一键导入性能测试概况

测试概述

项目背景

拆库以后一键导入大物料时常有超时报警发生,为了提升性能,当前开发对一键导入从单进程处理,采取了多线程处理,并在实现结构上进行了部分调整。目前老版本的一键导入导入10万个物料时用时约8个小时。

测试目的

通过对新版本的性能测试,与老版本性能相比较,确定是否有性能提升。在模拟一键导入 真实使用环境的压力负载,欲求提前发现系统的性能瓶颈及各应用服务器资源使用情况;评估 系统在当前的测试设备条件下,能够承受的负载量。通过监控应用服务器端性能概况,判断系 统整体性能及缺陷,协助开发人员调试并优化系统性能。

测试内容

以10万个物料为一个任务进行一键导入。

主要检测指标

- 1. 键导入 10 万个物料处理时间
- 2. 应用服务器的 CPU、Memory 使用情况
- 3. Jvm Memory 使用情况

系统架构

根据业务的需求,测试环境服务器分为内网应用服务器和数据库服务器。具体使用数量如下:

区域划分	服务器角色	数量	服务器型号	备注
内网区	APP 🗵	2		
	DB 🗵	8		

测试环境

序号	设备	硬件配置	软件配置
1	测试压力机	Intel(R) Core(TM)2 Duo CPU E7200 @2.53GHz 4Gb 内存	Windows server2003 Enterprise x64 Edition Service Pack2 LoadRunner8.1 sp4 Controller
2	应用服务器	Panama、apollo 各 1 台 Intel(R) Xeon(R) CPU E5645 @ 2.40GHz*8 8Gb 内存	Linux 2.6.18-194.32.1.el5xen Resin4
3	数据库服务器		

测试方法

本次性能测试共分 4 个场景,场景 1,单任务导入 10 万个物料;场景 2,并发两个任务导入,每个任务导入 10 万个物料;场景 3,并发 3 个任务导入,每个任务导入 10 万个物料;场景 4,连续 48 小时导入物料,作为系统的稳定性测试。

测试执行结果

测试场景 (一) 描述

本次测试场景执行一个任务,导入10万个物料。

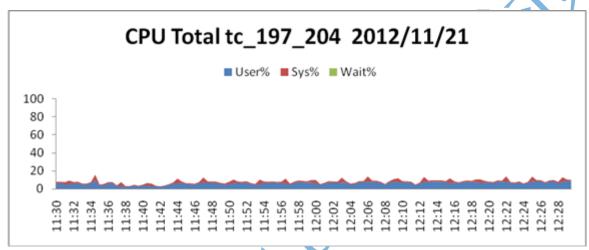
测试场景(一)执行结果

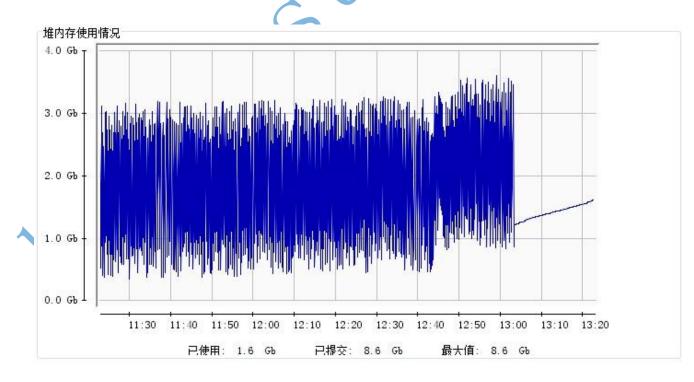
任务用时约1小时40分钟

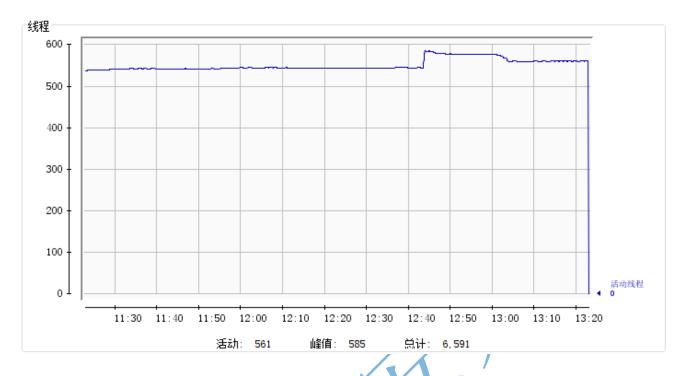
关键性能指标分析

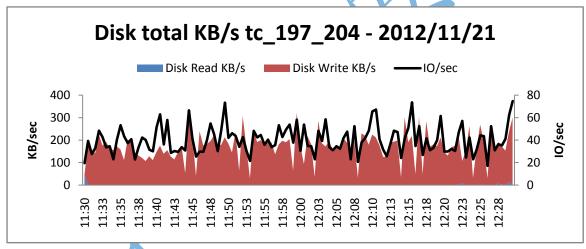
Apollo 应用服务资源使用情况

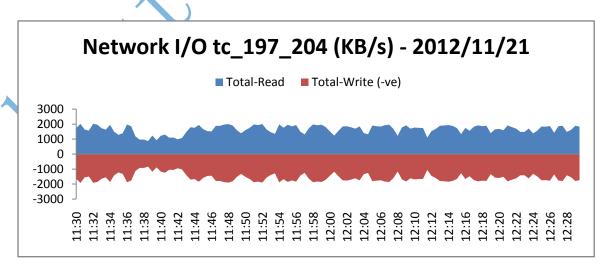




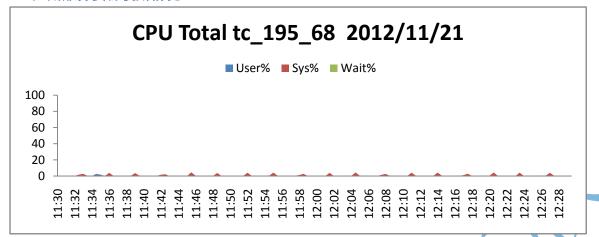


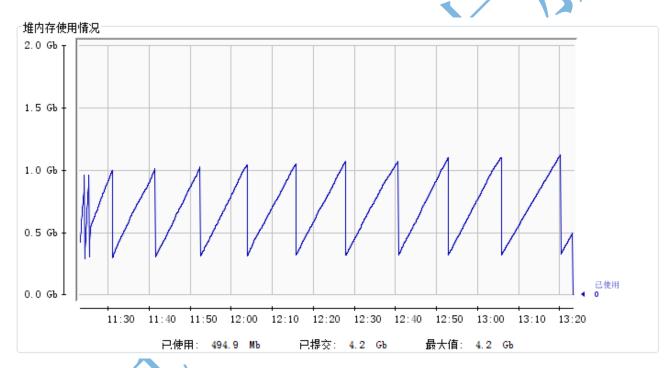


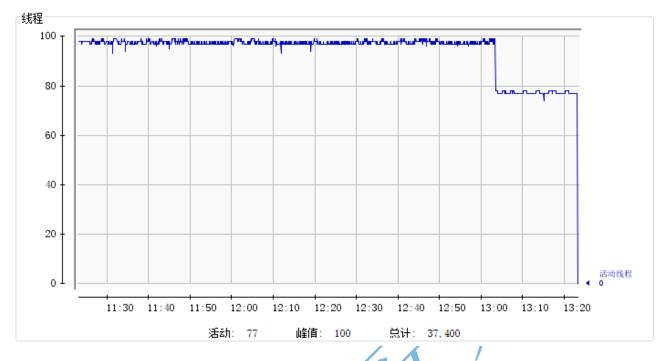


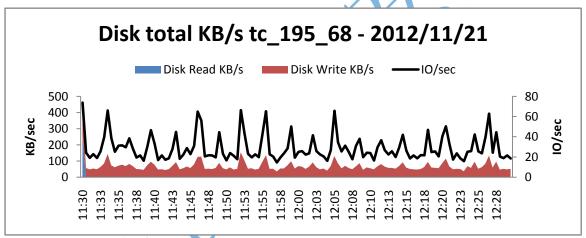


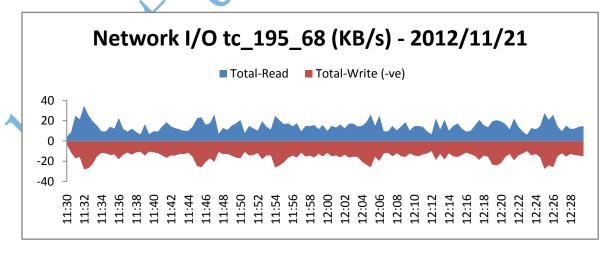
Panamz 应用服务资源使用情况











在单任务执行时,当前版本与线上版本比较,从原来的8小时,导入完成10万个物料,到1小时40分钟导入完成,性能有了4倍左右的提升。

Apollo 的 cpu 资源使用在 10%左右,主用用于用户态应用; 堆内存在 3G,来回浮动; 应用活动线程 550; 磁盘和网络 IO 都很低,整体负载压力比较低。

相比 Apollo, Panama 资源使用率更低,不论 cpu,内存还是 io 都远远低于 apollo。

测试场景 (二) 描述

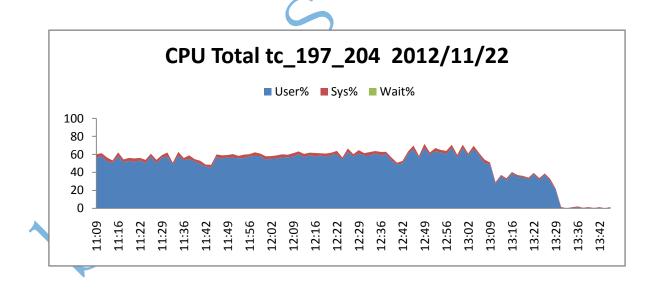
本次测试场景并发执行两个任务,每个任务导入10万个物料。

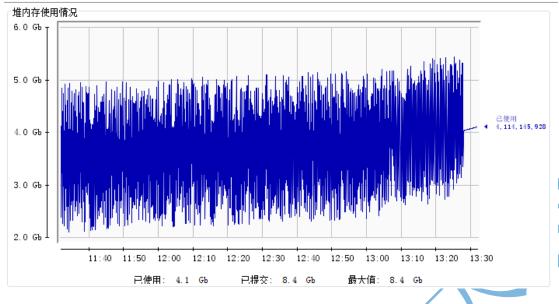
测试场景 (二) 执行结果

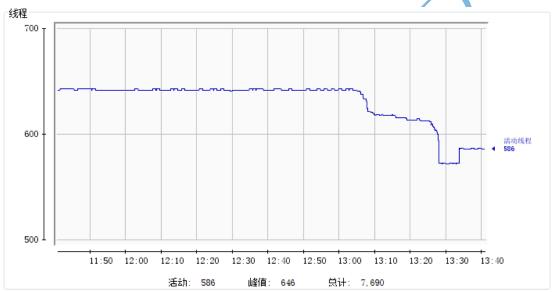
每个任务执行时间完成不一样, a 任务耗时 122 分钟, b 任务耗时 144 分钟。

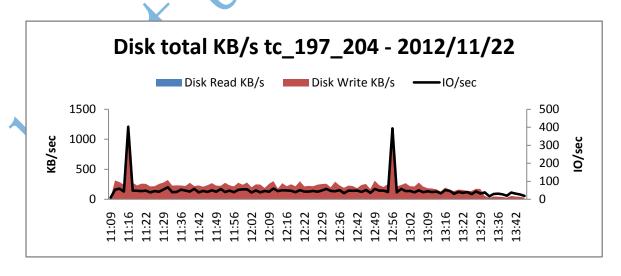
关键性能指标分析

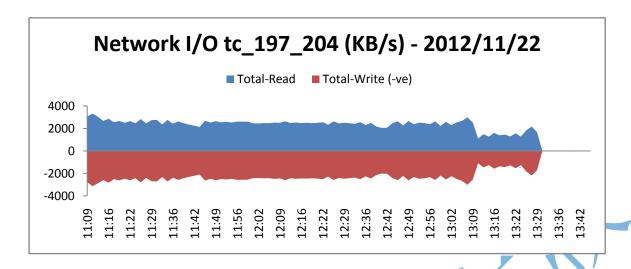
Apollo 应用服务资源使用情况



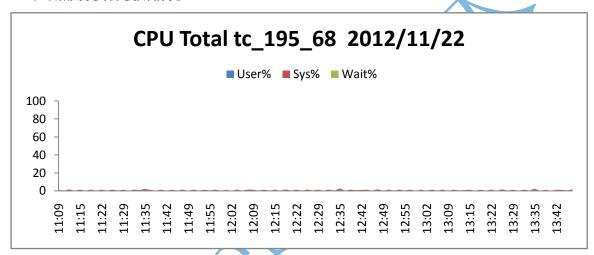


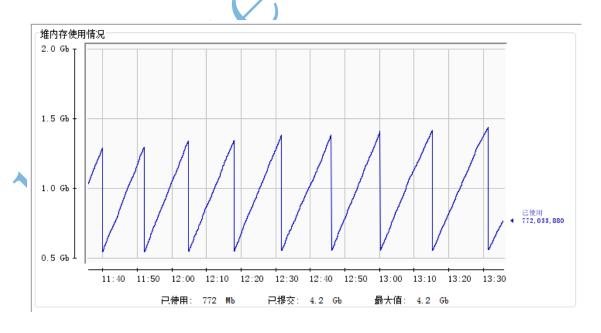


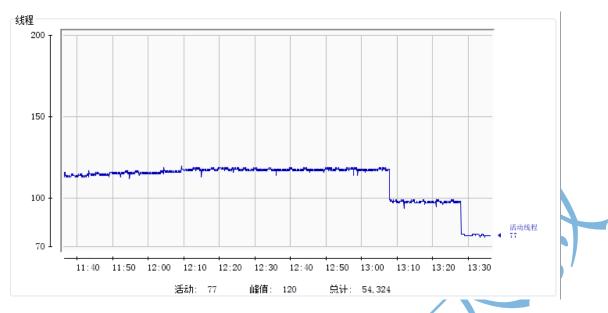


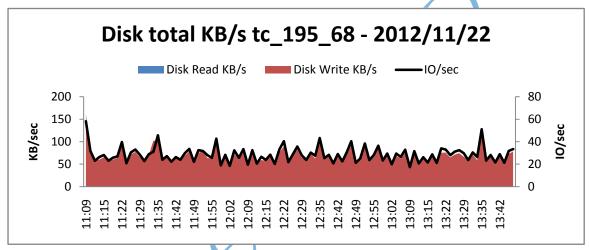


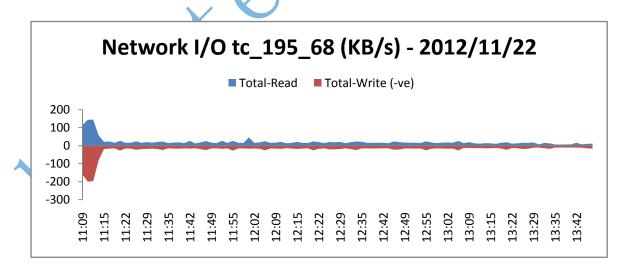
Panamz 应用服务资源使用情况











与场景 1 单任务执行相比,场景 2 的处理时间有所变长,场景 2 的任务平均完成时间是133 分钟(a 任务 122 分钟, b 任务 144 分钟),比场景 1 的 100 分钟多了 40 分钟。

Apollo 的 cpu 资源使用增长到了 60%, 主用用于用户态应用; 堆内存增峰值长到了 5G; 应用活动线程 650; 磁盘和网络 IO 都很低。

Panama 资源使用率很低,cpu 没有太多变化,内存有所增长,峰值 1.5G,磁盘和网络 IO 很小。

测试场景 (三) 描述

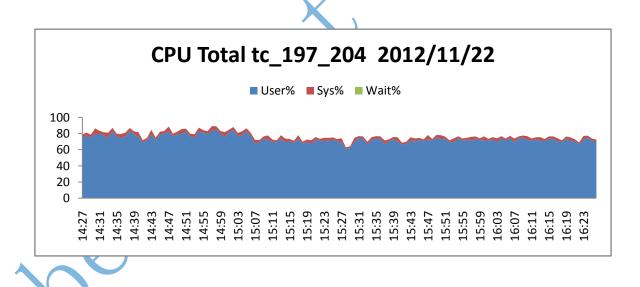
本次测试场景并发执行三个任务,每个任务导入10万个物料。

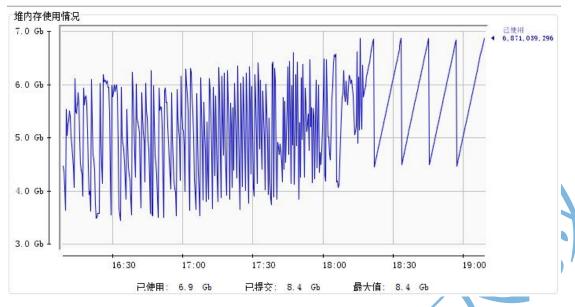
测试场景(三)执行结果

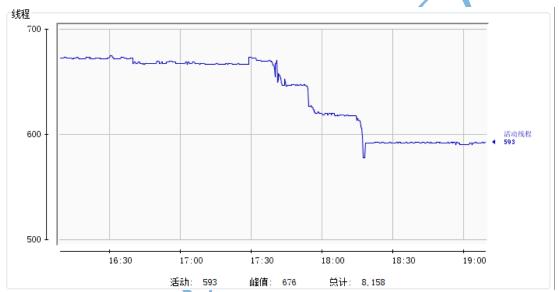
每个任务执行时间完成不一样, a 任务耗时 199 分钟, b 任务耗时 212 分钟, c 任务耗时 235 分钟。

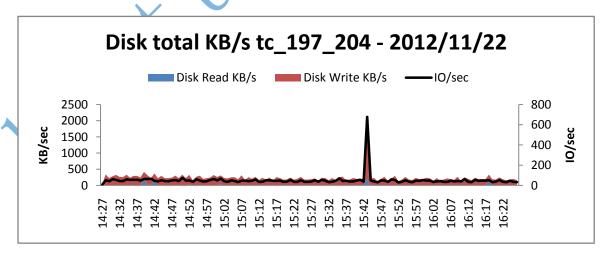
关键性能指标分析

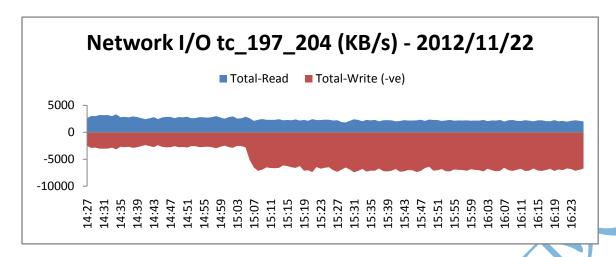
Apollo 应用服务资源使用情况



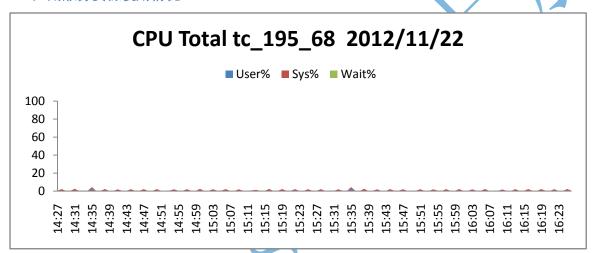


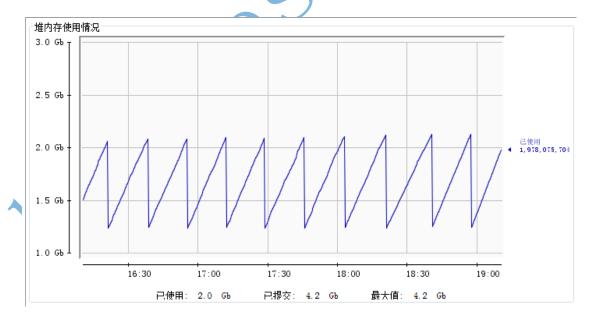


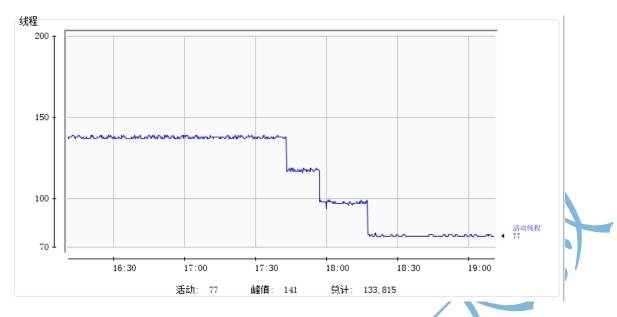


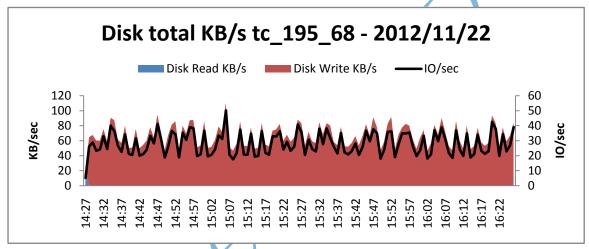


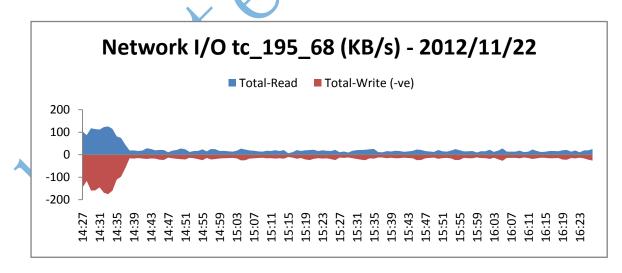
Panamz 应用服务资源使用情况











场景 3 的任务平均完成时间是 215 分钟 (a 任务 199 分钟, b 任务 212 分钟, c 任务 235),比场景 2 的 133 分钟多了 65 分钟。

Apollo 的 cpu 资源使用增长到了 80%, 主用用于用户态应用; 堆内存增峰值长到了 7G; 应用活动线程 670; 磁盘和网络 IO 都很低。

Panama 资源使用率很低, cpu 没有太多变化, 内存有所增长, 峰值 2.0G, 磁盘和网络 I0 很小。

测试场景(四)描述

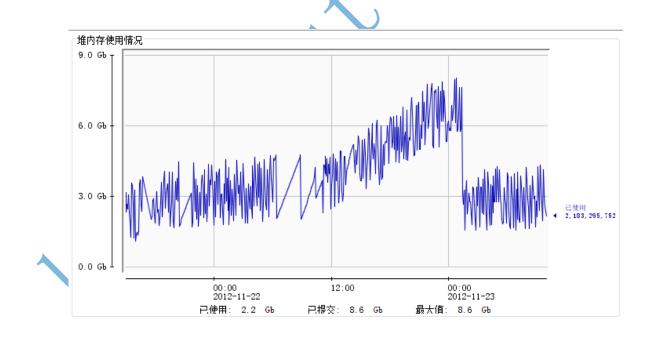
本次测试场景持续48小时导入任务,每个任务导入10万个物料,观察内存使用情况。

测试场景 (三) 执行结果

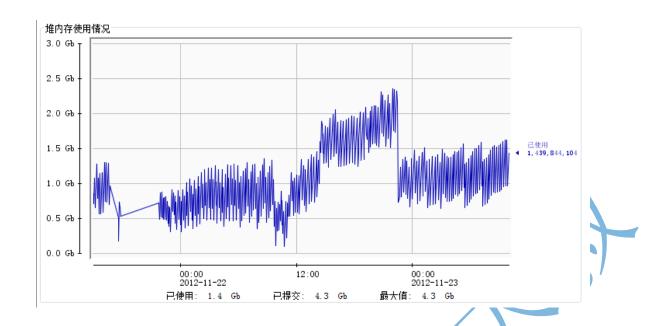
Apollo 和 panama 内存回收正常,未发现内存溢出。

关键性能指标分析

Apollo 应用服务器内存使用情况



Panamz 应用服务资源内存使用情况



持续 48 小时测试, apollo 内存回收正常, 在内存增长到 8G 的时候(当前 jvm 开设最大内存 8G) 执行了一次 full GC, 内存回收到了正常水平。

Panama 内存峰值 2.3G,实际可供内存使用 4G,内存回收正常。

问题及结论

综合几个场景测试,可知,一键导入服务器的性能压力主要在 apollo 的服务上,优化后的一键导入从原来 panama 的单线程调用到多线程调用,目前一个任务起 20 个线程,对 apollo 压力很大, apollo cpu 资源消耗非常高,因此对单台 apollo 来说,随着任务的增多,cpu 资源使用率越来越高,从而导致任务完成的时间几乎成倍增长。

虽然 panama 进行了优化,与线上版本比较,任务导入完成时间得到了 4 倍的性能提升,但在 apollo 没有完成优化前,在多任务并发情况下,系统上线要考虑 apollo 的负载能力,若 apollo 负载较高,可以采取减少 panama 任务的线程数等方式,来降低对 apollo 的负载。s

当前 Apollo 服务仍有优化提升的空间,下面提供 apollo 主要消耗 cpu 资源的具体代码和线程阻塞的具体代码调用视图。

1、cpu 消耗高的代码调用树木视图

```
© ■ 68.4% - 2,605 s com.sogou.apollo.aop.AopPointCuter
     ■ 86.1% - 3,280 s java.lang.Thread
©
        ■ 68.3% - 2,605 s org.springframework.aop.aspectj.MethodInvocationProceedingJoinPoint
  Ė. O =
          ■ 67.2% - 2,560 s com.sogou.apollo.facade.cpc.impl.CpcAPIImpl
     Ė...@ =

    65.8% - 2.508 s com.sogou.bizdev.platform.datasource.distributed.aspect.ServiceAspect

         65.3% - 2,487 s org.springframework.aop.aspectj.MethodInvocationProceedingJoinPoint
            ■ 63.3% - 2,411 s com.sogou.apollo.service.impl.CpcMgrImpl
              चि === 52.2% - 1,989 s $Proxy27
                52.2% - 1,989 s com.sogou.bizdev.platform.datasource.distributed.aspect.ServiceAspect
                   48.1% - 1,833 s org.springframework.aop.aspectj.MethodInvocationProceedingJoinPoint
                     = 0 = 48.1% - 1,833 s com.sogou.bizdev.platform.generalinterceptor.GeneralAspect
                       45.7% - 1,742 s org.springframework.aop.aspectj.MethodInvocationProceedingJoinPoint
                          😑 🧿 🚃 45.7% - 1,742 s com.sogou.bizdev.cpc.key.service.impl.CpcKeyServiceImpl
                            🗖 🧑 ■ 39.4% - 1,502 s $Proxy17
                              😑 🧿 🚃 39.4% - 1,502 s com.sogou.bizdev.platform.generalinterceptor.GeneralAspect
                                 🖟 💿 🕳 39.4% - 1,502 s org.springframework.aop.aspectj.MethodInvocationProceedingJoinPoint
                                   🚊 🧿 🚃 39.4% - 1,502 s com.sogou.bizdev.common.bizvalidate.impl.BizValidatorServiceImpl
                                     😑 🧿 🚃 39.4% - 1,502 s com.sogou.bizdev.common.bizvalidate.impl.CreateCpcKeyBizValidator
                                        🖟 🧑 ∎ 10.3% - 391 s $Proxy21
                                          🖆 🥝 🛮 10.2% - 390 s com.sogou.bizdev.platform.datasource.distributed.aspect.ServiceAspect
                                             🖃 🧿 🛮 9.0% - 342 s com.sogou.bizdev.platform.datasource.switchms.MySwitcherDst
                                               🖻 🧿 🛮 8.8% - 336 s com.sogou.bizdev.platform.datasource.switchms.SwitchUtils
                                                 🖶 🧿 🛮 8.6% - 327 s com.sogou.bizdev.utils.cache.MyCacheUtils

■ 0 ■ 8.6% - 326 s com.sogou.bizdev.utils.cache.MyMemCache

                                                      □ 0 ■ 8.6% - 326 s com.danga.MemCached.MemCachedClient
                                                        ⊟ © ■ 8.6% - 326 s com.schooner.MemCached.AscIIClient
                                                           😑 🧿 ı 4.5% - 173 s com.schooner.MemCached.SockInputStream
                                                               🧿 । ४.५% - १७२ s sun.nio.ch.SocketChannelImpl
                                                               0.0% - 401 ms java.lang.String
                                                             ± ©
                                                                  0.0% - 64,042 µs java.io.InputStream
                                                              0.0% - 56,018 µs java.io.ByteArrayOutputStream
                                                           ⊕ 🧿 0.6% - 21,852 ms com.schooner.MemCached.SchoonerSockIOPool$TCPSoc
                                                            0.1% - 5,060 ms java.net.URLEncoder
                                                            0.0% - 614 ms java.lang.String
                                                               0.0% - 175 ms java.lang.StringBuffer
```

2、线程阻塞代码调用树视图

```
■ 100.0% - 939 s java.lang.Thread.run
🚊 🕡 🚥 32.9% - 308 s com. sogou. apollo. aop. AopPointCuter. doSomething
   🖆 🧓 💳 32.9% - 308 s org. springframework. aop. aspectj. MethodInvocationProceedingJoinPoint. proceed
      🖮 🧰 == 32.9% - 308 s com. sogou. apollo. facade. cpc. impl. CpcAPIImpl. addCpc
         🚊 🔟 🚥 32.7% - 307 s com. sogou. apollo. facade. cpc. impl. CpcAPIImpl. batchAddCpcDIO
            🖃 🧓 == 32.7% - 306 s $Proxy52.addCpcDTO
               🖆 👊 💳 32.7% - 306 s com. sogou. bizdev. platform. datasource. distributed. aspect. ServiceAspect. doOperationBefore
                   🚊 🧓 💻 32.6% - 305 s org. springframework. aop. aspectj. MethodInvocationProceedingJoinPoint. proceed
                      🖨 🔟 💻 23.7% - 222 s com. mchange. v2. c3p0. impl. AbstractPoolBackedDataSource. getConnection
                         🚊 🔟 💻 23.7% - 222 s com. mchange. v2. c3p0. impl. C3P0PooledConnectionPool. checkoutPooledConnection
                            😑 🗓 💻 23.7% - 222 s com. mchange. v2. resourcepool. BasicResourcePool. checkoutResource
                               🖮 🔟 💻 23.7% - 222 s com. mchange. v2. resourcepool. BasicResourcePool. prelimCheckoutResource
                                   🖮 👊 💻 23.7% - 222 s com. mchange. v2. resourcepool. BasicResourcePool. awaitAvailable
                                       🧓 0.0% - 32,001 ⊥s com.mchange.v2.c3p0.impl.NewPooledConnection.getConnection
                      ⊞ · 📵 ■ 8.6% - 81,009 ms com. sogou. apollo. service. impl. CpcMgrImpl. addCpcDTO
                      🗓 🗓 0.2% - 2,167 ms com.mchange.v2.c3p0.impl.NewProxyConnection.commit
                      🗓 🧓 0.0% - 43,877 µs com.mchange.v2.c3p0.impl.NewProxyConnection.close
                   🔠 🧓 0.1% - 1,012 ms com. sogou. bizdev. platform. datasource. distributed. aspect. ServiceAspect. process
            ษ 🗓 0.0% - 177 ms com. sogou. cupid. util. APICache. putToDBCache
            in 0.0% - 136 ms $Proxy49.getCpcGrpId2CpcPlanIdByCpcGrpId
            🖮 🧓 0.0% - 95,989 🗠 com. sogou. apollo. util. TimeUtils. sleepSeconds
           0.1% - 1,068 ms $Proxy64.getCpcMatchTypes
         🗓 🗓 0.0% - 304 ms com. sogou. apollo. util. IDValidator. getIdAuthorizedErrorDTOsForAddUpdate
         👜 🗓 0.0% - 183 ms com. sogou. apollo. validator. objectvalidator. CpcValidator. validateCpc
         🖮 🗓 0.0% - 16,010 Hs com. sogou. cupid. response. BaseResponse. toString
🔠 👊 0.0% - 48,010 🗠 com. sogou. apollo. facade. unifytask. impl. UnifyTaskAPIImpl. getUnifyTaskByTypes
```