2019 级

《物联网数据存储与管理》课程

实 验 报 告

姓　名 黄欣宇

学　号 U201915014

班　号 计算机 1905 班

日　期 2022.04.14

目　　录

一、实验目的 1
二、 实验背景 2
三、实验环境 4
四、实验内容 5
4.1 对象存储技术实践 5
4.2 对象存储性能分析 5
五、实验过程 6
5.2 对象存储性能分析 8
六、实验总结 10
参考文献 11

一、实验目的

1. 熟悉对象存储技术，代表性系统及其特性；
2. 实践对象存储系统，部署实验环境，进行初步测试；
3. 基于对象存储系统，分析性能问题，架设应用实践。

二、实验背景

对象存储（Object-based Storage)是一种新的网络存储架构，基于对象存储技术的设备就是对象存储设备（Object-based Storage Device）简称 OSD。1999 年成立的全球网络存储工业协会（SNIA）的对象存储设备工作组发布了 ANSI 的 X3T10 标准。总体上来讲，对象存储综合了 NAS 和 SAN 的优点，同时具有 SAN 的高速直接访问和 NAS 的分布式数据共享等优势，提供了具有高性能、高可靠性、跨平台以及安全的数据共享的存储体系结构。

核心是将数据通路（数据读或写）和控制通路（元数据）分离，并且基于对象存储设备构建存储系统，每个对象存储设备具有一定的只能，能够自动管理其上的数据分布。对象存储结构由对象、对象存储设备、元数据服务器、对象存储系统的客户端四部分组成。

(1)对象

对象是系统中数据存储的基本单位，每个 Obeject 是数据和数据属性集的综合体，数据属性可以根据应用的需求进行设置，包括数据分布、服务质量等。在传统的存储系统中用文件或块作为基本的存储单位，块设备要记录每个存储数据块在设备上的位置。Object 维护自己的属性，从而简化了存储系统的管理任务，增加了灵活性。Object 的大小可以不同，可以包含整个数据结构，如文件、数据表项等。在存储设备中，所有对象都有一个对象标识，通过对象标识 OSD 命令访问对象。通常由多种类型的对象，存储设备上的根对象标识存储设备和该设备的各种属性，组队象是存储设上共享资源管理策略的对象集合等。

(2)对象存储设备

每个 OSD 都是一个智能设备，具有自己的存储介质、处理器、内存以及网络系统等，负责管理本地的 Object，是对象存储系统的核心。OSD 同块设备的不同不在于存储介质，而在于两者提供的访问接口。OSD 的主要功能包括数据存储和安全访问、目前国际上通常采用刀片式结构实现对象存储设备。OSD 提供三个主要功能：

① 数据存储。OSD 管理对象数据，并将它们放置在标准的磁盘系统上，OSD 不提供接口访问方式，Client 请求数据时用对象 ID、偏移进行数据读写。

② 智能分布。OSD 用其自身的 CPU 和内存优化数据分布，并支持数据的预取。由于 OSD 可以智能地支持对象的预取，从而可以优化磁盘的性能。

③ 每个对象数据的管理。OSD 管理存储在其它对象上的元数据，该元数据与传统的 inode 元数据相似，通常包括对象的数据块和对象的长度。而在传统的 NAS 系统中，这些元数据是由文件服务器提供的，对象存储架构将系统中主要的元数据管理工作由 OSD 来完成，降低了 Client 的开销。

(3) 元数据服务器（Metadata Server，MDS）

(4) MDS 控制 Client 与 OSD 对象的交互，为客户端提供元数据，主要是文件的逻辑视图，包括文件与目录的组织关系、每个文件所对应的 OSD 等。主要提供以下几个功能：

① 对象存储访问。MDS 构造、管理描述每个文件分布的视图，允许 Client 直接访问对象。MDS 为 Client 提供访问该文件所含对象的能力，OSD 在接收到每个请求时先验证该能力，然后才可以访问。

② 文件和目录访问管理。MDS 在存储系统上构建一个文件结构，包括限额控制、目录和文件的创建和删除、访问控制等。

③ Client Cache 一致性。为了提高 Client 性能，在对象存储系统设计时通常支持 Client 的 Cache。由于引入 Client 方的 Cache，带来了 Cache 一致性的问提，MDS 支持基于 Client 的文件 Cache，当 Cache 的文件发生改变时，将通知 Client 刷新 Cvashe，从而防止 Cache 不一致引发的问提。

(4)对象存储系统的客户端 Client

为了有效支持 Client 支持访问 OSD 上的对象，需要在计算节点实现对象存储系统的 Client。现有的应用对数据的访问大部分都是通过 POSIX 文件方式进行的，同时为了提高性能，也具有对数据的 Cache 功能和文件的条带功能。同时，文件系统必须维护不同客户端上 Cache 的一致性，保证文件系统的数据一致。文件系统访问流程：

① 客户端应用发出读请求；

② 文件系统向元数据服务器发送请求，获取要读取的数据所在的 OSD；

③ 然后直接向每个 OSD 发送数据读取请求；

④ OSD 得到请求后，判断要读取的 Object，并根据此 Object 的认证方式，对客户端进行认证，如果客户端得到收授权，则将 Object 的数据返回给客户端；

⑤ 文件系统收到 OSD 返回的数据以后，读操作完成。

三、实验环境

本次实验采用的是 Vmware Ubuntu18.04LTS 本地化部署。

配置如下：

处理器	Intel(R) Core(TM) i7-9750H CPU
机带 RAM	16.0 GB (15.9 GB 可用)
系统类型	64 位操作系统, 基于 x64 的处理器

具体环境如下（服务端&客户端）：

软件名	版本号
Gcc	7.5.0
Python	3.6
Python-swiftclient	3.13.1
Openstack-swift	x.x.x
Swift-bench	x.x.x

四、实验内容

实验的主要内容为，在 Linux 服务器上安装 Docker，在 Docker 上配置并部署单节点的 Openstack Swift，部署 swift-bench 性能测试软件。在本地部署 python-swiftclient 功能测试客户端，借用了往届学长的一个基于 Flask 的 Web 客户端。

4.1 对象存储技术实践

(1)在 Ubuntu 服务器上安装 Docker，在 Docker 上配置并部署一个单节 Openstack Swift。

(2)在本地上部署 python-swiftclient 和 CloudBerry 两个分别是命令行和图形化的客户端，进行功能性的测试。

(3)使用往届学长基于 Flask，使用 Python 语言调用 python-swiftclient 提供的 API 开发的一个轻量级的 Web 客户端，进行测试。

4.2 对象存储性能分析

使用 swift-bench 配合脚本进行测试。

五、实验过程

5.1.1 服务端部署 Openstack-swift

首先使用 curl -fsSL https://get.docker.com | bash -s docker --mirror Aliyun 进行一键化安装 docker。

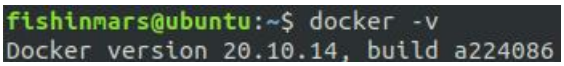

图 5.1 docker 安装成功

如图 5.1 所示，当前 docker 版本为 20.10.14, build a224086

然后按照实验中所给的 openstack-swift 容器开盒即用版地址，clone 仓库到本地，依此执行以下指令进行部署单节点。

- docker build -t openstack-swift-docker .
- docker run -v /srv --name SWIFT_DATA busybox
- docker run -d --name openstack-swift -p 12345:8080 --volumes-from SWIFT_DATA -t openstack-swift-docker

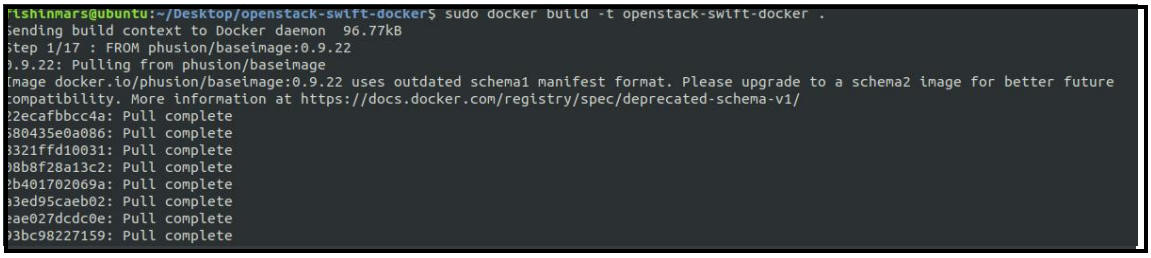

图 5.2 .

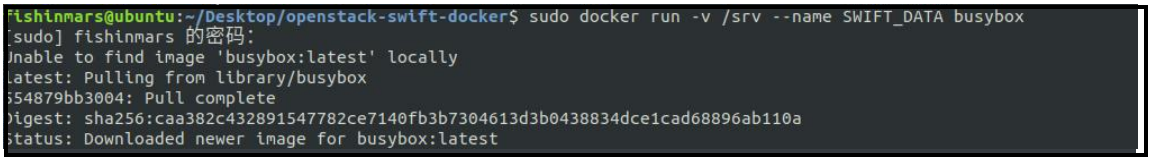

图 5.3

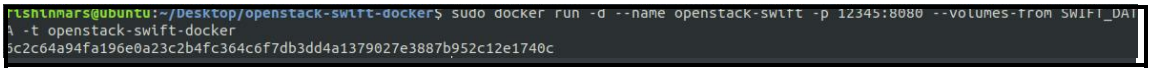

图 5.4

然后运行 docker ps，可以看到已经成功部署的节点

图 5.5

5.1.2 客户端部署 python-swiftclient 以及往届学长 Web 端测试

使用 python3 自带的 pip 指令即可完成安装，pip3 install python-swiftclient

安装完成后可以使用 swift -A http://127.0.0.1:12345/auth/v1.0 -U test:tester -K testing stat 可以测试成功部署后的节点，通过图 5.6 可以看到已经成功部署后的服务器 stat。

图 5.6

通过 clone 学长的库，可以获取一个轻量级的 Web 客户端，这个客户端基于社区内 python-swiftclient 提供的 API，使用 Python 语言，基于 Flask 框架作为后端，在 Bootstrap 4 规范上使用 Jinjia 模板作为 HTML 前端的控制，完成一个轻量级的 Web 端的 Swift 客户端，支持包括 CURD、生成临时下载链接在内的功能。启动后即可使用该 Web 端进行管理。在填写了相关内容后点击 Go 按钮即可进入管理界面。

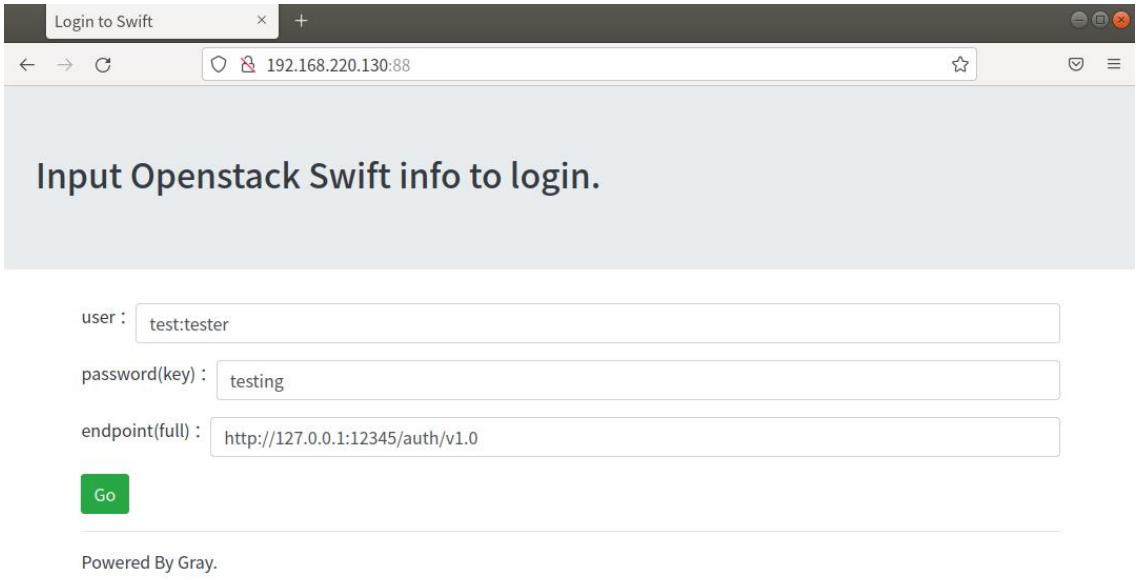

图 5.7

登陆之后，会进入到默认的 Container 上，上方是一个上传文件的界面块，下方是一个卡片列表，遍历展示这个 Container 内部的所有文件，展示包括其名字，hash 值，修改时间，文件大小，文件类型。卡片上有三个按钮，分别对应下载、获得临时下载链接、删除三个功能。

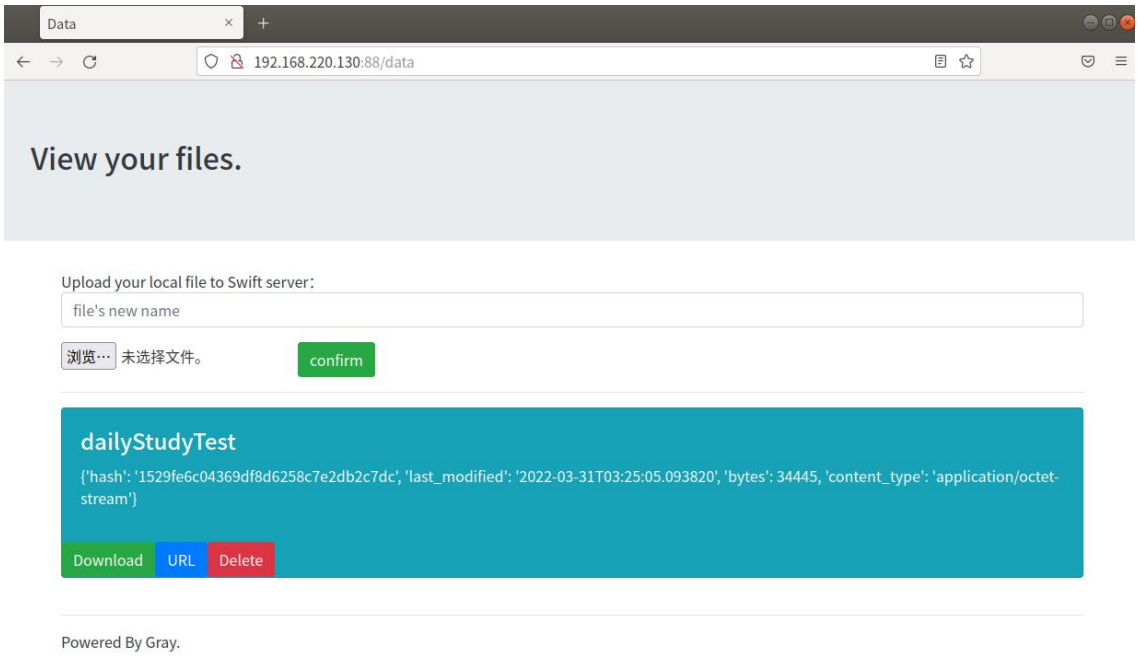

图 5.8

接下来对以下功能进行测试，首先是上传文件选取 gitPull.bat 进行上传，重命名 test1 后可以看到成功上传。

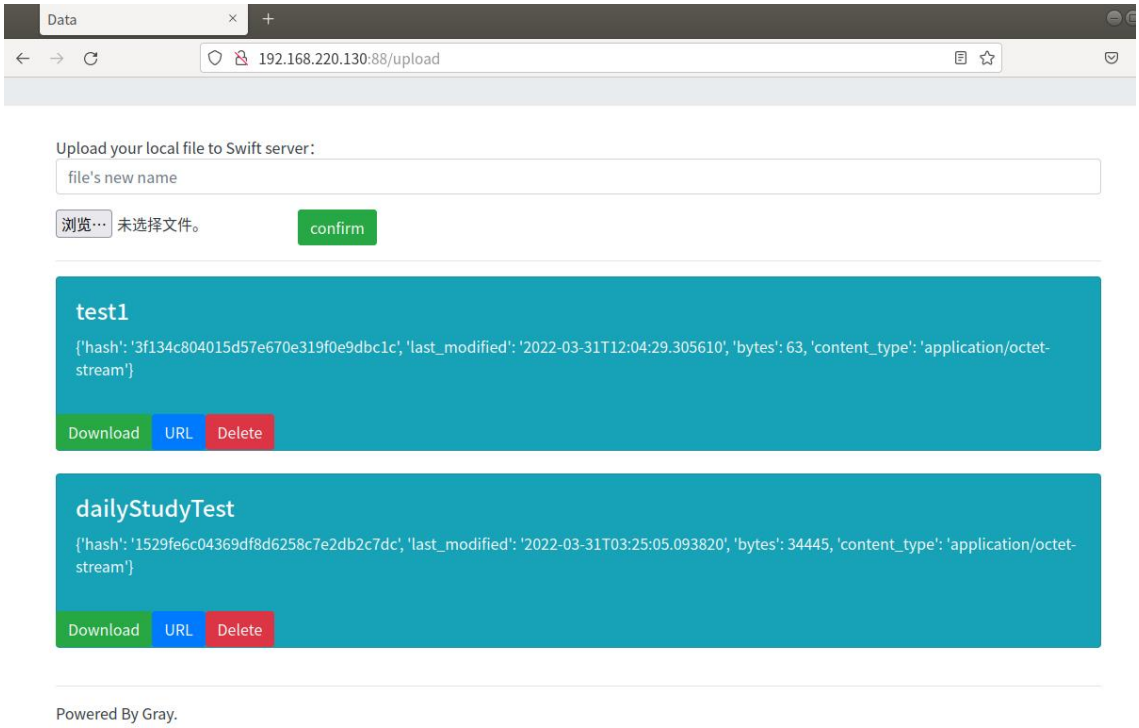

图 5.9

接下来测试下载，并使用自带的 diff 指令对下载文件和源文件进行比对，无异常。

```
fishinmars@ubuntu:~/Downloads$ diff test1 /home/fishinmars/Desktop/daily-study/gitPull.bat
fishinmars@ubuntu:~/Downloads$
```

图 5.10

接下来测试删除，在后台也是可以看到成功接受到了所有指令并进行相应的操作。

```
<FileStorage: 'gitPull.bat' ('application/x-msdos-program')>
192.168.220.130 - - [31/Mar/2022 20:04:29] "POST /upload HTTP/1.1" 200 -
192.168.220.130 - - [31/Mar/2022 20:04:29] "GET /static/bootstrap.min.css HTTP/1.1" 304 -
192.168.220.130 - - [31/Mar/2022 20:04:29] "GET /static/jquery.min.js HTTP/1.1" 304 -
192.168.220.130 - - [31/Mar/2022 20:04:29] "GET /static/bootstrap.min.js HTTP/1.1" 304 -
192.168.220.130 - - [31/Mar/2022 20:04:29] "GET /static/popper.min.js HTTP/1.1" 304 -
test1
True
192.168.220.130 - - [31/Mar/2022 20:05:06] "POST /download HTTP/1.1" 200 -
test1
192.168.220.130 - - [31/Mar/2022 20:07:21] "POST /delete HTTP/1.1" 200 -
192.168.220.130 - - [31/Mar/2022 20:07:21] "GET /static/bootstrap.min.css HTTP/1.1" 304 -
192.168.220.130 - - [31/Mar/2022 20:07:21] "GET /static/popper.min.js HTTP/1.1" 304 -
192.168.220.130 - - [31/Mar/2022 20:07:21] "GET /static/bootstrap.min.js HTTP/1.1" 304 -
192.168.220.130 - - [31/Mar/2022 20:07:21] "GET /static/jquery.min.js HTTP/1.1" 304 -
```

图 5.11

5.2 对象存储性能分析

5.2.1 Swift-Bench 测试

通过 pip install swift-bench 进行安装，成功安装后也可以看到 swiftbench 的初始 conf 为并发量 10，文件大小 1B，文件数量 10000

```
[bench]
auth = http://localhost:8080/auth/v1.0
user = test:tester
key = testing
concurrency = 10
object_size = 1
num_objects = 1000
num_gets = 10000
delete = yes
auth_version = 1.0
policy_name = gold
```

图 5.12

运行一次 swiftbench，并设置为 100 并发量，对象大小为 1KB，写入次数默认为 1000，读取次数默认为 10000。运行结果如图 5.13 所示，可以看到下载速度为 340.9/s，上传速度为 198.4/s

```
fishinmars@ubuntu:~$ swift-bench -A http://127.0.0.1:12345/auth/v1.0 -U test:tester -K testing -s 1024 -c 100
swift-bench 2022-03-31 20:17:02,627 INFO Auth version: 1.0
swift-bench 2022-03-31 20:17:03,039 INFO Auth version: 1.0
swift-bench 2022-03-31 20:17:05,050 INFO 303 PUTS [0 failures], 151.5/s
swift-bench 2022-03-31 20:17:08,091 INFO 1000 PUTS **FINAL** [0 failures], 198.4/s
swift-bench 2022-03-31 20:17:08,091 INFO Auth version: 1.0
swift-bench 2022-03-31 20:17:10,120 INFO 618 GETS [0 failures], 308.0/s
swift-bench 2022-03-31 20:17:25,124 INFO 5771 GETS [0 failures], 339.3/s
swift-bench 2022-03-31 20:17:37,444 INFO 10000 GETS **FINAL** [0 failures], 340.9/s
swift-bench 2022-03-31 20:17:37,445 INFO Auth version: 1.0
swift-bench 2022-03-31 20:17:39,465 INFO 400 DEL [0 failures], 200.0/s
swift-bench 2022-03-31 20:17:41,809 INFO 1000 DEL **FINAL** [0 failures], 230.2/s
swift-bench 2022-03-31 20:17:41,809 INFO Auth version: 1.0
```

图 5.13

通过 Python 脚本进行批量化执行多组测试并提取输出中总结的平均操作速率后整理测试结果表如下。

表 1

对象大小	并发数	写入操作速率	读取操作速率	删除操作速率	写入速度（Byte/s）	读取速度（Byte/s）	删除速度（Byte/s）
1B	10	175.49	298.54	191.59	175.49	298.54	191.59
1KB	10	163.3	304.98	192.74	167219.2	312299.52	197365.76
15KB	10	150.19	248.4	171.58	2306918.4	3815424	2635468.8
256KB	10	122.36	185.15	156.63	32075940.3	48535961.6	41059614.72
512KB	10	101.66	150.42	177.33	53299119	78863400.5	92971991.04
1MB	10	86.25	114.31	189.29	90439680	119862722.1	198484951
4MB	10	32.2	37.72	164.22	135056588.8	158209147.8	688788603.8
8MB	10	17.71	20.93	156.63	148562248.6	175573565.9	1313907672
16MB	10	10.81	10.35	147.66	181361704.5	173644185.6	2477323714
32MB	10	6.21	5.29	163.99	208373021.8	177502946.2	5502591305
1MB	1	55.89	110.4	147.43	58604913.1	115762790.4	154591559.7
1MB	10	86.25	114.31	189.29	90439680	119862722.1	198484951
1MB	20	81.88	107.64	189.06	85857403.8	112868721.1	198243778.6
1MB	30	83.49	101.89	180.32	87545610.7	106839409.1	189079224.3
1MB	40	80.73	106.03	175.26	84651541.4	111180514.2	183773429.8
1MB	60	81.65	105.34	177.56	85616230.4	110456996.3	186185154.6
1MB	80	75.9	95.91	174.34	79586918.4	100568923.7	182808739.8
1MB	100	61.41	105.11	182.16	64393051.7	110215822.9	191008604.2

六、实验总结

本次实验采用 Openstack Swift 作为服务端，python-swiftclient 以及学长的 WEB 端作为客户端，并用 swiftbench 来进行评测。

面向对象存储的入门实验也使我了解到了一种新的存储技术，并明白了其的优越性，以及为什么在已有基于块和文件的存储系统以及网络附加存储等技术后，仍需要面向对象存储技术。

总的来说，通过这次实验，对于对象存储的概念有了基本的认识，常用的对象存储工具有了初步的掌握，同时自己亲自动手测试了一些性能指标，培养了动手能力，也丰富了理论知识。

参考文献

[1] ZHENG Q, CHEN H, WANG Y等. COSBench: A Benchmark Tool for Cloud Object Storage Services[C]//2012 IEEE Fifth International Conference on Cloud Computing. 2012: 998－999.
[2] ARNOLD J. OpenStack Swift[M]. O’Reilly Media, 2014.
[3] WEIL S A, BRANDT S A, MILLER E L等. Ceph: A Scalable, High-performance Distributed File System[C]//Proceedings of the 7th Symposium on Operating Systems Design and Implementation. Berkeley, CA, USA: USENIX Association, 2006: 307－320.
[4] Dean J, Barroso L A. Association for Computing Machinery, 2013. The Tail at Scale[J]. Commun. ACM, 2013, 56(2): 74－80.
[5] Delimitrou C, Kozyrakis C. Association for Computing Machinery, 2018. Amdahl’s Law for Tail Latency[J]. Commun. ACM, 2018, 61(8): 65－72.
（可以根据实际需要更新调整）