



软件测试文档 **Software Test Documentation**  
撰写日期 **Date: 2024-02-15**

# 软件测试文档

## Software Test Documentation

“518 三缺一” 队

谢天祺 231307040028

王冬雨 231307040024

张捷 231307040030

(计算机科学与技术 学硕 2023 级)

实验课程：分布式计算

Distributed Computing

主讲教师：毛莺池 谢在鹏

复现论文：**Lunule: An Agile and Judicious Metadata Load Balancer for CephFS**

学 分 数：2 Credits

开设学院：计算机与软件学院 College of Computers & Software

开设时间：2023~2024 学年第一学期 1<sup>st</sup> Semester, 2023~2024

河 海 大 学

# 目录

<b>1 测试用例</b>	<b>1</b>
1.1 编译安装 Lunule	1
1.2 上传和解压数据集	2
1.3 配置环境并运行测试	3
<b>2 测试场景</b>	<b>4</b>
2.1 负载监视	4
2.2 工作负载感知和迁移	4
<b>3 测试报告</b>	<b>6</b>
3.1 配置实验服务器集群	6
3.2 CephFS 安装	7
3.2.1 服务器集群各节点基本设置	7
3.2.2 安装 Ceph 并配置和创建节点	8
3.3 Lunule 的安装	10
3.3.1 环境配置	10
3.3.2 遇到的问题	10
3.4 软件测试与结果	13
3.4.1 测试程序及其验证	13
3.4.2 数据集上传	13
3.4.3 服务器测试结果	14

根据原论文在相关章节的描述，为验证原文所述的负载均衡器 Lunule 在 CephFS 文件系统中对于多元数据服务器 MDS 间的动态资源负载的平衡和有效迁移作用，需要在创建好的 16 服务器节点集群上安装 CephFS 文件管理服务器，并在 MDS 服务节点中部署该负载均衡器后执行进行测试。综合以上过程，该测试用例包含以下两个过程：

1. 在 CephFS 文件系统的元数据服务器 MDS 上，构建 Lunule 负载均衡器；
2. 在 Client 节点上，运行测试项目

## 1.1 编译安装 Lunule

- 本部分在 MDS1-5 节点执行
- 此处以 MDS1 节点的测试为例

### 步骤 1 安装所需依赖

预期执行结果为全部安装成功无报错。全部执行完毕后执行“python3.4 --version”指令应输出版本号“Python 3.4.x”（x 对应一个数字，与 Cython 的版本相关）。

```
[root@MDS1 ~]# yum install wget libiconv curl curl-devel expat expat-devel gettext-devel openssl  
openssl-devel zlib zlib-devel bzip2-devel.x86_64 git-core -y  
[root@MDS1 ~]# wget ftp://ftp.pbone.net/mirror/archive.fedoraproject.org/epel/7.2020-04-20/x86_64  
/Packages/p/python34-Cython-0.28.5-1.el7.x86_64.rpm  
[root@MDS1 ~]# yum install -y python34-Cython-0.28.5-1.el7.x86_64.rpm
```

### 步骤 2 克隆 lunule 项目到服务器节点

预期执行结果为克隆完成，且执行“ls”的输出应包含“lunule”文件夹。

```
[root@MDS1 ~]# git clone https://github.com/mbal-lunule/lunule.git
```

### 步骤 3 进入项目目录，并执行编译准备脚本

预期执行结果为脚本运行并正常退出，执行“ls”的输出应出现“build”文件夹

```
[root@MDS1 ~]# cd ./lunule  
[root@MDS1 lunule]# ./install-deps.sh  
[root@MDS1 lunule]# ./do_cmake.sh
```

### 步骤 4 编译 Lunule 源码，并重启 MDS 服务

预期执行结果为“make”和“make install”正常执行并退出，执行“ceph mds stat”应包含“5up:standby”即 5 个 MDS 节点全部在线。

```
[root@MDS1 build]# cd ./build
```

```
[root@MDS1 build]# make -j 16
[root@MDS1 build]# make install -j 16
[root@MDS1 build]# systemctl restart ceph-mds.target
```

## 1.2 上传和解压数据集

步骤 1 从以下页面下载 ILSVRC2012 数据集

包含开发套件（Development Kit）和图像（Image）共约 150GB，下载页面为：

```
https://image-net.org/challenges/LSVRC/2012/2012-downloads.php
```

Development kit for Task1 & Task2:

```
https://image-net.org/data/ILSVRC/2012/ILSVRC2012_devkit_t12.tar.gz
```

Development kit for Task3:

```
https://image-net.org/data/ILSVRC/2012/ILSVRC2012_devkit_t3.tar.gz
```

Task1 和 2 的训练数据集(138GB):

```
https://image-net.org/data/ILSVRC/2012/ILSVRC2012_img_train.tar
```

Task3 的训练数据集(728MB):

```
https://image-net.org/data/ILSVRC/2012/ILSVRC2012_img_train_t3.tar
```

验证数据集(6.3GB):

```
https://image-net.org/data/ILSVRC/2012/ILSVRC2012_img_val.tar
```

测试数据集(13GB):

```
https://image-net.org/data/ILSVRC/2012/ILSVRC2012_img_test_v10102019.tar
```

步骤 2 上传数据集到 CephFS 客户端节点

方法一：使用 WinSCP 等软件，将本地文件上传到服务器；

方法二：使用 wget 等下载指令在服务器下载步骤 1 所述文件；

预期执行结果为在上传目录执行“ls”指令，输出结果应包含上传的所有文件。

步骤 3 在客户端解压文件

(1) 解压测试集 ILSVRC2012\_img\_test

测试集的压缩包包含所有测试图片，解压过程中应输出图像文件的名称，即测试集内的图片的名称。解压完成后，在目标文件夹内执行“ls”应输出全部图像的文件名。

```
[root@OSD10 Dataset]# mkdir ./ILSVRC2012_img_test
[root@OSD10 Dataset]# tar -xvf ./ILSVRC2012_img_test_v10102019.tar -C ./ILSVRC2012_img_test
```

(2) 解压验证集 ILSVRC2012\_val\_test

验证集的压缩包包含了所有的验证图片，解压过程中应输出图像文件的名称，即验证集的图片名称。解压完成后，在目标文件夹内执行“ls”应输出全部图像的文件名。

```
[root@OSD10 Dataset]# mkdir ./ILSVRC2012_img_val
[root@OSD10 Dataset]# tar -xvf ./ILSVRC2012_img_val.tar -C ./ILSVRC2012_img_val
```

### (3) 解压训练集

训练集的压缩包包含了所有的训练用的图像,但是这些图像随机组合形成多个训练批次,因此解压缩后是一个批次的训练用图像的压缩文件包。因此在解压过程中输出的是压缩包的名称,解压完成后在目标文件夹内执行“ls”的输出为全部的压缩包文件名。

```
[root@OSD10 Dataset]# mkdir ./ILSVRC2012_img_train
[root@OSD10 Dataset]# tar -xvf ./ILSVRC2012_img_train.tar -C ./ILSVRC2012_img_train
```

执行 unzip.py 程序对 train 目录下的文件进行进一步的解压,在执行过程中应输出图像名称。

```
[root@OSD10 Dataset]# cat > "unzip.py" << EOF

import glob
import os

filelist = glob.glob("./ILSVRC2012_img_train/*.tar")

for f in filelist:
    os.system("mkdir ." + f.split(".")[1])
    os.system("tar -xvf " + f + " -C ." + f.split('.')[1])
    os.system("rm " + f)

EOF

[root@OSD10 Dataset]# python3 unzip.py
```

## 1.3 配置环境并运行测试

➤ 本部分在 CephF 的客户端 OSD10 节点执行

### 步骤 1 安装测试依赖

```
[root@OSD10 ~]# python3 -m pip install scikit-build contextvars numpy mxnet opencv-python
```

### 步骤 2 克隆测试程序 MXNet 项目

```
[root@OSD10 ~]# git clone https://github.com/apache/mxnet.git
```

### 步骤 3 执行测试程序 img2rec.py

本部分需要根据上一步骤上传的数据集的具体位置,对最终执行的代码进行一定的调整,如数据集所处的文件夹目录等。

```
[root@OSD10 ~]# mkdir {your_ceph_client_path}/record
[root@OSD10 ~]# python3 ./mxnet/tools/img2rec.py --list --recursive {your_ceph_client_path}/record
{your_ceph_client_path}/Dataset
```

例如:

```
[root@OSD10 ~]# python3 ./mxnet/tools/img2rec.py --list --recursive /mnt/ILSVRC2012/record
/mnt/ILSVRC2012/Dataset
```

根据作者在论文《Lunule: An Agile and Judicious Metadata Load Balancer for CephFS》中描述，该 Lunule 负载均衡器程序应该具备三个特色功能：进行负载监视、感知工作负载变化、规划工作负载迁移。

## 2.1 负载监视器

根据 Lunule 体系分析结果和源代码分析，Lunule 通过在每个 MDS 上部署一个负载监视器（Load Monitor）以收集相应 MDS 每秒处理的元数据请求的数量来监视负载压力。该部分内容在 `src/mon/MDSMonitor.cc` 中实现，通过 `update_metadata`，`remove_from_metadata`，`load_metadata`，`count_metadata`，`dump_metadata` 五个函数共同完成对元数据的更新，撤销，载入，计算，挂载五个功能请求处理。

其中 `count_metadata` 函数存在两个不同的输入用来计算，分别是对应子树分片和目录分片，两者相互转化能够成功收集 MDS 的元数据请求，保证了 Lunule 能够监视负载压力。

## 2.2 工作负载感知和迁移器

另一个关键组件是感知工作负载的负载感知迁移计划器（Migration Planner），它也部署在每个 MDS 上独立运行，其功能主要在 `src/mon/MgrMonitor.cc` 文件中实现。文件中它主要依靠 `load_metadata`，`count_metadata`，`dump_metadata` 三个函数对元数据进行加载、计算、挂载等基本操作。之后在将其进一步划分成两部分，第一个是模式分析器（Pattern Analyzer），它学习不同工作负载的 I/O 模式，集中在相应 MDS 管理的子树上，并通过理解过去的工作负载的影响和预测未来元数据访问的差异来计算迁移的概率。第二个是，当触发重新平衡时，在输出的 MDS 上，有一个子树选择器（Subtree Selector）选择一组适当的子树进行迁移，该子树满足迁移启动器计算的数量，并适合目标工作负载的未来访问。第三个组件是在集群的 MDS 上有一个迁移启动器（Migration Initiator），用于实时决定何时应该进行迁移，以及应该在元数据服务器之间交换多少元数据等信息，这主要在 `src/mon/MgrStatMonitor.cc` 文件中实现，其中不仅对其进行了初始化等静态配置，同时对整个环境进行了监控，能够检测到服务器之间的元数据交流。所做的决策依赖于 IF 模型的值，此值汇总了 MDS 之间的负载分布，从负载监视器其中收集的统计信息根据 IF 分析模型计算得到。虽然迁移启动器是一个集中式组件，但它不会成为性能瓶颈，因为迁移过程发生在后台，每次 epoch 都运行，并且消耗很少的资源，这在 `src/tools/rbd_recover_tool/epoch_h` 中实现，默认情况下 epoch 是 10 秒钟，其中 Lunule 在一个 Epoch 的固定时间内做出迁移决定。在每个 epoch 中，每个负

载监视器将观察到的元数据吞吐量数发送给迁移启动器，当元数据集群的不平衡度，即 IF 值超过预定义的阈值时，迁移启动器触发负载重新平衡过程，并生成迁移计划，将输出（exporter）和输入（importer）的角色分配给 MDS，并将输出和输入的需求和能力进行配对。之后工作流感知迁移计划器将通知每个输出 MDS 上分配的迁移任务。在迁移任务的到达时，子树选择器选择合适的子树分区列表，然后将其输入到现有的迁移器中，从较重负载的 MDS 重新定位到负载较轻的 MDS。

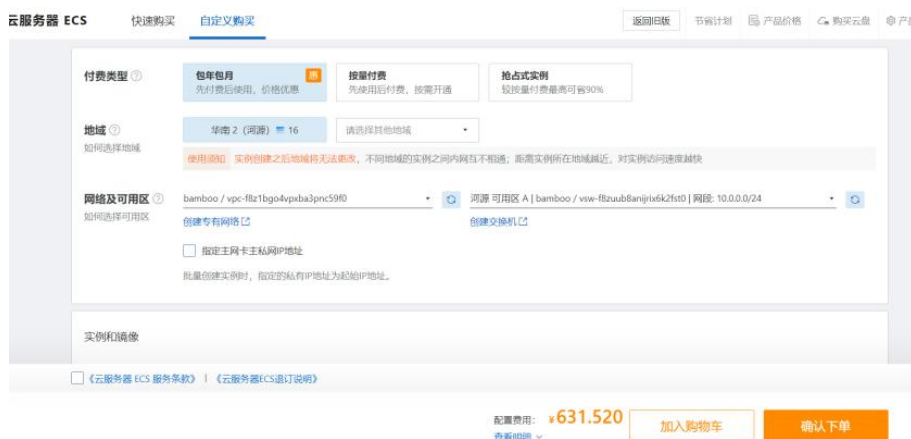
# 3

## 测试报告

本部分包含 CephFS 的安装和第 1 章节所述的测试用例的实际执行结果。在执行过程中，由于资源限制和网络限制，主要的问题存在与运行环境上。本章节将按照测试用例的相关内容，在设置的服务器集群上进行环境配置和测试，对测试中出现的包括环境配置问题和代码问题进行详细的描述并尝试提出一些可供参考的解决思路。

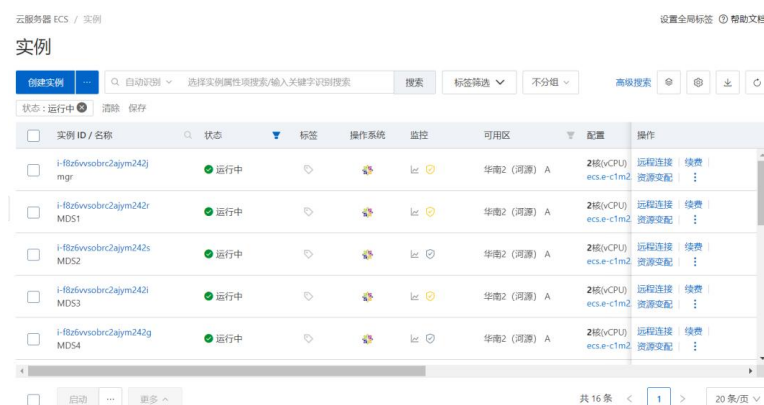
### 3.1 配置实验服务器集群

根据原论文在相关章节的描述，为了完成论文所述的实验需要使用 16 个服务器。由于我们没有服务器资源，结合实际需求和各公有云资源在价格、优惠情况等多因素考量，最终选择在阿里云上购买共计 16 个 ecs.c7.2xlarge 的云服务器 ECS，包含 2 个虚拟核心、4GB 内存。其中 1 个服务器作为管理员 mgr 节点并安装 CephFS，5 个服务器作为元数据存储 MDS 节点存储元数据，9 个服务器作为对象存储 OSD 节点，1 个服务器作为客户端 Client 节点。最终购买情况如下，因使用了阿里云提供的学生 300 元代金券，最终数额与图片有一定差异。



具体而言，我们选择使用华南 2（河源）地域的服务器以降低成本，采用包年包月（1 周）的形式，购买了短期服务。在其他资源的配置上，我们为每个服务器挂在了 150GB 的 ESSD 云盘，并购买和使用了指 IPv4 公网服务以方便下载资源。同时，创建了专有网络 VPC，构建网段 10.0.0.0/24 以方便内网互联互通。购买完毕后为更改各服务器的实例名称与主机名，方便后续为其配置 CephFS 文件系统。

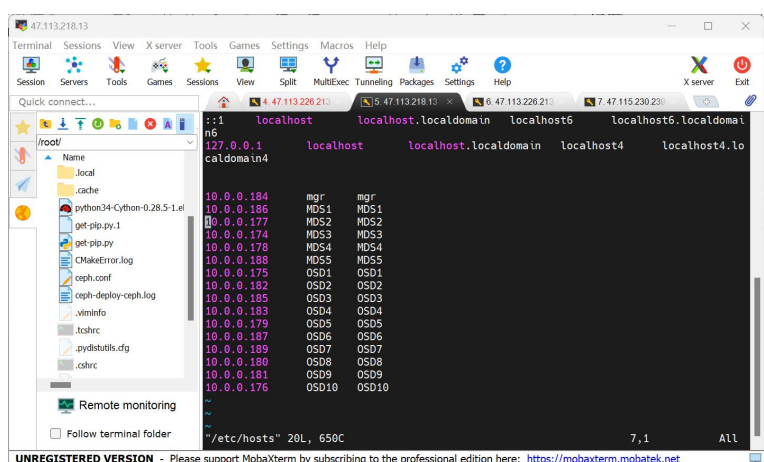




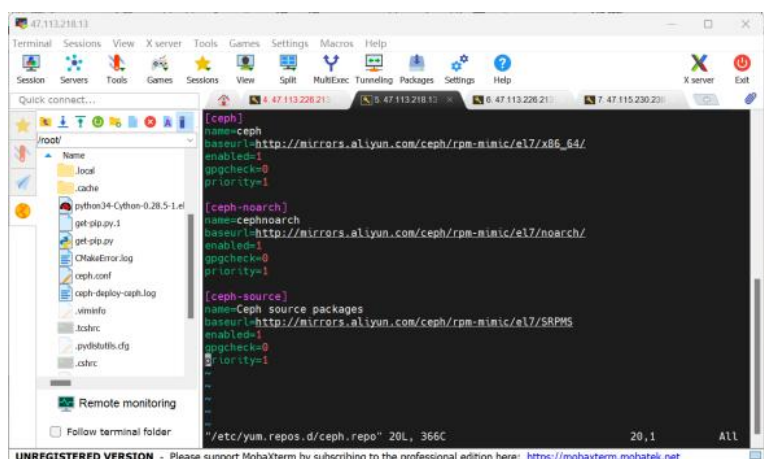
## 3.2 CephFS 安装

### 3.2.1 服务器集群各节点基本设置

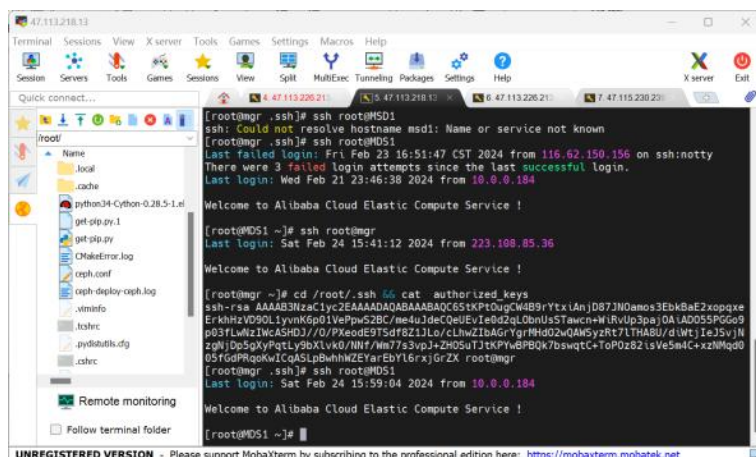
在配置 CephFS 前，需要添加各主机的 host 信息，将其的专有网络 IP 与主机名、实例名称对应。如 10.0.0.184 与 mgr 对应，10.0.0.186 与 MDS1 对应等。



随后关闭防火墙、对齐各服务器时间，并安装必要的软件包如 epel、ceph 等，可以通过制定下载源、设置下载镜像等加快下载速度。



为各主机节点配置公钥和私钥从而使得服务器之间能够利用 ssh 指令免密连接和登录，方便 CephFS 内的各主机之间灵活切换和消息传递。



```
[root@mgr ~]# ssh root@MSD1
[root@mgr ~]# ssh root@MSD1
Last failed login: Fri Feb 23 16:51:47 CST 2024 from 116.62.150.156 on ssh:notty
There were 3 failed login attempts since the last successful login.
Last login: Wed Feb 21 23:46:38 2024 from 10.0.0.184
Welcome to Alibaba Cloud Elastic Compute Service !

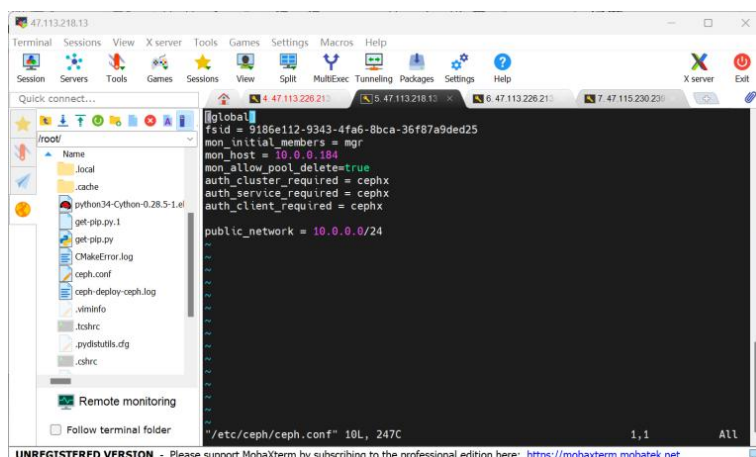
[root@MSD1 ~]# ssh root@mgr
Last login: Sat Feb 24 15:41:12 2024 from 223.108.85.36
Welcome to Alibaba Cloud Elastic Compute Service !

[root@mgr ~]# cd /root/.ssh && cat authorized_keys
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQCAKp01VePp52BC/me4u3deCQdEVeEd2qL0bnl5Stawcn+W(RvUp3pa)0A\AD055PGG09
p03fLwNzJMcASHD3//O/PXeadE9T5df8Z13Le/cLhw21bAgYgrMh02wQAN5yzRt7LTHA8U/d\Wt\IeJ5v\N
zqNjDp5qXyPqTly9bXlvk0/Wmf/Wm7a3vpJ+ZHO5uTJtkPYv6PBQK7bswtC+ToP0z82isVe5m4C+zzNMqd0
05f0dPRqKwIcQASLpBwhhWZYEaBYL6rxjGrZX root@mgr
[root@mgr ~]# ssh root@MSD1
Last login: Sat Feb 24 15:59:04 2024 from 10.0.0.184
Welcome to Alibaba Cloud Elastic Compute Service !

[root@MSD1 ~]#
```

### 3.2.2 安装 Ceph 并配置和创建节点

所有节点都要执行安装和配置 Ceph 节点的操作。以 mgr 节点为例，需要在该服务器上利用部署安装工具 ceph-deploy 创建集群。除此以外，mgr 节点需要为集群配置公共网络与 mon 监视节点。

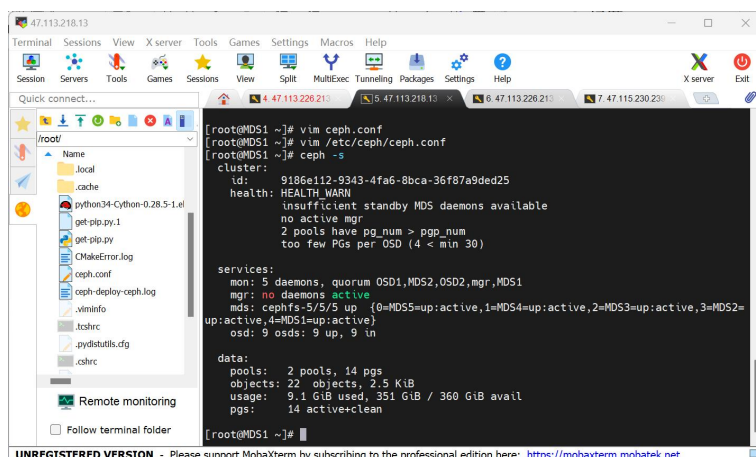


```
[global]
fsid = 9186e112-9343-4fa6-8bca-36f87a9ded25
mon_initial_members = mgr
mon_host = 10.0.0.184
mon_allow_pool_delete=true
auth_cluster_required = cephx
auth_service_required = cephx
auth_client_required = cephx

public_network = 10.0.0.0/24

/etc/ceph/ceph.conf 10L, 247C
```

安装完成后，可以使用指令 “ceph -s” 查看本集群当前的状态。



```
[root@MSD1 ~]# vim ceph.conf
[root@MSD1 ~]# vim /etc/ceph/ceph.conf
[root@MSD1 ~]# ceph -s
cluster:
  id: 9186e112-9343-4fa6-8bca-36f87a9ded25
  health: HEALTH_WARN
insufficient standby MDS daemons available
no active mgr
2 pools have pg_num > pgp_num
too few PGs per OSD (4 < min 30)

services:
  mon: 5 daemons, quorum OSD1,MSD2,OSD2,mgr,MSD1
  mgr: no daemons active
  mds: cephfs-5/5/5 up {0=MSD5=up:active,1=MSD4=up:active,2=MSD3=up:active,3=MSD2=up:active,4=MSD1=up:active}
  osd: 9 osds: 9 up, 9 in

data:
  pools: 2 pools, 14 pgs
  objects: 22 objects, 2.5 KiB
  usage: 9.1 GiB used, 351 GiB / 360 GiB avail
  pgs: 14 active+clean
```

接着创建 OSD 节点。在 Ceph 中，由于 OSD 节点的存储块必须是磁盘设备、分区、逻辑卷等，不能是普通目录（普通目录创建 OSD 时会报错。逻辑卷挂载的目录也可以，但在实验过程中，若使用逻辑卷挂载目录，则逻辑卷扩容后 OSD 并不会扩容，因此不如直接使用磁盘或分区创建 OSD 便利）。

为此我们选择购买额外阿里云 ESSD AutoPL 云盘。在阿里云 ECS 实例控制台中，选择实例并进行存储配置以创建云盘。创建云盘后需要在各节点分别对挂在的硬盘进行格式化操作，如使用 fdisk 工具查看数据盘的设备名称（vdb 即是我们新加的额外云盘）并其创建分区。创建分区后，进入 mgr 节点利用 ceph-deploy disk 相关指令格式化磁盘并创建对应的 OSD 存储块，配置开机自动挂载分区即可使用。

```
[root@OSD10 ~]# sudo fdisk -lu
Disk /dev/vda: 161.1 GB, 161061273600 bytes, 314572800 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk label type: dos
Disk identifier: 0x000bdc9e

   Device Boot      Start         End      Blocks   Id  System
/dev/vda1 *         2048     314572766     157285359+   83   Linux

Disk /dev/vdb: 322.1 GB, 322122547200 bytes, 629145600 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
```

MDS 是 CephFS 文件系统的元数据存储节点，虽然同为数据存储节点，其设置过程与 OSD 有较大差异。在 Linux 系统使用 ls 等操作查看某个目录下的文件的时候，会有保存在磁盘上的分区表记录文件的名称、创建日期、大小、inode 及存储位置等元数据信息，在 CephFS 由于数据是被打散为若干个离散的 object 进行分布式存储，因此并没有统一保存文件的元数据，而且将文件的元数据保存到一个单独的存储出 matedata pool，但是客户端并不能直接访问 matedata pool 中的元数据信息，而是在读写数据时由 MDS（matadata server）处理，读数据时由 MDS 从 matedata pool 加载元数据然后缓存在内存(用于后期快速响应其它客户端的请求)并返回给客户端，写数据的时候有 MDS 缓存在内存并同步到 matedata pool。

在配置完后，使用 “ceph mds stat” 查看 MDS 节点状态。

```
[root@OSD10 ~]# ceph mds stat
cephfs-5/5/5 up {0=MDS5=up:active,1=MDS4=up:active,2=MDS3=up:active,3=MDS2=up:active,4=MDS1=up:active}
[root@OSD10 ~]#
```

在 mgr 节点查看客户机 key，并在 Client 节点将信息保存到 admin.key 文件中。

```
[root@mgr ceph]# cat ceph.client.admin.keyring
[client.admin]
key = AQC0MdJlnkXkXjAAjrtKve4saaoimWHtRVq3mw==
caps mds = "allow *"
caps mgr = "allow *"
caps mon = "allow *"
caps osd = "allow *"
[root@mgr ceph]#
```

在完成所有的 CephFS 配置后，使用 “df-h” 查看整体情况。

```
[root@OSD10 ~]# df -h
Filesystem      Size  Used Avail Use% Mounted on
devtmpfs        1.8G   0   1.8G   0% /dev
tmpfs           1.8G   0   1.8G   0% /dev/shm
tmpfs           1.8G 656K   1.8G   1% /run
tmpfs           1.8G   0   1.8G   0% /sys/fs/cgroup
/dev/vda1       148G  19G  124G  13% /
10.0.0.184:6789:/ 111G   0  111G   0% /cephfs_data
tmpfs           365M   0  365M   0% /run/user/0
/dev/vdb        296G  159G  122G  57% /mnt
[root@OSD10 ~]#
```

## 3.3 Lunule 的安装

### 3.3.1 环境配置

Linux 平台上安装 Git 的工作需要调用 curl, zlib, openssl, expat, libiconv 等库的代码, 所以需要先安装这些依赖工具。

在有 yum 的系统上 (比如 Fedora、CentOS) 或者有 apt-get 的系统上 (比如 Debian、Ubuntu), 可以利用特定指令如 “yum install xxx”、 “apt install” 等执行相关软件依赖的安装工作。

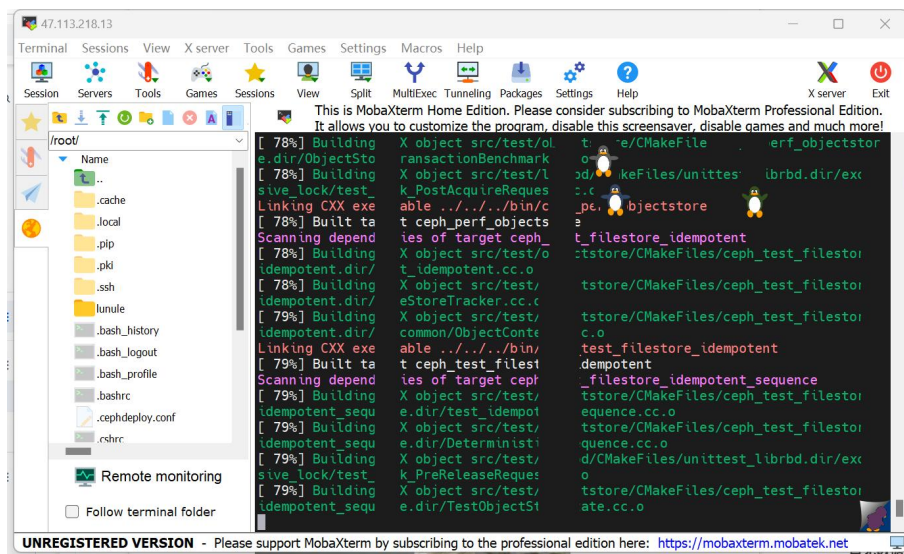
例如, 在 CentOS 下可以执行以下语句安装相关依赖:

```
[root@MDS1 ~]# yum install curl wget zlib openssl expat libiconv curl-devel expat-devel gettext-devel openssl-devel zlib-devel git-core bzip2-devel -y
```

使用 git 工具下载 Lunule 项目开源的代码文件:

```
[root@MDS1 ~]# git clone https://github.com/mdbal-lunule/lunule.git
```

根据论文对应的开源项目代码, 需要在利用 install-deps.sh 和 do\_cmake.sh 分别安装编译依赖和执行 cmake 准备, 再利用 make 和 make install 指令编译和执行, 并重启 CephFS 文件系统的 mds 服务。



### 3.3.2 遇到的问题

在执行相关编译指令的过程中, 我们遇到了如下几个问题, 在此对遇到的问题和我们的解决方案进行详述。

**问题 1:** 在执行 “./install-deps.sh” 时, 报错 “ERROR: No matching distribution found for setuptools<36,>=0.8”



```

hl (11 kB)
Collecting pyparsing>=2.0.2
  Downloading http://mirrors.cloud.aliyuncs.com/pypi/packages/8a/bb/488841f56197b13700afd5658fc279a2025a39e22449b7cf29864669b15d/pyparsing-2.4.7-py2.py3-none-any.whl (67 kB)
##### 67 kB 87.2 MB/s
Installing collected packages: pyparsing, packaging, appdirs, six, setuptools
  Attempting uninstall: setuptools
    Found existing installation: setuptools 44.1.1
    Uninstalling setuptools-44.1.1:
      Successfully uninstalled setuptools-44.1.1
Successfully installed appdirs-1.4.4 packaging-20.9 pyparsing-2.4.7 setuptools-35.0.2 six-1.16.0
DEPRECATION: Python 2.7 reached the end of its life on January 1st, 2020. Please upgrade your Python as Python 2.7 is no longer maintained. pip 21.0 will drop support for Python 2.7 in January 2021. More details about Python 2 support in pip can be found at https://pip.pypa.io/en/latest/development/release-process/#python-2-support pip 21.0 will remove support for this functionality.
Looking in indexes: http://mirrors.cloud.aliyuncs.com/pypi/simple/
WARNING: The repository located at mirrors.cloud.aliyuncs.com is not a trusted or secure host and is being ignored. If this repository is available via HTTPS we recommend you use HTTPS instead, otherwise you may silence this warning and allow it anyway with '--trusted-host mirrors.cloud.aliyuncs.com'.
ERROR: Could not find a version that satisfies the requirement setuptools<36,>=0.8 (from versions: none)

```

在查看本地的所有 python 运行环境，并设置包含清华源、阿里源、科大源在内的多个 pypi 镜像源，并在一般环境下确认能够正确安装所需版本的 setuptools 后，该问题仍未能得到解决。在查看相关脚本文件后，怀疑是在使用 venv 虚拟环境中执行 pip 时，相关进程无法访问 pypi.org 的包仓库导致的问题，不影响后续实验。该问题已向项目原作者提出，但截至本文档撰写时未能得到回复。

## 问题 2：执行 “make”时，报错 “c++: internal compiler error: killed (program cc1plus)”

```

Scanning dependencies of target cls_references_objs
[ 15%] Building CXX object src/CMakeFiles/cls_references_objs.dir/objclass/class_api.cc.o
/root/.lunule/src/osd/OSDMap.cc: In member function 'int OSDMap::validate_crush_rules(CrushWrapper*, std::ostream*) const':
/root/.lunule/src/osd/OSDMap.cc:3449:71: warning: comparison between signed and unsigned integer expressions [-Wsign-compare]
    if (pool.get_size() < (int)newcrush->get_rule_mask_min_size(ruleno) ||
                                                                    ^
/root/.lunule/src/osd/OSDMap.cc:3450:64: warning: comparison between signed and unsigned integer expressions [-Wsign-compare]
    pool.get_size() > (int)newcrush->get_rule_mask_max_size(ruleno)) {
                                                                    ^
[ 15%] Building CXX object src/CMakeFiles/common-objs.dir/osd/OSDMapMapping.cc.o
[ 15%] Built target cls_references_objs
Scanning dependencies of target mds
[ 15%] Building CXX object src/CMakeFiles/common-objs.dir/common/histogram.cc.o
[ 15%] Building CXX object src/CMakeFiles/common-objs.dir/osd/osd_types.cc.o
[ 15%] Building CXX object src/mds/CMakeFiles/mds.dir/Capability.Cc.o
[ 15%] Building CXX object src/mds/CMakeFiles/mds.dir/MDSDaemon.cc.o
c++: internal compiler error: Killed (program cc1plus)
Please submit a full bug report,
with preprocessed source if appropriate.
See <http://bugzilla.redhat.com/bugzilla> for instructions.
make[2]: *** [src/CMakeFiles/common-objs.dir/osd/osd_types.cc.o] Error 4
make[1]: *** [src/CMakeFiles/common-objs.dir/all] Error 2
make[1]: *** Waiting for unfinished jobs....
[ 15%] Building CXX object src/mds/CMakeFiles/mds.dir/MDSRank.cc.o
[ 15%] Building CXX object src/mds/CMakeFiles/mds.dir/Beacon.cc.o
c++: internal compiler error: Killed (program cc1plus)
Please submit a full bug report,
with preprocessed source if appropriate.
See <http://bugzilla.redhat.com/bugzilla> for instructions.
make[2]: *** [src/mds/CMakeFiles/mds.dir/MDSRank.cc.o] Error 4
make[2]: *** Waiting for unfinished jobs....
make[1]: *** [src/mds/CMakeFiles/mds.dir/all] Error 2
make: *** [all] Error 2

```

该问题为内存不足的问题，在实验中由于购买的服务器内存资源不够执行源代码的编译和安装，在执行到一些语句时会出现类似问题。一方面可以通过更换资源更多的服务器，另一方面可以执行以下代码创建 swap 交换分区解决：

```

[root@MDS1 ~]# sudo dd if=/dev/zero of=/swapfile bs=8M count=1024
[root@MDS1 ~]# sudo mkswap /swapfile
[root@MDS1 ~]# sudo swapon /swapfile

```

## 问题 3：执行 make 时，报错

```

make[2]: [src/test/CMakeFiles/ceph_test_librgw_file_marker.dir/librgw_file_marker.cc.o] Error 1
make[1]: *** [src/test/CMakeFiles/ceph_test_librgw_file_marker.dir/all] Error 2
make: [all] Error 2

```

该问题为执行 make 和 make install 时遇到了这样两个的 TEST 函数报错，不影响 make、make install 的继续执行。

可以忽视程序报错或尝试删除以下两个文件内的相关函数：

\\src\\test\\librgw\_file\_marker.cc

```
TEST(LibRGW, CLEANUP) {
    int rc;
    if (do_marker1) {
        cleanup_queue.push_back(
            obj_rec{bucket_name, bucket_fh, fs->root_fh, get_rgwfh(fs->root_fh)});
    }
    for (auto& elt : cleanup_queue) {
        if (elt.fh) {
            rc = rgw_fh_rele(fs, elt.fh, 0 /* flags */);
            ASSERT_EQ(rc, 0);
        }
    }
    cleanup_queue.clear();
}
```

\\src\\test\\librgw\_file\_nfsns.cc

```
TEST(LibRGW, CLEANUP) {
    int rc;

    if (do_marker1) {
        cleanup_queue.push_back(
            obj_rec{bucket_name, bucket_fh, fs->root_fh, get_rgwfh(fs->root_fh)});
    }

    for (auto& elt : cleanup_queue) {
        if (elt.fh) {
            rc = rgw_fh_rele(fs, elt.fh, 0 /* flags */);
            ASSERT_EQ(rc, 0);
        }
    }
    cleanup_queue.clear();
}
```

**问题 4：**执行 “make install” 时，报错 “undefined reference to 'pthread\_create'”

```
/usr/bin/cc CMakeFiles/cmTryCompileExec1918115415.dir/CheckSymbolExists.c.o -o cmTryCompileExec1918115415 -rdynamic
CMakeFiles/cmTryCompileExec1918115415.dir/CheckSymbolExists.c.o: In function 'main':
CheckSymbolExists.c:(.text+0x16): undefined reference to 'pthread_create'
```

出现该报错的原因是 pthread 库不是 Linux 系统默认的库。因此在执行 “make install” 时，若需要调用静态库 “libpthread.a” 中的函数 “pthread\_create()” 创建线程或是函数 “pthread\_atfork()” 建立 fork 处理程序时，需要在编译时添加 “-lpthread” 参数。这种情况类似于 “<math.h>” 的使用，需在编译时加 “-m” 参数。

例如：在加了头文件 “#include <pthread.h>” 之后执行 “pthread.c” 文件，需要使用如下命令：

```
gcc pthread.c -o thread -lpthread
```

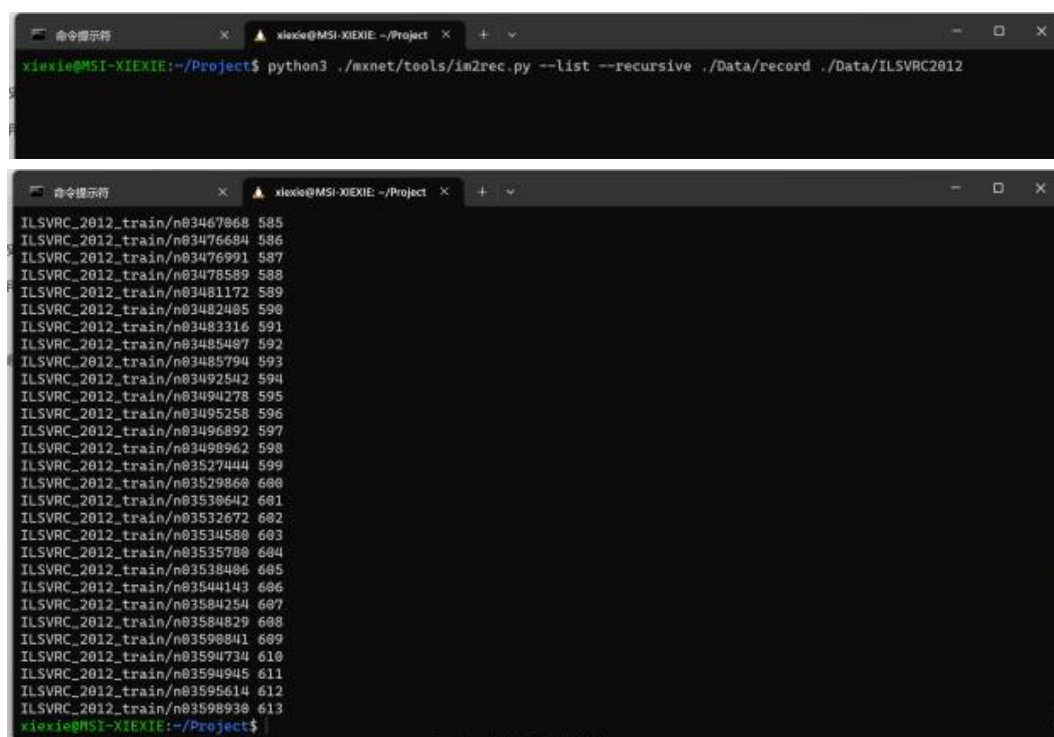
## 3.4 软件测试与结果

### 3.4.1 测试程序及其验证

本次复现过程选择的测试用例是 MXNet 开源项目的 img2rec.py 文件。MXNet 是由亚马逊开发的一个高效且灵活的开源深度学习框架,具备处理大规模深度学习模型和海量数据的能力。当训练数据包含大量图片的时候,一次性将所有数据载入内存很容易导致内存溢出。因此 MXNet 框架利用 img2rec.py 文件将原始的图片数据集转换为一个批次的训练数据记录并保存。

为了保证测试用例的准确性,我们首先在本地运行环境中选择了部分训练集和测试集、验证集对 img2rec.py 程序进行验证和测试。MXNet 的 img2rec.py 程序通过对 ILSVRC2012 数据集进行划分,生成 rec 记录文件对每个批次的图像的路径和内容进行标记和记录。

本地实验环境实验 AMD Ryzen5 5600X 处理器,利用 WSL 安装 Ubuntu-20.04 虚拟机,并执行测试用例的实验。实验输入和输出如下图所示,共耗时约 11 秒。



```
xixie@MSI-XIEXIE: ~/Project
python3 ./mxnet/tools/img2rec.py --list --recursive ./Data/record ./Data/ILSVRC2012

ILSVRC_2012_train/n03467868 585
ILSVRC_2012_train/n03476684 586
ILSVRC_2012_train/n03476991 587
ILSVRC_2012_train/n03478589 588
ILSVRC_2012_train/n03481172 589
ILSVRC_2012_train/n03482405 590
ILSVRC_2012_train/n03483316 591
ILSVRC_2012_train/n03485407 592
ILSVRC_2012_train/n03485794 593
ILSVRC_2012_train/n03489254 594
ILSVRC_2012_train/n03489427 595
ILSVRC_2012_train/n03489528 596
ILSVRC_2012_train/n03496892 597
ILSVRC_2012_train/n03498962 598
ILSVRC_2012_train/n03527444 599
ILSVRC_2012_train/n03529868 600
ILSVRC_2012_train/n03530642 601
ILSVRC_2012_train/n03532672 602
ILSVRC_2012_train/n03534580 603
ILSVRC_2012_train/n03535780 604
ILSVRC_2012_train/n03538406 605
ILSVRC_2012_train/n03544143 606
ILSVRC_2012_train/n03584254 607
ILSVRC_2012_train/n03584829 608
ILSVRC_2012_train/n03590841 609
ILSVRC_2012_train/n03594734 610
ILSVRC_2012_train/n03594945 611
ILSVRC_2012_train/n03595614 612
ILSVRC_2012_train/n03598930 613
xixie@MSI-XIEXIE:~/Project$
```

### 3.4.2 数据集上传

在执行测试用例时,由于 MXNet 使用的 ILSVRC2012 数据集包含各类别的数十万张图片,因此其数据集大小超过我们最初购买的服务器云盘大小。为了解决该问题,我们购买了 300GB 的 ESSD AutoPL 云盘并挂在到客户端节点服务器上。格式化云盘后,将已经下载在本地硬盘中的 ILSVRC2012 数据集利用 WinSCP 等远程服务器文件管理软件上传数据集文

件。为了减少上传所需时间，我们希望通过临时升级带宽的方式以增加文件的传输速率，但是实际上传中并没有省去多少时间。



由于该测试项目使用的 ILSVRC2012 的训练数据集为随机设置的多 epoch 文件，其经过一次解压后仍为压缩包文件，因此编写 unzip.py 文件用于进一步解压数据集，并在解压后删除压缩包以节约存储空间。

```
import glob
import os

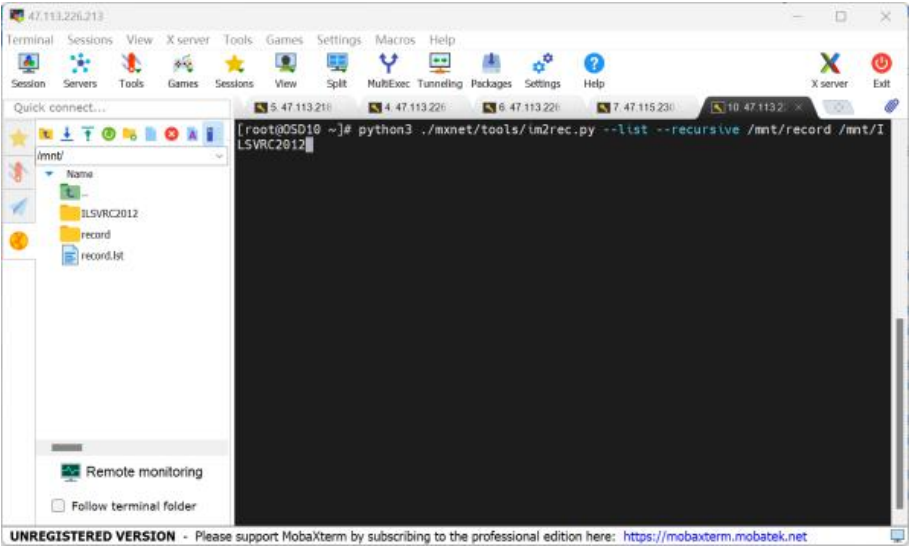
filelist = glob.glob("./train/*.tar")

for f in filelist:
    os.system("mkdir ." + f.split('.')[1])
    os.system("tar -xvf " + f + " -C ." + f.split('.')[1])
    os.system("rm " + f)
```

### 3.4.3 服务器测试结果

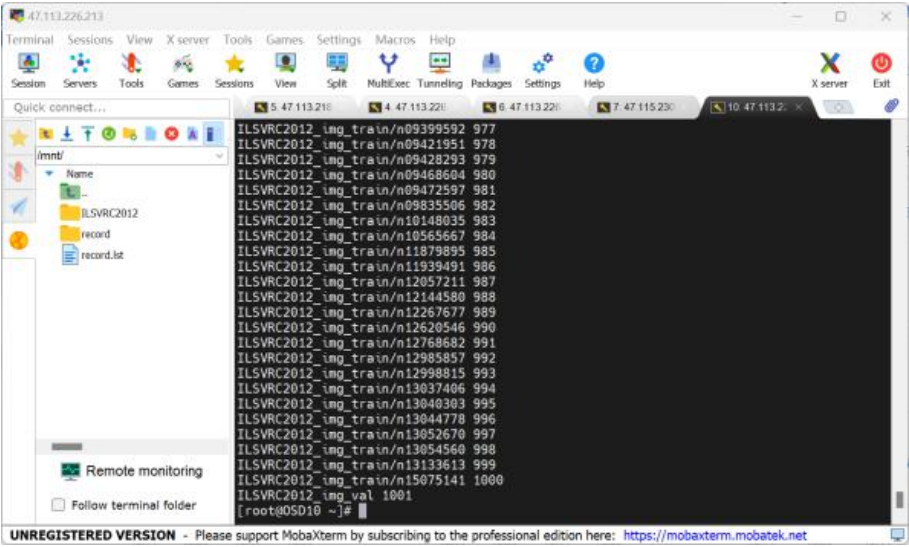
将数据集上传到服务器后，我们在服务器环境下，对该论文对应的开源项目的软件进行了测试，测试的输入和输出如下图所示。由于我们使用的服务器均为 2 核 CPU，且内存大小为 4GB，因此较原论文所述的运行结果有较大差异性，从指令输入到产生输出结果的时间约为 28 秒。利用阿里云服务器的服务器资源监控能够清晰直观的查看在指令执行前后的资源占有率变化情况。该指令执行前后的资源占用变化情况展示如下图所示。

输入截图：

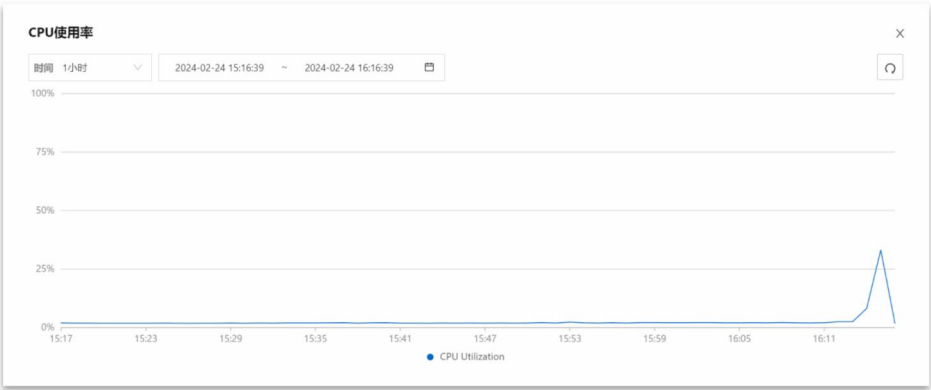




输出截图：



CPU 占有率变化：



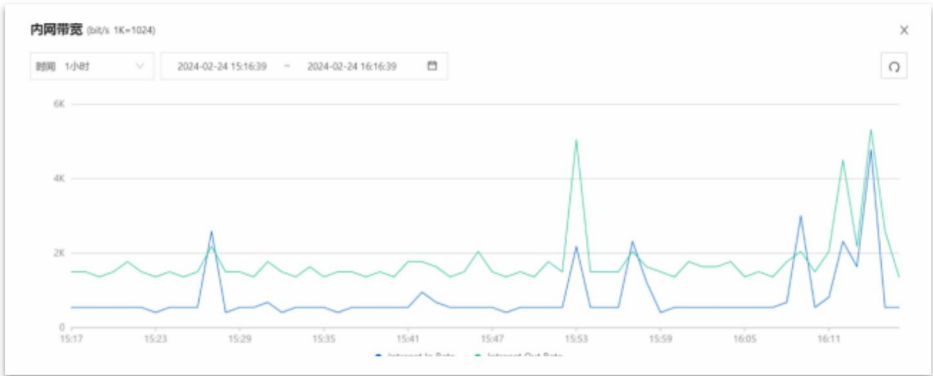
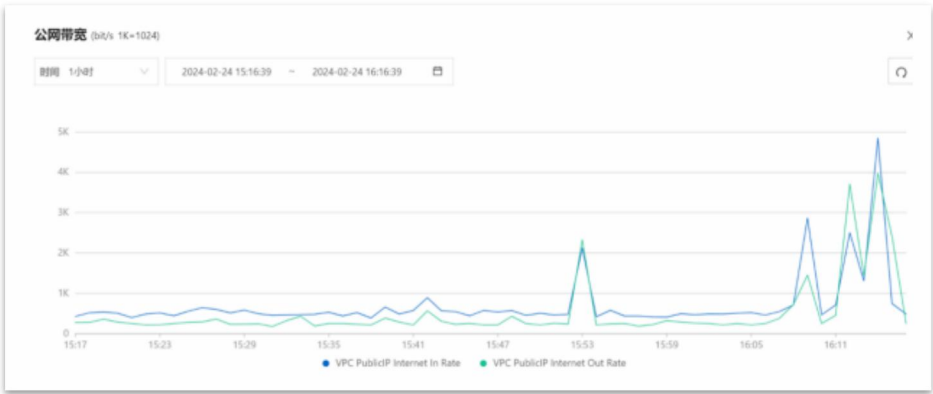
云盘读写变化：



云盘 I/O 变化情况：



带宽使用情况：



从图中可看出，在 16:15 时，CPU 使用率有一个高峰达 35%，随后急转直下；实例云盘读写 BPS 也达 4M；实例云盘 I/O 变化情况最高值也近 1k 次每秒；公网带宽也达 5k，内网带宽也同样达到了 5k。