



2015 级

《物联网数据存储与管理》课程

## 实 验 报 告

姓 名 李自东

学 号 U201514876

班 号 物联网 1501 班

日 期 2018.04.23

# 目 录

1	实验目的 .....	1
2	实验背景 .....	1
3	实验环境 .....	1
4	实验内容 .....	2
5	实验过程 .....	2
	5.1 对象存储技术实践 .....	4
	5.2 对象存储性能分析 .....	7
6	实验总结 .....	8
7	参考文献 .....	8

## 1 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。

## 2 实验背景

现在的个人计算机，主要的存储设备都是以机械硬盘与固态硬盘为主，机械硬盘具备容量大，价格比较低廉的特点，是现在的主流存储设备，固态硬盘读写速度远超机械硬盘，但是价格比较昂贵，主要在一些需求比较高的用户手中进行操作，我们采用的存储系统也基本上是基于块和基于文件存储系统，这些问题操作系统就已经帮我们解决好，不需要我们额外的进行设计。

有大量的基于块和基于文件的存储系统可供选择，一个明显的问题是，我们为什么还需要对象存储呢？块和文件都是成熟且经过验证的，所以也许看起来好像他们可以增强以满足日益增长的分布式生态系统的需求。

基于块的存储系统，磁盘通过底层存储协议访问，像 SCSI 命令，开销很小而且没有其它额外的抽象层。这是访问磁盘数据最快的方式，所有高级别的任务，像多用户访问、共享、锁定和安全通常由操作系统负责。换句话讲，基于块的存储关心所有底层的问题，但其它事情都要依靠高层的应用程序实现。所有的对象存储拥有基于块存储的节点，利用对象存储软件集合提供所有其它的功能。

基于块的存储系统是对象存储系统的补充，而基于文件的存储系统一般被认为是直接的竞争者。横向扩展的 NAS 系统的关键属性就是扩展性，对象存储也是这样，通过增加节点实现水平扩展。但由于 NAS 系统是基于分层文件结构的有限的命名空间，它们对于有着接近无限扩展能力的、具有扁平结构的存储来讲，所受的约束更多，对象受到对象 ID 的位数限制。尽管限制多多，但横向扩展的 NAS 系统仍然具备对象存储的诸多特性，而其欠缺的功能，例如表征状态转移协议的支持，厂商们正在快速的完善中，这样他们就可以把横向扩展的 NAS 系统划归到对象存储的类别中了。

对象存储的特点，在于他可以在一个持久稳固且高度可用的系统中存储任意的对象，且独立于虚拟机的实例之外，应用和用户可以在对象存储中使用简单的 API 访问数据，这些通常都基于表属性状态转移（REST）架构，同样也有面向编程语言的界面。

简单一点来说，对象存储有点像我们平常使用的网盘，只不过网盘的使用者主要是我们个人，但是对象存储的使用者主要是企业级，他们存储的数据量更大，对于数据的性能要求也更高，是为这些更大的玩家发明的新一代的存储技术。

## 3 实验环境

计算机硬件环境：

联想小新笔记本一台：

- 1) CPU: i7-6500U
- 2) 内存: 8G
- 3) 硬盘: 500G 机械硬盘
- 4) 虚拟机分配内存 2G, 磁盘 40G

操作系统：

Ubuntu16.04.3 LTS 版 64 位虚拟机环境

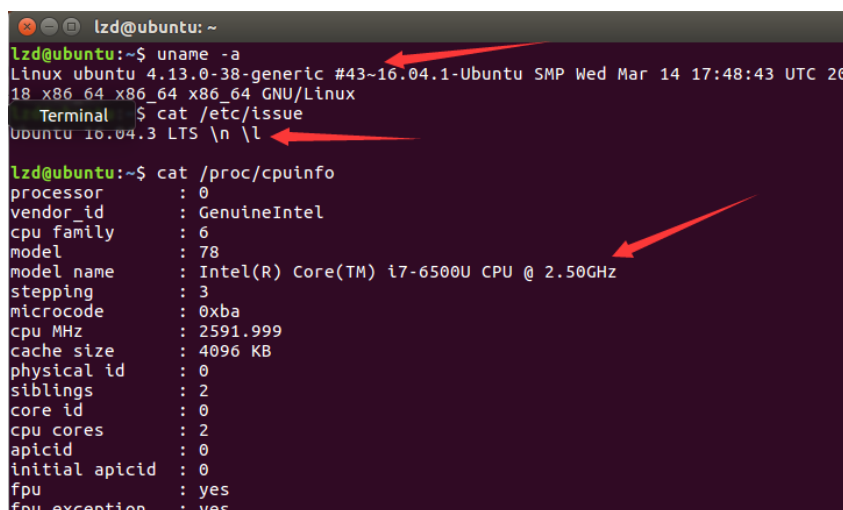
软件环境:

对象存储服务端: minio

对象存储客户端: mc

对象存储测试端: COSBENCH

系统配置如图 3-1 所示。



```
lzd@ubuntu: ~  
lzd@ubuntu:~$ uname -a  
Linux ubuntu 4.13.0-38-generic #43~16.04.1-Ubuntu SMP Wed Mar 14 17:48:43 UTC 20  
18 x86_64 x86_64 x86_64 GNU/Linux  
Terminal $ cat /etc/issue  
Ubuntu 16.04.3 LTS \n \l  
lzd@ubuntu:~$ cat /proc/cpuinfo  
processor       : 0  
vendor_id      : GenuineIntel  
cpu family     : 6  
model          : 78  
model name     : Intel(R) Core(TM) i7-6500U CPU @ 2.50GHz  
stepping       : 3  
microcode      : 0xba  
cpu MHz        : 2591.999  
cache size     : 4096 KB  
physical id    : 0  
siblings       : 2  
core id        : 0  
cpu cores      : 2  
apicid         : 0  
initial apicid : 0  
fpu            : yes  
fpu_exception  : yes
```

图 3-1 系统配置

## 4 实验内容

配置好实验所需要的各种环境,运行对象存储服务端和客户端,然后启动 COSBENCH,添加负载测试读写效率。

## 5 实验过程

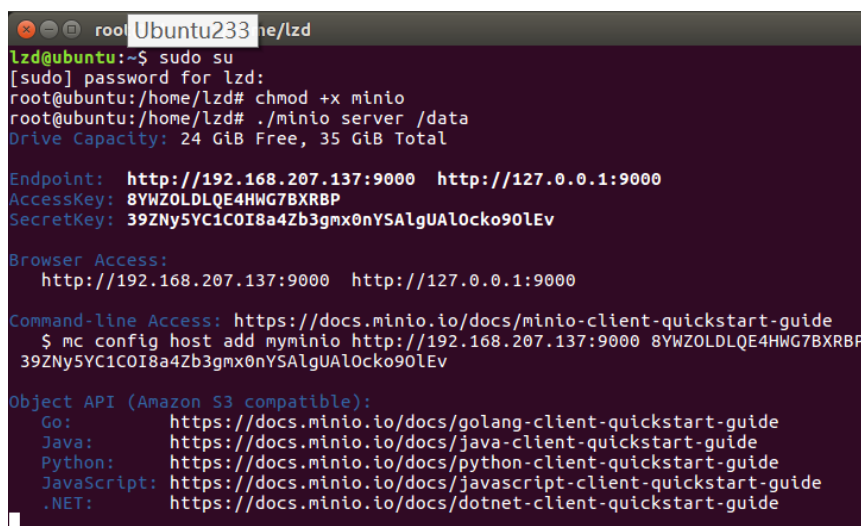
首先打开 minio 服务端。命令行操作:

以管理员权限运行以下命令:

```
chmod +x minio
```

```
./minio server /data
```

开启 minio 服务端如图 5-1 所示。



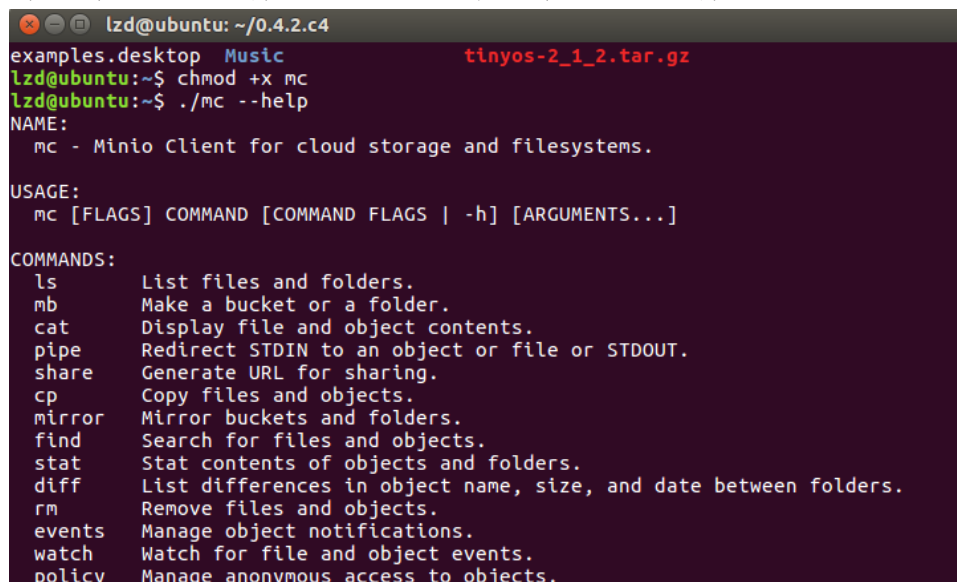
```
root@Ubuntu233: /home/lzd  
lzd@ubuntu:~$ sudo su  
[sudo] password for lzd:  
root@ubuntu:/home/lzd# chmod +x minio  
root@ubuntu:/home/lzd# ./minio server /data  
Drive Capacity: 24 GiB Free, 35 GiB Total  
  
Endpoint: http://192.168.207.137:9000 http://127.0.0.1:9000  
AccessKey: 8YWZOLDLQE4HWG7BXRBP  
SecretKey: 39ZNy5YC1COI8a4Zb3gmX0nYSA1gUAL0cko90LEv  
  
Browser Access:  
http://192.168.207.137:9000 http://127.0.0.1:9000  
  
Command-line Access: https://docs.minio.io/docs/minio-client-quickstart-guide  
$ mc config host add myminio http://192.168.207.137:9000 8YWZOLDLQE4HWG7BXRBP  
39ZNy5YC1COI8a4Zb3gmX0nYSA1gUAL0cko90LEv  
  
Object API (Amazon S3 compatible):  
Go: https://docs.minio.io/docs/golang-client-quickstart-guide  
Java: https://docs.minio.io/docs/java-client-quickstart-guide  
Python: https://docs.minio.io/docs/python-client-quickstart-guide  
JavaScript: https://docs.minio.io/docs/javascript-client-quickstart-guide  
.NET: https://docs.minio.io/docs/dotnet-client-quickstart-guide
```

图 5-1 minio 服务端

然后打开对象存储客户端，此处采用 mc 作为客户端。另外开一个终端，输入以下命令：

```
wget https://dl.minio.io/client/mc/release/linux-amd64/mc
chmod +x mc
./mc -help
```

第一条命令下载客户端，运行后面两个命令开启客户端，如图 5-2 所示。



```
lzd@ubuntu: ~/0.4.2.c4
examples.desktop Music tinyos-2_1_2.tar.gz
lzd@ubuntu:~$ chmod +x mc
lzd@ubuntu:~$ ./mc --help
NAME:
  mc - Minio Client for cloud storage and filesystems.

USAGE:
  mc [FLAGS] COMMAND [COMMAND FLAGS | -h] [ARGUMENTS...]

COMMANDS:
  ls      List files and folders.
  mb      Make a bucket or a folder.
  cat     Display file and object contents.
  pipe    Redirect STDIN to an object or file or STDOUT.
  share   Generate URL for sharing.
  cp      Copy files and objects.
  mirror  Mirror buckets and folders.
  find    Search for files and objects.
  stat    Stat contents of objects and folders.
  diff    List differences in object name, size, and date between folders.
  rm      Remove files and objects.
  events  Manage object notifications.
  watch   Watch for file and object events.
  policy  Manage anonymous access to objects.
```

图 5-2 开启对象存储客户端 mc

接下来打开对象存储测试部分 cos-bench 脚本：

下载安装 cos-bench 软件：

```
wget
https://github.com/intel-cloud/cosbench/releases/download/v0.4.2.c4/0.4.2.c4.zip
```

解压：

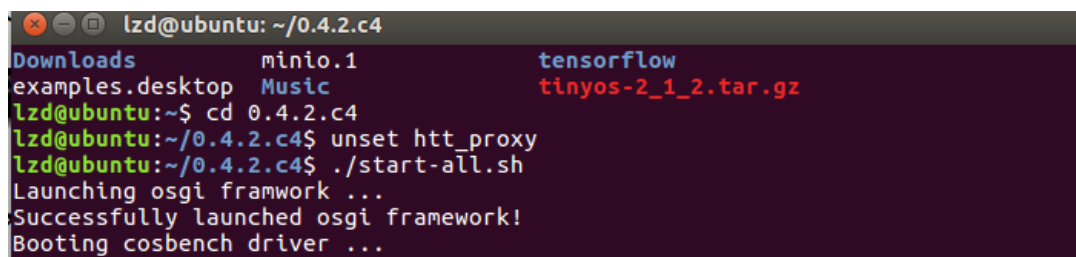
```
unzip 0.4.2.c4.zip
```

进入 cos-bench 目录，将脚本添加可执行权限：

```
cd 0.4.2.c4
```

```
chmod +x *.sh
```

启动 COSBench 之前，运行 `unset http_proxy`，运行 `./start-all.sh` 启动脚本启动 COSBench，如图 5-3 所示。



```
lzd@ubuntu: ~/0.4.2.c4
Downloads minio.1 tensorflow
examples.desktop Music tinyos-2_1_2.tar.gz
lzd@ubuntu:~$ cd 0.4.2.c4
lzd@ubuntu:~/0.4.2.c4$ unset http_proxy
lzd@ubuntu:~/0.4.2.c4$ ./start-all.sh
Launching osgi framework ...
Successfully launched osgi framework!
Booting cosbench driver ...
```

图 5-3 启动 COSBench

出现如下信息，启动成功！由图 5-4 可知，driver 的访问端口为 18088，service 的访问端口为 19088。

```
lzd@ubuntu: ~/0.4.2.c4
!!! Service will listen on web port: 18088 !!!

=====

Launching osgi framework ...
Successfully launched osgi framework!
Booting cosbench controller ...
..
Starting    cosbench-log_0.4.2    [OK]
..
Starting    cosbench-tomcat_0.4.2  [OK]
Starting    cosbench-config_0.4.2  [OK]
Starting    cosbench-core_0.4.2    [OK]
Starting    cosbench-core-web_0.4.2 [OK]
Starting    cosbench-controller_0.4.2 [OK]
Starting    cosbench-controller-web_0.4.2 [OK]
Successfully started cosbench controller!
Listening on port 0.0.0.0/0.0.0.0:19089 ...
Persistence bundle starting...
Persistence bundle started.

=====
!!! Service will listen on web port: 19088 !!!
```

图 5-4 启动成功

在浏览器中输入当前的地址，端口号为 19088，即可访问 COSBench 工作界面。



图 5-5 访问 COSBench 界面

## 5.1 对象存储技术实践

对 minio 服务端进行测试，点击 submit new workloads 添加负载进行测试，这里加载 workload-exmple.xml 进行测试，首先将 accesskey 和 secretkey 修改为 minio 服务端的对应内容，key 值如图 5-6 所示。

```
root@ubuntu:/home/lzd# ./minio server /data
Drive Capacity: 24 GiB Free, 35 GiB Total

Endpoint: http://192.168.207.137:9000 http://127.0.0.1:9000
AccessKey: 8YWZ0LDLQE4HWG7BXRBP
SecretKey: 39ZNy5YC1C0I8a4Zb3gmX0nYSA1gUA10cko90lEv
Browser Access:
http://192.168.207.137:9000 http://127.0.0.1:9000
```

图 5-6 密钥

修改后的负载如图 5-7 所示。

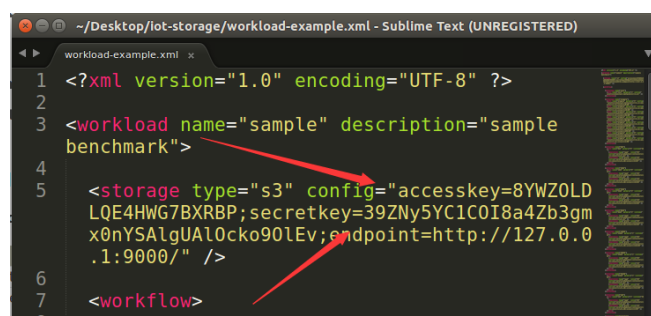


图 5-7 workload-example

测试结果如下：

① 127.0.0.1:19088/controller/workload.html?id=w4

⋮🔔🔍Search

COSBENCH - CONTROLLER WEB CONSOLE

time: Sun Apr 22 20:52:15 CST 2018  
version: 0.4.2.20160615

[index](#) -> workload

Workload

Basic Info

ID: w4 Name: sample Current State: finished

Submitted At: Apr 22, 2018 8:46:56 PM Started At: Apr 22, 2018 8:46:56 PM Stopped At: Apr 22, 2018 8:52:13 PM

[more info](#)

Final Result

General Report

Op-Type	Op-Count	Byte-Count	Avg-ResTime	Avg-ProcTime	Throughput	Bandwidth	Succ-Ratio
op1: init -write	0 ops	0 B	N/A	N/A	0 op/s	0 B/s	N/A
op1: prepare -write	8 ops	64 KB	344.25 ms	322.38 ms	24.26 op/s	194.09 KB/s	100%
op2: prepare -write	8 ops	128 KB	377.88 ms	329.12 ms	21.93 op/s	350.83 KB/s	100%
op3: prepare -write	8 ops	256 KB	384.5 ms	253.12 ms	22.1 op/s	707.24 KB/s	100%
op4: prepare -write	8 ops	512 KB	543.88 ms	382.88 ms	15.19 op/s	972.29 KB/s	100%
op5: prepare -write	8 ops	1.02 MB	674.75 ms	295.25 ms	12.15 op/s	1.56 MB/s	100%
op6: prepare -write	8 ops	2.05 MB	799.88 ms	346.38 ms	10.17 op/s	2.6 MB/s	100%
op7: prepare -write	8 ops	4.1 MB	859.25 ms	350.5 ms	9.3 op/s	4.76 MB/s	100%
op8: prepare -write	8 ops	8 MB	907.62 ms	455 ms	8.87 op/s	8.87 MB/s	100%
op1: read	12.99 kops	103.9 MB	11.14 ms	10.76 ms	433.07 op/s	3.46 MB/s	97.51%
op2: write	3.32 kops	26.6 MB	26.99 ms	26.25 ms	110.87 op/s	886.96 KB/s	100%
op1: read	22.22 kops	355.44 MB	6.71 ms	6.18 ms	740.78 op/s	11.85 MB/s	96.5%
op2: write	5.81 kops	92.94 MB	14.63 ms	14.5 ms	193.71 op/s	3.1 MB/s	100%
op1: read	16.35 kops	523.3 MB	4.21 ms	3.89 ms	549.99 op/s	17.6 MB/s	95.69%
op2: write	4.33 kops	138.66 MB	10.73 ms	10.5 ms	145.73 op/s	4.66 MB/s	100%
op1: read	9.89 kops	632.9 MB	5.13 ms	4.7 ms	333.78 op/s	21.36 MB/s	91.9%

图 5-8 测试性能

<a href="#">hide performance details</a>						
Stages						
Current Stage	Stages completed	Stages remaining	Start Time	End Time	Time Remaining	
ID	Name	Works	Workers	Op-Info	State	Link
w4-s1-init	init	1 wks	1 wkrs	init	<span>completed</span>	<a href="#">view details</a>
w4-s2-prepare	prepare	8 wks	64 wkrs	prepare	<span>completed</span>	<a href="#">view details</a>
w4-s3-8kb	8kb	1 wks	8 wkrs	read, write	<span>completed</span>	<a href="#">view details</a>
w4-s4-16kb	16kb	1 wks	8 wkrs	read, write	<span>completed</span>	<a href="#">view details</a>
w4-s5-32kb	32kb	1 wks	4 wkrs	read, write	<span>completed</span>	<a href="#">view details</a>
w4-s6-64kb	64kb	1 wks	4 wkrs	read, write	<span>completed</span>	<a href="#">view details</a>
w4-s7-128kb	128kb	1 wks	1 wkrs	read, write	<span>completed</span>	<a href="#">view details</a>
w4-s8-256kb	256kb	1 wks	1 wkrs	read, write	<span>completed</span>	<a href="#">view details</a>
w4-s9-512kb	512kb	1 wks	1 wkrs	read, write	<span>completed</span>	<a href="#">view details</a>
w4-s10-1mb	1mb	1 wks	1 wkrs	read, write	<span>completed</span>	<a href="#">view details</a>
w4-s11-cleanup	cleanup	1 wks	1 wkrs	cleanup	<span>completed</span>	<a href="#">view details</a>
w4-s12-dispose	dispose	1 wks	1 wkrs	dispose	<span>completed</span>	<a href="#">view details</a>

There are 12 stages in this workload.

[show error statistics details](#)

Performance Graph

Actions

[download-log](#)
[download-config](#)

[go back to index](#)

图 5-9 测试结果

由图 5-8 所示结果可知，minio 服务端对于写的成功率均为 100%，对于读的成功率不是 100%，最低为 91.9%，且任务越大成功率越小。写成功率虽然较高，写速率远低于读取速率。

8kb 任务读成功率为 97.51%，16kb 任务读成功率为 96.5%，32kb 任务读成功率为 95.69%，64kb 任务读成功率为 91.9%，结合图 5-10 源码分析 8kb，16kb 任务 workers=8，32kb，64kb 任务 workers=4，其他任务 workers=1，可得以下结果：

任务越大读成功率越小，另外后面较大的任务读成功率反而较高，均为 100%，可知 workers 越多，即任务并发度越高，读写性能越差。

```

<workstage name="8kb">
  <work name="8kb" workers="8" runtime="30">
    <operation type="read" ratio="20" config="cprefix=s3obstest;containers=c(1);objects=u(1,8)" />
    <operation type="write" ratio="80" config="cprefix=s3obstest;containers=c(1);objects=u(9,16);sizes=c(8)KB" />
  </work>
</workstage>

<workstage name="16kb">
  <work name="16kb" workers="8" runtime="30">
    <operation type="read" ratio="20" config="cprefix=s3obstest;containers=c(1);objects=u(1,8)" />
    <operation type="write" ratio="80" config="cprefix=s3obstest;containers=c(1);objects=u(9,16);sizes=c(8)KB" />
  </work>
</workstage>

<workstage name="32kb">
  <work name="32kb" workers="4" runtime="30">
    <operation type="read" ratio="20" config="cprefix=s3obstest;containers=c(1);objects=u(1,8)" />
    <operation type="write" ratio="80" config="cprefix=s3obstest;containers=c(1);objects=u(9,16);sizes=c(8)KB" />
  </work>
</workstage>

<workstage name="64kb">
  <work name="64kb" workers="4" runtime="30">
    <operation type="read" ratio="20" config="cprefix=s3obstest;containers=c(1);objects=u(1,8)" />
    <operation type="write" ratio="80" config="cprefix=s3obstest;containers=c(1);objects=u(9,16);sizes=c(8)KB" />
  </work>
</workstage>

<workstage name="128kb">
  <work name="128kb" workers="1" runtime="30">
    <operation type="read" ratio="20" config="cprefix=s3obstest;containers=c(1);objects=u(1,8)" />
    <operation type="write" ratio="80" config="cprefix=s3obstest;containers=c(1);objects=u(9,16);sizes=c(8)KB" />
  </work>
</workstage>

```

图 5-10 源码分析

修改读写任务占比，初始为读占 80%，写占 20%，如图 5-11 所示。

```

24 <workstage name="8kb">
25   <work name="8kb" workers="8" runtime="30">
26     <operation type="read" ratio="80" config="cprefix=s3obstest;containers=c(1);objects=u(1,8)" />
27     <operation type="write" ratio="20" config="cprefix=s3obstest;containers=c(1);objects=u(9,16);sizes=c(8)KB" />
28   </work>
29 </workstage>

```

图 5-11 读写占比分析

修改读写任务比为读占 20%，写占 80%，测试结果如下：

COSBENCH - CONTROLLER WEB CONSOLE							
General Report				time: Mon Apr 23 22:17:58 CST 2018 version: 0.4.2.20160615			
Op-Type	Op-Count	Byte-Count	Avg-ResTime	Avg-ProcTime	Throughput	Bandwidth	Succ-Ratio
op1: init -write	0 ops	0 B	N/A	N/A	0 op/s	0 B/s	N/A
op1: prepare -write	8 ops	64 KB	2965.12 ms	2965.12 ms	2.71 op/s	21.7 KB/s	100%
op2: prepare -write	8 ops	128 KB	2748 ms	2747.12 ms	2.91 op/s	46.6 KB/s	100%
op3: prepare -write	8 ops	256 KB	2957 ms	2860.62 ms	2.73 op/s	87.27 KB/s	100%
op4: prepare -write	8 ops	512 KB	3067.38 ms	3066.88 ms	2.64 op/s	169.06 KB/s	100%
op5: prepare -write	8 ops	1.02 MB	2728.62 ms	2723.12 ms	2.94 op/s	375.77 KB/s	100%
op6: prepare -write	8 ops	2.05 MB	3277.62 ms	2822.62 ms	2.46 op/s	630.35 KB/s	100%
op7: prepare -write	8 ops	4.1 MB	2911 ms	2748.88 ms	2.77 op/s	1.42 MB/s	100%
op8: prepare -write	8 ops	8 MB	2741.75 ms	2719.25 ms	2.92 op/s	2.92 MB/s	100%
op1: read	2.67 kops	21.38 MB	5.94 ms	5.63 ms	90.62 op/s	724.97 KB/s	99.33%
op2: write	10.95 kops	87.62 MB	19.95 ms	19.87 ms	371.43 op/s	2.97 MB/s	100%
op1: read	2.44 kops	38.96 MB	7.21 ms	6.73 ms	81.21 op/s	1.3 MB/s	99.51%
op2: write	9.77 kops	156.34 MB	22.51 ms	22.37 ms	325.89 op/s	5.21 MB/s	100%
op1: read	2.24 kops	71.52 MB	5.04 ms	4.69 ms	74.5 op/s	2.38 MB/s	99.07%
op2: write	9 kops	287.9 MB	11.98 ms	11.75 ms	299.91 op/s	9.6 MB/s	100%
op1: read	1.84 kops	117.95 MB	5.46 ms	4.93 ms	62.3 op/s	3.99 MB/s	98.71%
op2: write	7.64 kops	488.77 MB	14.06 ms	13.65 ms	258.19 op/s	16.52 MB/s	100%
op1: read	891 ops	114.05 MB	4.47 ms	3.66 ms	29.7 op/s	3.8 MB/s	100%
op2: write	3.68 kops	471.55 MB	6.97 ms	6.2 ms	122.8 op/s	15.72 MB/s	100%
op1: read	834 ops	213.5 MB	5.11 ms	3.92 ms	27.8 op/s	7.12 MB/s	100%
op2: write	3.2 kops	818.94 MB	7.96 ms	6.56 ms	106.63 op/s	27.3 MB/s	100%
op1: read	688 ops	352.26 MB	6.84 ms	4.75 ms	22.94 op/s	11.74 MB/s	100%
op2: write	2.65 kops	1.36 GB	9.47 ms	6.68 ms	88.25 op/s	45.18 MB/s	100%
op1: read	484 ops	484 MB	8.67 ms	4.43 ms	16.13 op/s	16.13 MB/s	100%
op2: write	1.73 kops	1.73 GB	14.81 ms	8.44 ms	57.7 op/s	57.7 MB/s	100%
op1: cleanup -delete	128 ops	0 B	14.45 ms	14.45 ms	67.76 op/s	0 B/s	100%
op1: dispose -delete	0 ops	0 B	N/A	N/A	0 op/s	0 B/s	N/A

图 5-12 修改读写比后性能

由图 5-12 可知，修改读写比后读成功率有所提高，写成功率依然保持在 100%。



接下来将所有任务的 workers 修改为 8，测试高并发状态下的读写性能，结果如图 5-13 图 5-14 所示，修改 workers 为 8 后，写成功率仍为 100%，但读成功率大幅下降，256kb 任务读成功率下降至 57.71%，而 512kb 和 1mb 任务读成功率仍为 100%。可以看出在高并发的情况下，比较小的任务量还是可以保证一定的成功率，但是在任务情况较大的时候就会产生 fail 的情况。

localhost:19088/controller/workload.html?id=w9

COSBENCH - CONTROLLER WEB CONSOLE

time: Mon Apr 23 22:45:09 CST 2018  
version: 0.4.2.20160615

General Report

Op-Type	Op-Count	Byte-Count	Avg-ResTime	Avg-ProcTime	Throughput	Bandwidth	Succ-Ratio
op1: init -write	0 ops	0 B	N/A	N/A	0 op/s	0 B/S	N/A
op1: prepare -write	8 ops	64 KB	3030.88 ms	3030.88 ms	2.82 op/s	22.55 KB/S	100%
op2: prepare -write	8 ops	128 KB	3687.75 ms	3687.5 ms	2.28 op/s	36.54 KB/S	100%
op3: prepare -write	8 ops	256 KB	937 ms	935.25 ms	10.68 op/s	341.68 KB/S	100%
op4: prepare -write	8 ops	512 KB	678.88 ms	658.62 ms	11.78 op/s	754.05 KB/S	100%
op5: prepare -write	8 ops	1.02 MB	3837.88 ms	2809.75 ms	2.17 op/s	277.56 KB/S	100%
op6: prepare -write	8 ops	2.05 MB	3668.75 ms	3664.25 ms	2.28 op/s	584.56 KB/S	100%
op7: prepare -write	8 ops	4.1 MB	4283.12 ms	3512.5 ms	1.88 op/s	964.08 KB/S	100%
op8: prepare -write	8 ops	8 MB	4164.25 ms	3386.62 ms	1.98 op/s	1.98 MB/S	100%
op1: read	22.39 kops	179.09 MB	6 ms	5.73 ms	746.39 op/s	5.97 MB/S	97.5%
op2: write	5.7 kops	45.62 MB	17.48 ms	17.4 ms	190.15 op/s	1.52 MB/S	100%
op1: read	13.93 kops	222.86 MB	5.9 ms	5.54 ms	845.04 op/s	13.52 MB/S	95.27%
op2: write	3.76 kops	60.11 MB	12.06 ms	11.94 ms	227.92 op/s	3.65 MB/S	100%
op1: read	20.78 kops	664.8 MB	6.53 ms	6.04 ms	695.27 op/s	22.25 MB/S	93.98%
op2: write	5.46 kops	174.66 MB	17.19 ms	16.96 ms	182.66 op/s	5.85 MB/S	100%
op1: read	18.1 kops	1.16 GB	7.45 ms	6.9 ms	603.6 op/s	38.63 MB/S	88.91%
op2: write	5.22 kops	334.34 MB	16.77 ms	16.22 ms	174.17 op/s	11.15 MB/S	100%
op1: read	13.43 kops	1.72 GB	9.18 ms	8.31 ms	447.77 op/s	57.31 MB/S	76.95%
op2: write	4.41 kops	563.97 MB	17.02 ms	15.08 ms	146.91 op/s	18.8 MB/S	100%
op1: read	7.76 kops	1.99 GB	11.78 ms	10.41 ms	258.76 op/s	66.24 MB/S	57.51%
op2: write	3.25 kops	831.74 MB	21.64 ms	17.67 ms	108.39 op/s	27.75 MB/S	100%
op1: read	3.84 kops	1.97 GB	5.43 ms	3.53 ms	128.03 op/s	65.55 MB/S	100%
op2: write	917 ops	469.5 MB	9.84 ms	6.75 ms	30.57 op/s	15.65 MB/S	100%
op1: read	2.47 kops	2.47 GB	8.08 ms	4.24 ms	82.44 op/s	82.44 MB/S	100%
op2: write	609 ops	609 MB	16.3 ms	10.25 ms	20.3 op/s	20.3 MB/S	100%
op1: cleanup -delete	128 ops	0 B	68.66 ms	68.66 ms	14.56 op/s	0 B/S	100%
op1: dispose	0 ops	0 B	N/A	N/A	0 op/s	0 B/S	N/A

图 5-13 修改 workers 后结果

[show peformance details](#)

Stages

Current Stage	Stages completed	Stages remaining	Start Time	End Time	Time Remaining
ID	Name	Works	Workers	Op-Info	State
w9-s1-init	init	1 wks	1 wkrs	init	completed
w9-s2-prepare	prepare	8 wks	64 wkrs	prepare	completed
w9-s3-8kb	8kb	1 wks	8 wkrs	read, write	completed
w9-s4-16kb	16kb	1 wks	8 wkrs	read, write	completed
w9-s5-32kb	32kb	1 wks	8 wkrs	read, write	completed
w9-s6-64kb	64kb	1 wks	8 wkrs	read, write	completed
w9-s7-128kb	128kb	1 wks	8 wkrs	read, write	completed
w9-s8-256kb	256kb	1 wks	8 wkrs	read, write	failed
w9-s9-512kb	512kb	1 wks	1 wkrs	read, write	completed
w9-s10-1mb	1mb	1 wks	1 wkrs	read, write	completed
w9-s11-cleanup	cleanup	1 wks	1 wkrs	cleanup	completed
w9-s12-dispose	dispose	1 wks	1 wkrs	dispose	completed

There are 12 stages in this workload.

[show error statistics details](#)

Performance Graph

Actions

[download-log](#) [download-config](#)

[go back to index](#)

图 5-14 修改 workers 后结果 2

## 5.2 对象存储性能分析

通过此次对象存储实验的跑分测试，我总结出了以下几点规律：读写比例对于

写成功率没有影响，会对读成功率造成一定影响，降低读的占比，可以提高读取成功率，但是影响并不大，原先读写成功率也不是特别低。读写文件越大，速率越低，读成功率下降。对读取成功率影响最大的因素在于 `workers` 数目，即并发访问量，并发访问量上去之后读成功率明显下降甚至 `fail`。这告诉我们对象存储在读取任务较小时，可保持高并发访问，但读写大文件时，必须保证低并发访问。

## 6 实验总结

经过此次实验，我了解了基本的对象存储的原理，了解了基本的对象存储访问操作，一些评测对象存储系统性能相关的指标以及他们如何影响系统性能。对象存储系统只需设备联网即可将数据上传至云端，有效解决了物联网设备本地存储空间较小的问题，显然是一个较好的应对物联网海量数据存储挑战的存储解决方案。本次实验只简单的对一个较小的对象存储系统进行跑分测试，初步了解了对象存储系统，实际的对象存储系统更加庞大，面临的问题也更多，希望在以后的学习中能够进一步接触研究。同时，本次实验也让我掌握了基本的分析方法，相信对以后的相关学习研究大有帮助。

## 7 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）