Redis 缓存的效率

摘　要:　本实验旨在探究 Redis 缓存在提升系统性能方面的效果。通过在 Ubuntu 18.04 服务器上搭建实验环境，使用 JDK 17、Maven、git 以及 Redis 6.2.4 等工具，我们设计并实现了两个 API：查询商品完整信息。实验分为两套方案，一套使用 Redis 缓存，另一套不使用缓存，通过 JMeter 进行性能测试，并监控主机性能。实验结果表明，使用 Redis 缓存可以显著提高系统的响应速度和处理能力。
关键词:　Redis 缓存，系统性能，API，JMeter 测试，主机监控

Based on MySQLBased on MyBatis Association Implementation Plan

Abstract:　This experiment aims to investigate the effect of Redis caching on system performance enhancement. By setting up an experimental environment on Ubuntu 18.04 servers with JDK 17, Maven, git, and Redis 6.2.4, we designed and implemented two APIs: querying complete product information. The experiment was divided into two schemes, one using Redis caching and the other without caching, with performance testing conducted using JMeter and host performance monitoring. The results indicate that using Redis caching significantly improves system response speed and processing capability.
Key words:　Redis Caching, System Performance, API, JMeter Testing, Host Monitoring

在现代互联网应用中，随着用户数量的增加和数据量的增长，系统性能成为影响用户体验的关键因素。缓存技术，尤其是 Redis 缓存，因其高效的数据处理能力而被广泛应用于提升系统性能。本实验旨在通过对比使用和不使用 Redis 缓存的系统性能，来评估 Redis 缓存在实际应用中的效果和效率。

主要问题：

1.使用和不使用 Redis 缓存的 JMeter 测试结果

2.使用和不使用 Redis 缓存的 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 10min，通过华为云的云监控服务，比较相同线程与循环状态下使用和不使用 Redis 缓存的方案的 MySQL 与 node1 服务器的负载情况

2.2　方案设计

（1）Redis/非 Redis 方案

Github仓库链接：https://github.com/X-Believer/JavaEE-2-4/tree/master/Exp6

```java
@GetMapping("{id}")
public ReturnObject getProductById(@PathVariable("id") Long id, @RequestParam(required = false,
    logger.debug("getProductById: id = {}", id);
    ReturnObject retObj = null;
    Product product = null;
    if("redis".equals(type))
    {

        String key = String.format(KEY, id);
        product = (Product) redisUtil.get(key); // 使用 RedisUtil 的 get 方法

        if (Objects.isNull(product)) {
            logger.info("Cache MISS for product ID: {}", id);
            product = productService.retrieveProductByID(id, all: true);
            if (!Objects.isNull(product))
                redisUtil.set(key, product, timeout: 3600);
        }
        else {
            logger.info("Cache HIT for product ID: {}", id);
        }
    }
    else {
        product = productService.retrieveProductByID(id, all: true);
    }
```

选择 redis 方案时，先查缓存是否命中，命中直接返回，否则使用之前的 MyBatis 方案查询，查询后写入缓存。选择非 redis 方案时，直接使用之前的 MyBatis 方案查询。

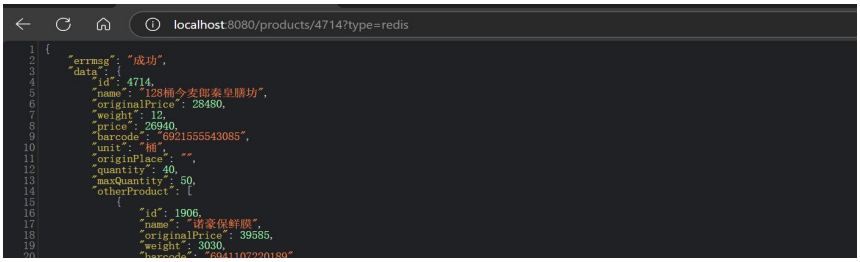
（2）运行效果

缓存命中，不再使用数据库查询

```
2024-11-26 16:02:30.792 [http-nio-8080-exec-1] DEBUG c.e.x.j.p.controller.ProductController - getProductById: id = 5440
2024-11-26 16:02:31.776 [http-nio-8080-exec-1] INFO  c.e.x.j.p.controller.ProductController - Cache HIT for product ID: 5440
```

缓存不命中，使用数据库查询后写入缓存

```
2024-11-24 12:54:02.366 [http-nio-8080-exec-281] INFO  c.e.x.j.p.controller.ProductController - Cache HIT for product ID: 4048
2024-11-24 12:54:02.366 [http-nio-8080-exec-281] DEBUG c.e.x.j.p.controller.ProductController - getProductById: id = 1903
2024-11-24 12:54:02.369 [http-nio-8080-exec-111] INFO  c.e.x.j.p.controller.ProductController - Cache MISS for product ID: 4100
2024-11-24 12:54:02.369 [http-nio-8080-exec-241] INFO  c.e.x.j.p.controller.ProductController - Cache HIT for product ID: 2434
2024-11-24 12:54:02.370 [http-nio-8080-exec-241] DEBUG c.e.x.j.p.controller.ProductController - getProductById: id = 5203
2024-11-24 12:54:02.370 [http-nio-8080-exec-111] DEBUG c.e.x.javaee.productdemoaop.service.ProductService - findProductById: id = 4100,
 all = true
2024-11-24 12:54:02.370 [http-nio-8080-exec-111] DEBUG c.e.x.j.p.m.g.ProductPoMapper.selectByPrimaryKey - ==>  Preparing: select `id`,
`shop_id`, `goods_id`, `category_id`, `template_id`, `sku_sn`, `name`, `original_price`, `weight`, `barcode`, `unit`, `origin_place`, `
creator_id`, `creator_name`, `modifier_id`, `modifier_name`, `gmt_create`, `gmt_modified`, `status`, `commission_ratio`, `shop_logistic
_id`, `free_threshold` from goods_product where `id` = ?
2024-11-24 12:54:02.370 [http-nio-8080-exec-111] DEBUG c.e.x.j.p.m.g.ProductPoMapper.selectByPrimaryKey - ==> Parameters: 4100(Long)
```



3 结果分析与讨论

3.1 1000-60-13（无redis最大不阻塞线程数）

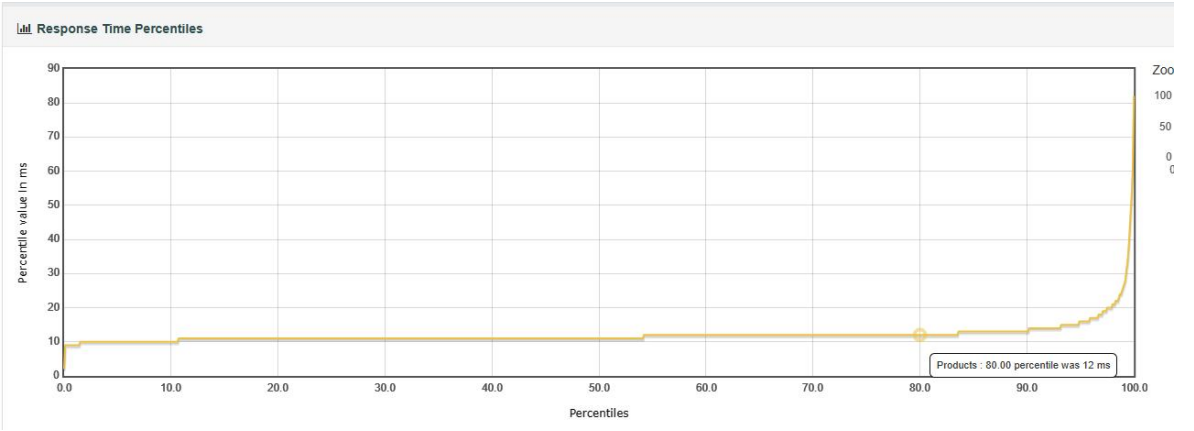


图 1noredis-1000-60-13 Response Time Percentiles

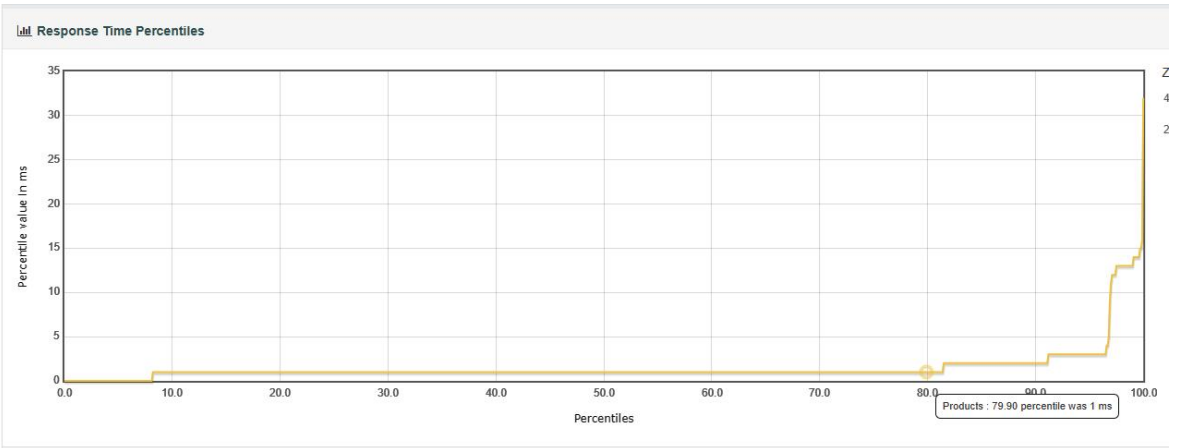


图 2redis-1000-60-13 Response Time Percentiles

noredis80%的请求在 12ms 内响应，redis 在 1ms 内响应

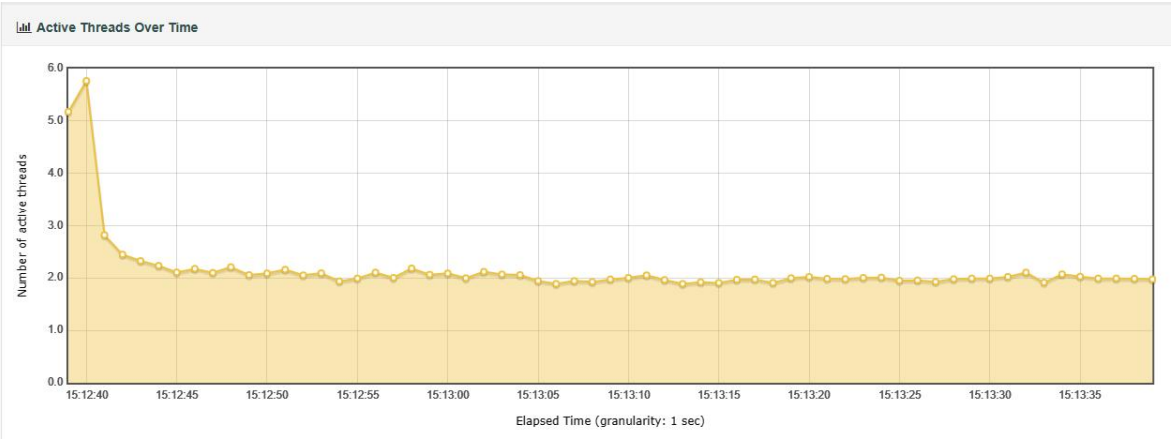


图 3noredis-1000-60-13 Active Threads Over Time

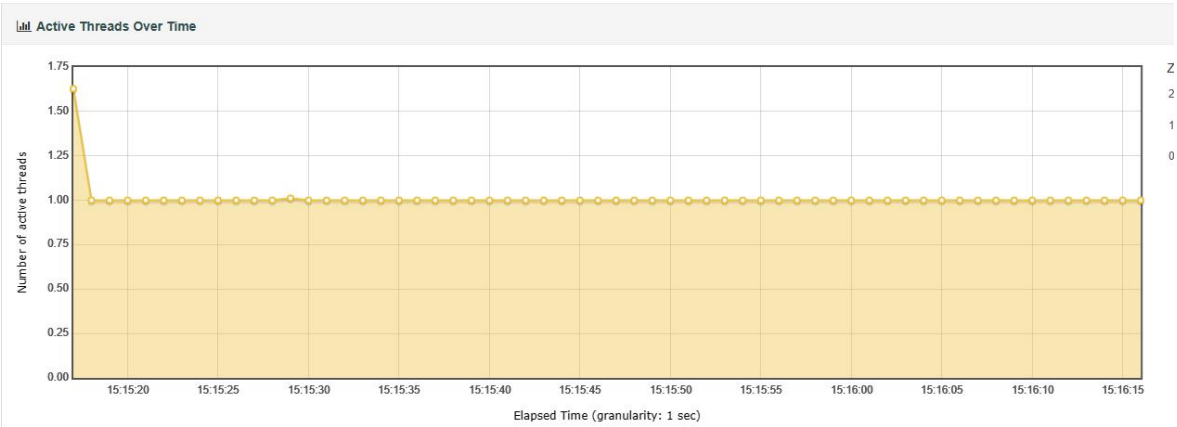


图 4redis-1000-60-13 Active Threads Over Time

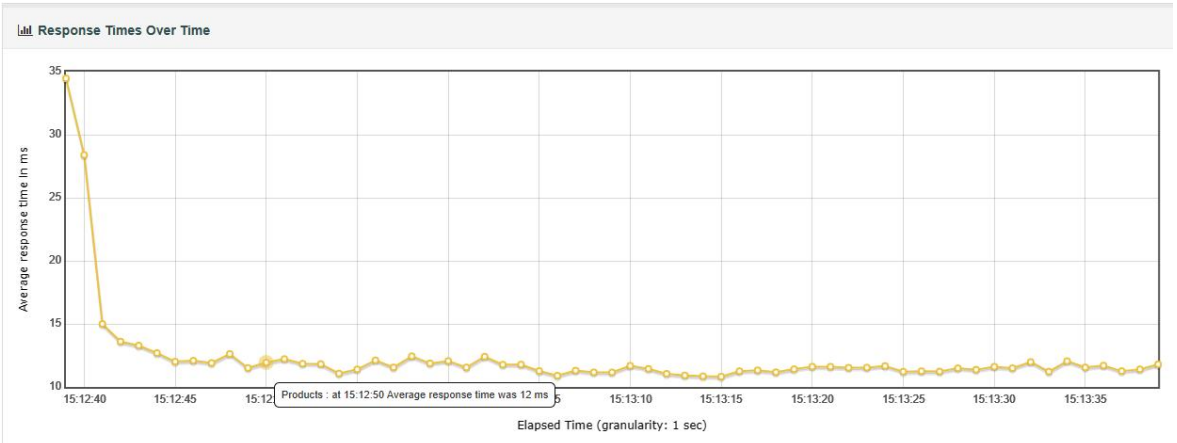


图 5noredis-1000-60-13 Response Times Over Time

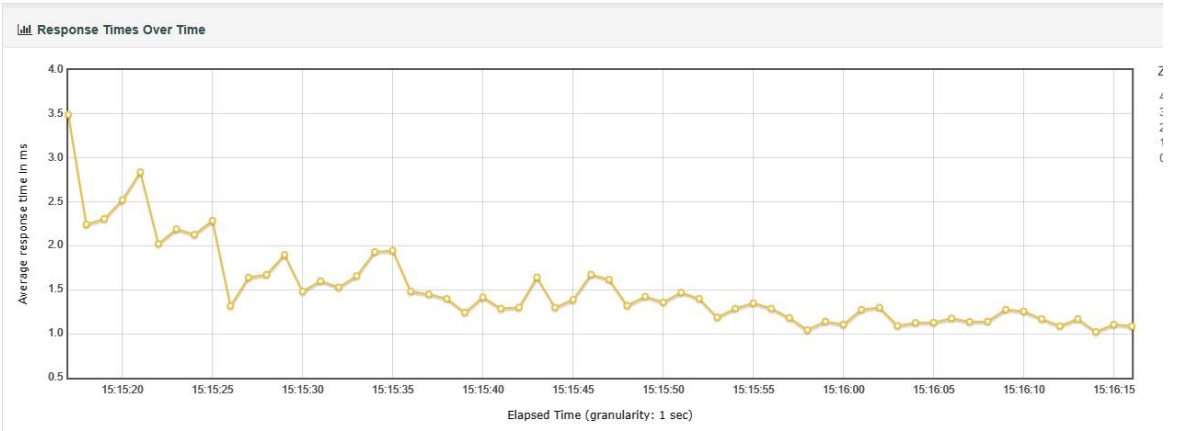


图 6redis-1000-60-13 Response Times Over Time

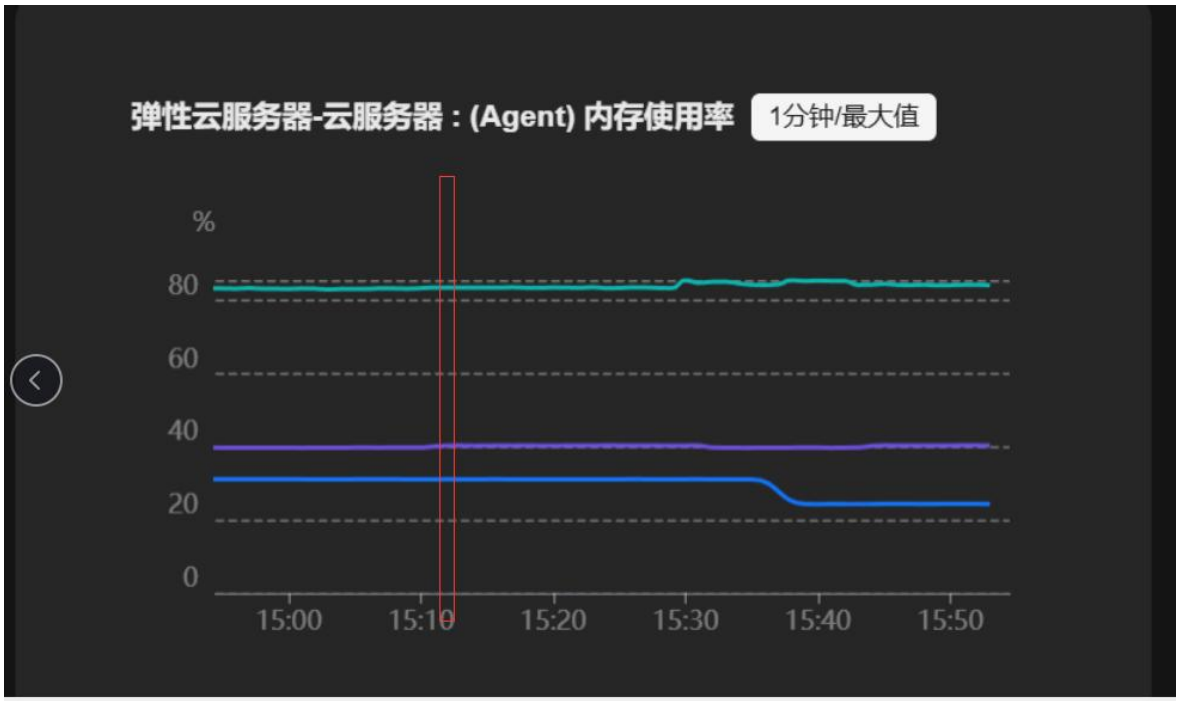


图 9noredis-1000-60-13 内存使用率

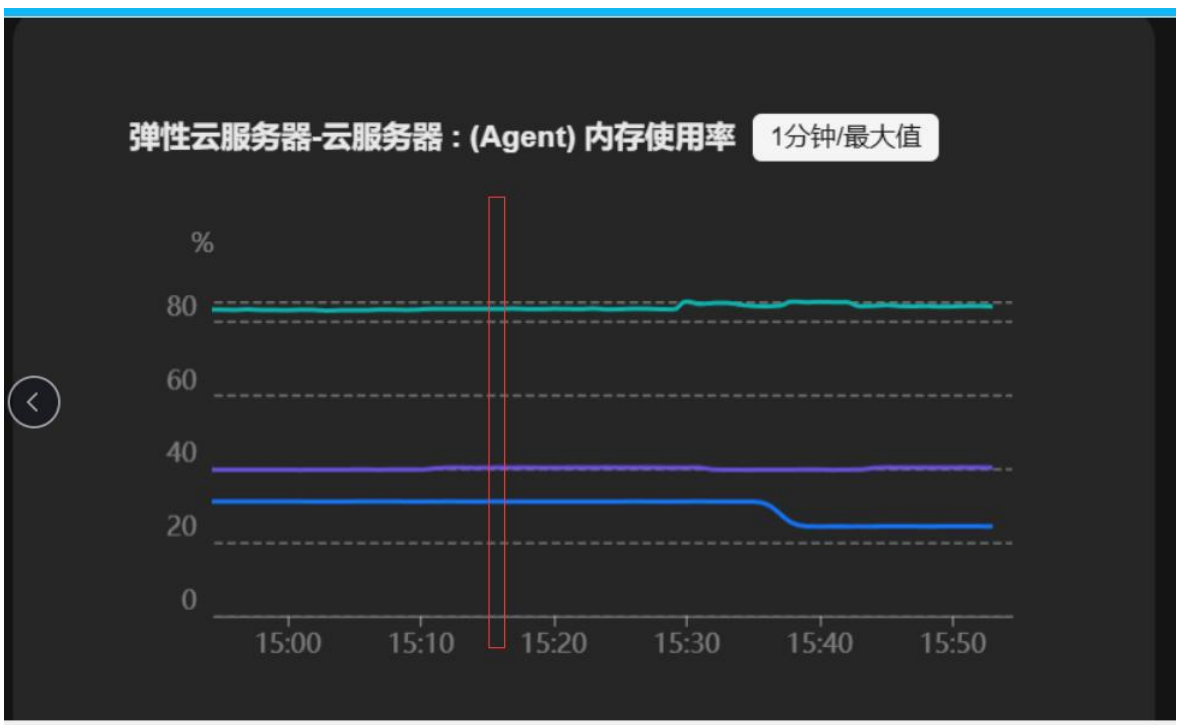


图 10redis-1000-60-13 内存使用率

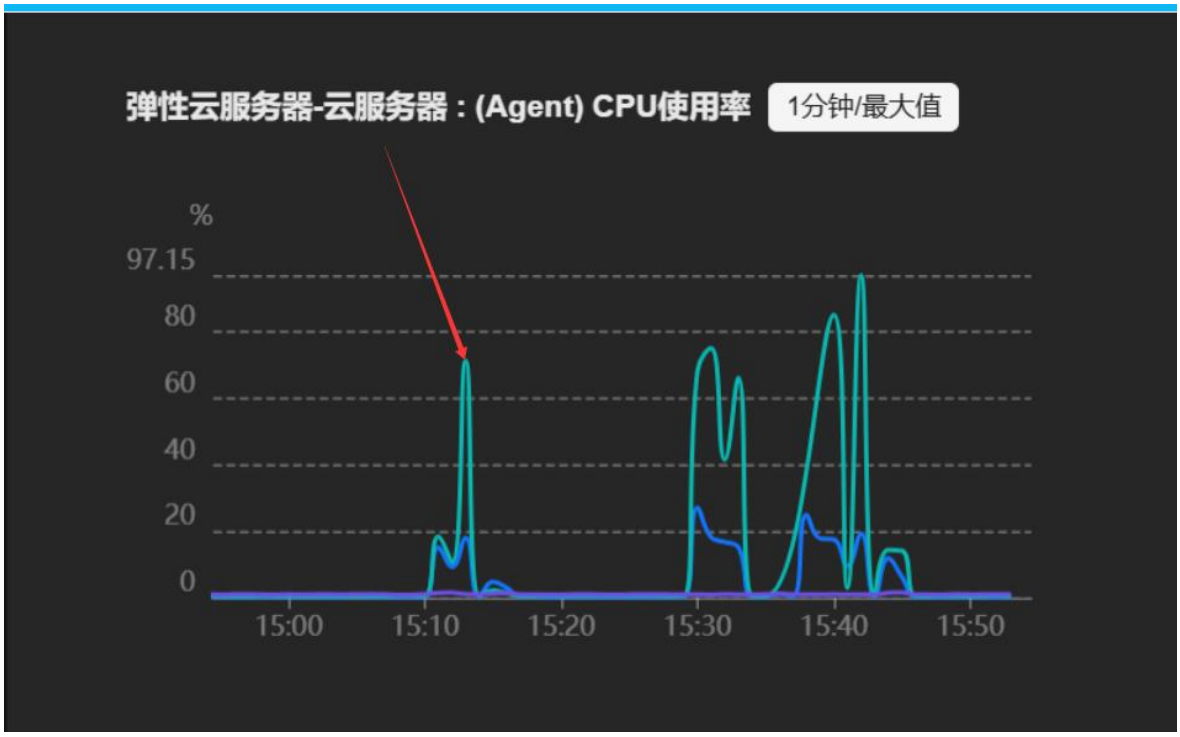


图 11noredis-1000-60-13 CPU 使用率

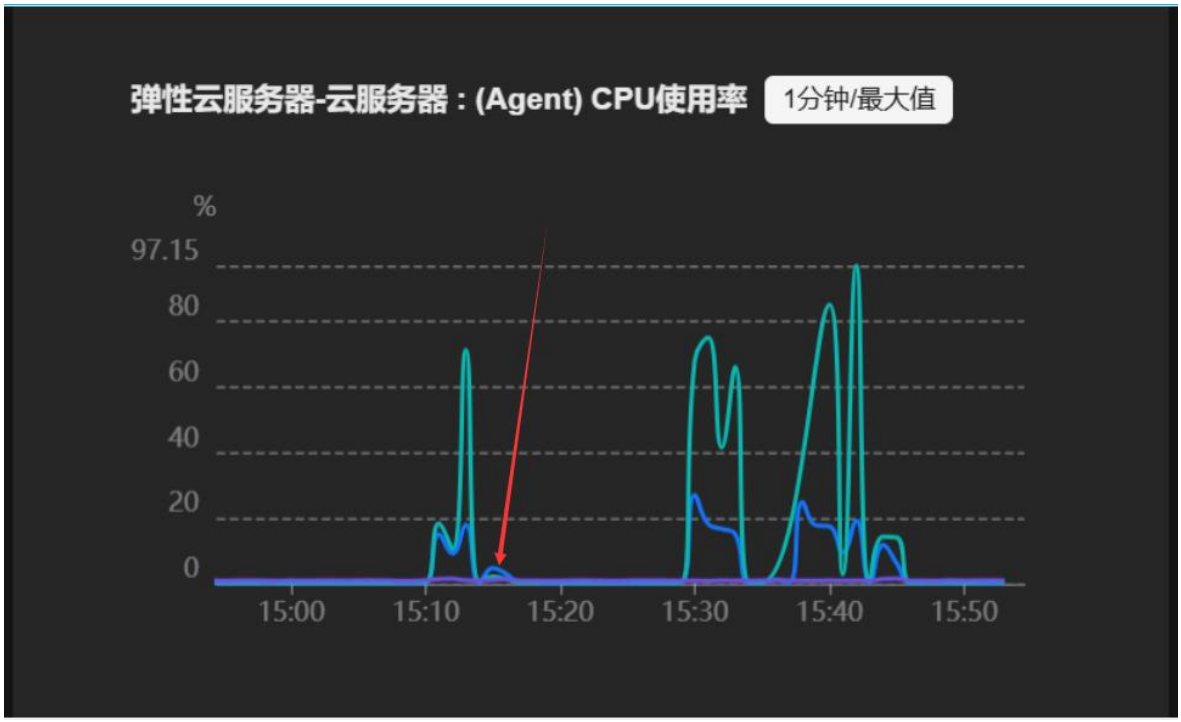


图 12redis-1000-60-13 CPU 使用率

3.2 1000-60-14（无redis在此时开始阻塞）

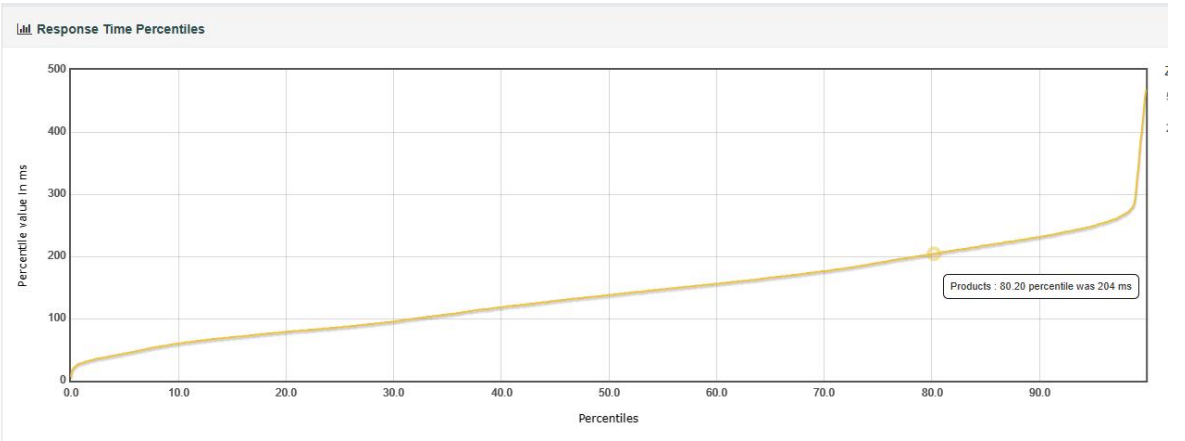


图 1noredis-1000-60-14 Response Time Percentiles

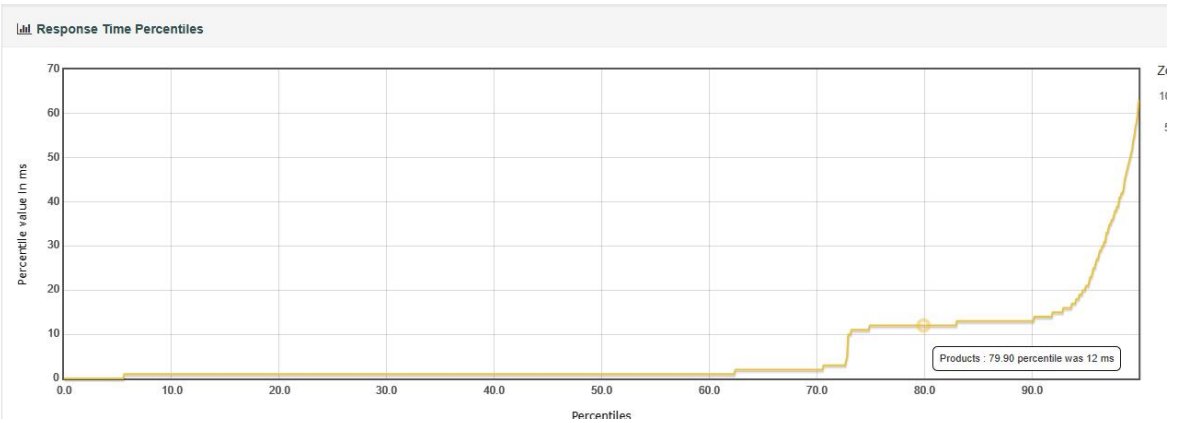


图 2redis-1000-60-14 Response Time Percentiles

noredis80%的请求在 204ms 内响应，redis 在 12ms 内响应
由于此时不使用缓存的方案发生阻塞，故认为 NoRedis 方案的临界可处理线程为 1000-60-13

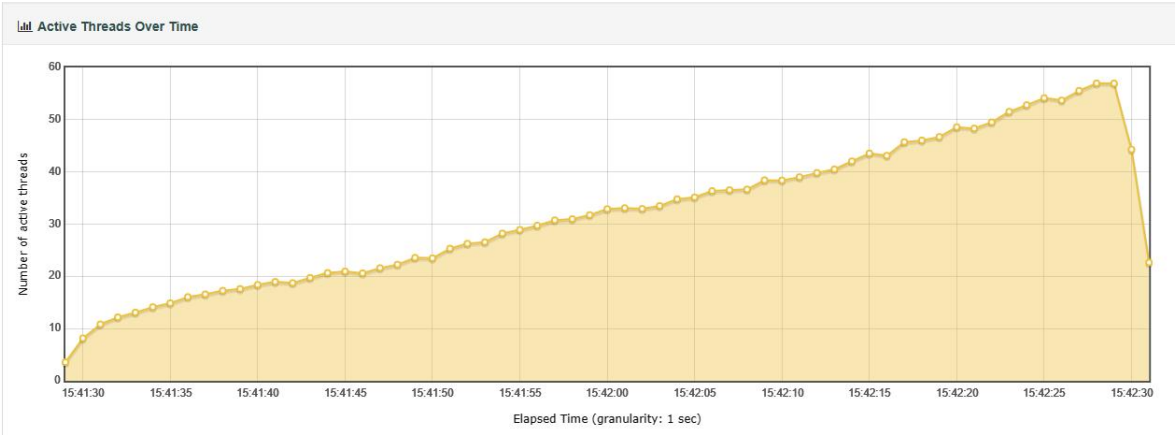


图 3noredis-1000-60-14 Active Threads Over Time

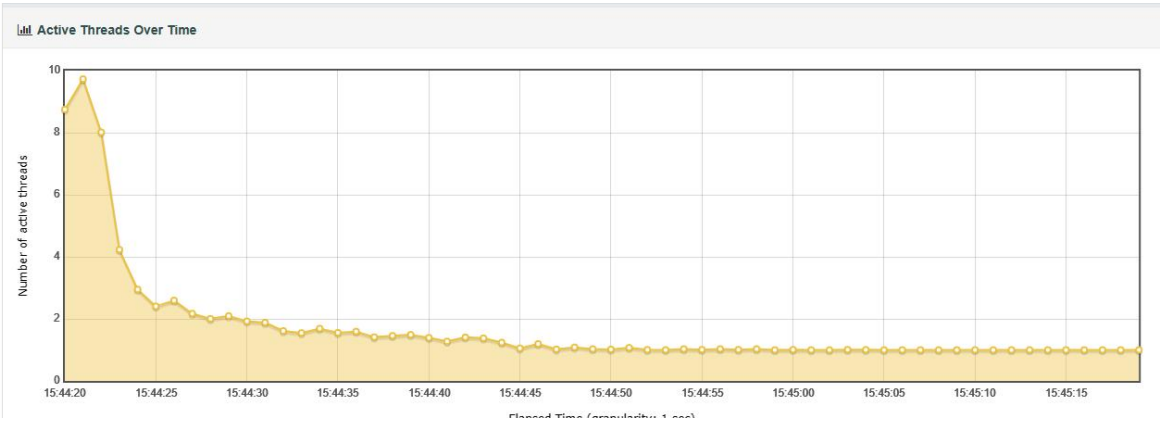


图 4redis-1000-60-14 Active Threads Over Time

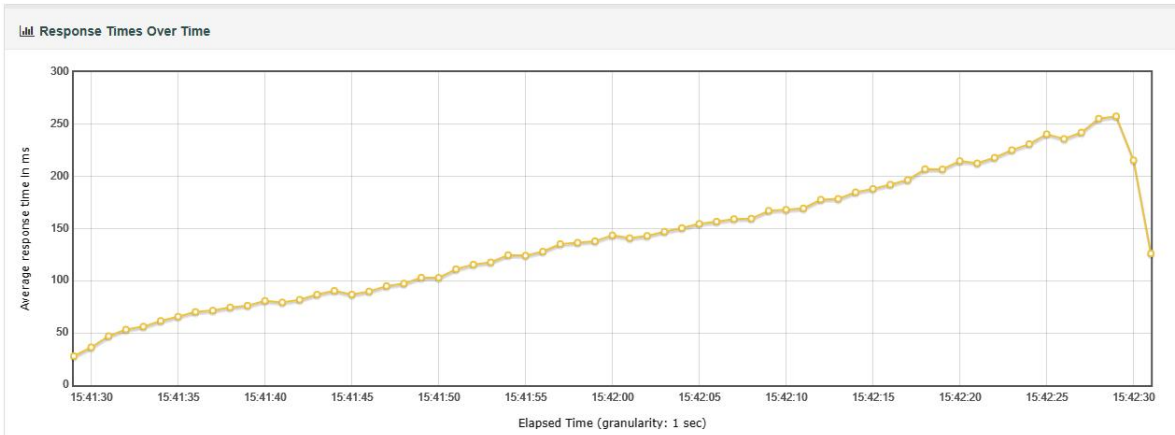


图 5noredis-1000-60-14 Response Times Over Time

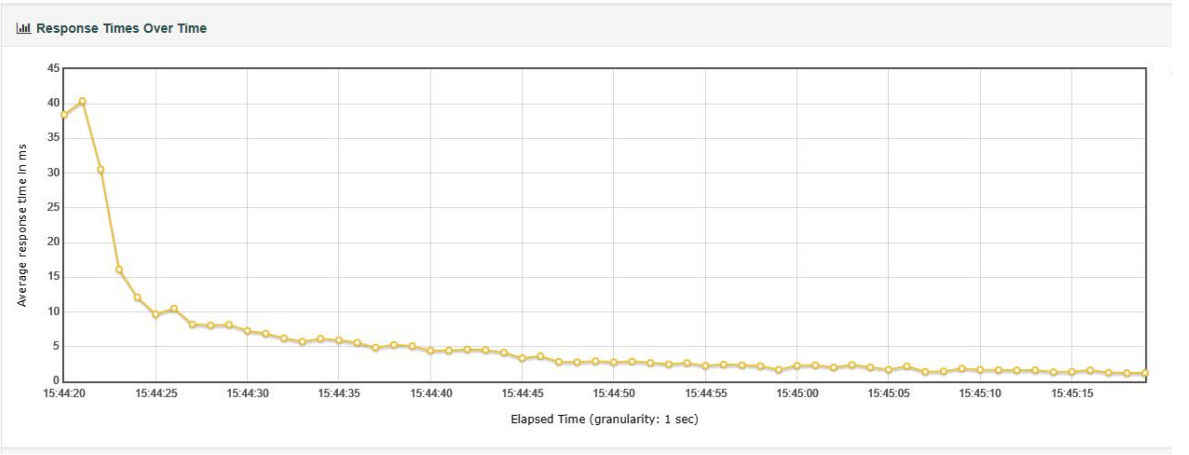


图 6redis-1000-60-14 Response Times Over Time

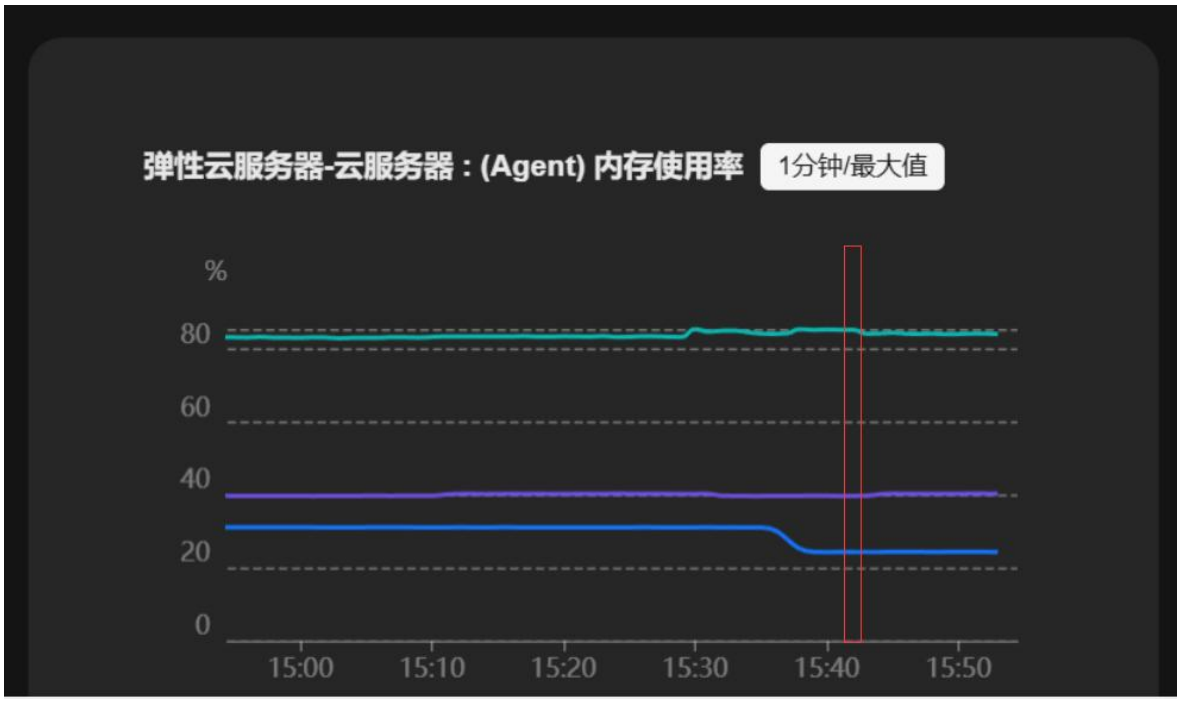


图 9noredis-1000-60-14 内存使用率

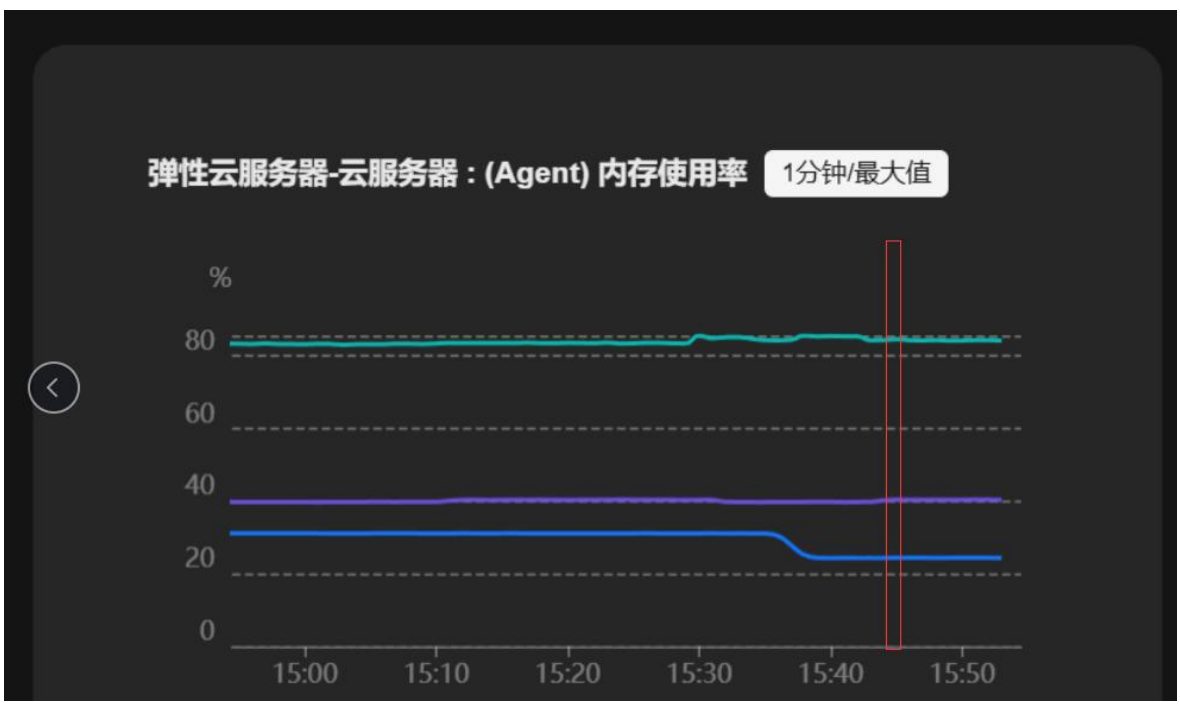


图 10redis-1000-60-14 内存使用率

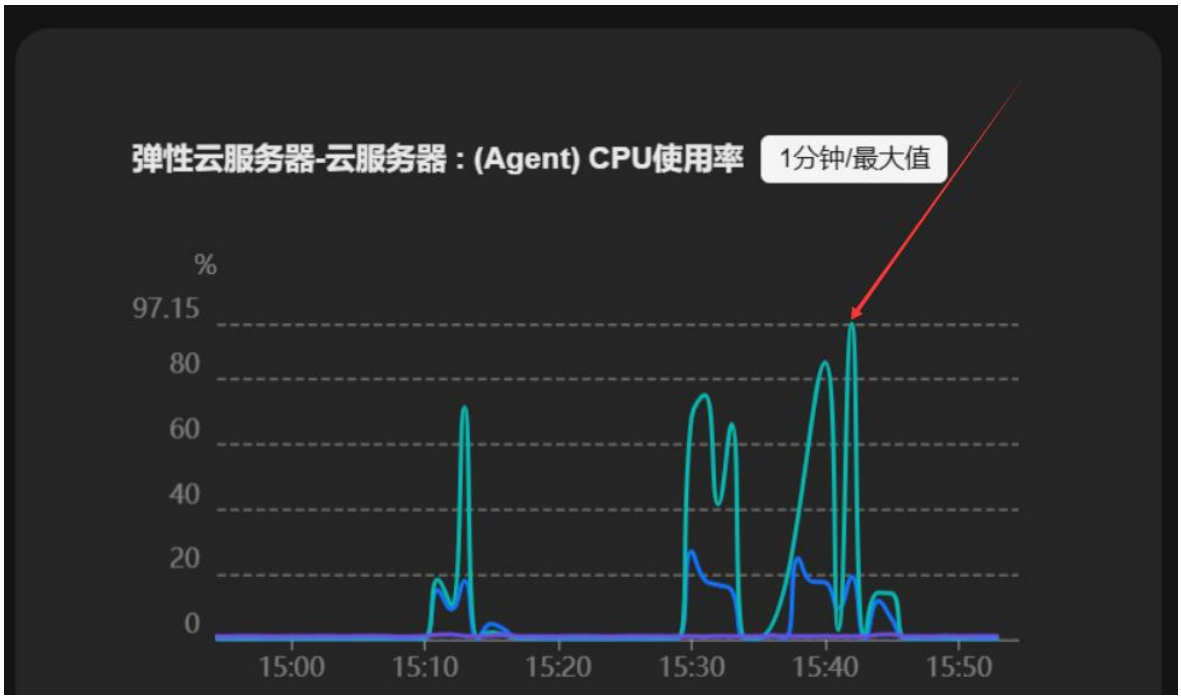


图 11noredis-1000-60-14 CPU 使用率

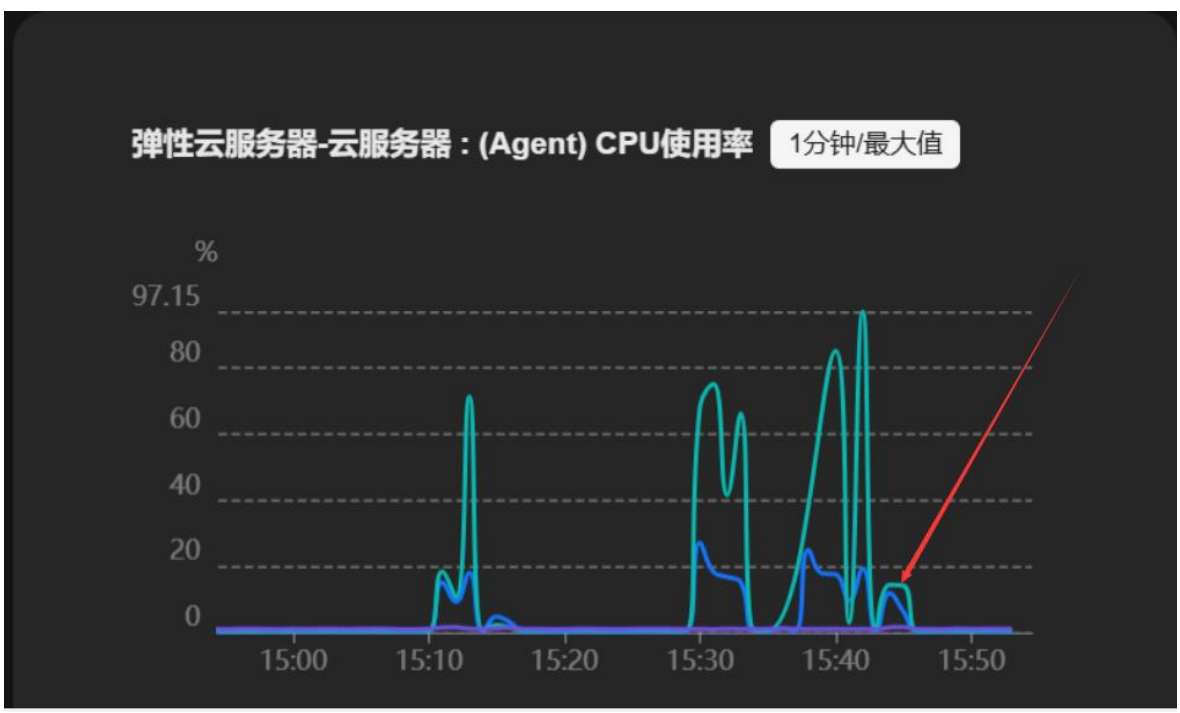


图 12redis-1000-60-14 CPU 使用率

3.3 1000-60-100（仅测试Redis探究其临界性能）

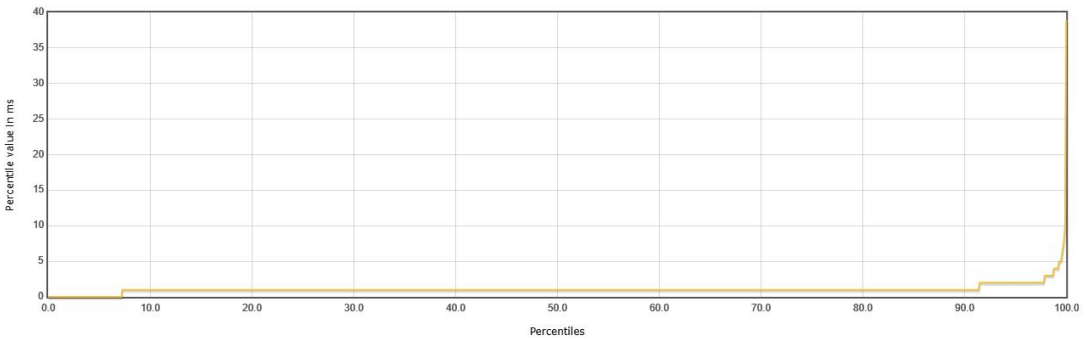


图 1redis-1000-60-100 Response Time Percentiles

redis80%的请求在 1ms 内响应

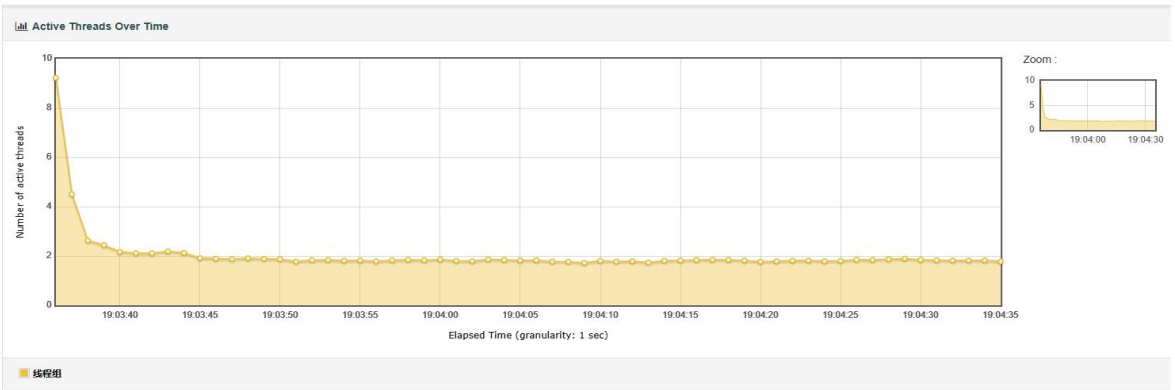


图 2redis-1000-60-100 Active Threads Over Time

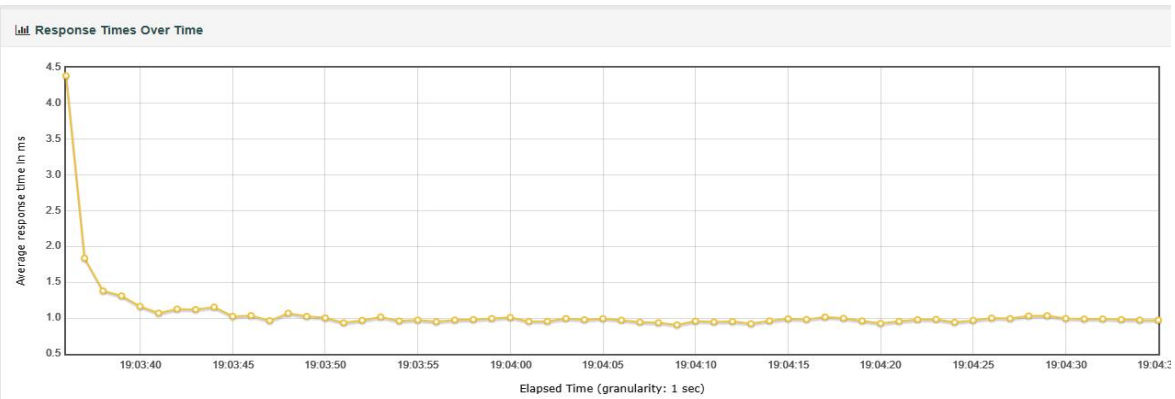


图 3redis-1000-60-100 Response Times Over Time

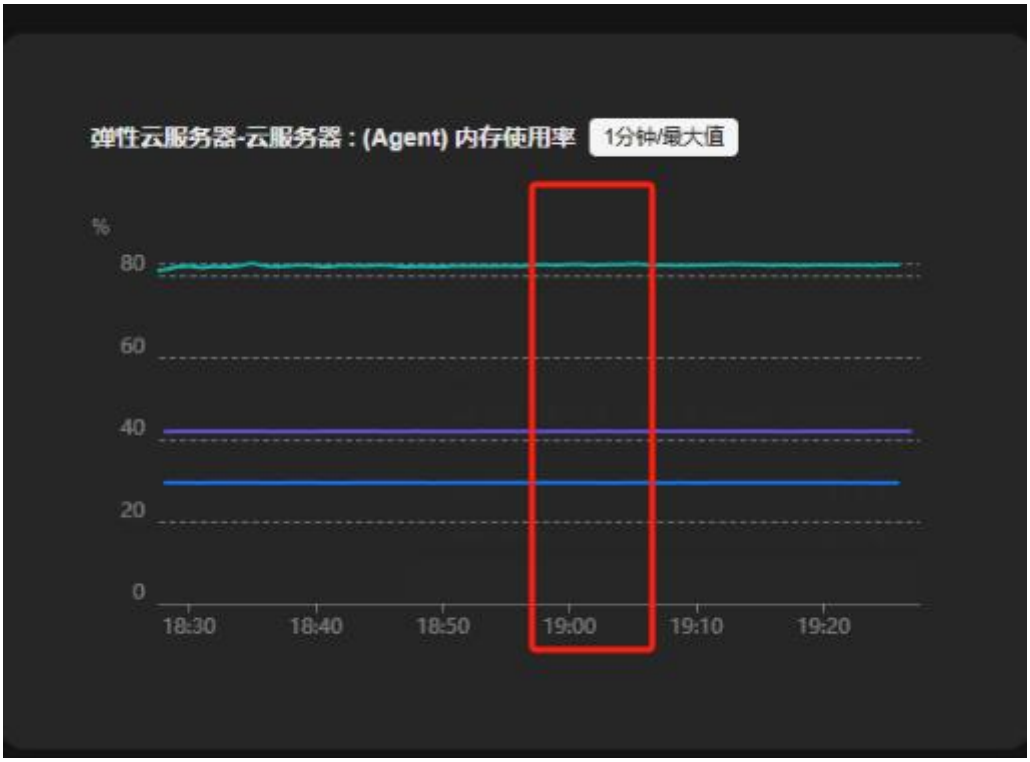


图 4noredis-1000-60-100 内存使用率

图 5redis-1000-60-100 CPU 使用率

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	Redis	NoRedis
1000-60-13	1	12
1000-60-14	12	204（已经产生阻塞）
1000-60-100	1	（完全阻塞）

其中使用缓存方案 1000-60-100 比 1000-60-14 更快的原因是测试 1000-60-14 时有部分缓存被清除。

（1）性能比较

根据实验结果，使用 Redis 缓存的系统在响应速度和处理能力上均优于不使用缓存的系统。具体来说，使用 Redis 缓存的系统在不同负载条件下，80%的请求响应时间均显著低于不使用缓存的系统。数据表明 Redis 缓存能够显著减少数据检索时间，提高系统的响应速度。

（2）临界性能探究

不使用缓存的方案在 1000-60-13 的测试条件下就已经达到临界性能，在 1000-60-14 的条件下已经阻塞，而使用缓存的方案在 1000-60-100 的条件下仍可以平稳运行，即其性能已经超过 1660 次请求/s，之所以没有测到极限负载，是因为 jmeter 服务器内存不足，更多的并发线程数无法正常发送。

4.2 服务器负载：

通过华为云云监控服务对服务器 B 和 C 的负载情况进行了监控，结果显示在相同负载条件下，使用 Redis 缓存的方案相比不使用缓存的方案，服务器的内存使用率在完全阻塞前，而这并无太大区别；而 CPU 使用率明显更低。这表明 Redis 缓存能够有效减轻数据库的查询压力，降低服务器资源消耗，从而提高系统的整体稳定性和可扩展性。

参考文献:

[1] Redis——缓存. https://blog.csdn.net/lwl2014100338/article/details/107876741?ops_request_misc=&request_id=&biz_id=102&utm_term=Redis%20%E7%BC%93%E5%AD%98%E7%9A%84%E6%95%88%E7%8E%87&utm_medium=distribute.pc_search_result.none-task-blog-2~all~sobaiduweb~default-1-107876741.142^v100^pc_search_result_base3&spm=1018.2226.3001.418

[2] Redis 为什么这么快？ Https://blog.csdn.net/qq_45607784/article/details/134875595?ops_request_misc=&request_id=&biz_id=102&utm_term=Redis%20%E7%BC%93%E5%AD%98%E7%9A%84%E6%95%88%E7%8E%87&utm_medium=distribute.pc_search_result.none-task-blog-2~all~sobaiduweb~default-3-134875595.142^v100^pc_search_result_base3&spm=1018.2226.3001.4187

[3] 关于 Redis 缓存的使用和优缺点 https://blog.csdn.net/alice_tl/article/details/75905429?ops_request_misc=&request_id=&biz_id=102&utm_term=Redis%20%E7%BC%93%E5%AD%98%E7%9A%84%E6%95%88%E7%8E%87&utm_medium=distribute.pc_search_result.none-task-blog-2~all~sobaiduweb~default-2-75905429.142^v100^pc_search_result_base3&spm=1018.2226.3001.4187