2017 级

《物联网数据存储与管理》课程

实 验 报 告

姓　名　林宇华

学　号　U201714736

班　号　物联网 1701 班

日　期　2020.05.27

目　　录

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，架设实际应用，示范主要功能。

二、实验背景

①对象存储需求：随着互联网，Web 应用创建出数百亿的小文件;人们上传海量的照片、视频、音乐，Facebook 每天都新增数十亿条内容，人们每天发送数千亿封电子邮件。据 IDC 统计未来在 10 年间数据将增长 44 倍，到 2020 年全球数据将增加到 35ZB，其中 80%是非结构化数据，且大部分是非活跃数据；面对如此庞大的数据量，仅具备 PB 级扩展能力的块存储(SAN)和文件存储(NAS)显得有些无能为力：通常块存储(SAN)的一个 LUN 容量仅数 TB。单个文件系统最优性能情况下支持的文件数量通常只在百万级别。人们需要一种全新的架构的存储系统，这种存储系统需要具备极高的可扩展性，能够满足人们对存储容量 TB 到 EB 规模的扩展的需求；2006 年 Amazon 发布 AWS, S3 服务及其使用的 REST、SOAP 访问接口成为对象存储的事实标准。Amazon S3 成功为对象存储注入云服务基因。

②对象存储的特性：

1、优秀的扩展性

扁平化的数据结构允许对象存储容量从 TB 级扩展到 EB 级，管理数十个到百亿个存储对象，支持从数字节(Byte)到数万亿字节(TB)范围内的任意大小对象，解决了文件系统复杂的 iNode 的机制带来的扩展性瓶颈，并使得对象存储无需像 SAN 存储那样管理数量庞大的逻辑单元号(LUN)。对象存储系统通常在一个横向扩展(或网格硬件)架构上构建一个全局的命名空间，这使得对象存储非常适用在云计算环境中使用。某些对象存储系统还可支持升级、扩容过程中业务零中断。

2、基于策略的自动化管理

由于云环境中的数据往往是动态、快速增长的，所以基于策略的自动化将变得非常重要。对象存储支持从应用角度基于业务需求设置对象/容器的属性(元数据)策略，如数据保护级别，保留期限，合规状况，远程复制的份数等。这使得对象存储具备云的自服务特征同时，有效的降低运维管理的成本，使得客户在存储容量从 TB 增长到 ZB 时，运维管理成本不会随之飙升。

3、多租户技术

多租户特性可以使用同一种架构,同一套系统为不同用户和应用提供存储服务,并分别为这些用户和应用设置数据保护、数据存储策略，并确保这些数据之间相互隔离。

③对象存储结构如图 2-1 所示

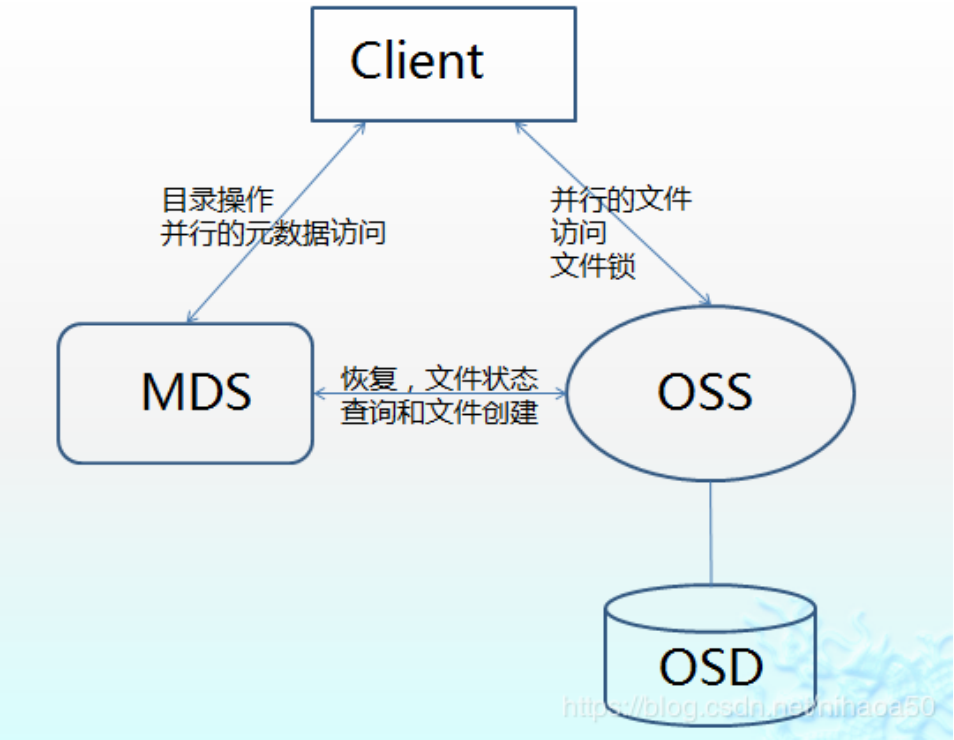

图 2-1 对象存储结构

④实验所用相关平台介绍

a) mock-s3：适用于 AWS S3 的模拟 API，模仿 aws-sdk 的 S3 包装器的 API。支持分段上传以及 getObject 从 AWSRequest 返回的流式传输。completeMultipartUpload， createBucket， createMultipartUpload，deleteObject，deleteObjects，getObject，putObject，uploadPart。还有许多其他功能不受支持，例如匹配错误以及 S3 对象所需的所有返回属性，但这已经很接近了。

b) s3cmd：S3cmd 是一款免费的命令行工具和客户端，用于在 Amazon S3 和其他使用 S3 协议的云存储服务提供商中上传，检索和管理数据。

c) s3bench：s3bench 可以针对 S3 兼容端点运行非常基本的吞吐量基准测试。它执行一系列 put 操作，然后执行一系列 get 操作并显示相应的统计信息。该工具使用 AWS Go SDK。

三、实验环境

（用图、表描述实验系统软硬件构成，系统内组件关系）

硬件环境	处理器：Intel®Core(TM) i5-6300HQ CPU @2.30GHZ 内存：8.00GB
虚拟机版本	Oracle VM VirtualBox5.2.26
虚拟机操作系统信息	Linux version 4.15.0-101-generic Ubuntu 18.04.4 LTS
Java 版本	OpenJDK 1.8.0_252
Python 版本	Python 3.6.9
Go 版本	Go version go1.14.2 linux/amd64
Git 版本	Git version 2.17.1

服务器端	mock-s3
客户端	s3cmd
评测工具	s3bench

四、实验内容

4.1 对象存储技术实践

1. 搭建基础环境，包括 java、python、git、go 环境的配置
2. 搭建对象存储服务器端和客户端：下载 mock-s3 和 s3cmd
3. 开启服务器，配置 s3cmd，创建 bucket 并上传文件
4. 尝试使用 zfile

4.2 对象存储性能分析

1. 安装 s3bench
2. 修改 s3bench 运行脚本
3. 使用 s3bench 进行测试
4. 探究对象大小如何影响传输率
5. 探究并发数如何影响传输率
6. 探究并发数如何让影响访问延迟

五、实验过程

1) 环境搭建：这里主要记录一下 go 语言环境配置：首先在 GO 中文语言网取得相应对的 go 安装包，下载安装，然后进行 GOROOT、GOPATH 配置，搭建好的环境如图 4-1 所示

```
lyh@lyh-VirtualBox:~$ git version
git version 2.17.1
lyh@lyh-VirtualBox:~$ go version
go version go1.14.2 linux/amd64
lyh@lyh-VirtualBox:~$ java -version
openjdk version "1.8.0_252"
OpenJDK Runtime Environment (build 1.8.0_252-8u252-b09-1~18.04-b09)
OpenJDK 64-Bit Server VM (build 25.252-b09, mixed mode)
lyh@lyh-VirtualBox:~$ python3 --version
Python 3.6.9
```

图 4-1 环境配置结果展示

2) 安装对象存储服务器端 mock-s3：克隆老师所给的 obs-tutorial 中的 mock-s3 到本地，即可完成安装，使用详见 mock-s3github 页，注意，要用 python3 进行运行。键入：cd mock_s3 ； python3 main.py --hostname 0.0.0.0 --port 9000 --root ./root
3) 安装并使用对象存储客户端 s3cmd
   a) 使用 pip3 下载 s3cmd：键入 sudo pip3 install s3cmd
   b) 对 s3cmd 进行配置：键入 s3cmd –configure，配置号后如图 4-2 所示

```
HTTP Proxy server name:

New settings:
  Access Key: hust
  Secret Key: hust_U201714736
  Default Region: US
  S3 Endpoint: 127.0.0.1:9000
  DNS-style bucket+hostname:port template for accessing a bucket: 9000
  Encryption password: hust_U201714736
  Path to GPG program: /usr/bin/gpg
  Use HTTPS protocol: False
  HTTP Proxy server name:
  HTTP Proxy server port: 0

Test access with supplied credentials? [Y/n] Y
Please wait, attempting to list all buckets...
ERROR: Test failed: [Errno 111] Connection refused

Retry configuration? [Y/n] n
```

图 4-2 s3cmd 配置结果图

c) 配置文件存在/home/maggie/.s3cfg 中，如果需要修改则打开修改

d) 创建 bucket，键入：s3cmd mb s3://lyh，结果如图 4-3 所示

```
lyh@lyh-VirtualBox:~$ s3cmd mb s3://lyhbucket
Bucket 's3://lyhbucket/' created
```

图 4-3 make bucket 结果

e) 上传文件，键入 s3cmd put /home/lyh/Go_learn/demo.go s3://lyhbucket, 结果如图 4-4 所示，接着在浏览器输入 127.0.0.1：9000/lyhbucket，浏览器显示如图 4-5 所示（如果

```
yh@lyh-VirtualBox:~$ s3cmd put /home/lyh/Go_learn/demo.go s3://lyhbucket
pload: '/home/lyh/Go_learn/demo.go' -> 's3://lyhbucket/demo.go'  [1 of 1]
72 of 72   100% in    0s    24.69 KB/s  done
yh@lyh-VirtualBox:~$
```

图 4-4 put 结果

```
-<ListBucketResult>
   <Name>lyhbucket</Name>
   <Prefix/>
   <Marker/>
   <MaxKeys>1000</MaxKeys>
   <IsTruncated>false</IsTruncated>
  -<Contents>
     <Key>demo.go</Key>
     <LastModified>2020-05-11T09:10:51.000Z</LastModified>
     <ETag>"fa915842fe8a4f38b9c919834b3f55cb"</ETag>
     <Size>72</Size>
     <StorageClass>STANDARD</StorageClass>
    -<Owner>
       <ID>123</ID>
       <DisplayName>MockS3</DisplayName>
     </Owner>
   </Contents>
 </ListBucketResult>
```

图 4-5 浏览器显示

4) 安装并使用 s3bench 进行实验

a) 安装：在这记录一下安装过程：键入 go get -u github.com/igneous-systems/s3bench,由于需要的 aws-sdk-go 过大，从而需要 ctrl+d 退出，通过 gitee 下载 aws，然后在 go/src/github.com 下解压，并执行 go install（相关首次使用要 install 一下），同理装好 golang.org 下的 build 和 net，然后可以用 go get 下载 jmespath（这是执行时，错误提示加上的），最后 install s3bench，相关生成文件在 go/bin 下

b) 运行脚本 run-s3bench.sh，具体代码见附录，命令为：sh run-s3bench.sh 执行性能分析实验

i.探究对象大小如何影响传输率，主要看写入和读取的速率，如图 4-6、图 4-7、图 4-8 所示

```
Test parameters
endpoint(s):      [http://127.0.0.1:9000]
bucket:           lyhbucket
objectNamePrefix: lyhbucket
objectSize:       0.0039 MB
numClients:       8
numSamples:       256
verbose:       %!d(bool=false)


Results Summary for Write Operation(s)
Total Transferred: 1.000 MB
Total Throughput:  0.64 MB/s
Total Duration:    1.569 s
Number of Errors:  0
-----------------------------------
```

图 4-6 4K 大小

```
Test parameters
endpoint(s):      [http://127.0.0.1:9000]
bucket:           lyhbucket
objectNamePrefix: lyhbucket
objectSize:       0.0156 MB
numClients:       8
numSamples:       256
verbose:       %!d(bool=false)


Results Summary for Write Operation(s)
Total Transferred: 4.000 MB
Total Throughput:  3.82 MB/s
Total Duration:    1.047 s
Number of Errors:  0
-----------------------------------
```

图 4-7 16K 大小

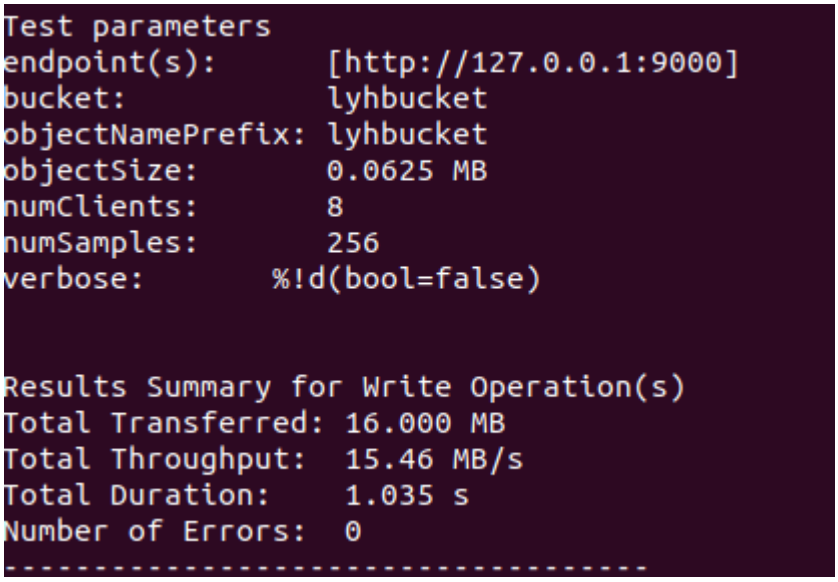

图 4-8 64K 大小

ii.探究并发数如何影响传输率（大小为 1024 * 32），主要看写入和读取的速率指标，结果如图 4-9、图 4-10、图 4-11 所示

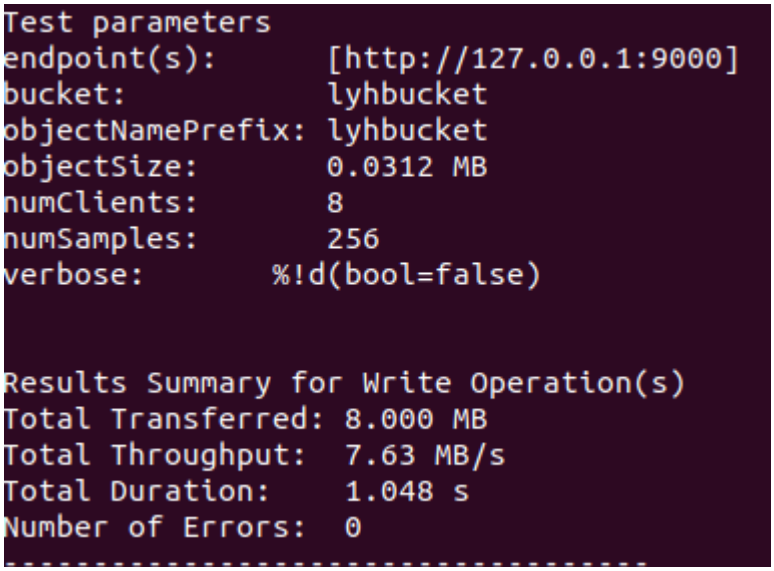

图 4-9 8 个客户端

```
Test parameters
endpoint(s):      [http://127.0.0.1:9000]
bucket:           lyhbucket
objectNamePrefix: lyhbucket
objectSize:       0.0312 MB
numClients:       32
numSamples:       256
verbose:       %!d(bool=false)


Results Summary for Write Operation(s)
Total Transferred: 8.000 MB
Total Throughput:  3.60 MB/s
Total Duration:    2.221 s
Number of Errors:  0
-----------------------------------
```

图 4-10 32 个客户端

```
Test parameters
endpoint(s):      [http://127.0.0.1:9000]
bucket:           lyhbucket
objectNamePrefix: lyhbucket
objectSize:       0.0312 MB
numClients:       64
numSamples:       256
verbose:       %!d(bool=false)


Results Summary for Write Operation(s)
Total Transferred: 8.000 MB
Total Throughput:  3.40 MB/s
Total Duration:    2.354 s
Number of Errors:  0
-----------------------------------
```

图 4-11 64 个客户端

iii.探究并发数如何影响访问延迟，文件大小为 1024 * 32，我选择 read times 99 th %file(99 百分位文件)为观测点，结果如图 4-12、图 14-13、图 4-14 所示

```
Results Summary for Read Operation(s)
Total Transferred: 8.000 MB
Total Throughput:  7.84 MB/s
Total Duration:    1.020 s
Number of Errors:  0
-----------------------------------
Read times Max:       1.019 s
Read times 99th %ile: 0.025 s
Read times 90th %ile: 0.017 s
Read times 75th %ile: 0.015 s
Read times 50th %ile: 0.012 s
Read times 25th %ile: 0.010 s
Read times Min:       0.004 s
```

图 4-12 8 个客户端

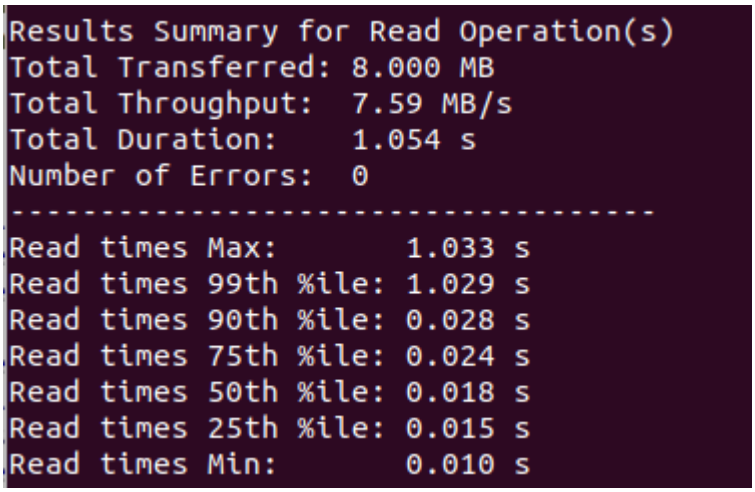

图 4-13 16 个客户端

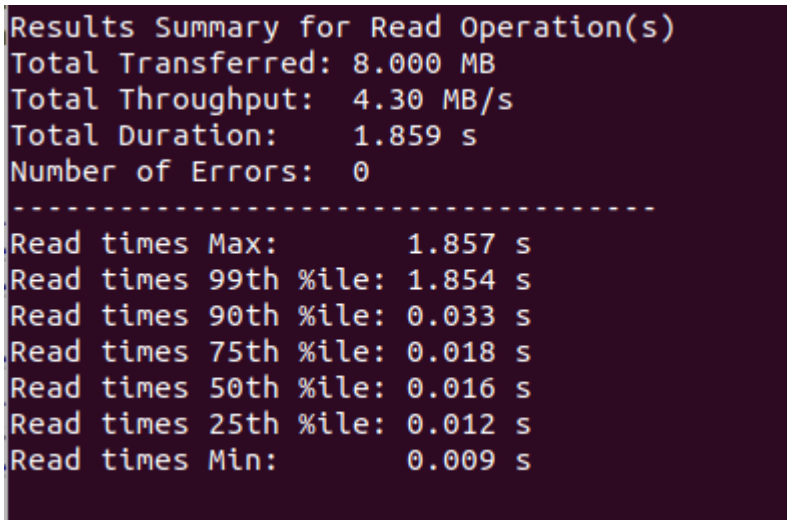

图 4-14 32 个客户端

实验数据补充：（写完报告后还有时间，就多做了几次测试，想找出阈值，数据以表格形式给出，实验图片(S 前缀)随报告一起提交）

i.对象大小

对象大小	读取速率(MB/S)
512K	79.57
1M	177.11
1.5M	182.72
1.75M	189.97
2M	167.59
4M	131.32
8M	37

ii.并发数影响传输速率（注意客户端数不能大于案例数）（B 前缀）

客户端数量	读取速率（MB/S）
1	212.52
2	276.65
4	276.31

8	226.24
16	125.19
64	137.68
128	94.62

iii.并发数影响延迟（Y 前缀）

并发数	99 %ile readtime
1	0.011
2	0.020
4	0.038
8	1.025
16	1.050
32	1.230
64	1.894
128	3.054

六、实验总结

由图 4-1 环境配置结构图知，实验基本环境已经配置完成

由图 4-5 浏览器显示所示，对象存储客户端以及服务器端都已正确安装，并能连接成功

对象大小如何影响传输率：如图 4-6、图 4-7、图 4-8 所示，传输速率随着文件大小的增加而增大，但经过老师课上的讲解，得知这不是线性增大的，当文件大小达到某一临界值是，传输速率基本不变，这也符合基于常识的推断。加上后来的实验数据推断，确实有这个阙值存在，该实验中的对象大小的阙值在 1.75M-2M 之间。

探究并发数如何影响传输率：由图 4-9、图 4-10、图 4-11 得，并发数越多，传输率越低，但我觉得应该是在一定范围内才会有这线性关系，在某一阙值之前，传输率应该是随着客户端数增加而增加的。在本实验中，当客户端数增加到 64 时，可以感受到明显的延迟。而补充的实验数据也能说明我猜想的观点，但实验数据中的 64 个客户端的数据存在偏差，可能是由于当时的网络条件不稳定所造成的。

探究并发数如何让影响访问延迟：由图 4-12、图 4-13、图 4-14 得，观察 99 百分位文件得传输有，并发数越多，延迟越明显（从至少 8 个客户端开始，有这种线性关系），同并发数影响传输率分析类似。由补充的实验数据，可以得到更细致的分析，由于不确定性，相同条件下得到 99%ile 的指标并不相同，但是，仍然可以大体观察出并发数达到在一定数量会有比较明显的延迟现象。

七、心得体会

总的来说，这次得实验还是比较简单得，可能是自己没有花时间进行深入的探究，只是完成了老师每一堂课规定的实验任务，但还是学到了新的知识，让我切身体会到了对象存储的整个过程，虽然其中环境的配置（如 go 语言），平台的搭建（如 s3bench）都花费了不少时间，但是也增长了知识，还有要感谢老师课前的讲授，细致、全面，给了实验过程的一个示

范，从而让自己知道实验该怎么做，提高了实验效率，对于最后的性能分析，自己只是简单的对比了一下指标，没有深入研究。总而言之，需要学习的地方还有很多，俗话说“师傅领进门，修行靠个人”，过后也要不断学习、不断积累！

附录

Run-s3bench.sh:

```
#!/bin/sh

# Locate s3bench

s3bench=~/go/bin/s3bench
#s3bench=/home/lyh/go/bin/s3bench

if [ -n "$GOPATH" ]; then
    s3bench=$GOPATH/bin/s3bench
fi
# -accessKey        Access Key
# -accessSecret     Secret Key
# -bucket=loadgen   Bucket for holding all test objects.
# -endpoint=http://127.0.0.1:9000 Endpoint URL of object storage service being tested.
# -numClients=8     Simulate 8 clients running concurrently.
# -numSamples=256   Test with 256 objects.
# -objectNamePrefix=loadgen Name prefix of test objects.
# -objectSize=1024          Size of test objects.

$s3bench \
    -accessKey=hust \
    -accessSecret=hust_U201714736 \
    -bucket=lyhbucket \
    -endpoint=http://127.0.0.1:9000 \
    -numClients=32 \
    -numSamples=256 \
    -objectNamePrefix=lyhbucket \
    -objectSize=$(( 1024*32 ))

# build your own test script with designated '-numClients', '-numSamples' and '-objectSize'
# 1. Use loop structure to generate test batch (E.g.: to re-evaluate multiple s3 servers under
the same configuration, or to gather data from a range of parameters);
# 2. Use redirection (the '>' operator) for storing program output to text files;
# 3. Observe and analyse the underlying relation between configuration parameters and
performance metrics.
```

参考文献

[1] ARNOLD J. OpenStack Swift[M]. O’Reilly Media, 2014.
[2] 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.
[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.
（可以根据实际需要更新调整）