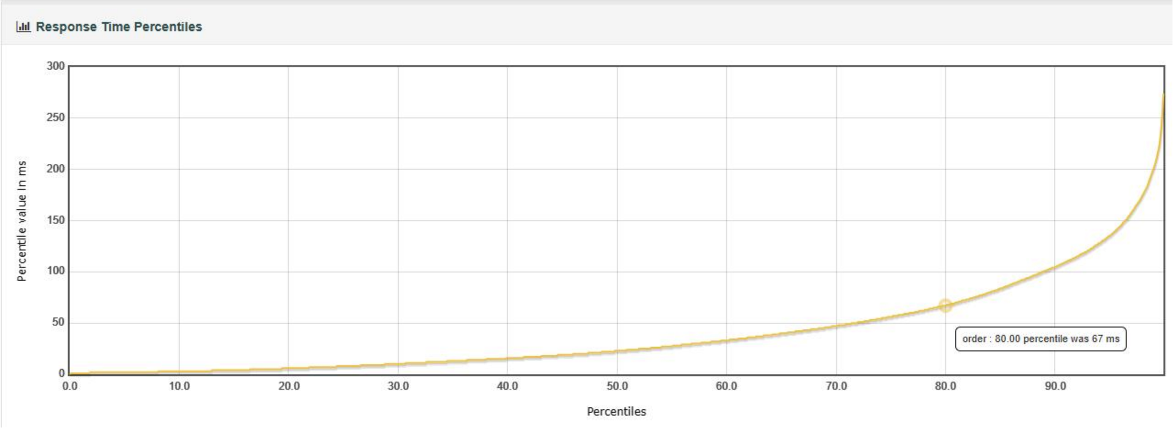


图 7readOrder-1000-60-127 Response Time Percentiles

80%的请求在 67ms 内响应

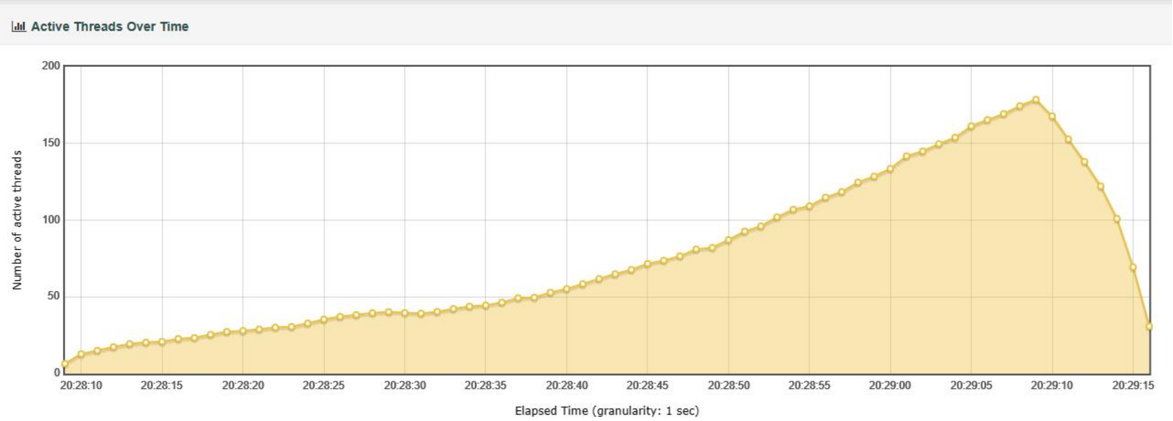


图 8readOrder-1000-60-127 Active Threads Over Time

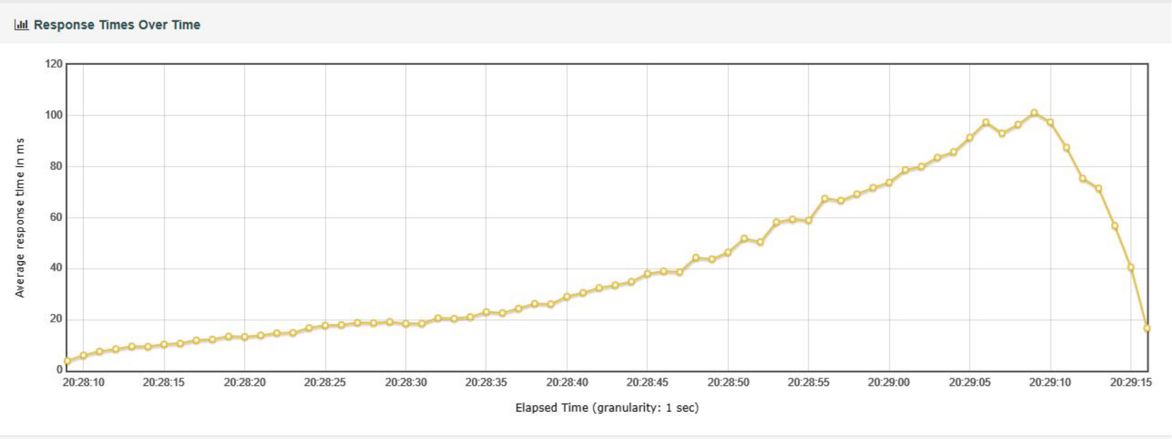
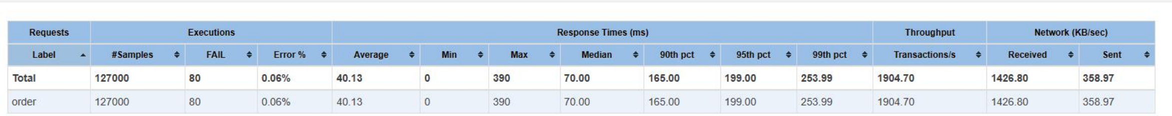


图 9readOrder-1000-60-127 Response Times Over Time



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	**127000**	**80**	**0.06%**	**40.13**	**0**	**390**	**70.00**	**165.00**	**199.00**	**253.99**	**1904.70**	**1426.80**	**358.97**
order	127000	80	0.06%	40.13	0	390	70.00	165.00	199.00	253.99	1904.70	1426.80	358.97

图 10readOrder-1000-60-127 Statistics

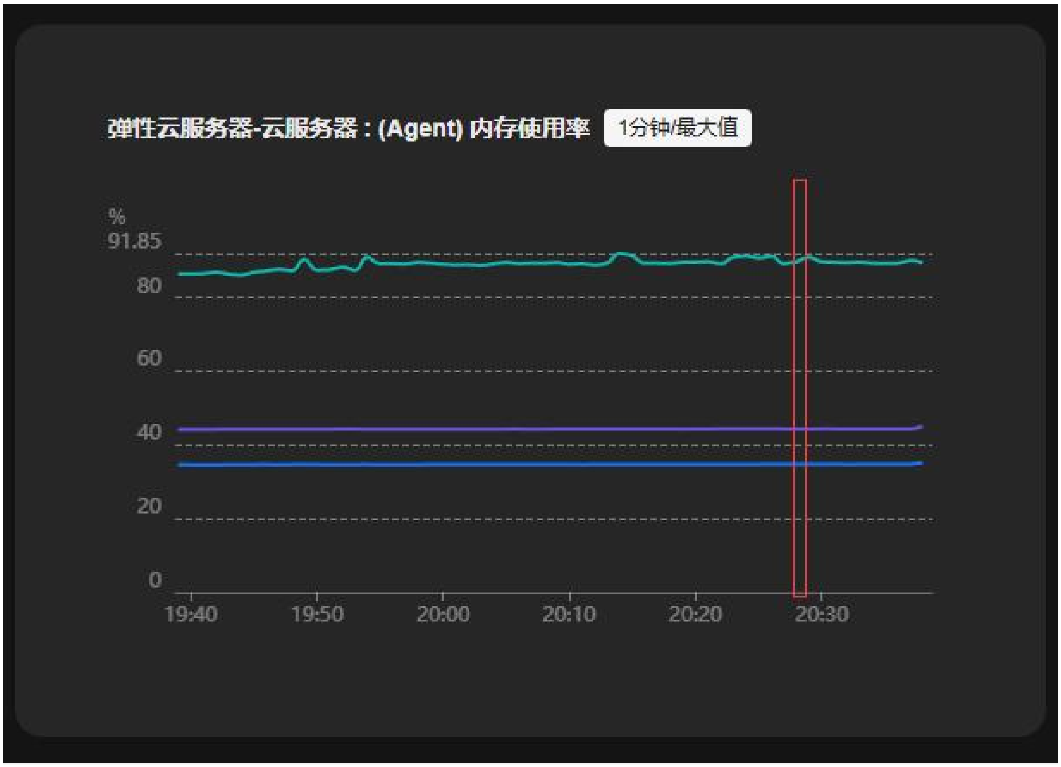


图 11readOrder-1000-60-127 内存使用率（绿 mongo2,紫 mongo1,蓝 node1）

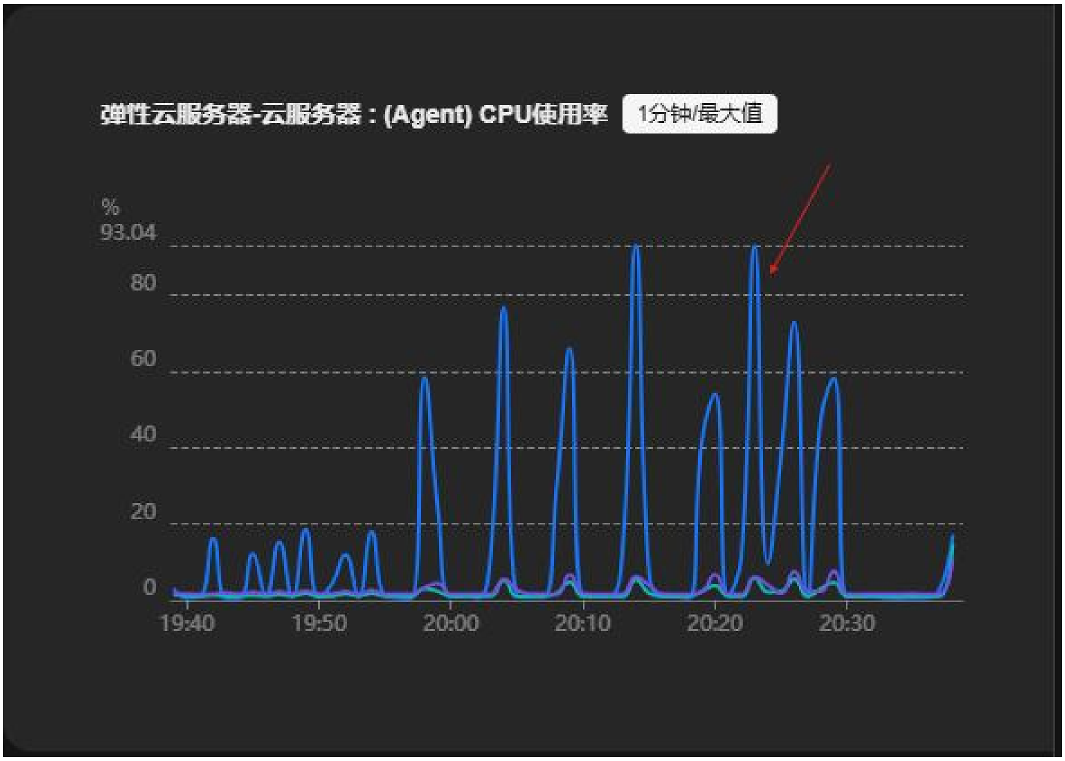


图 12readOrder-1000-60-127 CPU 使用率（绿 mongo2,紫 mongo1,蓝 node1）

可以看到，导致查询性能瓶颈为 node1

3.3 writeOrder-1000-60-137（60sMongo写订单达到的峰值）

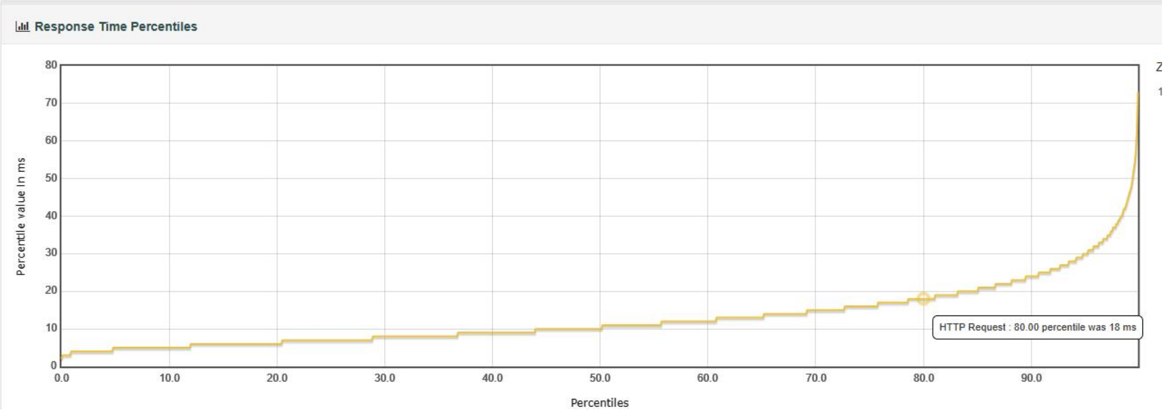


图 13writeOrder-1000-60-137 Response Time Percentiles

80%的请求在 18ms 内响应

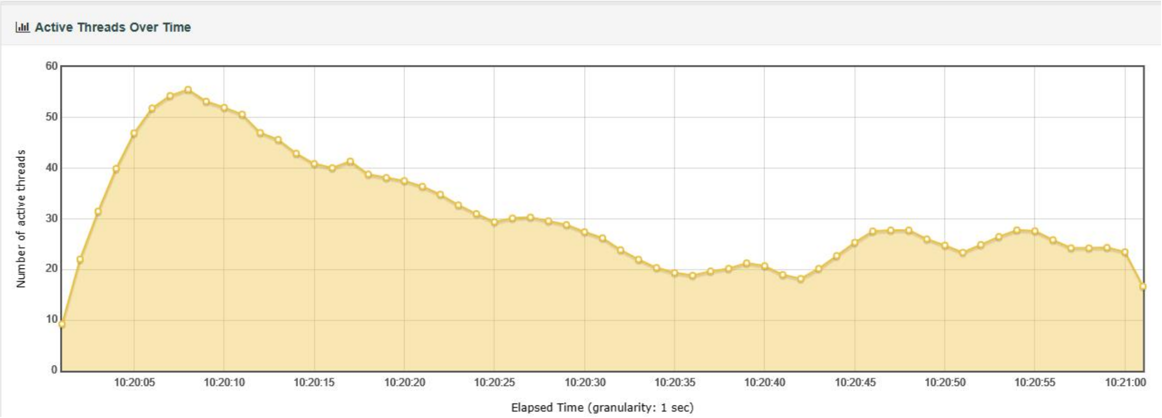


图 14writeOrder-1000-60-137 Active Threads Over Time

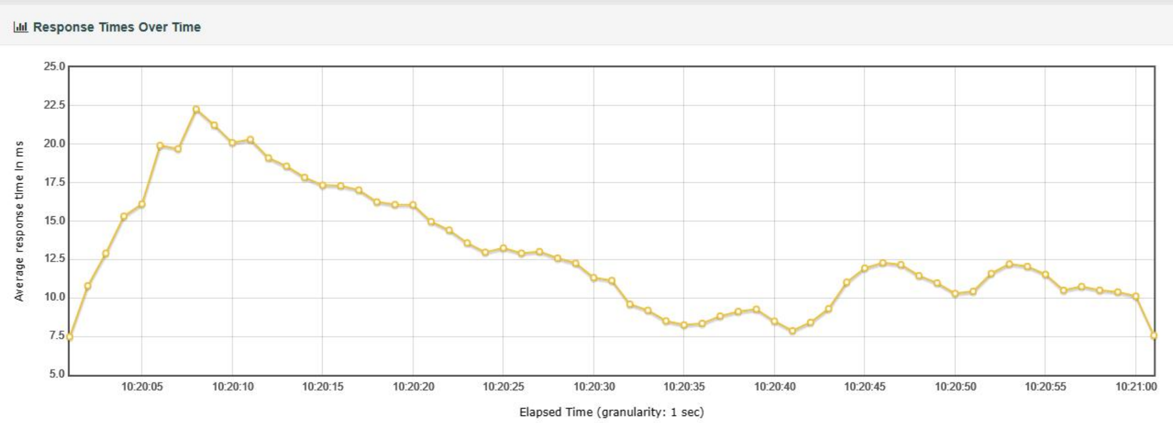
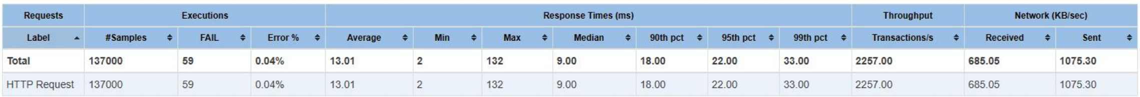


图 15writeOrder-1000-60-137 Response Times Over Time



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	**137000**	**59**	**0.04%**	**13.01**	**2**	**132**	**9.00**	**18.00**	**22.00**	**33.00**	**2257.00**	**685.05**	**1075.30**
HTTP Request	137000	59	0.04%	13.01	2	132	9.00	18.00	22.00	33.00	2257.00	685.05	1075.30

图 16writeOrder-1000-60-137 Statistics

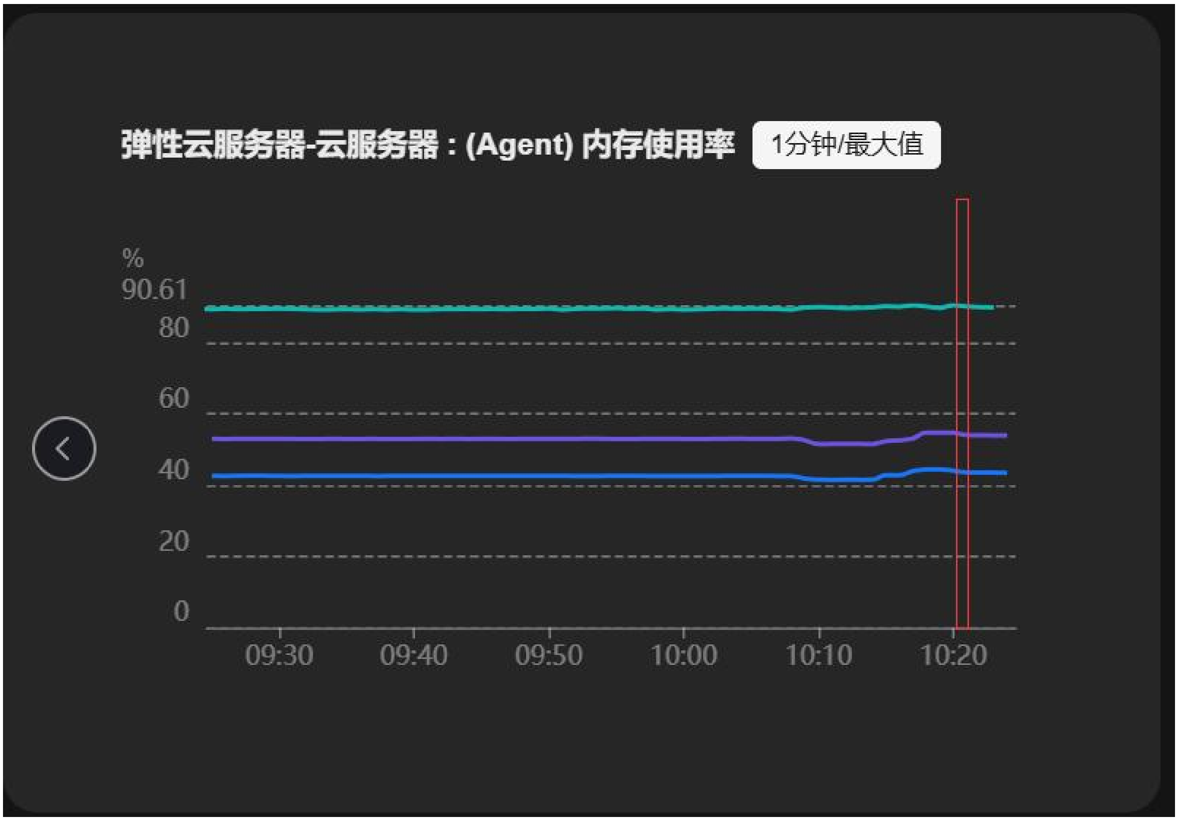


图 17writeOrder-1000-60-137 内存使用率（绿 mongo2,紫 mongo1,蓝 node1）

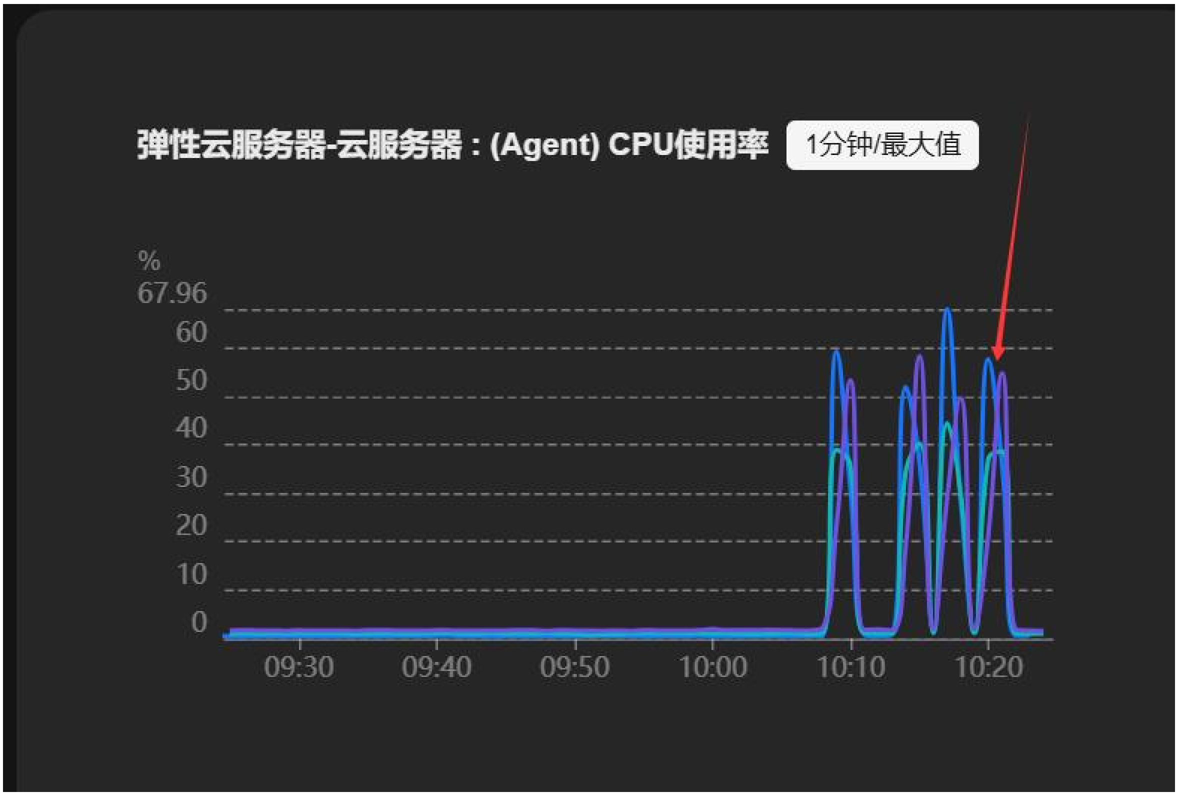


图 18writeOrder-1000-60-137 CPU 使用率（绿 mongo2,紫 mongo1,蓝 node1）

3.4 writeOrder-1000-60-138（60sMongo写订单阻塞）

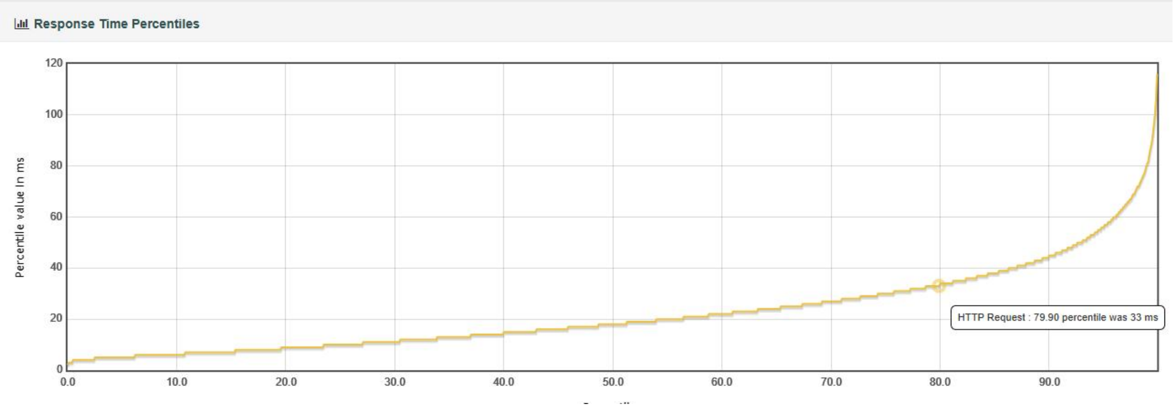


图 19writeOrder-1000-60-138 Response Time Percentiles

80%的请求在 33ms 内响应

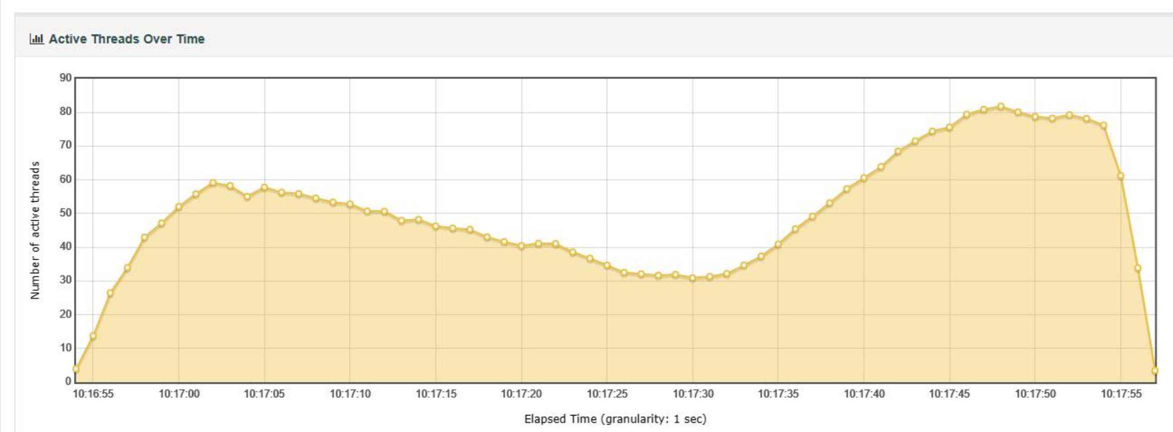


图 20writeOrder-1000-60-138 Active Threads Over Time

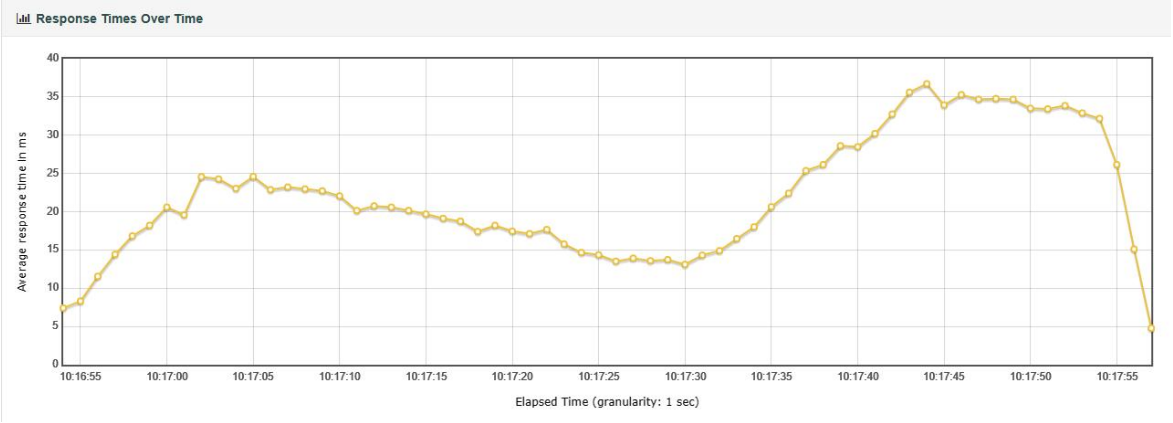
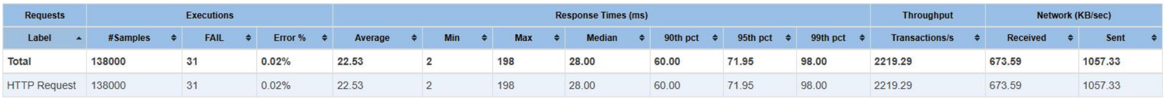


图 21writeOrder-1000-60-138 Response Times Over Time



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	**138000**	**31**	**0.02%**	**22.53**	**2**	**198**	**28.00**	**60.00**	**71.95**	**98.00**	**2219.29**	**673.59**	**1057.33**
HTTP Request	138000	31	0.02%	22.53	2	198	28.00	60.00	71.95	98.00	2219.29	673.59	1057.33

图 22writeOrder-1000-60-138 Statistics

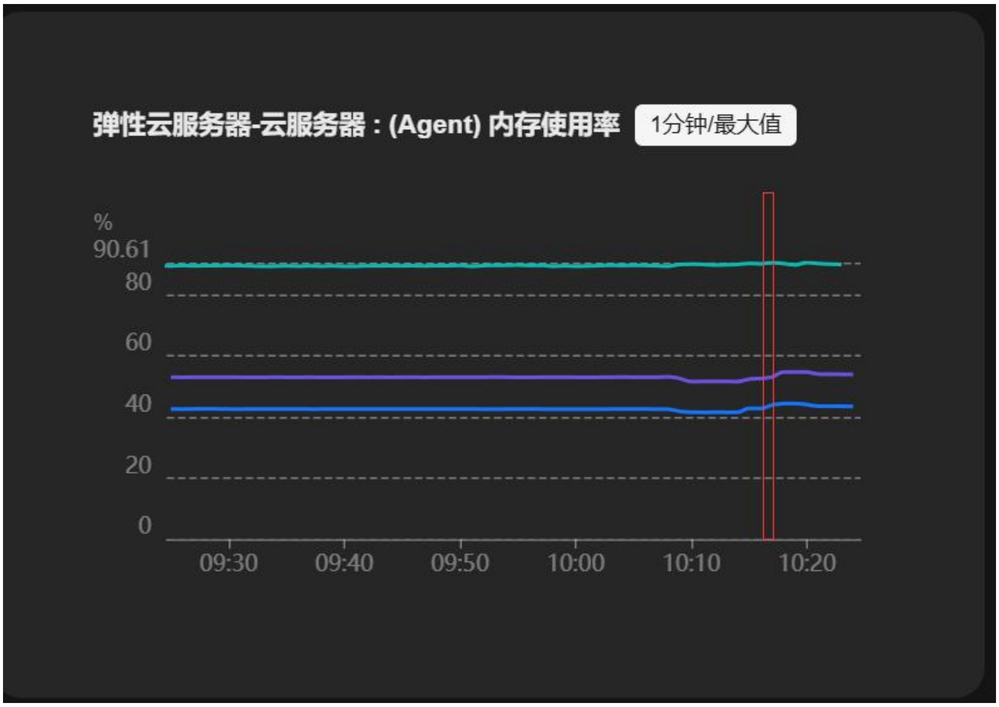


图 23writeOrder-1000-60-138 内存使用率（绿 mongo2,紫 mongo1,蓝 node1）

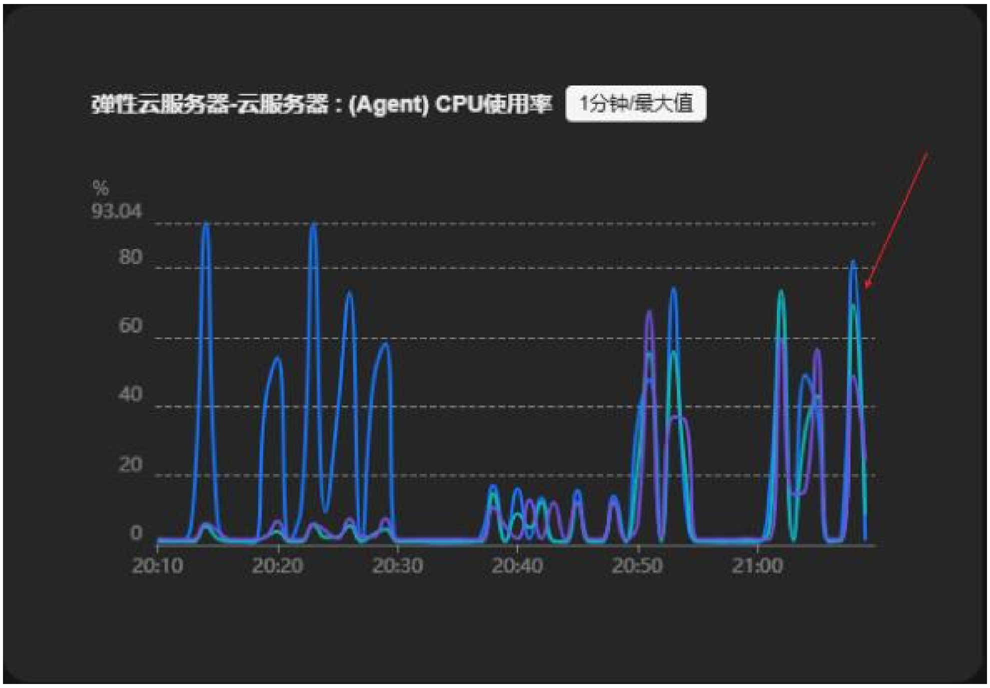


图 24writeOrder-1000-60-138 CPU 使用率（绿 mongo2,紫 mongo1,蓝 node1）

可以看到，导致插入性能瓶颈为 node1

3.5 Mybatis-read-2000-10-7（10sMybatis读达到峰值）

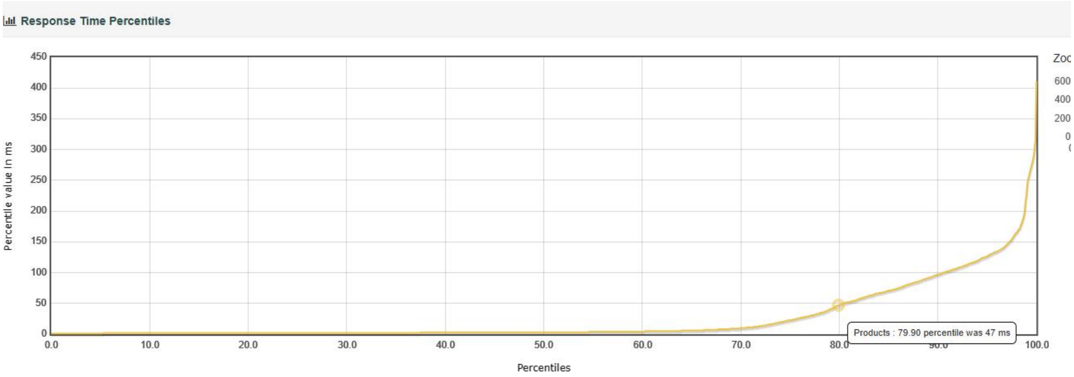


图 19Mybatis-read-2000-10-7 Response Time Percentiles

80%的请求在 47ms 内响应

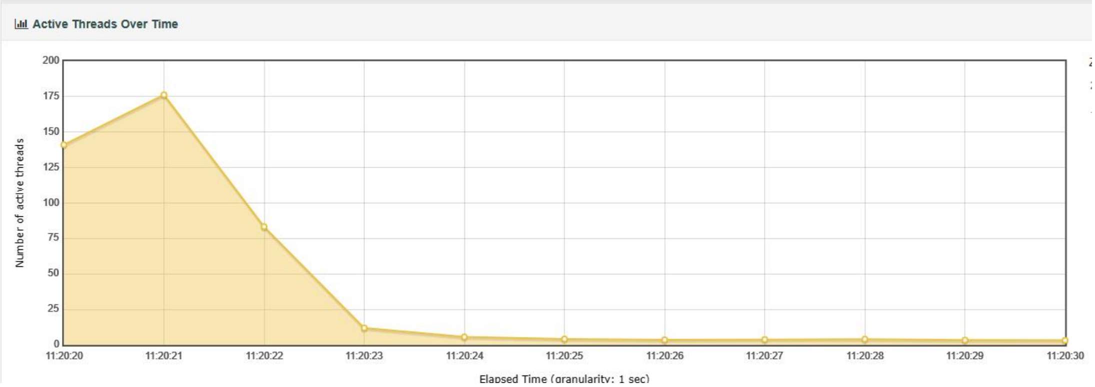


图 20Mybatis-read-2000-10-7 Active Threads Over Time

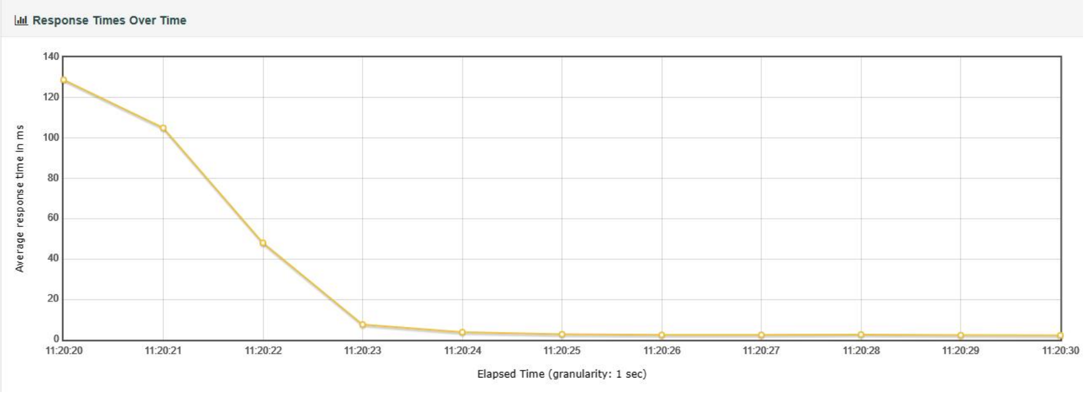
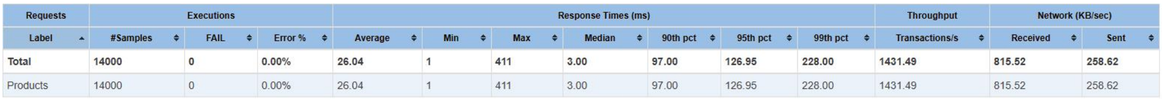


图 21Mybatis-read-2000-10-7 Response Times Over Time



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**	**815.52**	**258.62**
Products	14000	0	0.00%	26.04	1	411	3.00	97.00	126.95	228.00	1431.49	815.52	258.62

图 22Mybatis-read-2000-10-7 Statistics

3.6 Mybatis-read-2000-10-8（10sMybatis读阻塞）

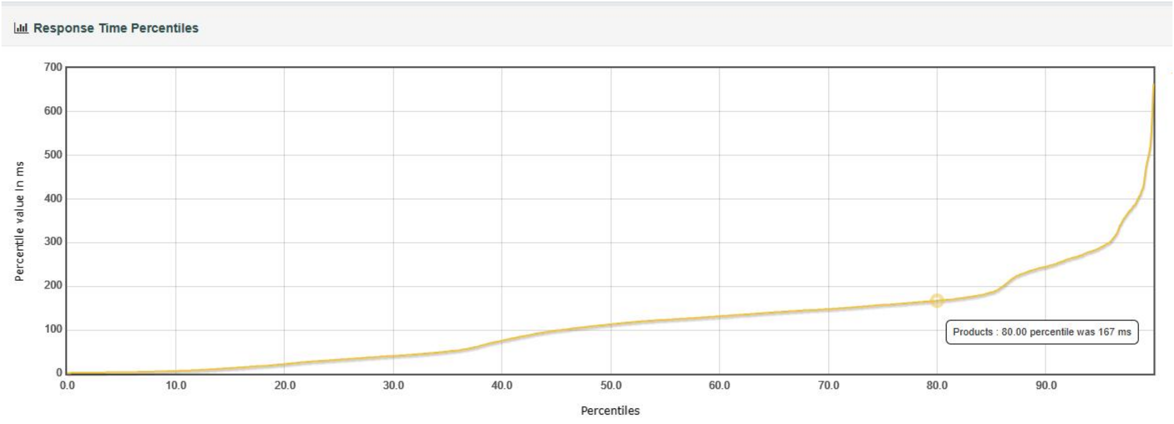


图 19Mybatis-read-2000-10-8 Response Time Percentiles

80%的请求在 167ms 内响应

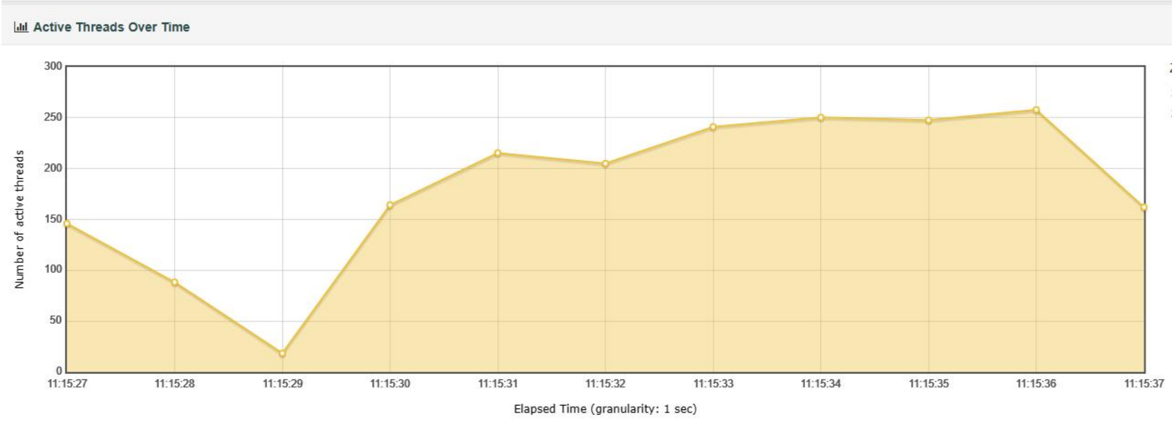


图 20Mybatis-read-2000-10-8 Active Threads Over Time

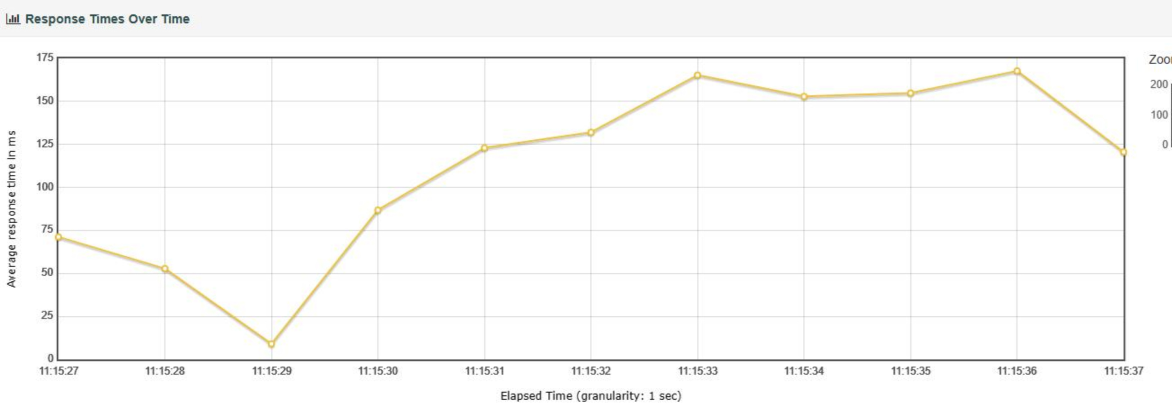
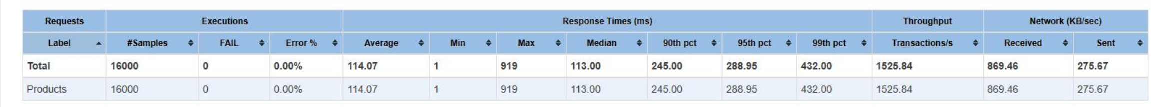


图 21Mybatis-read-2000-10-8 Response Times Over Time



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

图 22Mybatis-read-2000-10-8 Statistics

3.7 Mybatis-write-2000-10-4（10sMybatis写达到峰值）

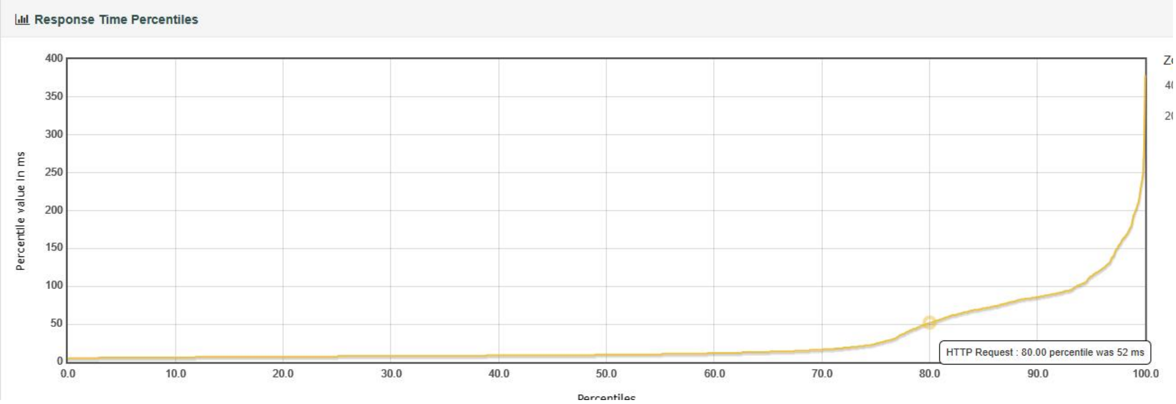


图 19Mybatis-read-2000-10-4 Response Time Percentiles

80%的请求在 52ms 内响应

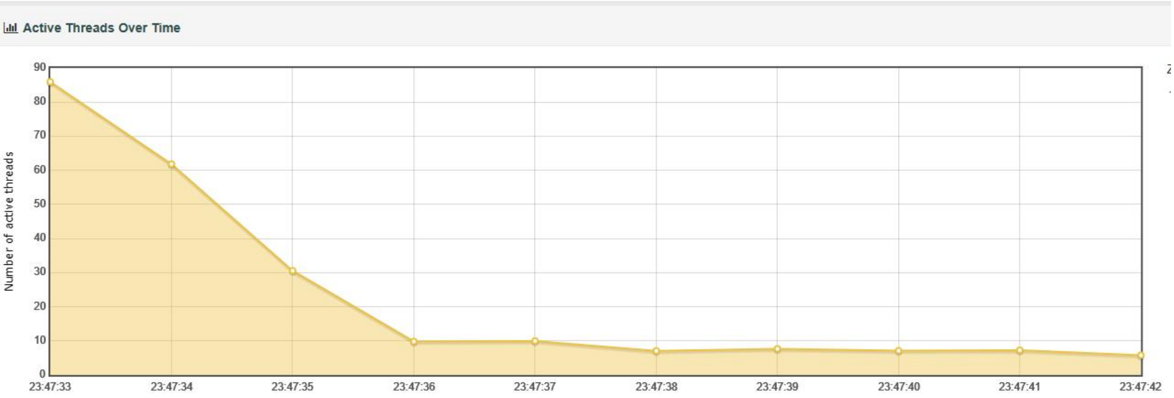


图 20Mybatis-read-2000-10-4 Active Threads Over Time

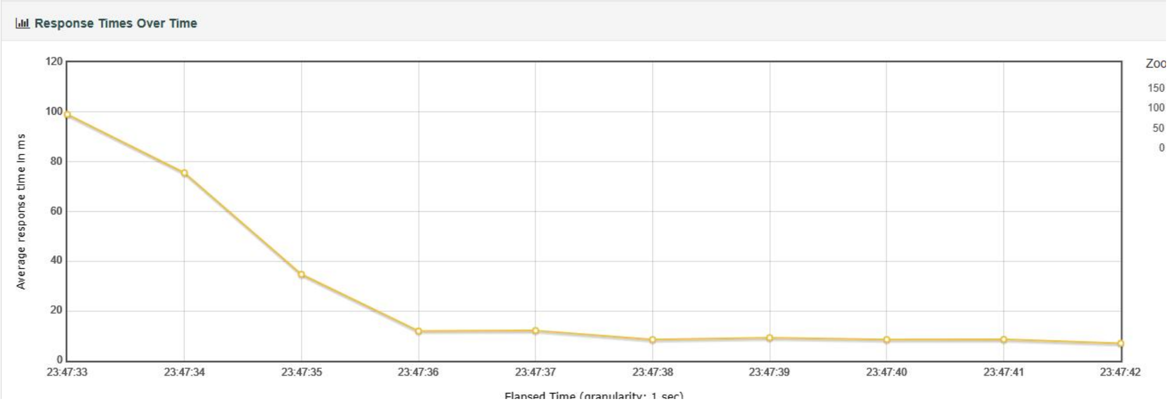
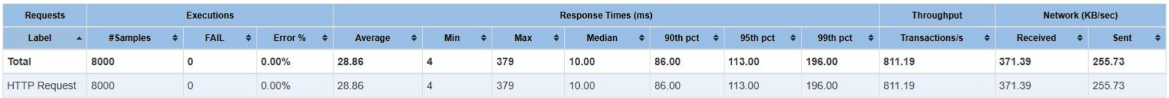


图 21Mybatis-read-2000-10-4 Response Times Over Time



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

图 22Mybatis-read-2000-10-4 Statistics

3.8 Mybatis-read-2000-10-5（10sMybatis写阻塞）

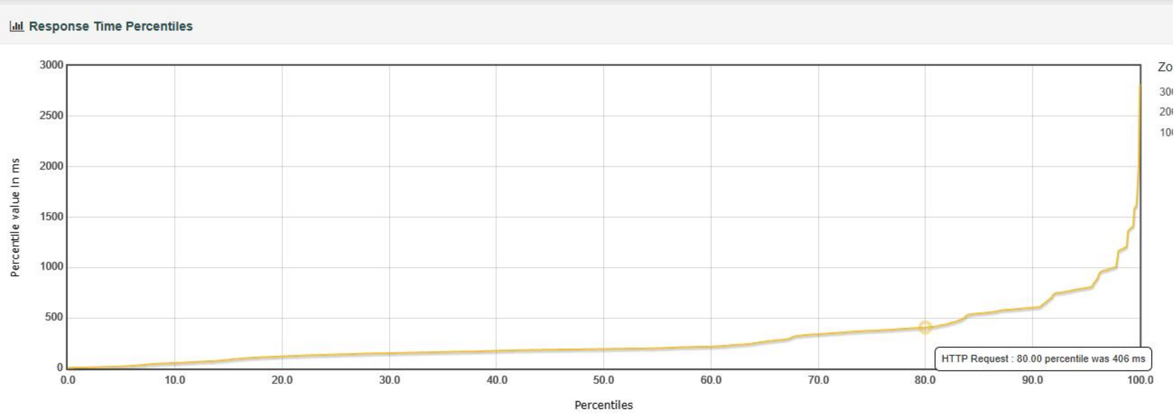


图 19Mybatis-read-2000-10-5 Response Time Percentiles

80%的请求在 406ms 内响应

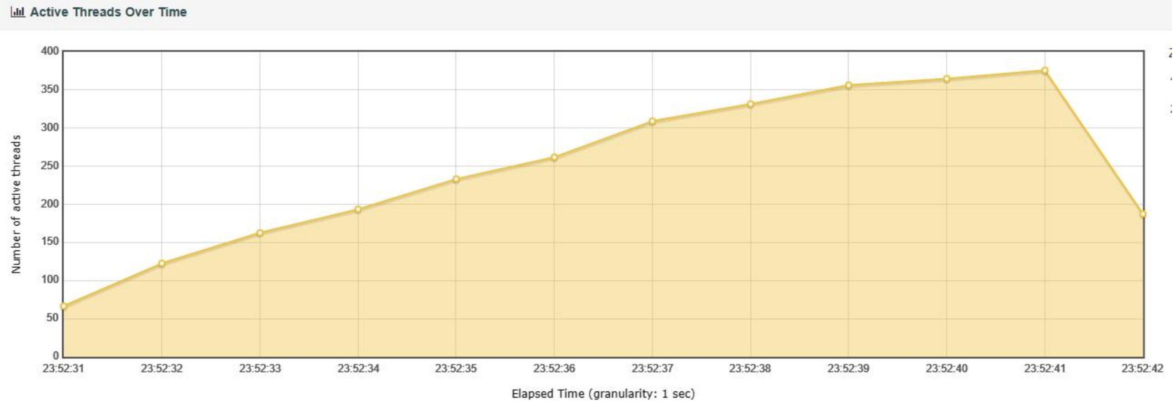


图 20Mybatis-read-2000-10-5 Active Threads Over Time

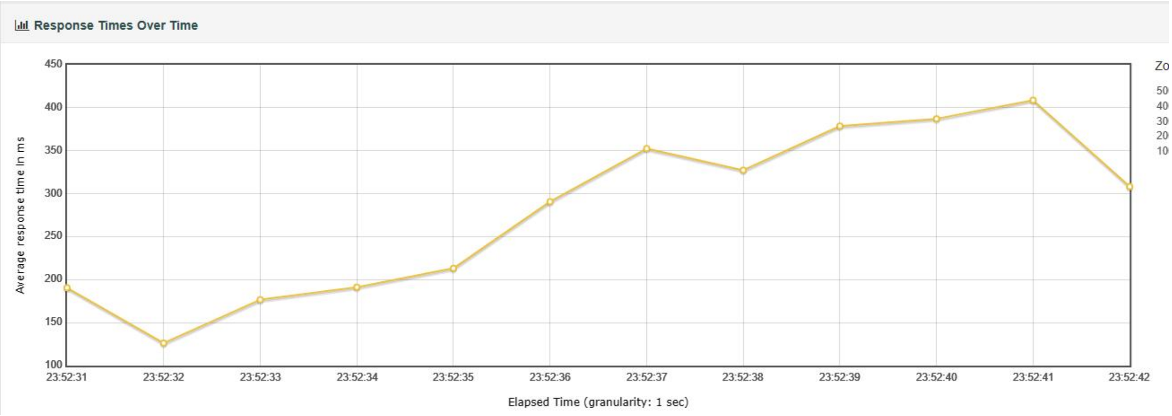
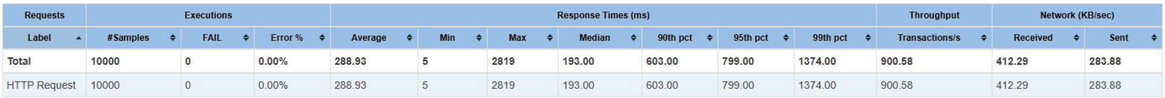


图 21Mybatis-read-2000-10-5 Response Times Over Time

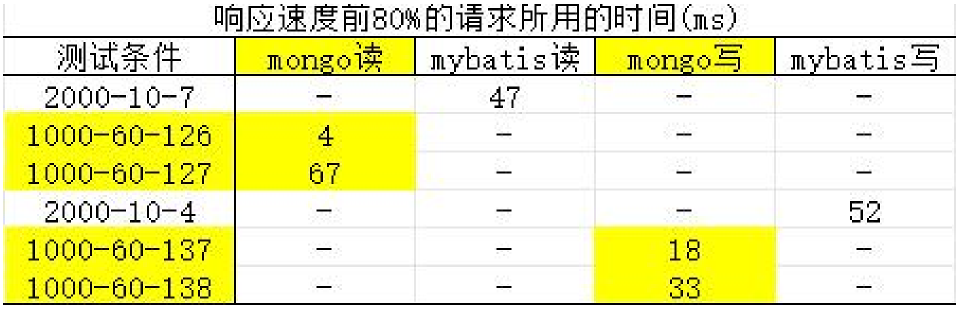


Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label ▲	#Samples ◆	FAIL ◆	Error % ◆	Average ◆	Min ◆	Max ◆	Median ◆	90th pct ◆	95th pct ◆	99th pct ◆	Transactions/s ◆	Received ◆	Sent ◆
Total	**10000**	**0**	**0.00%**	**288.93**	**5**	**2819**	**193.00**	**603.00**	**799.00**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

图 22Mybatis-read-2000-10-5 Statistics

4　总　结

4.1　性能表现：



响应速度前80%的请求所用的时间(ms)

测试条件	mongo读	mybatis读	mongo写	mybatis写
2000-10-7	–	47	–	–
1000-60-126	4	–	–	–
1000-60-127	67	–	–	–
2000-10-4	–	–	–	52
1000-60-137	–	–	18	–
1000-60-138	–	–	33	–

经过换算，

Mongo 方案读的峰值为 2100 请求/s，但是实际的性能瓶颈是 node1CPU 无法处理更多请求

Mongo 方案写的峰值为 2300 请求/s，但是实际的性能瓶颈是 node1CPU 无法处理更多请求

Mybatis 读的峰值为 1400 请求/s

Mybatis 写的峰值为 800 请求/s

实验结果显示，尽管由于 node1 性能不足导致 mongo 方案的读写效率受到限制，Mongo 数据库还是能够比 Mybatis 更加快速响应请求，这表明 Mongo 数据库在处理大规模数据读写时具有远高于 mysql 的效率。在读写对比方面，我们发现写的效率明显地高于读的效率，这在参考文献中也能找到依据。

4.2　服务器负载：

通过对服务器的负载情况进行监控，我们发现 Mongo 数据库在处理高并发读写请求时，尽管 node1 已经出现性能不足的情况，mongo 服务器的 CPU 和内存使用率仍保持在合理范围内，没有出现资源瓶颈。这进一步证实了 Mongo 数据库在分布式数据库系统中的优秀性能，尤其是在集群环境下能够有效地分散负载，保持系统的稳定性和响应速度。

参考文献:

[1] mongodb 读写性能分析 https://blog.csdn.net/zero1036/article/details/70153765?ops_request_misc=%257B%2522request%255Fid%2522%253A%2522c420e79539cb2c26596f004040e92699%2522%252C%2522scm%2522%253A%252220140713.130102334..%2522%257D&request_id=c420e79539cb2c26596f004040e92699&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~sobaiduend~defa

ult-1-70153765-null-null.142^v100^pc_search_result_base3&utm_term=mongo%20%E8%AF%BB%E5%86%99%E6%95%88%E7%8E%87&spm=1018.2226.3001.4187

[2] MongoDB 读写效率问题
https://blog.csdn.net/u011353623/article/details/140852279?ops_request_misc=&request_id=&biz_id=102&utm_term=mongo%20%E8%AF%BB%E5%86%99%E6%95%88%E7%8E%87&utm_medium=distribute.pc_search_result.none-task-blog-2~all~sobaiduweb~default-0-140852279.142^v100^pc_search_result_base3&spm=1018.2226.3001.4187

[3] MongoDB 的认识，优缺点和使用场景，原理
https://blog.csdn.net/cctvcqupt/article/details/82346571?ops_request_misc=&request_id=&biz_id=102&utm_term=mongo%20%E8%AF%BB%E5%86%99%E6%95%88%E7%8E%87&utm_medium=distribute.pc_search_result.none-task-blog-2~all~sobaiduweb~default-4-82346571.142^v100^pc_search_result_base3&spm=1018.2226.3001.4187