**Docker稳定性测试报告**

**版本<0.1>**

**文档信息**

|  |
| --- |
| 标题: |
| 作者:丁文军 |
| 创建日期:2015/09 |
| 上次更新日期: |
| 版本:0.1 |
|  |
| 部门名称:测试部 |

**修订历史记录**

|  |  |  |  |
| --- | --- | --- | --- |
| **日期** | **版本** | **说明** | **作者** |
| 2015.9.30 | 0.1 |  | 丁文军 |
|  |  |  |  |

目录

[1. 简介 3](#_Toc432009746)

[1.1. 背景 3](#_Toc432009747)

[1.2. 测试目的 3](#_Toc432009748)

[1.3. 参考资料 3](#_Toc432009749)

[2. 测试概要 3](#_Toc432009750)

[2.1. 测试工具 3](#_Toc432009751)

[2.2. 测试环境配置 3](#_Toc432009752)

[3. 测试过程及结果分析 4](#_Toc432009753)

[3.1. 用例结果分析 4](#_Toc432009754)

[3.2. 缺陷分析 4](#_Toc432009755)

[3.2.1. 缺陷统计 4](#_Toc432009756)

[3.2.2. 缺陷列表 4](#_Toc432009757)

[3.3. 测试分析 4](#_Toc432009758)

[3.3.1. 接口测试 4](#_Toc432009759)

[3.3.2. 并发测试 6](#_Toc432009760)

[3.3.3. 压力测试 10](#_Toc432009761)

[4. 结论和建议 23](#_Toc432009762)

# 简介

## 背景

docker作为民生金融通用云主要使用的云计算技术，可以把开发者从日常部署应用的繁杂工作中解脱出来，让开发者能专心写好程序；从系统工程师的角度来看也是一样，他们迫切需要从各种混乱的部署文档中解脱出来，让系统工程师专注在应用的水平扩展、稳定发布的解决方案上。

## 测试目的

本测试的测试对象是docker及可调dockerapi。主要对docker进行稳定性测试，同时对Docker集群devops管理平台可调dockerapi进行调用的可靠性测试，及调用的并发性能测试，得到docker在应用过程中的性能和功能边界，并初步分析测试结果是否符合期望。并将测试结果作为技术选型的重要标准。

## 参考资料

1. <<民生发布平台概要设计说明书2.0>>
2. Docker remote api: <http://docs.docker.com/reference/api/docker_remote_api/>
3. Locust: http://locust.io/

# 测试概要

## 测试工具

压力测试工具：nstress

网络测试工具：rsync

并发测试工具：locust

## 测试环境配置

测试环境为民生内部服务器，信息如下：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **主机名** | **IP** | **角色** | **CPU** | **内存** |
| DCBS8PJQ | 197.3.84.237 | Docker主机 | 64core | 128G |
| DCBS9PJQ | 197.3.84.238 | Docker主机 | 64core | 14G |
| DCBSAPJQ | 197.3.84.239 | Docker主机 | 64core | 512G |
| DCBS8PJQ | 197.3.155.154 | Swarm集群管理服务器 | - | - |
| DCBS9PJQ | 197.3.155.155 | 镜像仓库服务器 | - | - |

# 测试过程及结果分析

## 用例结果分析

|  |  |  |  |
| --- | --- | --- | --- |
| **用例数目** | **测试类型** | **成功率** | **备注** |
| 42 | 功能 | 92.8% | 测试Docker主机 |
| 42 | 功能 | 90.5% | 测试swarm集群 |
| 6 | 并发 | N/A |  |
| 12 | 性能 | N/A |  |

## 缺陷分析

### 缺陷统计

|  |  |  |
| --- | --- | --- |
| **关闭缺陷数** | **遗留缺陷数** | **总计** |
| 1 | 5 | 6 |

### 缺陷列表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **缺陷号** | **缺陷标题** | **缺陷类型** | **缺陷严重性** | **缺陷状态** |
| Bug1 | 在并发停止容器时出现部分容器无法检索到导致停止请求失败 | 性能 | 主要 | 打开 |
| Bug2 | 在删除容器的过程中创建容器的操作会非常缓慢 | 性能 | 主要 | 打开 |
| Bug3 | 进行50个容器写请求操作导致大部分数据无法从cache中sync到硬盘，且无法进行容器创建关闭等操作 | 性能 | 主要 | 打开 |
| Bug4 | 插入"key": "value"键值对格式的无效参数，容器也意外被创建 | 功能 | 主要 | 打开 |
| Bug5 | 仓库镜像搜索功能不可用 | 功能 | 主要 | 关闭 |
| Bug6 | 使用curl发送请求到swarm主机，调用/containers/create创建容器，部分参数不能被成功解析而创建失败 | 功能 | 主要 | 打开 |

## 测试分析

### 接口测试

#### 测试原理

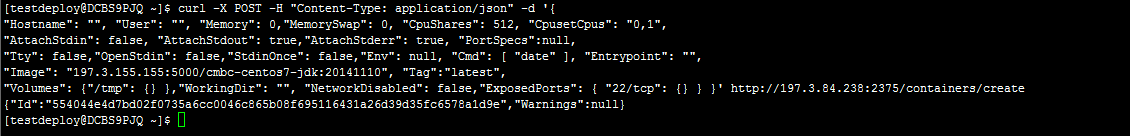
Docker Remote API 是Docker自带的一个REST API管理，本次测试使用的curl命令，通过标准的HTTP方法GET、POST、DELETE等，发送RESTful请求给Docker主机和Swarm集群管理服务，得到服务器返回码，验证Docker Remote API的功能。

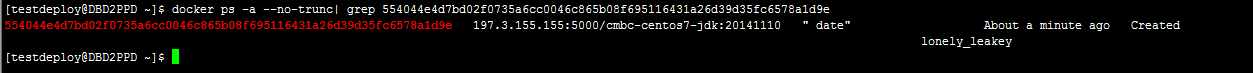
同时通过向Docker追和Swarm集群管理服务发送错误参数的Docker Remote API请求，得到服务器返回码，验证Docker Remote API的容错性。

#### 测试场景

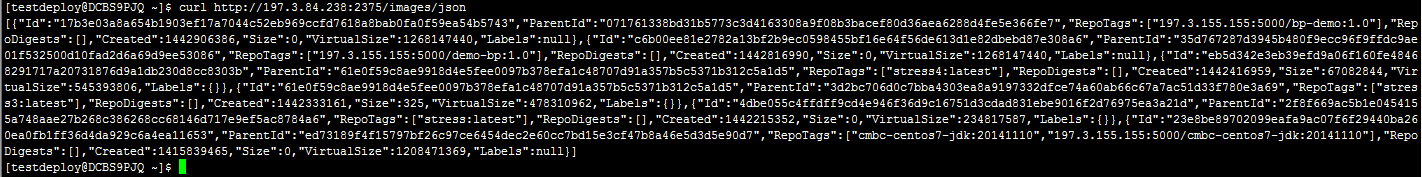
1. Docker Remote API功能验证

通过向Docker 主机发送对容器进行创建、开启、关闭、删除、获取日志、获取容器列表等操作的API，验证Docker Remote API对容器操作的功能有效性。





通过向Docker 主机发送对镜像进行创建、删除、操作、删除、获取容器列表等操作的API，验证Docker Remote API对镜像操作的功能有效性。



1. Docker RemoteAPI容错性验证

在使用curl向Docker主机或者Swarm集群服务发送RESTful请求过程中，使用或者加入错误的参数，验证Docker Remote API的容错性。

C:\Users\herong\Desktop\CMBC\current\0924-0928_1st-test-scrs\41-1.PNG

C:\Users\herong\Desktop\CMBC\current\0924-0928_1st-test-scrs\43-3.PNG

1. Swarm集群功能验证

通过向Swarm集群服务发送Docker Remote API请求，对容器和镜像进行基本操作和带错误参数的请求链接，验证Swarm集群服务对Docker Remote API的功能支持。



#### 测试结果分析

1. Docker Remote API功能验证

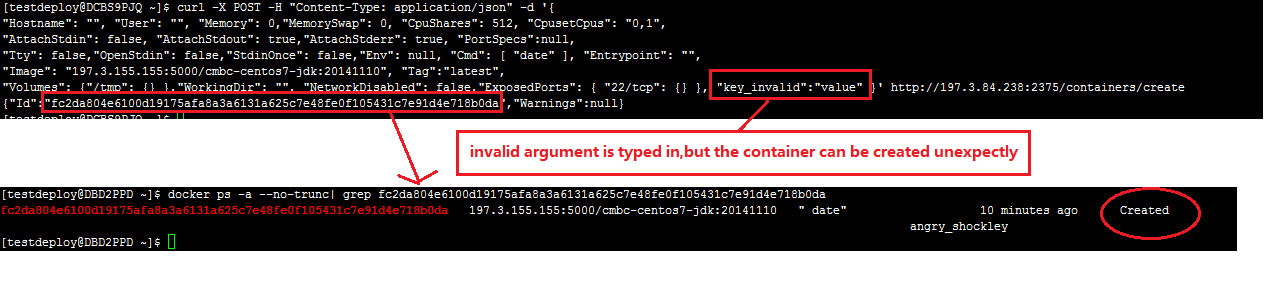
此场景测试，共运行测试案例23条，通过的案例为22条，验证了对容器的操作 (创建、开启、关闭、删除、获取日志、获取容器列表)都可以通过Docker Remote API完成，对镜像的操作(创建、删除、获取镜像列表等) 都可以通过Docker Remote API完成。

失败的用例为1条，记录为Bug5，Bug5已通过配置Docker参数的方式修复。

1. Docker RemoteAPI容错性验证

此场景测试，共运行测试案例19条，通过的案例为17条，验证了在发送创建、开启、关闭、删除、获取日志、获取容器列表等RESTfull请求时，带入错误的参数，Docker能够返回相应的错误返回码，供前台应用平台捕获。

失败的用例为2条，发现创建容器过程中，带入不可识别的参数创建容器，容器仍然能创建成功，但是会导致创建的容器无法开启，始终为Created状态，此问题已记录为Bug4，影响一般。



1. Swarm集群功能验证

此场景测试，共运行用例42条，通过的案例为38条，验证了发送给Docker主机进行容器和镜像操作的API，除创建容器外同样可以由Swarm进行处理和接收，并返回相应的执行结果。

失败的用例为4条，通过向Swarm发送创建容器的请求，会提示参数错误而导致发送请求失败，问题记录为Bug 6，影响较大。



### 并发测试

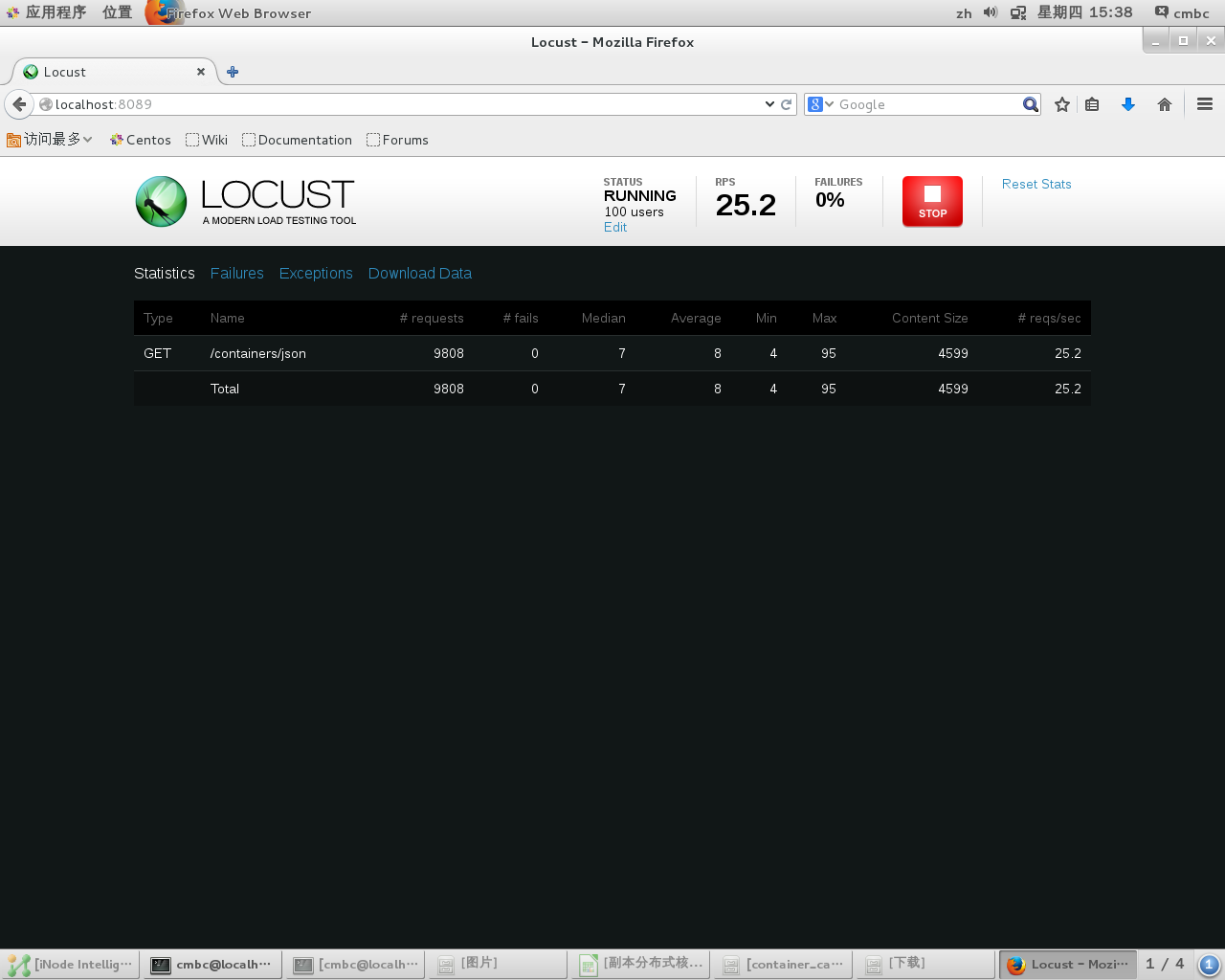
#### 测试原理

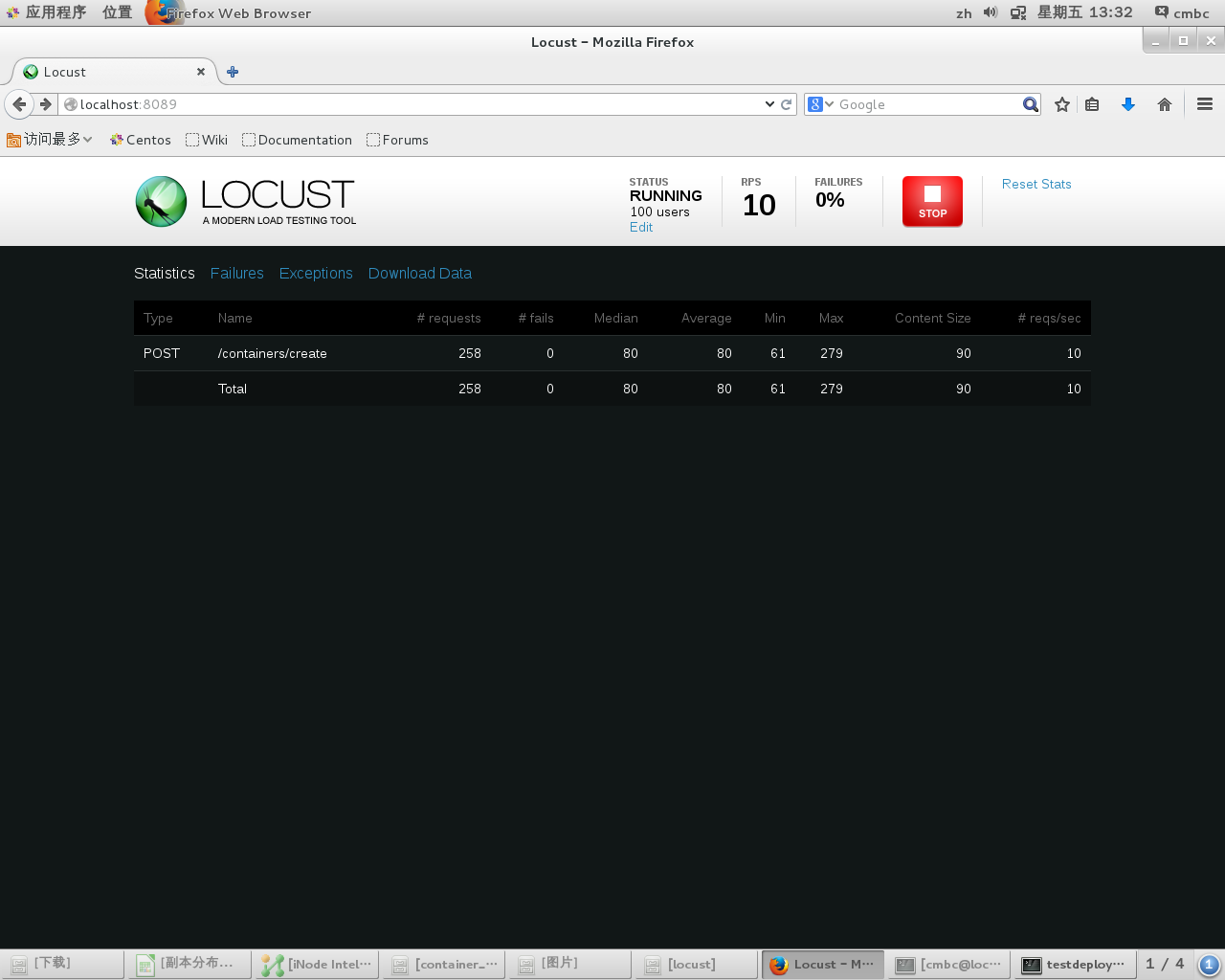
Dcoker Remote API是Docker自带的一个REST API管理。轻量级并发测试工具Locust，可模拟多用户并发发送RESTful请求。通过编辑Locust启动脚本，根据Docker Remote API的URL格式，编写Locust可发送的容器操作 (创建、开启、关闭、删除、获取日志、获取容器列表)脚本，由Locust模拟10/20/50/100个用户并发对Docker主机进行容器基本操作和镜像基本操作的请求。通过获得请求的RPS(每秒处理请求数)和请求失败率，评估Docker主机的并发处理能力。

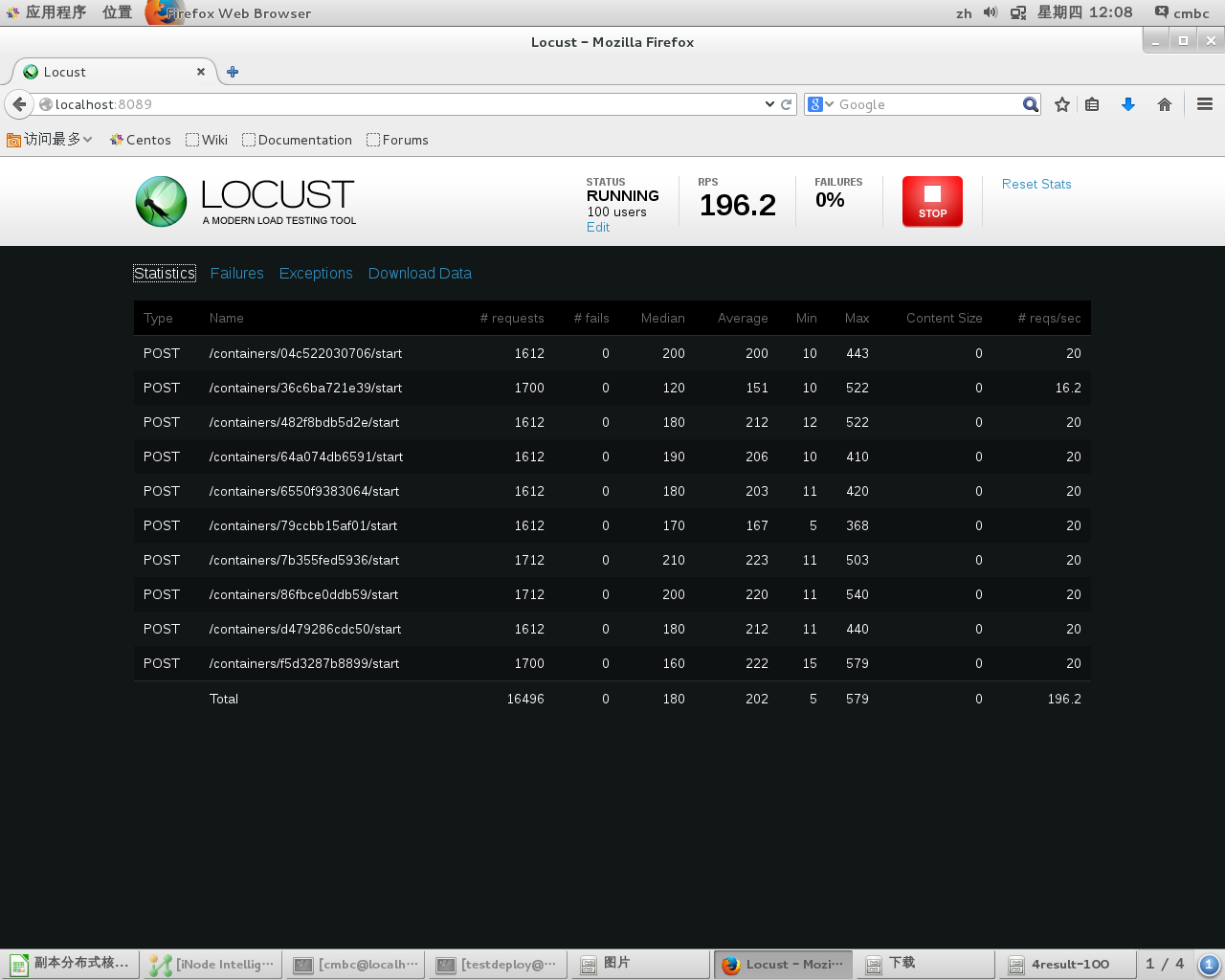
#### 测试场景及结果

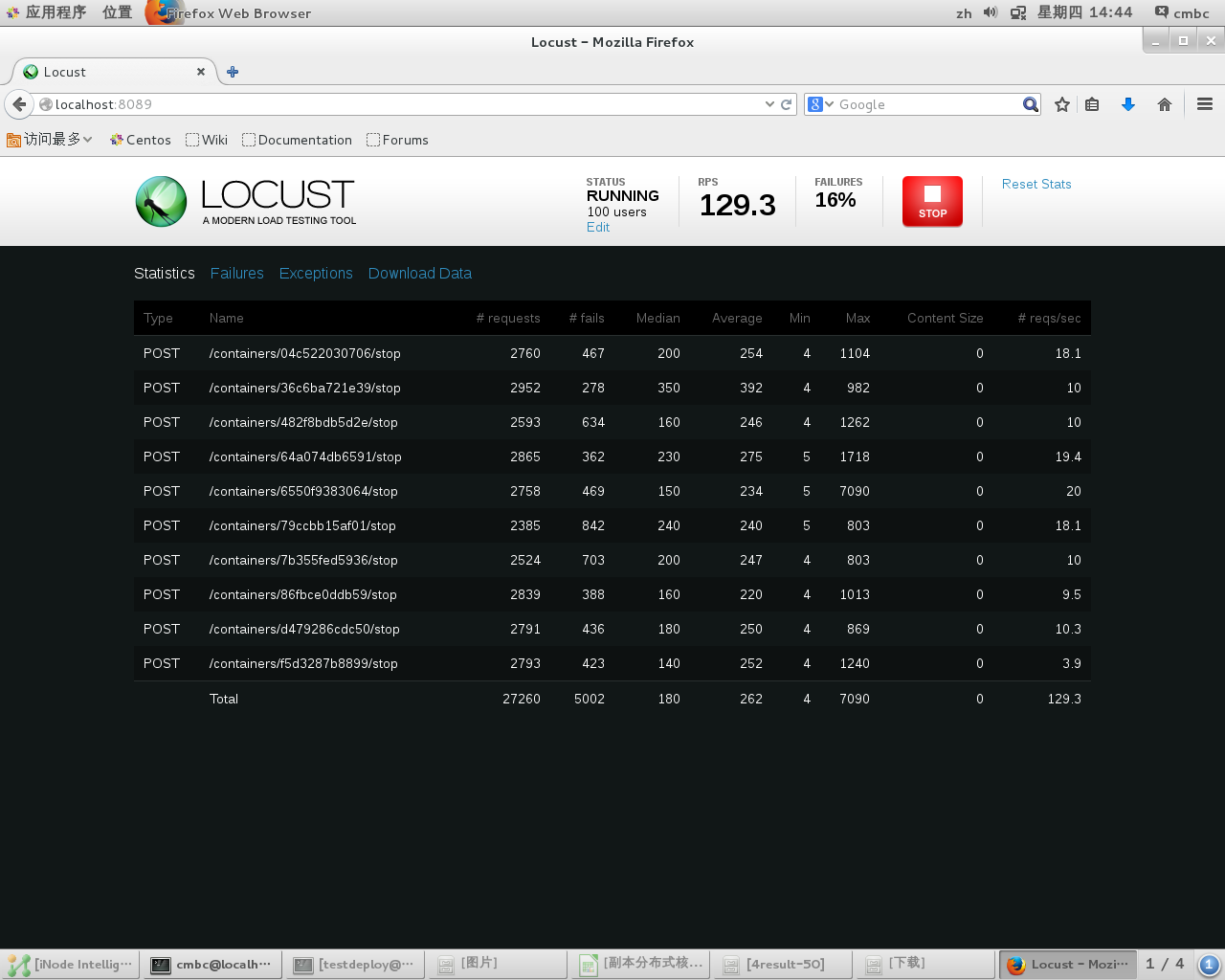
1. 单个容器和镜像操作

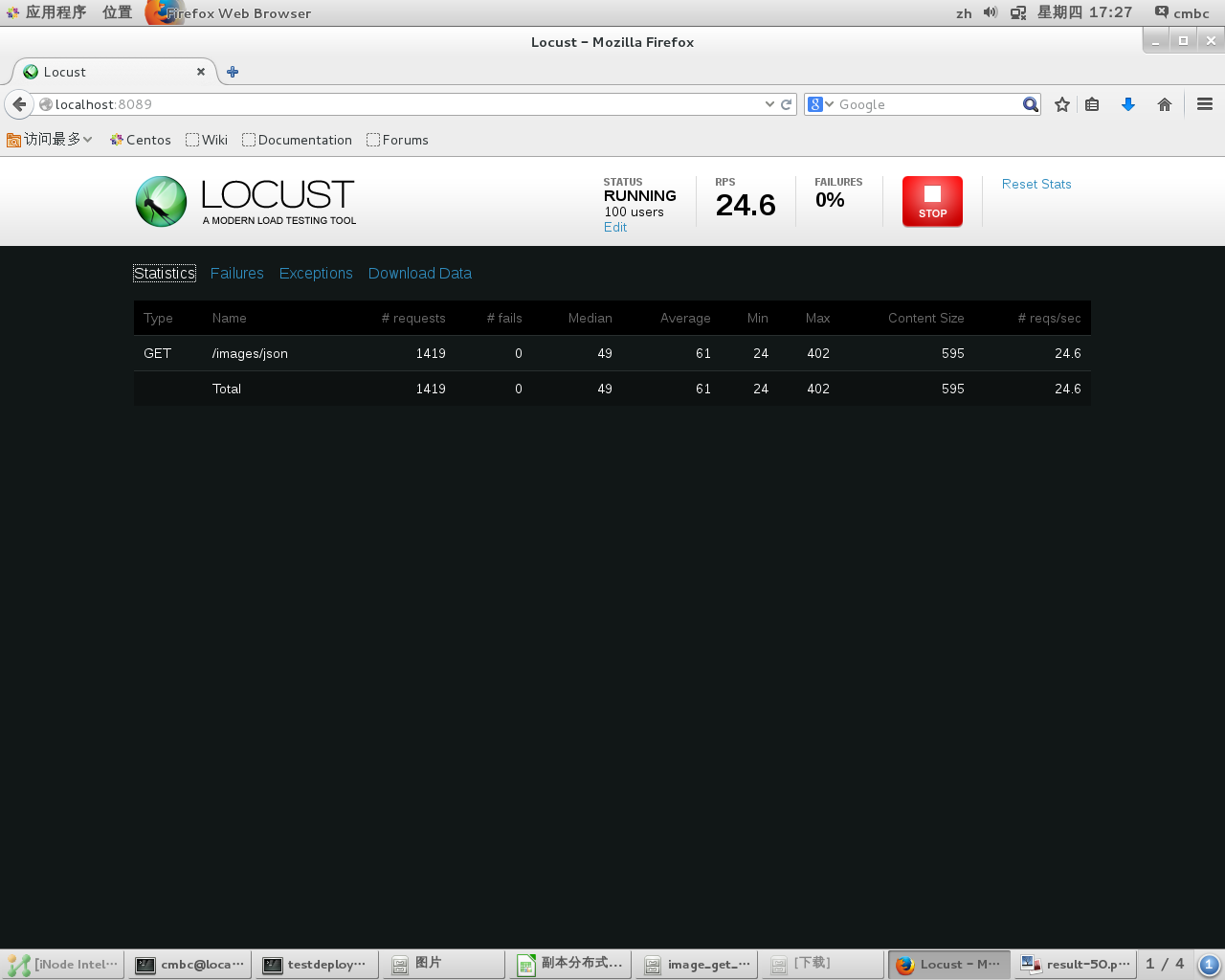
|  |  |  |  |
| --- | --- | --- | --- |
| **测试场景** | **并发量** | **错误率** | **每秒处理请求数** |
| 每秒并发10/20/50/100个用户创建容器 | 10 | 0 | 1 |
| 20 | 0 | 2 |
| 50 | 0 | 4.9 |
| 100 | 0 | 10 |
| 每秒并发10/20/50/100个用户开启容器 | 10 | 3% | 21 |
| 20 | 7% | 40 |
| 50 | 7% | 100 |
| 100 | 16% | 196 |
| 每秒并发10/20/50/100个用户关闭容器 | 10 | 0 | 10 |
| 20 | 0 | 40 |
| 50 | 0 | 65.1 |
| 100 | 0 | 129.3 |
| 每秒并发10/20/50/100个用户获取容器列表 | 10 | 0 | 2.5 |
| 20 | 0 | 5 |
| 50 | 0 | 12.1 |
| 100 | 0 | 25.2 |
| 每秒并发10/20/50/100个用户获取容器日志 | 10 | 0 | 15.4 |
| 20 | 0 | 13.8 |
| 50 | 0 | 13.8 |
| 100 | 0 | 13.9 |
| 每秒并发10/20/50/100个用户获取镜像列表 | 10 | 0 | 2.6 |
| 20 | 0 | 4.8 |
| 50 | 0 | 12.7 |
| 100 | 0 | 24.6 |





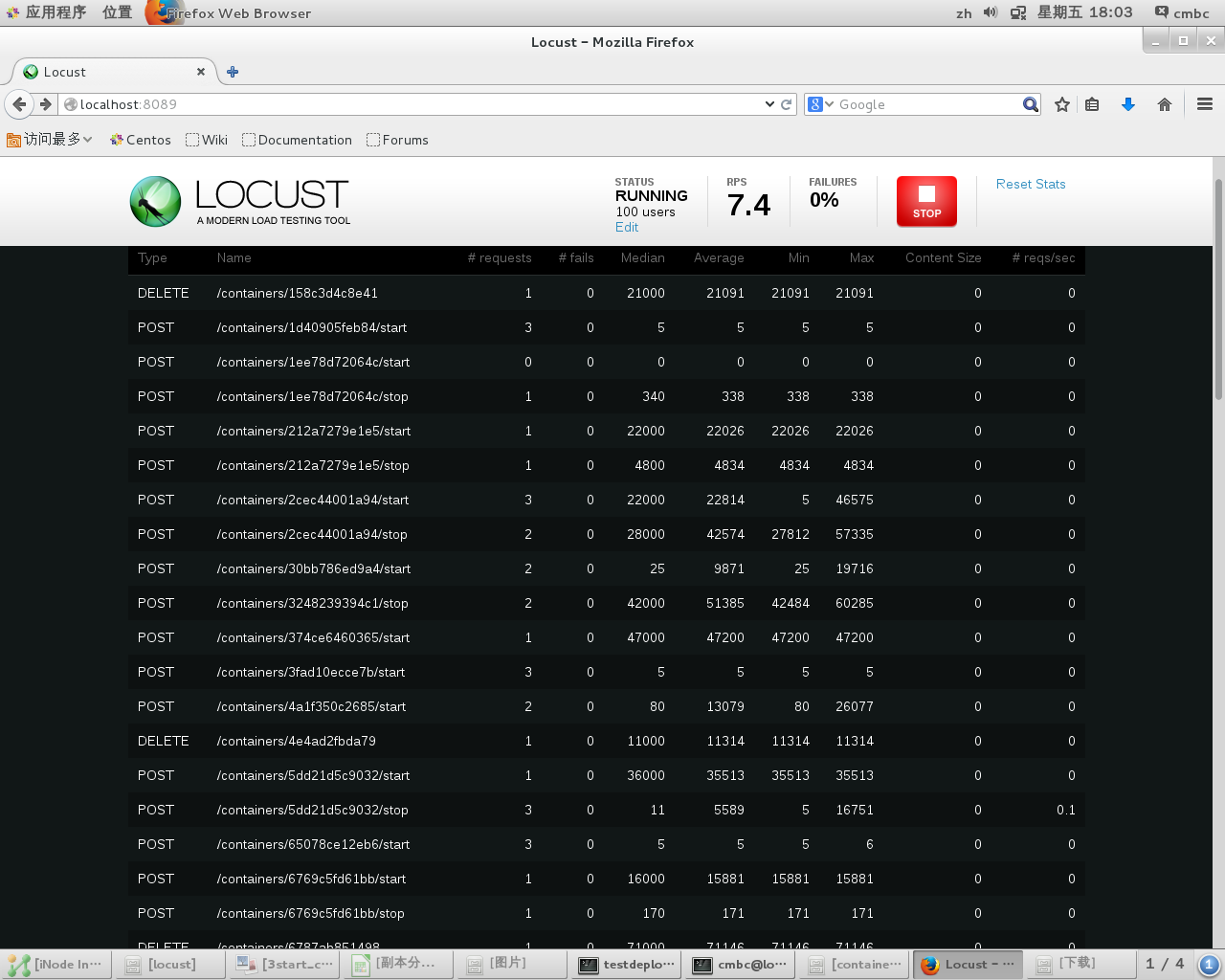






1. 模拟真实容器和镜像操作场景

|  |  |  |  |
| --- | --- | --- | --- |
| **测试场景** | **并发量** | **错误率** | **每秒处理请求数** |
| 每秒并发10/20/50/100个用户创建/开启/关闭/删除/列出容器(权重10:5:2:1:20) | 10 | 0 | 0.7 |
| 20 | 0 | 0.3 |
| 50 | 0 | 3.7 |
| 100 | 0 | 7.4 |



#### 测试结果分析

在10-100并发用户操作容器的情况下，RPS(每秒处理请求数)随并发量的增加而增加。且测试过程中CPU和内存资源并没有明显的增加。

获取容器日志的请求处理数并不受并发量的增加而增加，从结果判断出平均响应时间随着并发量的增加而递减。

在启动容器过程中，随着并发量的提高，会出现关闭容器的请求失败的情况，已记录为Bug 1，影响为中等。

在删除容器的过程中，做创建容器的操作过程非常缓慢，与容器本身秒级创建的特性有出入，已记录为Bug 2。

### 压力测试

#### 仓库负载测试场景

1. Docker主机push镜像至仓库，利用CPU和网络监测脚本，监测仓库主机网络IO和系统的负载情况。
2. 多台Docker主机从同仓库服务器pull镜像，利用CPU和网络监测脚本，监测仓库主机网络IO和系统的负载情况，并监测镜像获取机制。

#### CPU压力测试场景

1. 生成未绑定CPU的容器，在容器中使用nstress工具耗损CPU资源，监测并记录docker服务和其他容器运行状态
2. 生成绑定特定CPU的容器，在容器中使用nstress工具耗损CPU资源，监测并记录docker服务和其他绑定不同CPU的容器运行状态
3. 开启cpu-1 数量个容器，对容器绑定特定CPU，在容器中发布一个httpd应用，利用nstress耗损CPU资源，利用locust监测httpd访问请求的平均响应时间及每秒处理事务数。

#### 内存压力测试场景

1. 生成一个未限制内存使用的容器，在容器中使用nstress工具耗损内存资源，监测并记录docker服务运行状态和内存耗损情况。
2. 生成一个限制内存使用的容器，在容器中使用nstress工具耗损内存资源，监测并记录docker服务运行情况，内存耗损情况和内存突破最大使用限制时容器的运行情况。
3. 开启cpu-1 数量个容器，每个容器限制为(主机内存-2)/(cpu-1)的内存容量，在容器中发布一个httpd应用，利用nstress耗损内存资源，利用locust监测httpd访问请求的平均响应时间及每秒处理事务数。

#### 磁盘IO性能测试场景

1. 创建一个容器，对容器不进行磁盘IO限制，在容器中利用IOZONE进行读写性能测试。在docker主机中利用IOZONE进行读写性能测试。对比两次测试结果。
2. 创建cpu-1 数量个容器，每个容器限制为(主机内存-2)/(cpu-1)的内存容量，容器绑定不同的CPU，对每个容器进行磁盘IO限制，在容器中利用IOZONE进行读写性能测试。关注每个容器磁盘IO读写性能，及测试过程中docker主机的运行情况。

#### 网络性能测试场景

1. 创建cpu-1 数量个容器，每个容器限制为(主机内存-2)/(cpu-1)的内存容量，在容器中并行利用rsync进行大文件上传和下载测试，监测容器网卡接收/发送，丢包率和网络延时情况。
2. 创建cpu-1 数量个容器，每个容器限制为(主机内存-2)/(cpu-1)的内存容量，在容器中并行利用rsync进行小文件上传和下载测试，监测容器网卡接收/发送，丢包率和网络延时情况。

#### 测试执行过程和结果分析

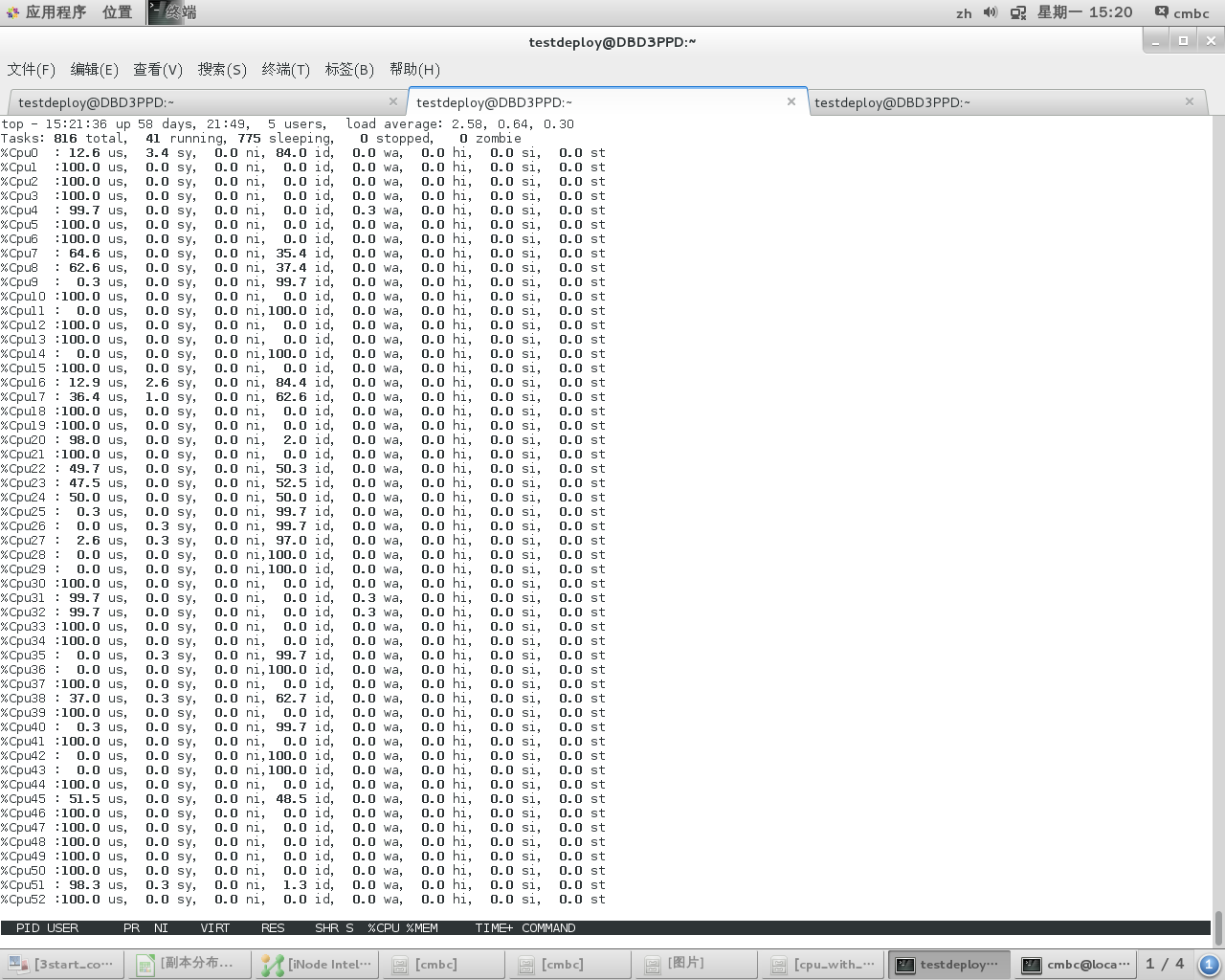
1. 仓库负载测试

Push镜像到仓库中，平均上传速度达到20~30Mbps，且仓库主机的CPU和内存资源没有明显增加，几乎没有负载。

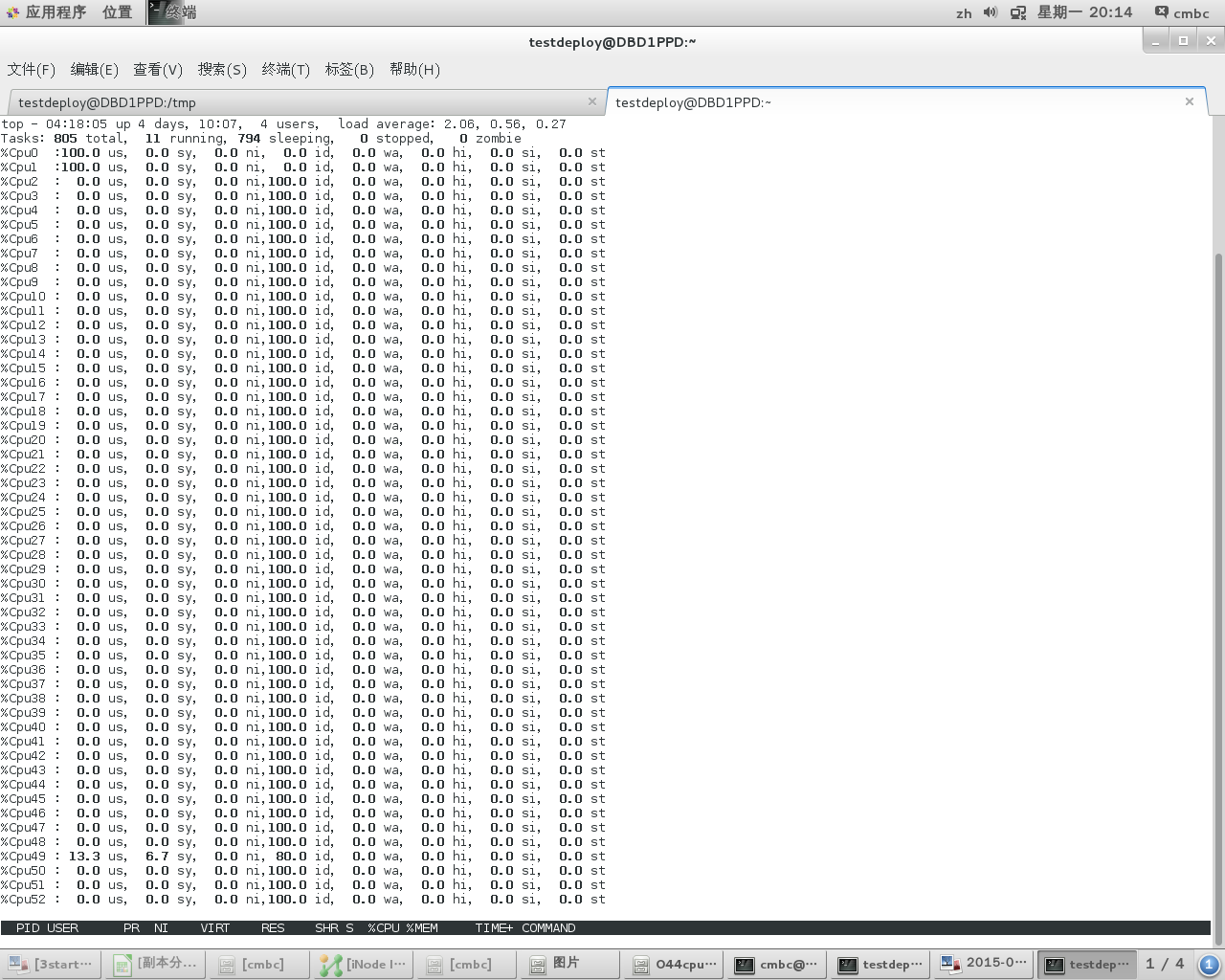
5台Docker主机同时从一台仓库pull同个镜像文件时，每台docker主机的下载速度达到20-30Mbps，且下载镜像的过程为并行下载，Docker主机能够同时从一台仓库中得到镜像，仓库服务器几乎没有负载。

1. CPU压力测试结果

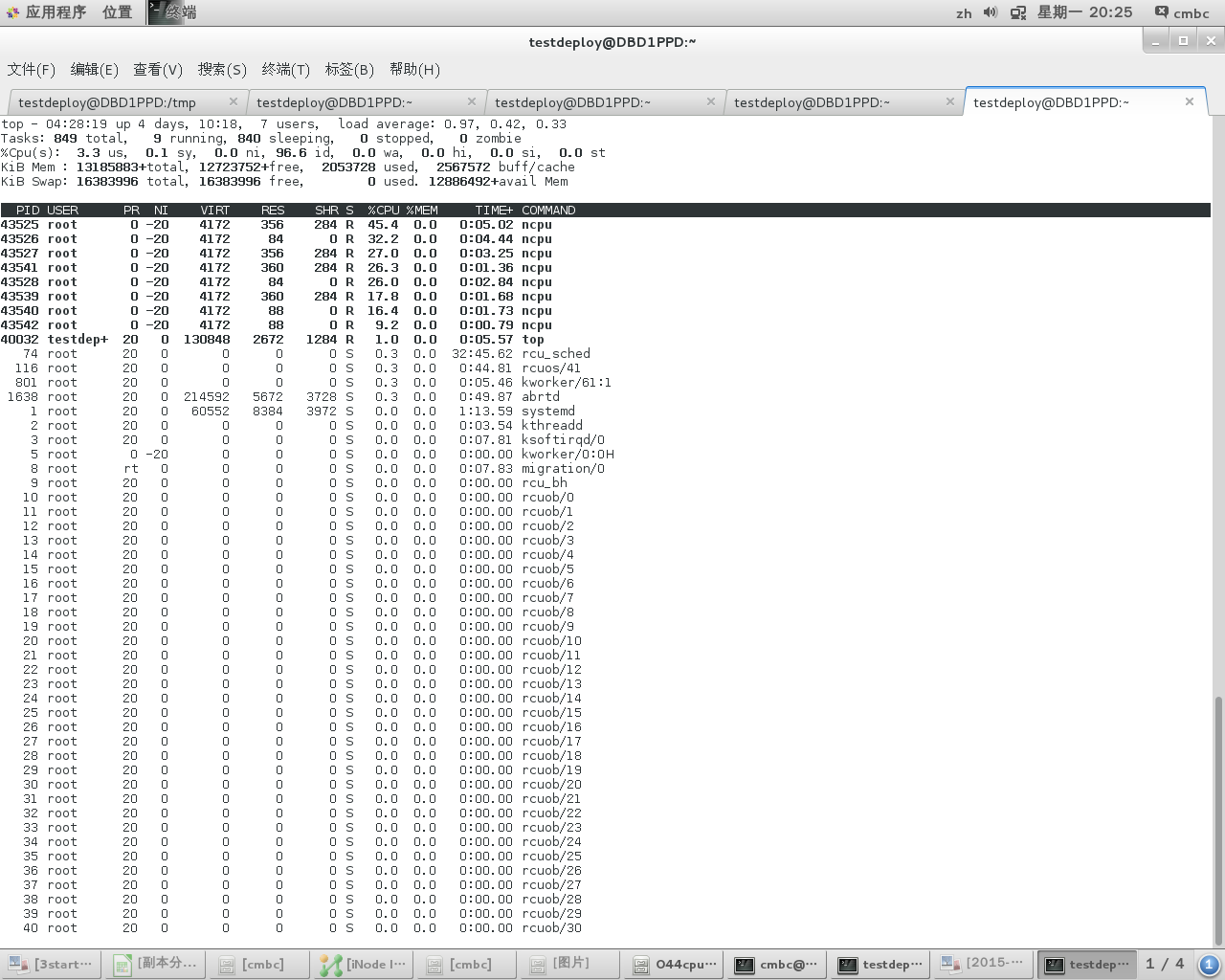
容器未绑定特定CPU情况下，会使用docker主机的所有CPU资源。

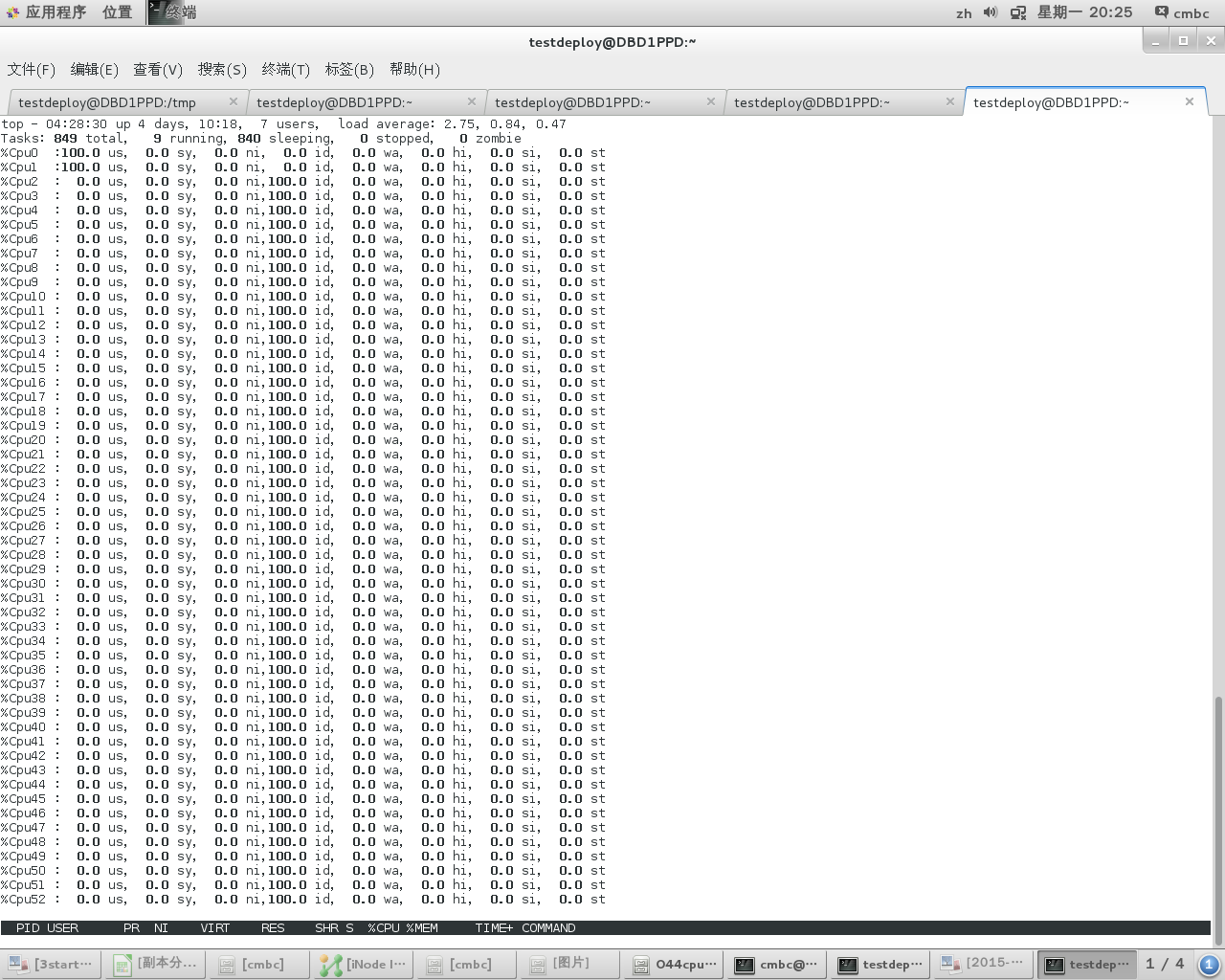


单个容器绑定特定CPU情况下，容器CPU资源的使用会限制在特定CPU上，即使用完也不会占用其他CPU。

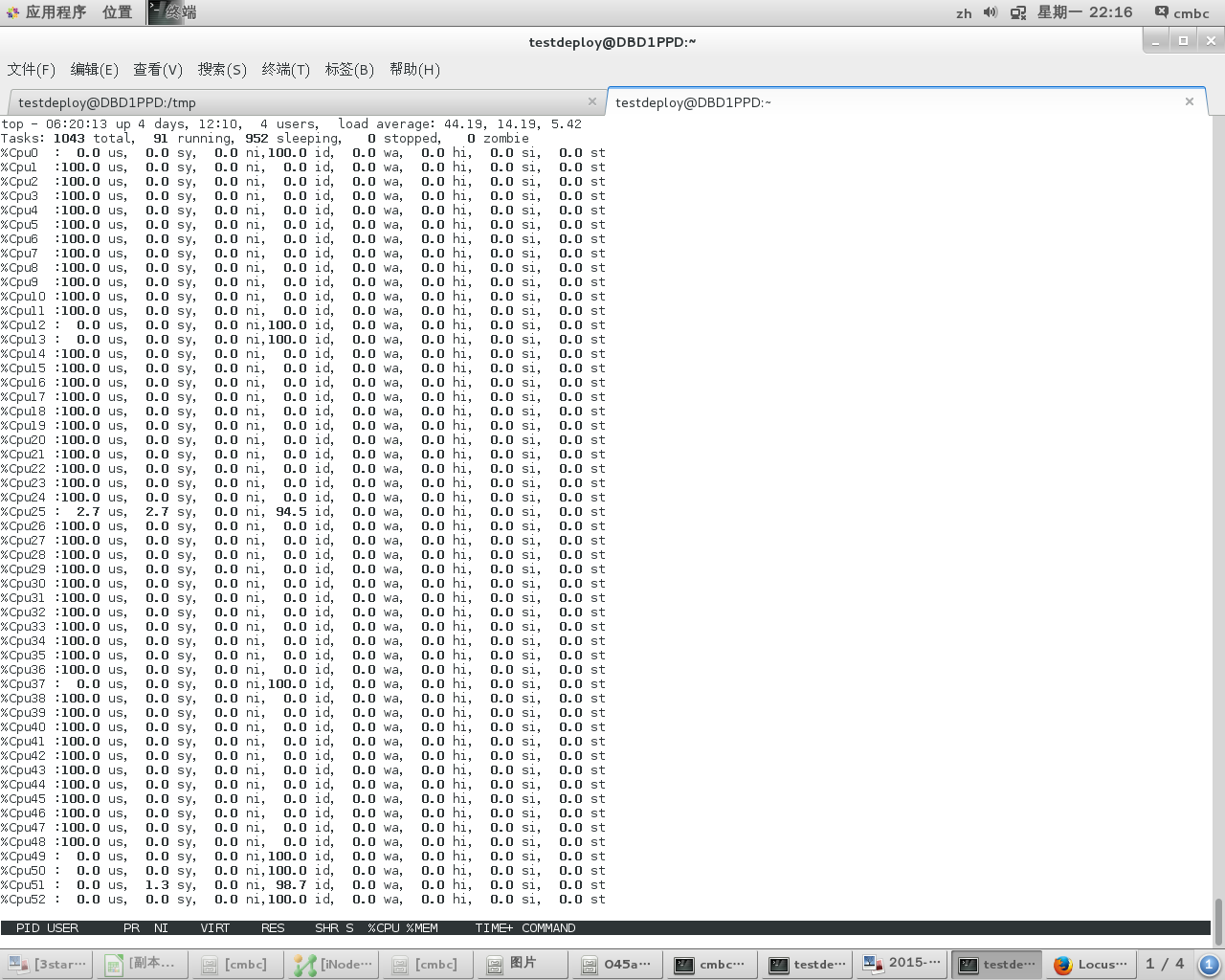


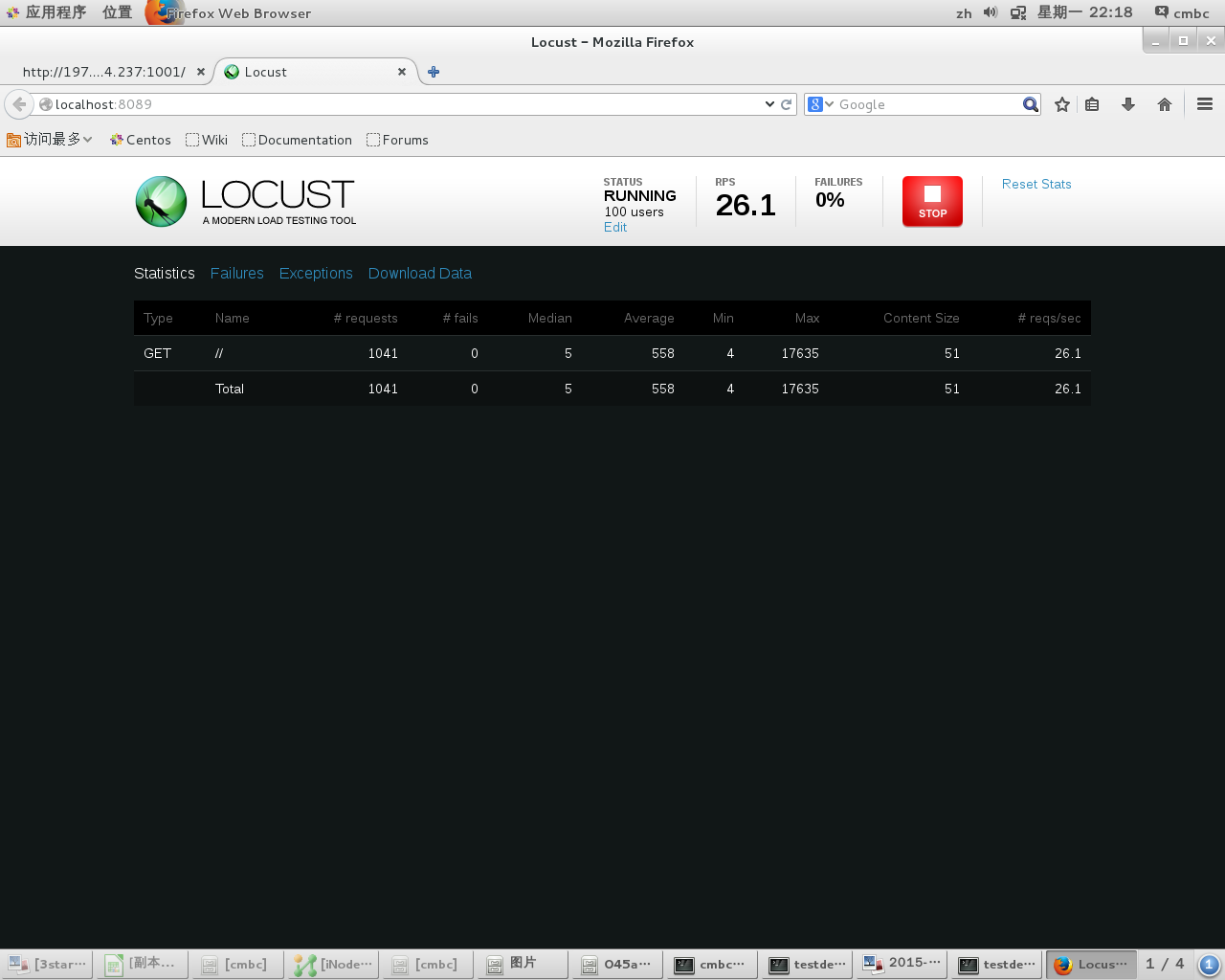
多个容器共享特定CPU，可设置CPU占用比例，单隔离性不是很稳定。





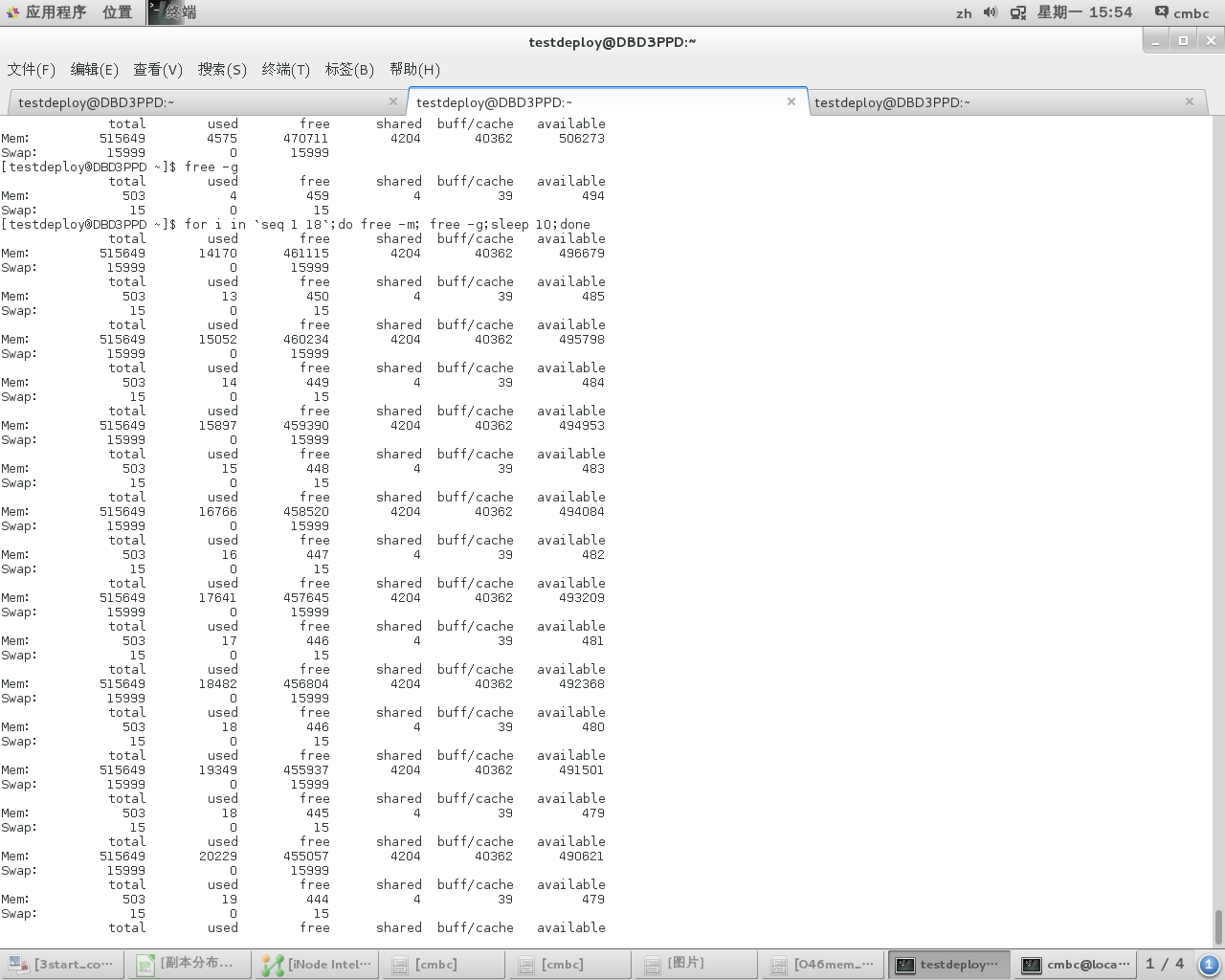
当容器CPU使用率满负荷时，不影响容器中应用的访问性能(测试过程中，100并发访问量场景下，apache的每秒处理请求为26左右，和CPU使用为0的场景下没有差别)。



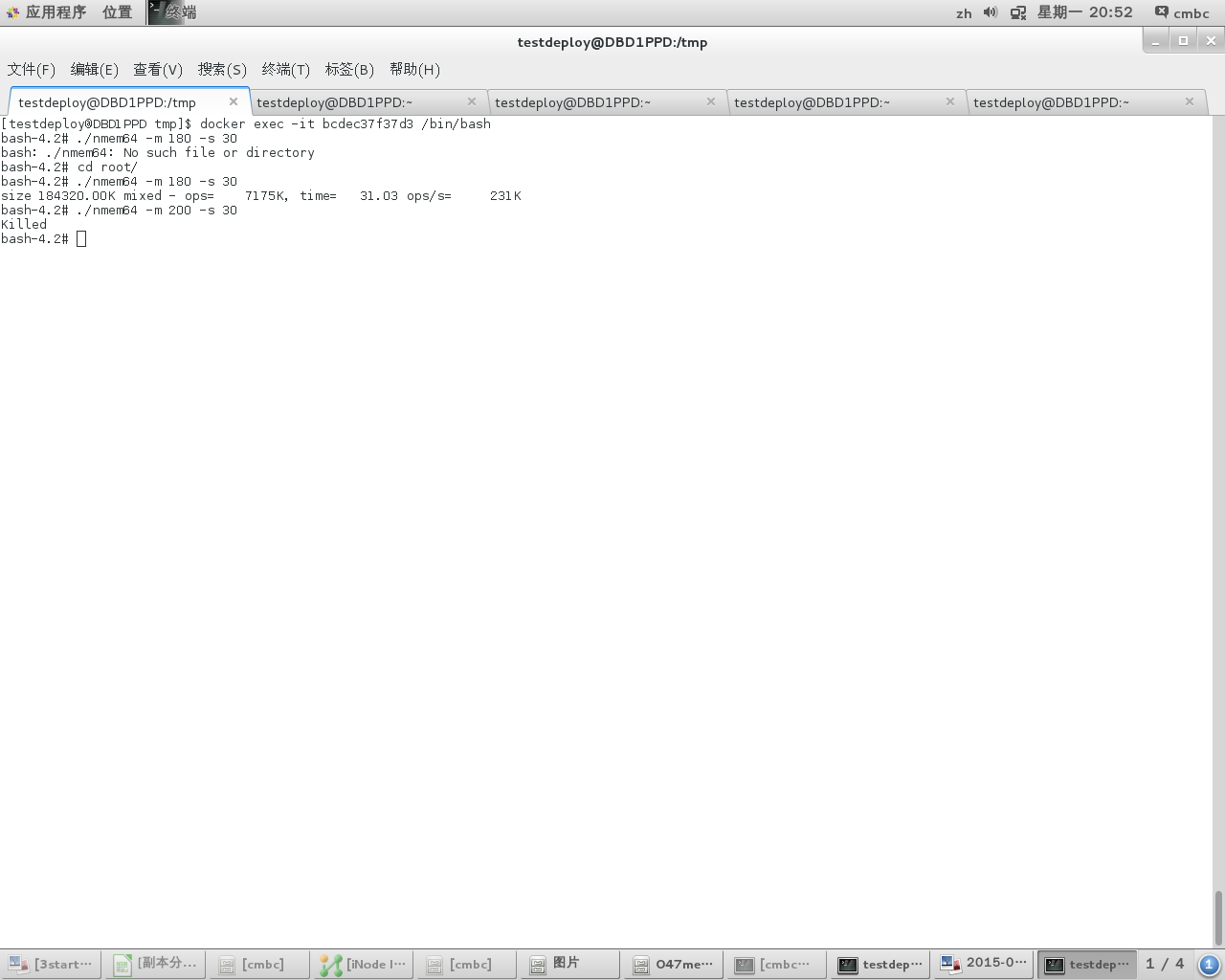


1. 内存压力测试结果

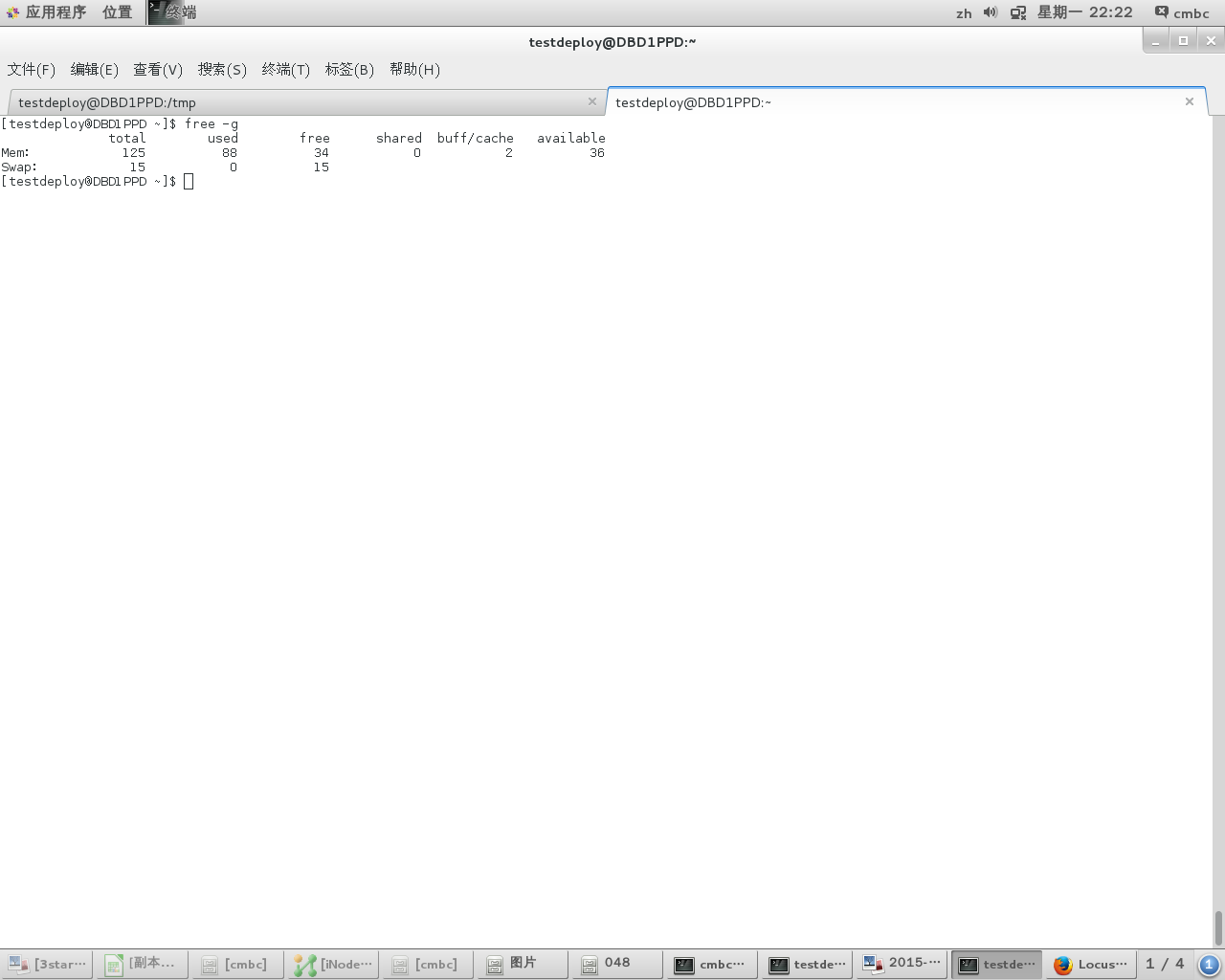
容器未限制使用内存时，会使用docker主机的所有内存资源，对容器进行负载前，docker主机的free内存为459G，随着负载一直增加，3分钟后free内存为435G，且还在不断减少。

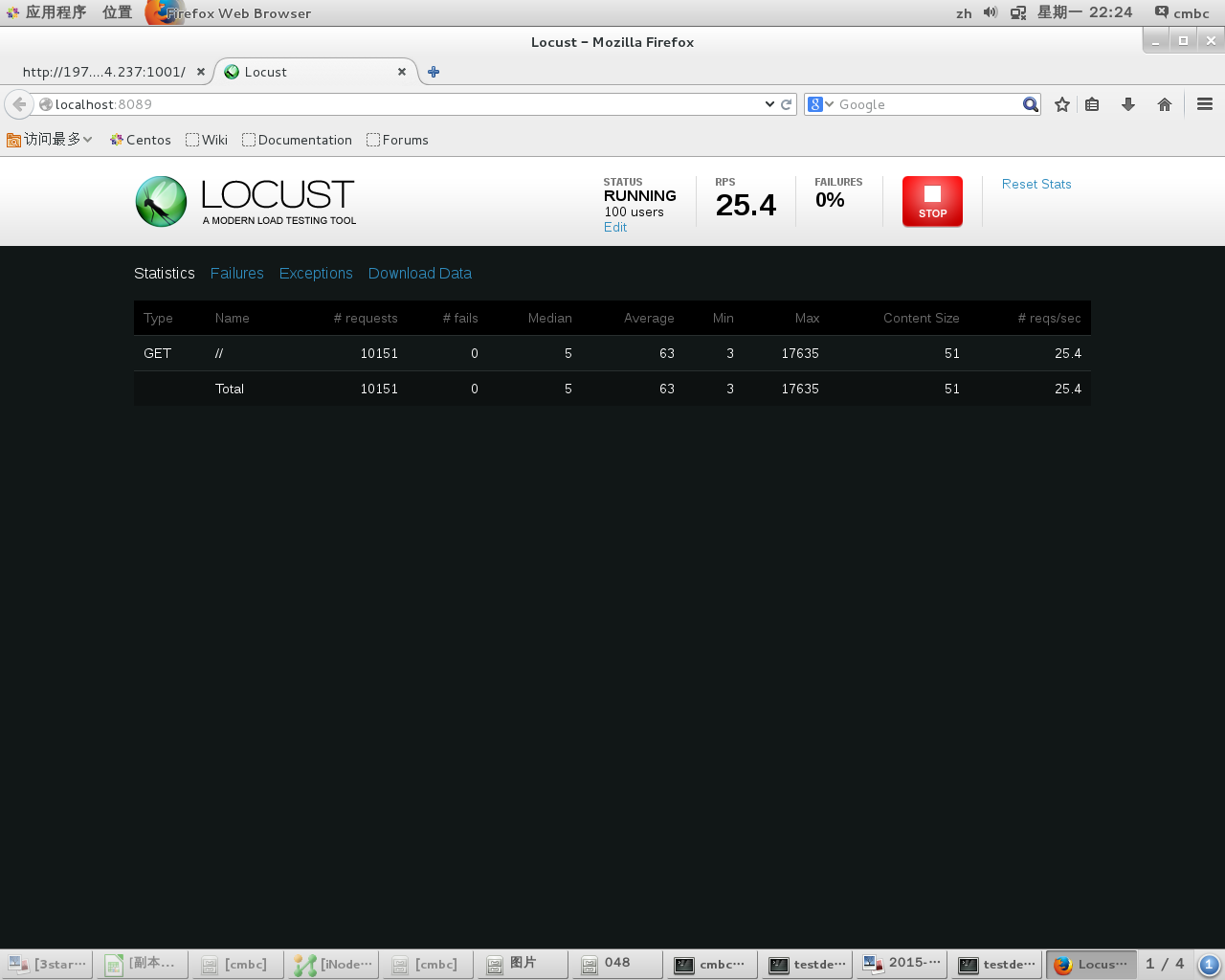


限制容器的使用内存时，当容器中的进程超过限制的内存上限，进程会被杀死，测试过程过程中，限定容器的内存限制为128M，当容器内进程使用率超过200M时，进程被杀死。



当容器内存使用率接近容器使用限制时，不影响容器中应用的访问性能，100并发访问量场景下，apache的每秒处理请求为25左右，和容器中未进行内存负载测试的场景下没有差别。

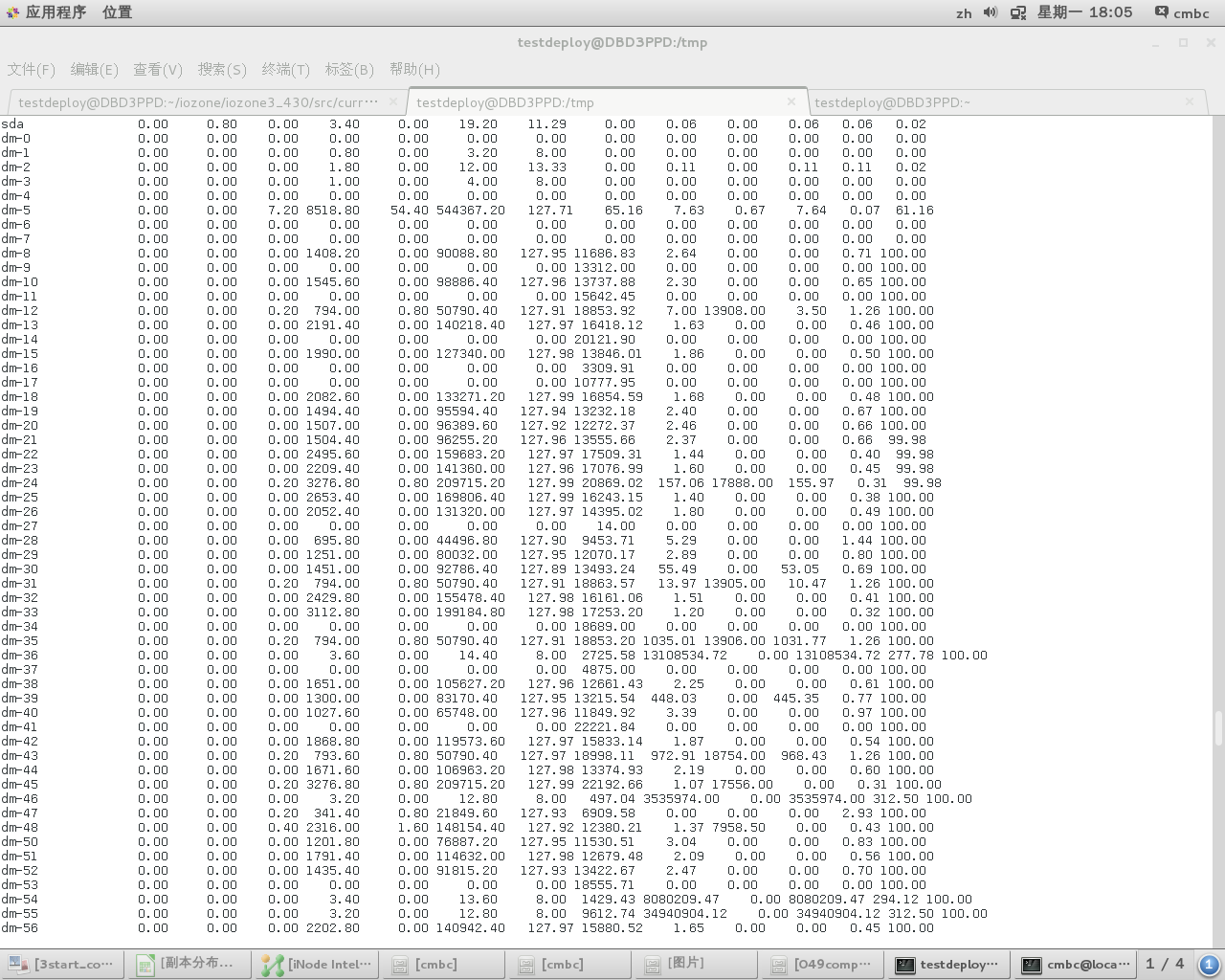


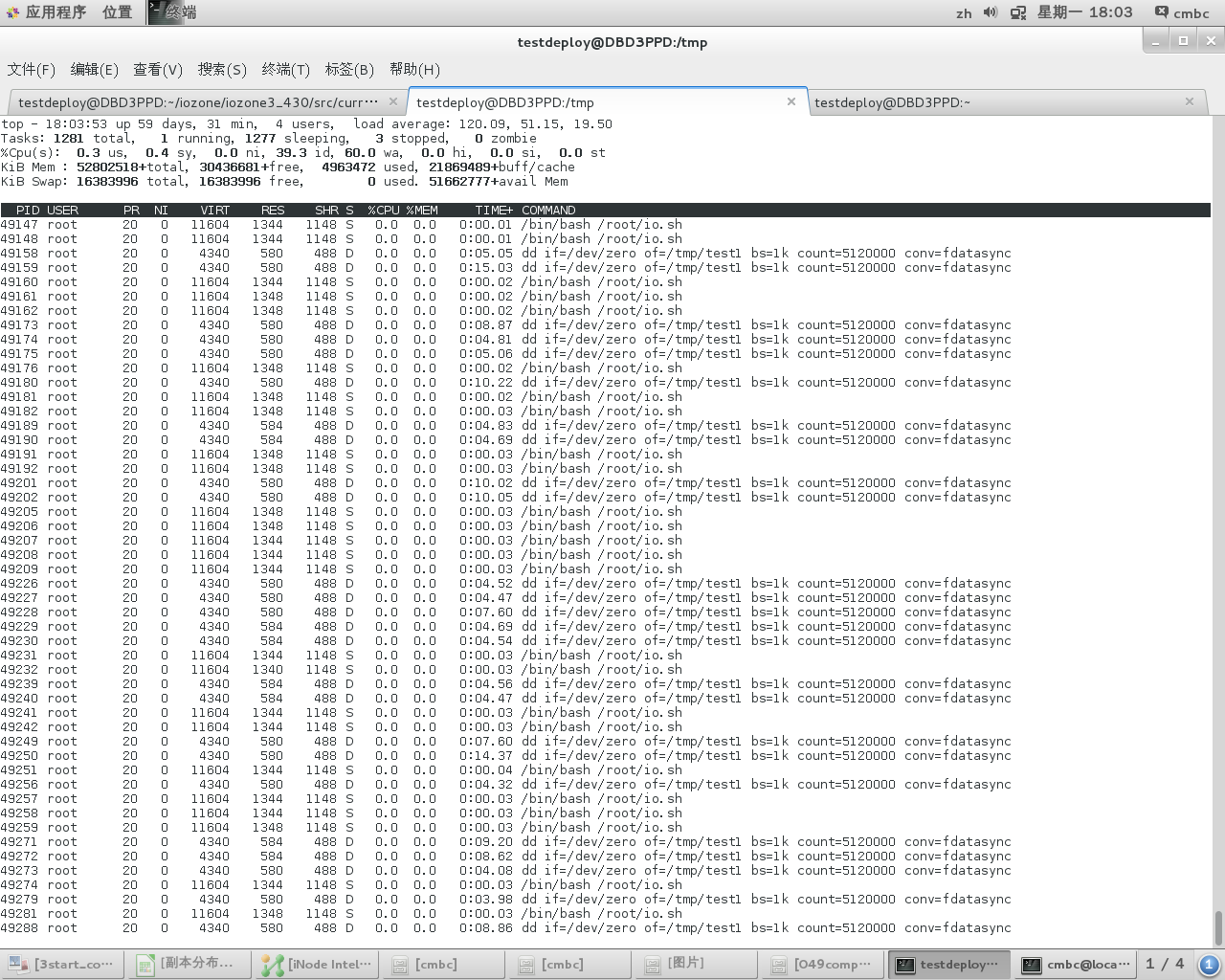


1. 磁盘IO性能测试结果

单个容器的io性能和docker主机的io性能无明显差别，随机读/顺序读/重读的性能在docker主机中和在容器中几乎没有差别(报告中仅列出贴近当前实际应用的随机读/写性能结果，顺序读写参加测试政绩中的数据)。

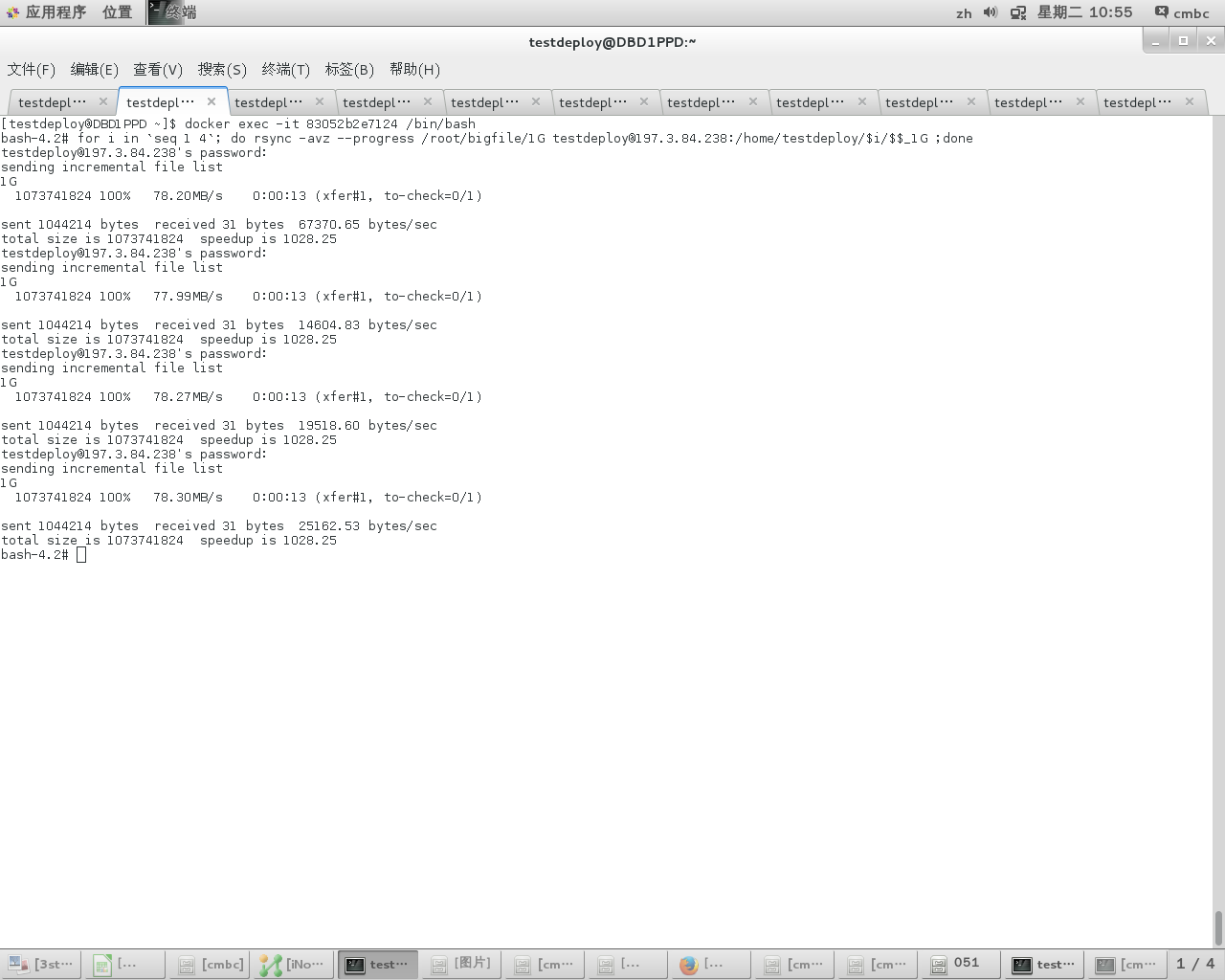
多个容器中进行写操作时，测试过程中遇到了一个较严重的Bug，多个容器并行写大文件时导致容器中的数据占用过多cache，且docker出现无法操作。经过一天的缓冲时间，容器中写入的数据仍然cache在内存中，无法被sync到物理磁盘，此时通过iostat监控io队列，发现iowait和io队列较高，且不会下降。此bug被记录到测试缺陷的bug 3。



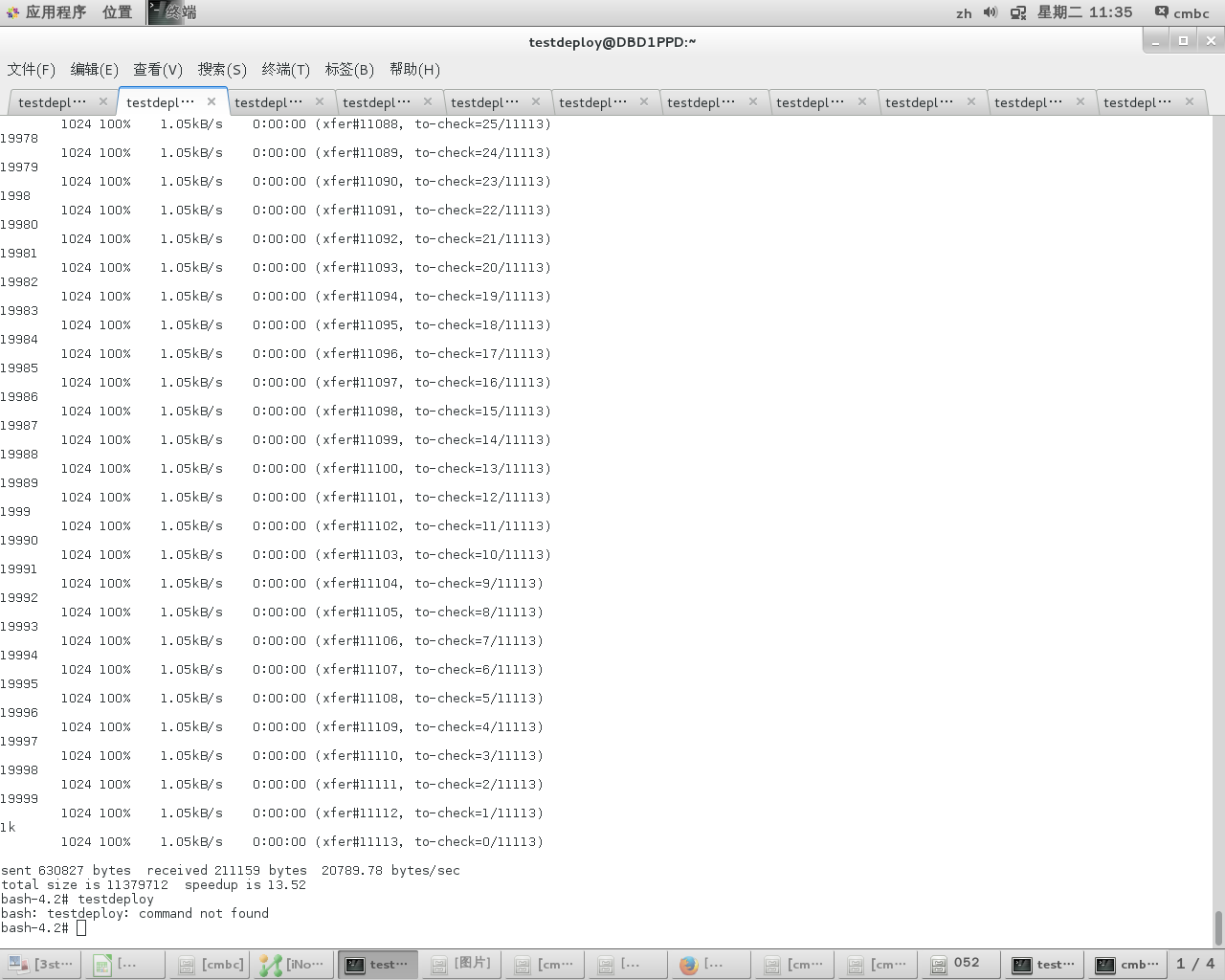


1. 网络性能测试结果

在12个容器中(图示仅列出一个容器的数据信息，其他容器请参考测试政绩)，并行操作大文件拷贝，监控虚拟机网卡的传输速度，本次测试的是容器传输相同文件情况下的传输速度，监控到传输速度达到78MBps，且容器未出现网络延时情况，但遇到容器的虚拟网卡和实际数据传输速度不一致的问题，但实际docker主机的物理网卡传输速度和docker容器虚拟网卡的传输速度一致。



在12个容器中(图示仅列出一个容器的数据信息，其他容器请参考测试政绩)，并行操作小文件拷贝，监控虚拟机网卡的传输速度，本次测试的是容器传输不同文件情况下的传输速度，容器未出现网络延时情况，容器的虚拟网卡和实际数据传输速度一致。



# 结论和建议

通过本次测试，可以了解到Docker容器这项新技术，实现了在相同的旧服务器上运行更多的应用程序，使得它很容易打包和发布程序。我们可以得到相同的硬件上比其他技术运行更多的应用，这些应用仅需要很小的开销内存/ CPU/硬盘，这意味着为企业降低硬件成本和运维成本。还可以使开发人员能够快速创建简单的，现成的运行容器化应用，这使管理和部署应用程序更加容易。

作为一个发布2年左右的新技术，docker容器也存在一些功能和性能上的缺陷，按照目前docker版本的更新速度和越来越多的公司关注这项新技术，利益和风险并存。使用它需要我们投入一定的学习成本，同时紧跟docker开源社区，并能够有一定的源码提供能力，以便加大公司在开源社区的知名度，使得这项技术能够更好的服务于民生银行金融云的需求目标。