logo_hhu **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 1st Semester, 2023~2024

河 海 大 学

**目录**

**[1 测试用例 1](#_Toc29094)**

[1.1 编译安装Lunule 1](#_Toc28729)

[1.2 上传和解压数据集 2](#_Toc19899)

[1.3 配置环境并运行测试 3](#_Toc22578)

**[2 测试场景 4](#_Toc968)**

[2.1 负载监视 4](#_Toc17476)

[2.2 工作负载感知和迁移 4](#_Toc3206)

**[3 测试报告 6](#_Toc23630)**

[3.1 配置实验服务器集群 6](#_Toc25146)

[3.2 CephFS安装 7](#_Toc25564)

[3.2.1 服务器集群各节点基本设置 7](#_Toc2868)

[3.2.2 安装Ceph并配置和创建节点 8](#_Toc7465)

[3.3 Lunule的安装 10](#_Toc26943)

[3.3.1 环境配置 10](#_Toc29001)

[3.3.2 遇到的问题 10](#_Toc32185)

[3.4 软件测试与结果 13](#_Toc28272)

[3.4.1 测试程序及其验证 13](#_Toc3874)

[3.4.2 数据集上传 13](#_Toc30136)

[3.4.3 服务器测试结果 14](#_Toc17505)

2. **测试用例**

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
3. 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

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

1. 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/mdbal-lunule/lunule.git

1. 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

1. 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. 上传和解压数据集
2. 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)

Devlopment kit for Task1 & Task2：

https://image-net.org/data/ILSVRC/2012/ILSVRC2012\_devkit\_t12.tar.gz

Devlopment 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

1. 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。

1. 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. 配置环境并运行测试

* 本部分在CephF的客户端OSD10节点执行

1. 安装测试依赖

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

1. 克隆测试程序MXNet项目

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

1. 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/im2rec.py --list --recursive {your\_ceph\_client\_path}/record {your\_ceph\_client\_path}/Dataset

例如：

[root@OSD10 ~]# python3 ./mxnet/tools/im2rec.py --list --recursive /mnt/ILSVRC2012/record /mnt/ILSVRC2012/Dataset

1. **测试场景**

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。

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，挂载五个功能的请求处理。

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。

1. 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。

1. **测试报告**

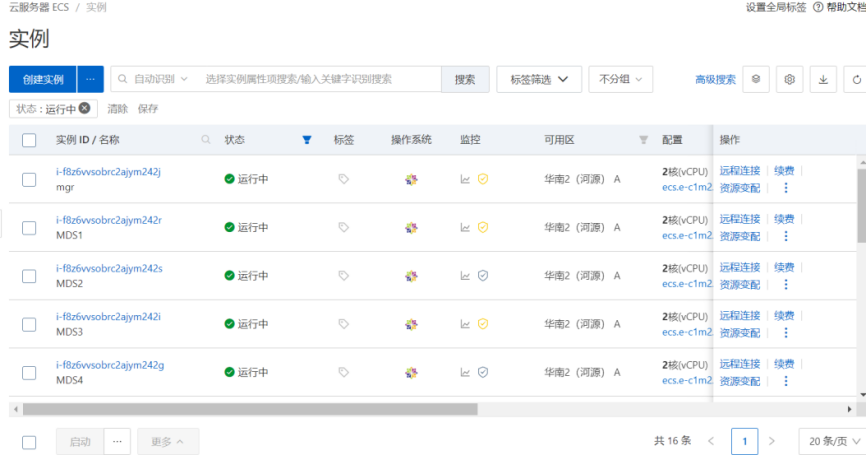
本部分包含CephFS的安装和第1章节所述的测试用例的实际执行结果。在执行过程中，由于资源限制和网络限制，主要的问题存在与运行环境上。本章节将按照测试用例的相关内容，在设置的服务器集群上进行环境配置和测试，对测试中出现的包括环境配置问题和代码问题进行详细的描述并尝试提出一些可供参考的解决思路。

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。



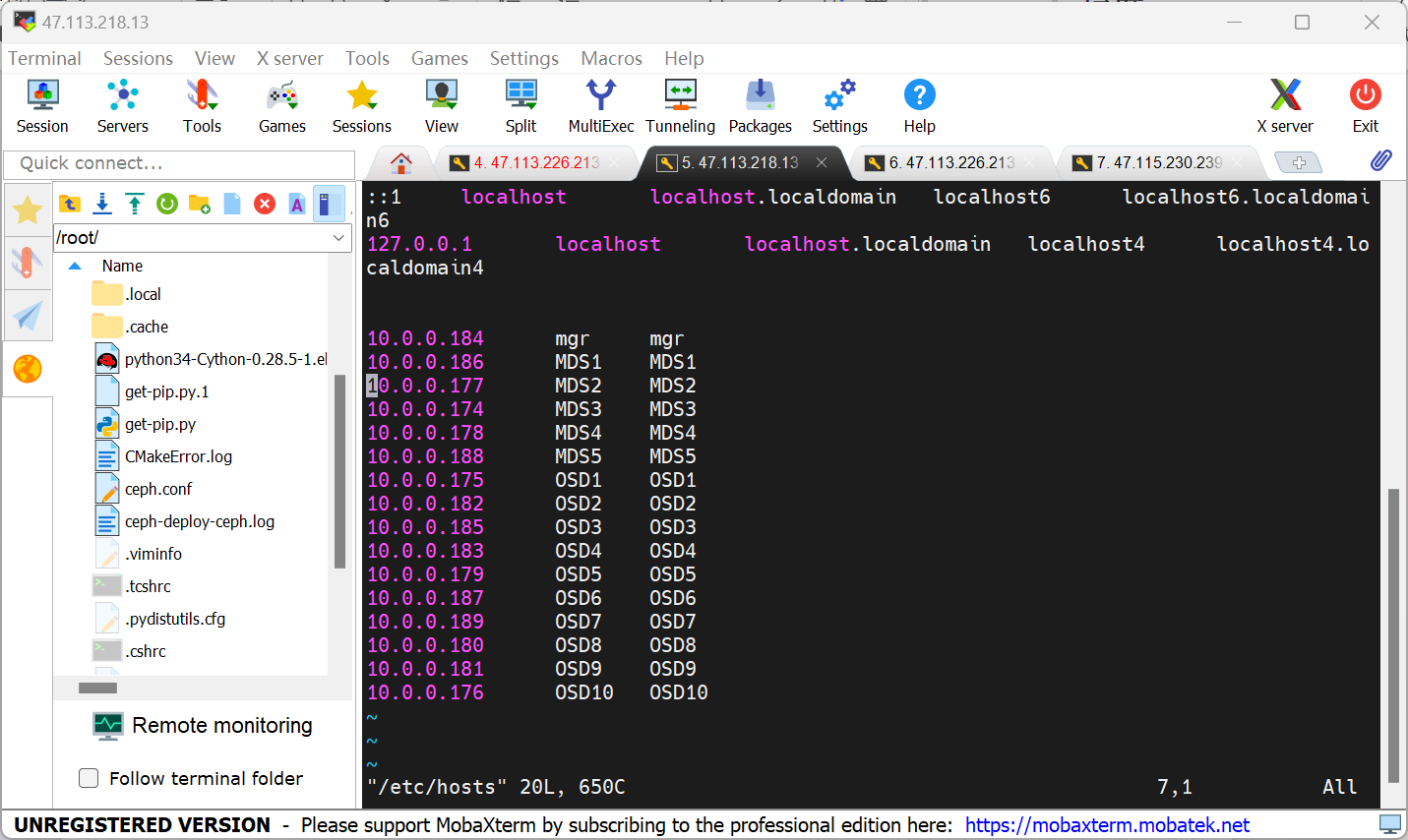
具体而言，我们选择使用华南2（河源）地域的服务器以降低成本，采用包年包月（1周）的形式，购买了短期服务。在其他资源的配置上，我们为每个服务器挂在了150GB的ESSD云盘，并购买和使用了ipv4公网服务以方便下载资源。同时，创建了专有网络VPC，构建网段10.0.0.0/24以方便内网互联互通。购买完毕后为更改各服务器的实例名称与主机名，方便后续为其配置CephFS文件系统。



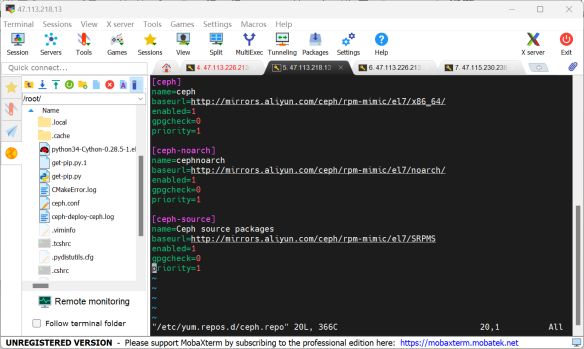
1. CephFS安装

## 服务器集群各节点基本设置

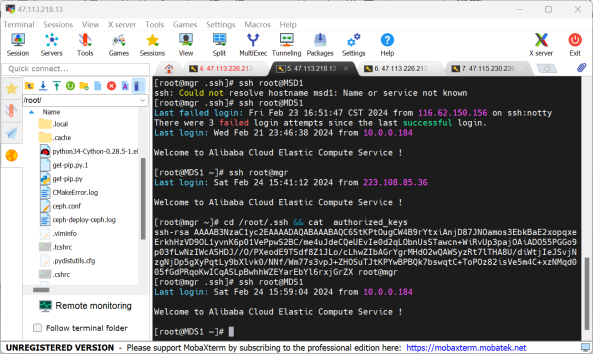
在配置CephFS前，需要添加各主机的host信息，将其的专有网络IP与主机名、实例名称对应。如10.0.0.184与mgr对应，10.0.0.186与MDS1对应等。



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。

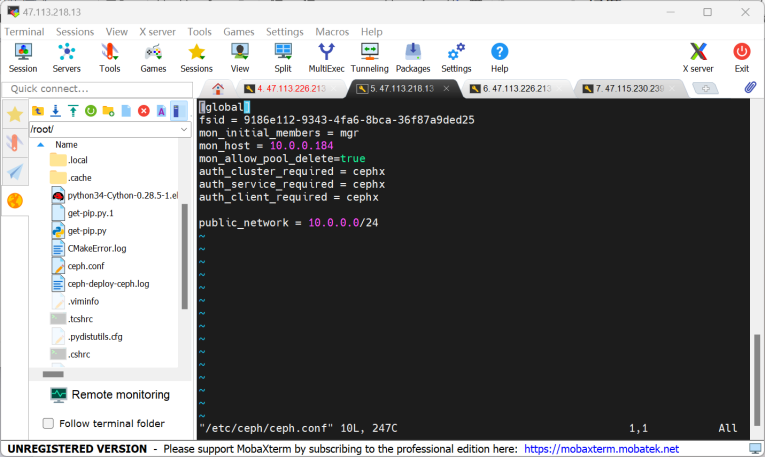


为各主机节点配置公钥和私钥从而使得服务器之间能够利用ssh指令免密连接和登录，方便CephFS内的各主机之间灵活切换和消息传递。

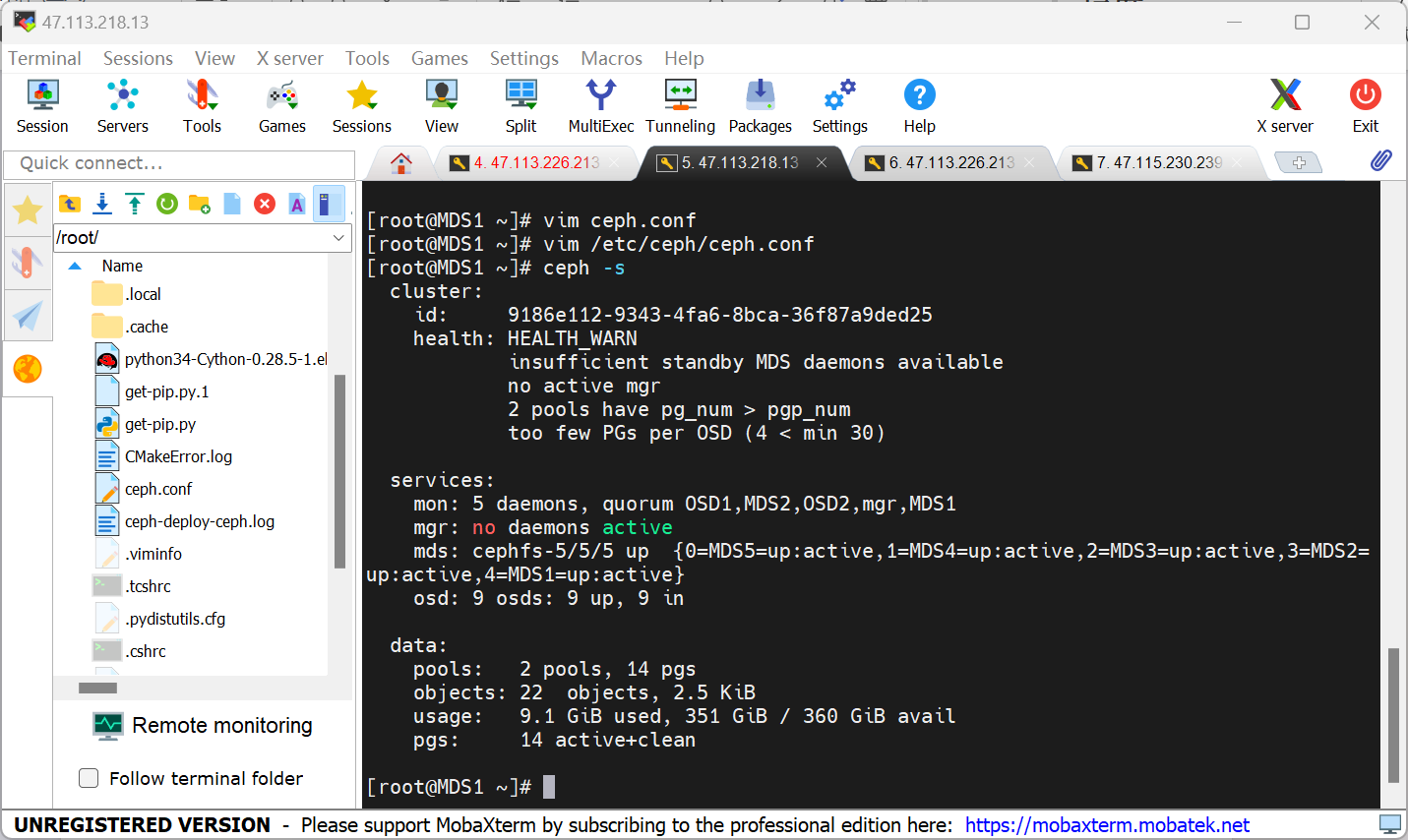


## 安装Ceph并配置和创建节点

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-deloy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。

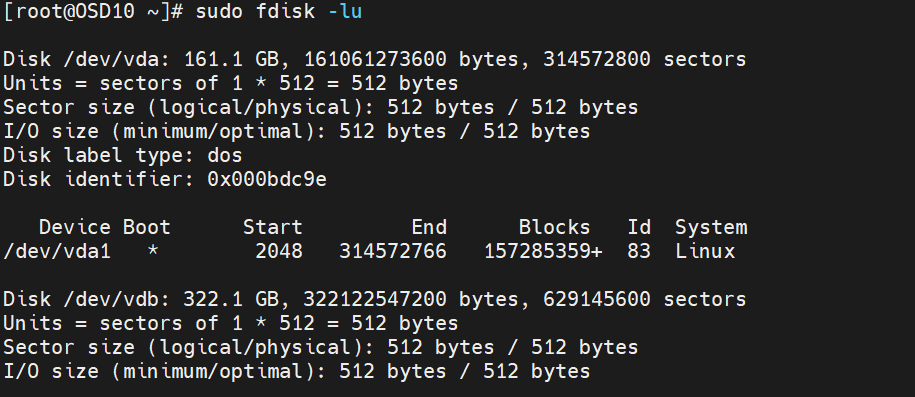


安装完成后，可以使用指令“ceph -s”查看本集群当前的状态。



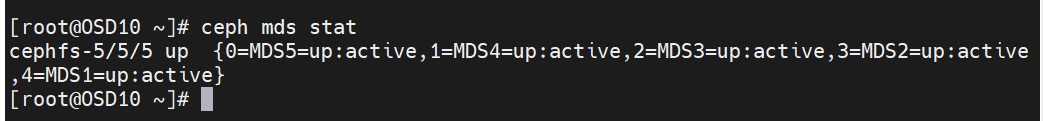
接着创建OSD节点。在Ceph中，由于OSD节点的存储块必须是磁盘设备、分区、逻辑卷等，不能是普通目录（普通目录创建OSD时会报错。逻辑卷挂载的目录也可以，但在实验过程中，若使用逻辑卷挂载目录，则逻辑卷扩容后OSD并不会扩容，因此不如直接使用磁盘或分区创建OSD便利）。

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。

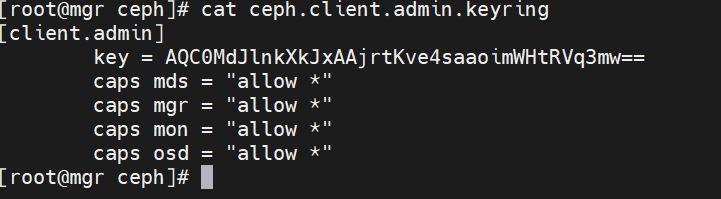


MDS是CephFS文件系统的元数据存储节点，虽然同为数据存储节点，其设置过程与OSD有较大差异。在Linux系统使用ls等操作查看某个目录下的文件的时候，会有保存在磁盘上的分区表记录文件的名称、创建日期、大小、inode及存储位置等元数据信息，在CephFS由于数据是被打散为若干个离散的object进行分布式存储，因此并没有统一保存文件的元数据，而且将文件的元数据保存到一个单独的存储出matedata pool，但是客户端并不能直接访问matedata pool中的元数据信息，而是在读写数据时由MDS（matadata server）处理，读数据时由MDS从matedata pool加载元数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

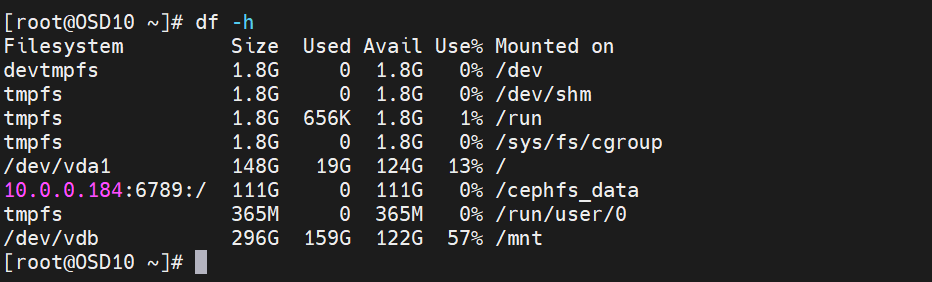
在配置完后，使用“ceph mds stat”查看MDS节点状态。



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。



在完成所有的CephFS配置后，使用“df-h”查看整体情况。



1. Lunule的安装

## 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。

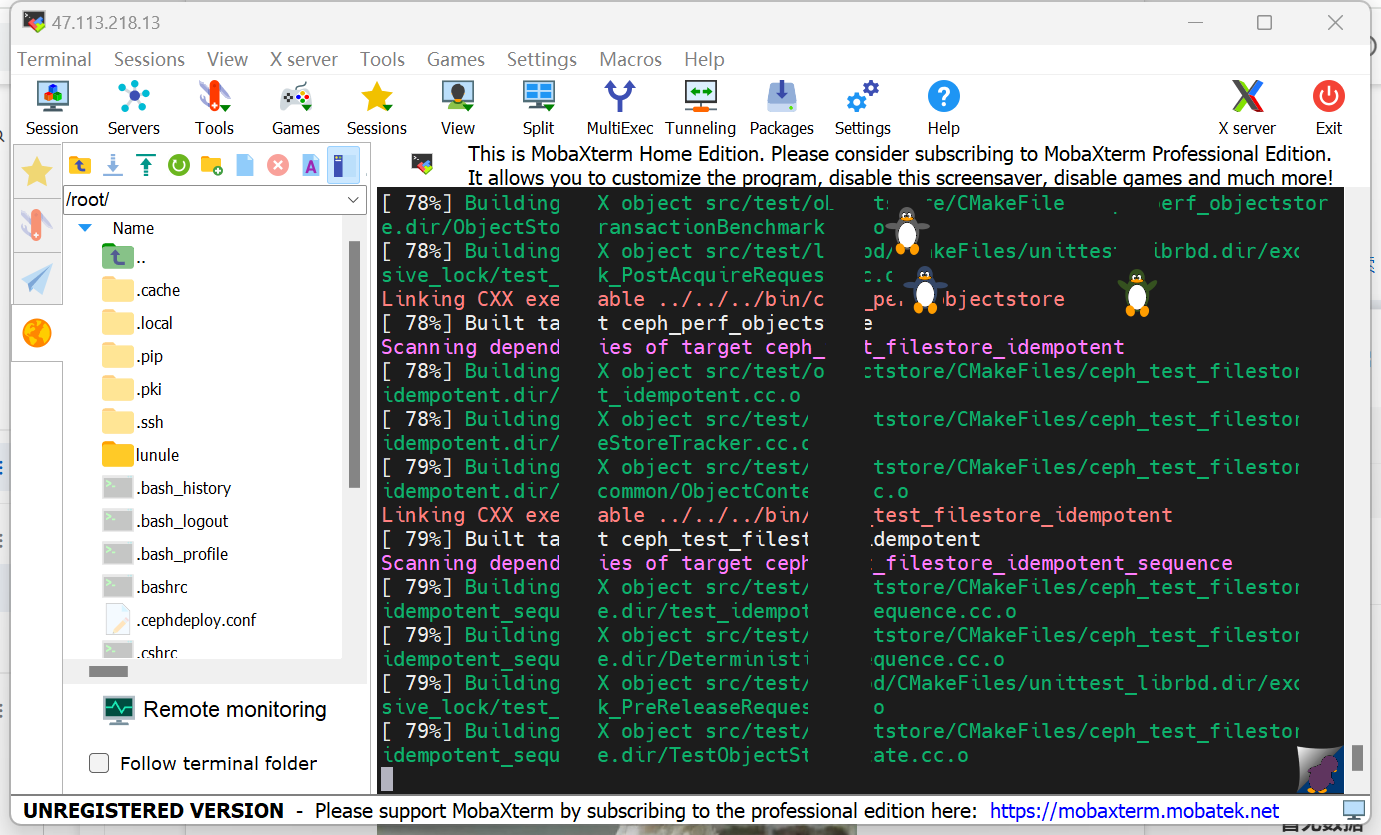
例如，在CentOS下可以执行以下语句安装相关依赖：

[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

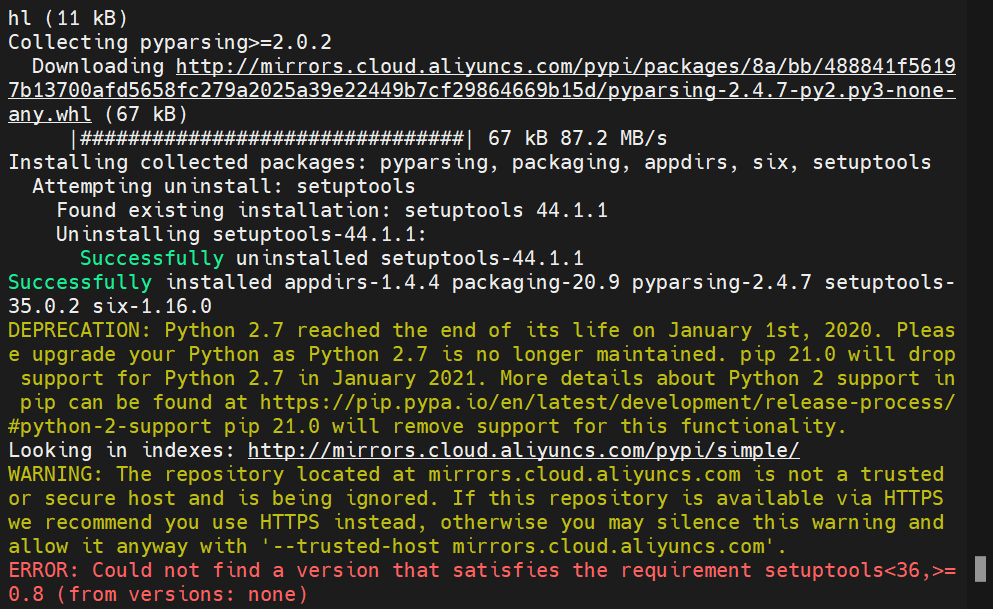
根据论文对应的开源项目代码，需要在利用install-deps.sh和do\_cmake.sh分别安装编译依赖和执行cmake准备，再利用make和make install指令编译和执行，并重启CephFS文件系统的mds服务。



## 遇到的问题

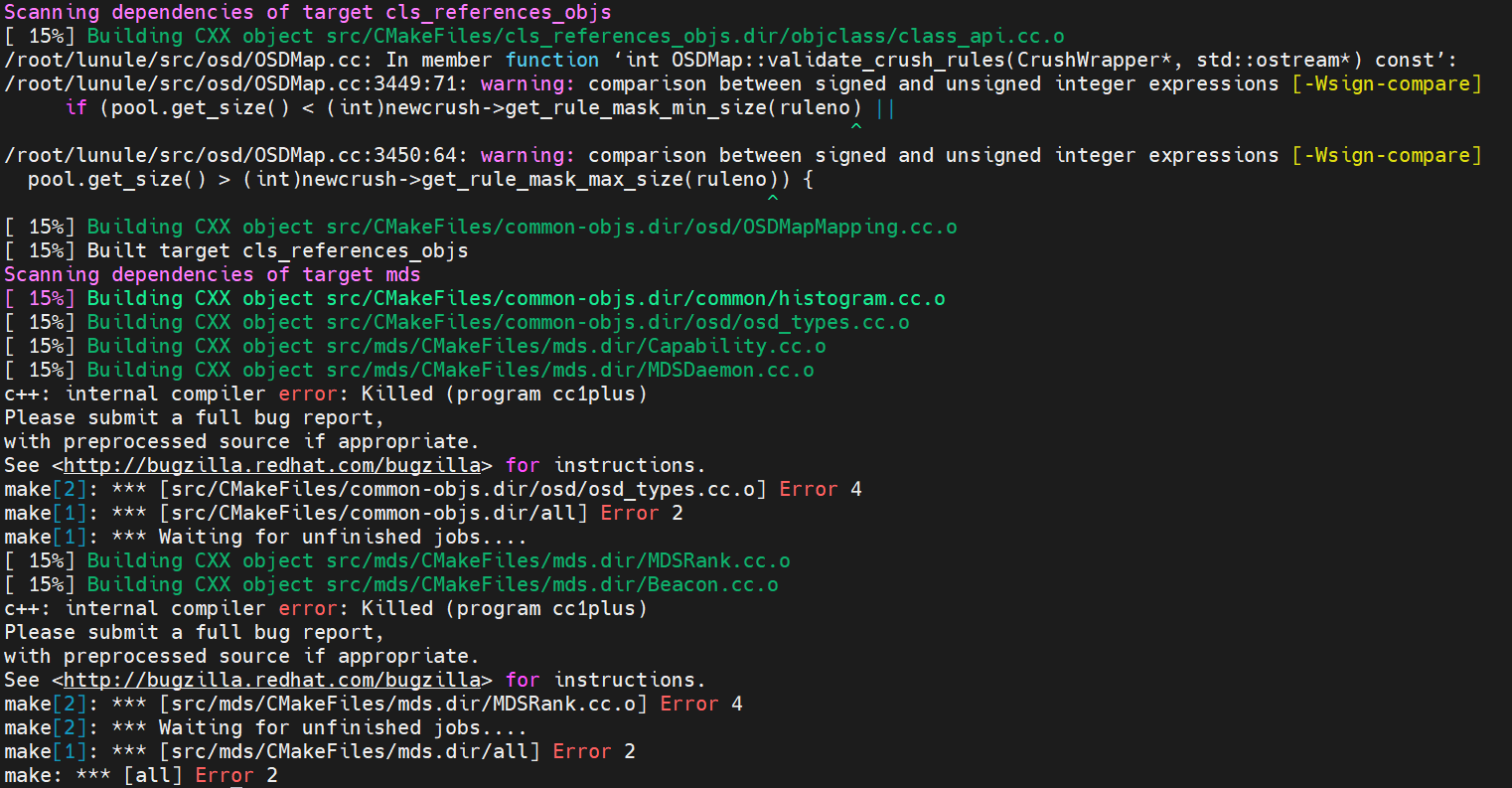
在执行相关编译指令的过程中，我们遇到了如下几个问题，在此对遇到的问题和我们的解决方案进行详述。

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



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) ”



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);

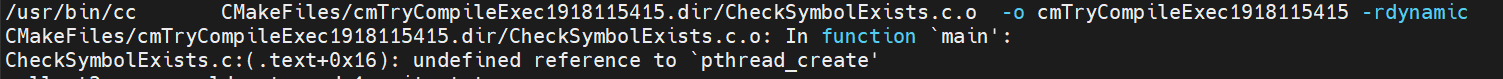
}

}

cleanup\_queue.clear();

}

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



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

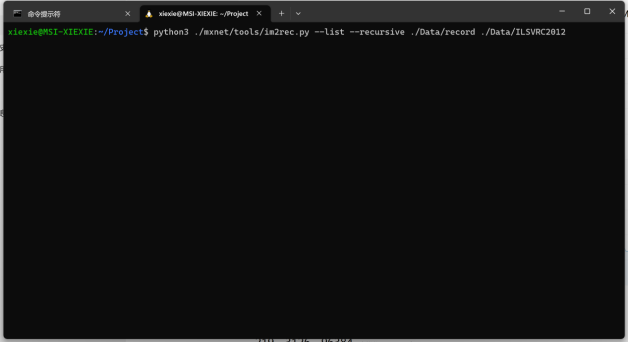
1. 软件测试与结果

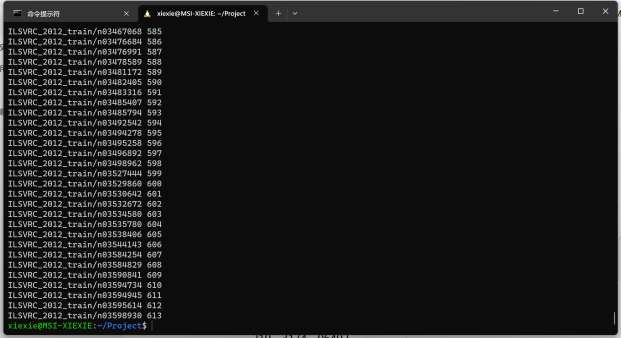
## 测试程序及其验证

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。



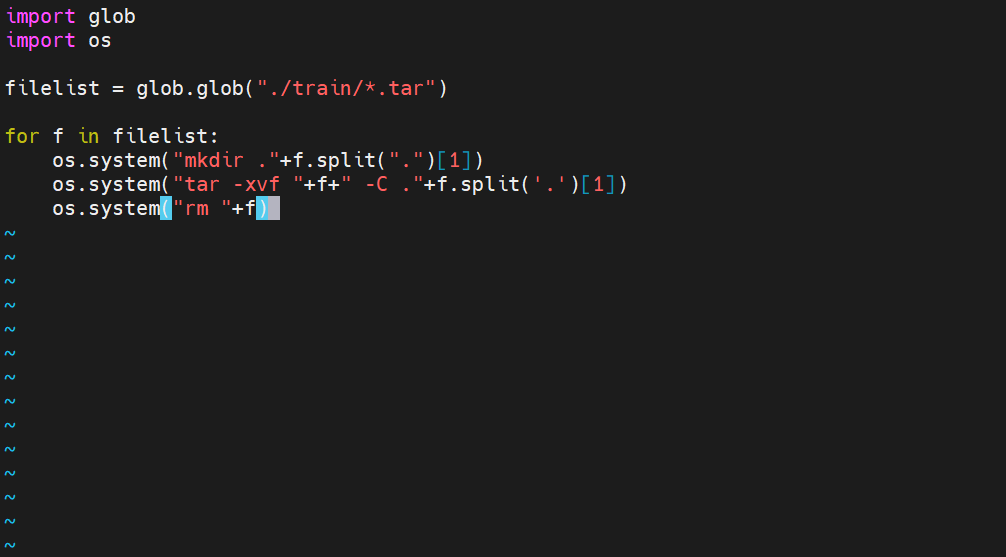


## 数据集上传

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。



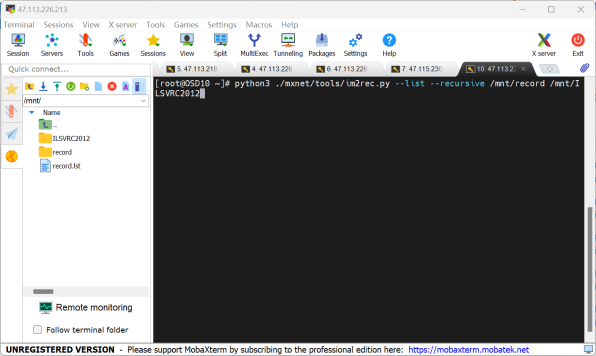
由于该测试项目使用的ILSVRC2012的训练数据集为随机设置的多epoch文件，其经过一次解压后仍为压缩包文件，因此编写unzip.py文件用于进一步解压数据集，并在解压后删除压缩包以节约存储空间。



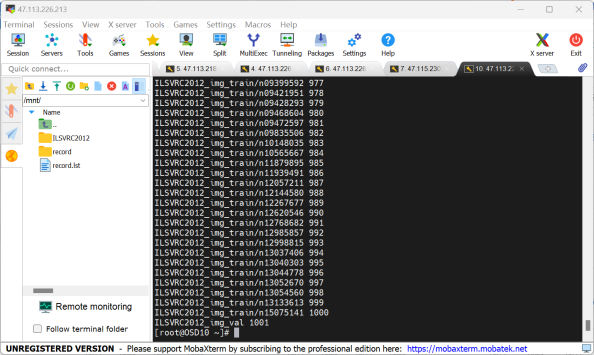
## 服务器测试结果

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。

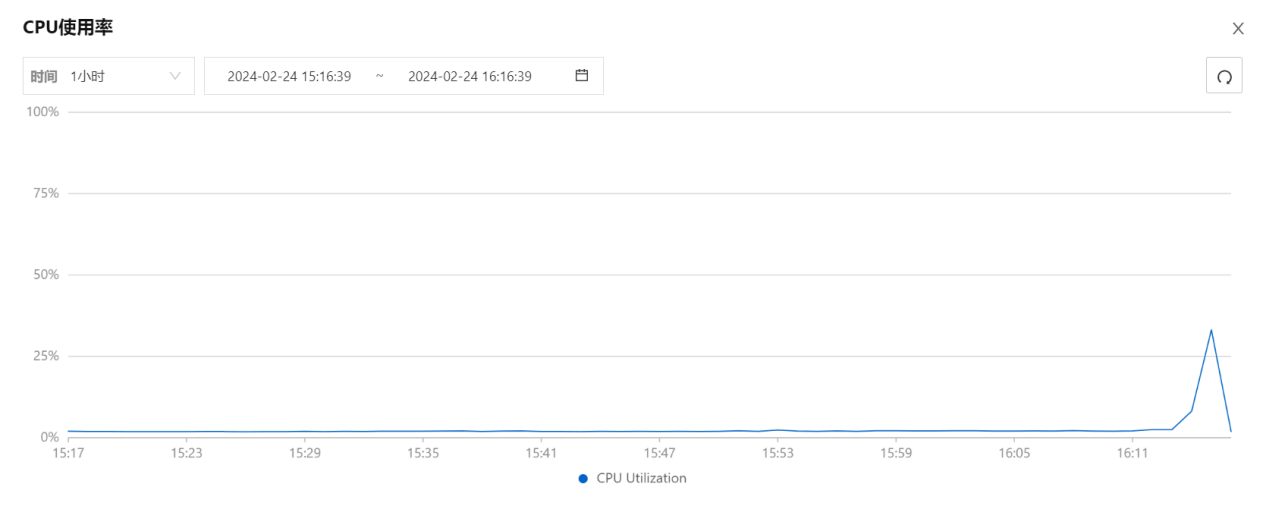
输入截图：



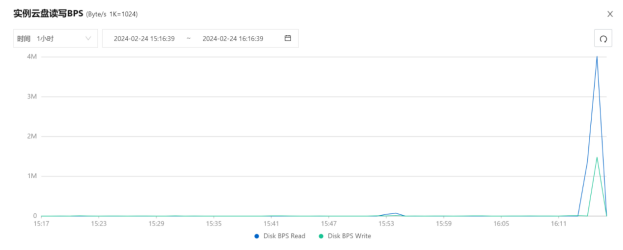
输出截图：



CPU占有率变化：



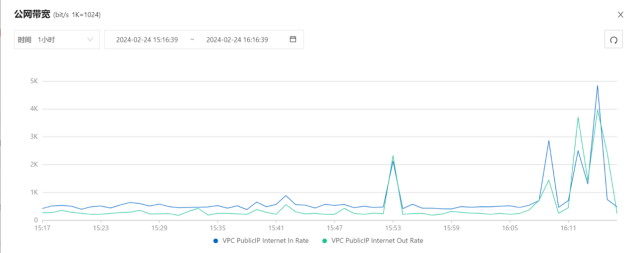
云盘读写变化：



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。