2015 级

《物联网数据存储与管理》课程

实 验 报 告

姓　名　庞 天 元

学　号　U201514869

班　号　物联网 1501 班

日　期　2018.04.23

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

Minio：一款私有的对象存储服务器，可以提供大规模的云存储服务。

Cosbench：由 INTEL 开发的一款分布式的对象存储系统的测试软件，支持一系列对象存储系统。由 Drivers 和 Controllers 两个部件构成。其主要目的一方面是提供给用户用以比较不同对象存储服务器的性能优劣，根据自己的应用需求选择合适的对象存储服务器；另一方面是给对象存储服务器的提供商一个测试自己产品的服务器。Cosbench 可以测试服务求增删读写的性能并提供测试程序个性化的支持。

Mock-s3：使用 Python 对 fakes3 的重写，而 fakes3 是对 Apache s3 的模仿。Apache s3 是一款经典的对象存储服务器架构，但是它的部署和应用的开发并不容易，有很多要求而且需要付费。fakes3 是对这款商用架构的模仿，可以提供与 s3 兼容的绝大部分 API 且容易部署。这样就使得对 s3 的开发简便得多。缺点则是 Mock-s3 以及 fakes3 无法应用于实际应用。

三、实验环境

操作系统 Ubuntu 16.04.1，JAVA 1.8.0_162，python

图 3-1 操作系统与语言环境

四、实验内容

4.1 对象存储技术实践

1.部署 Minio 服务器和客户端，验证 Minio 对象存储的功能。

2.安装 Cosbench，学习配置 Cosbench 的方法。

3.使用 Cosbench 对 Minio 进行性能测试。

4.安装 Mock-s3

5.使用 Cosbench 对 Mock-s3 进行性能测试。

4.2 对象存储性能分析

1.对象尺寸对性能的影响

S3 对 8KB 的对象进行读写操作而 S10 对 1MB 的对象进行读写测试，比较它们的性能报告。发现对尺寸大的对象的操作的带宽更高，但是处理时间更长而且操作数的吞吐量比较低，打开其余任务的报告可以得到相似结论。

在实际应用中，如果应用对于实时性的要求比较高，则分配对象的粒度要小，如果应用对于实时性的要求则分配对象的粒度可以大些。比如 QQ 等在线社交软件，其通话记录的存储粒度要小，而群文件的存储粒度可以大些。

ID: w15-s3 Name: 8kb Interval: N/A Current State: completed

more info

Final Result

General Report

Op-Type	Op-Count	Byte-Count	Avg-ResTime	Avg-ProcTime	Throughput	Bandwidth	Succ-Ratio
read	6.41 kops	51.3 MB	26.71 ms	25.55 ms	213.92 op/s	1.71 MB/S	99.44%
write	1.62 kops	12.96 MB	40.78 ms	40.15 ms	54.04 op/s	432.32 KB/S	100%

图 4-1 对象尺寸对于性能的影响 1

ID: w15-s10 Name: 1mb Interval: N/A Current State: completed

more info

Final Result

General Report

Op-Type	Op-Count	Byte-Count	Avg-ResTime	Avg-ProcTime	Throughput	Bandwidth	Succ-Ratio
read	934 ops	934 MB	18.14 ms	9.86 ms	31.18 op/s	31.18 MB/S	100%
write	239 ops	239 MB	54.09 ms	14.74 ms	7.98 op/s	7.98 MB/S	100%

图 4-2 对象尺寸对于性能的影响 2

2.Mocks3 和 Minio 性能的比较

通过将 Mock-s3 和 Minio 的性能报告比较发现，Minio 的吞吐量、反应时间和处理时间都在 Mock-s3 之上。

Final Result

General Report

Op-Type	Op-Count	Byte-Count	Avg-ResTime	Avg-ProcTime	Throughput	Bandwidth	Succ-Ratio
init -write	0 ops	0 B	N/A	N/A	0 op/s	0 B/S	N/A
prepare -write	8 ops	8 MB	2095.75 ms	1659.62 ms	3.17 op/s	3.17 MB/S	100%
read	934 ops	934 MB	18.14 ms	9.86 ms	31.18 op/s	31.18 MB/S	100%
write	239 ops	239 MB	54.09 ms	14.74 ms	7.98 op/s	7.98 MB/S	100%
cleanup -delete	128 ops	0 B	12.8 ms	12.8 ms	75.12 op/s	0 B/S	100%
dispose -delete	0 ops	0 B	N/A	N/A	0 op/s	0 B/S	N/A

图 4-3 Minio 的性能报告

Final Result

General Report

Op-Type	Op-Count	Byte-Count	Avg-ResTime	Avg-ProcTime	Throughput	Bandwidth	Succ-Ratio
init -write	0 ops	0 B	N/A	N/A	0 op/s	0 B/S	N/A
prepare -write	8 ops	8.05 MB	3083.75 ms	2144.75 ms	2.95 op/s	2.96 MB/S	100%
read	132 ops	132 MB	165.15 ms	142.99 ms	4.42 op/s	4.42 MB/S	100%
write	43 ops	43 MB	184.91 ms	170.84 ms	1.44 op/s	1.44 MB/S	100%
cleanup -delete	128 ops	0 B	74.8 ms	74.8 ms	12.83 op/s	0 B/S	100%
dispose -delete	0 ops	0 B	N/A	N/A	0 op/s	0 B/S	N/A

图 4-4 Mock-s3 的性能报告

五、实验过程

1.Minio server 以及 client 的安装和运行

```
oscar@ubuntu:~$ ./minio server /home/oscar
Drive Capacity: 9.0 GiB Free, 18 GiB Total

Endpoint:  http://172.16.225.190:9000  http://127.0.0.1:9000
AccessKey: OCDKQ4P2UFIIAHUMNULG
SecretKey: EFJIJLIUViOA+hGL0q9mcs1gGCGPlJIaE5iLsrSK

Browser Access:
   http://172.16.225.190:9000  http://127.0.0.1:9000

Command-line Access: https://docs.minio.io/docs/minio-client-quickstart-guide
   $ mc config host add myminio http://172.16.225.190:9000 OCDKQ4P2UFIIAHUMNULG
EFJIJLIUViOA+hGL0q9mcs1gGCGPlJIaE5iLsrSK

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 server 的启动

2.Cosbench 的安装并启动

```
oscar@ubuntu: ~/0.4.2.c4
-----------------------------------------------

=================================================

Launching osgi framwork ...
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 !!!
-----------------------------------------------
oscar@ubuntu:~/0.4.2.c4$
```

图 5-2 Cosbench 的安装与启动

3.运行 Cosbench 对 Minio 的 Server 进行测试

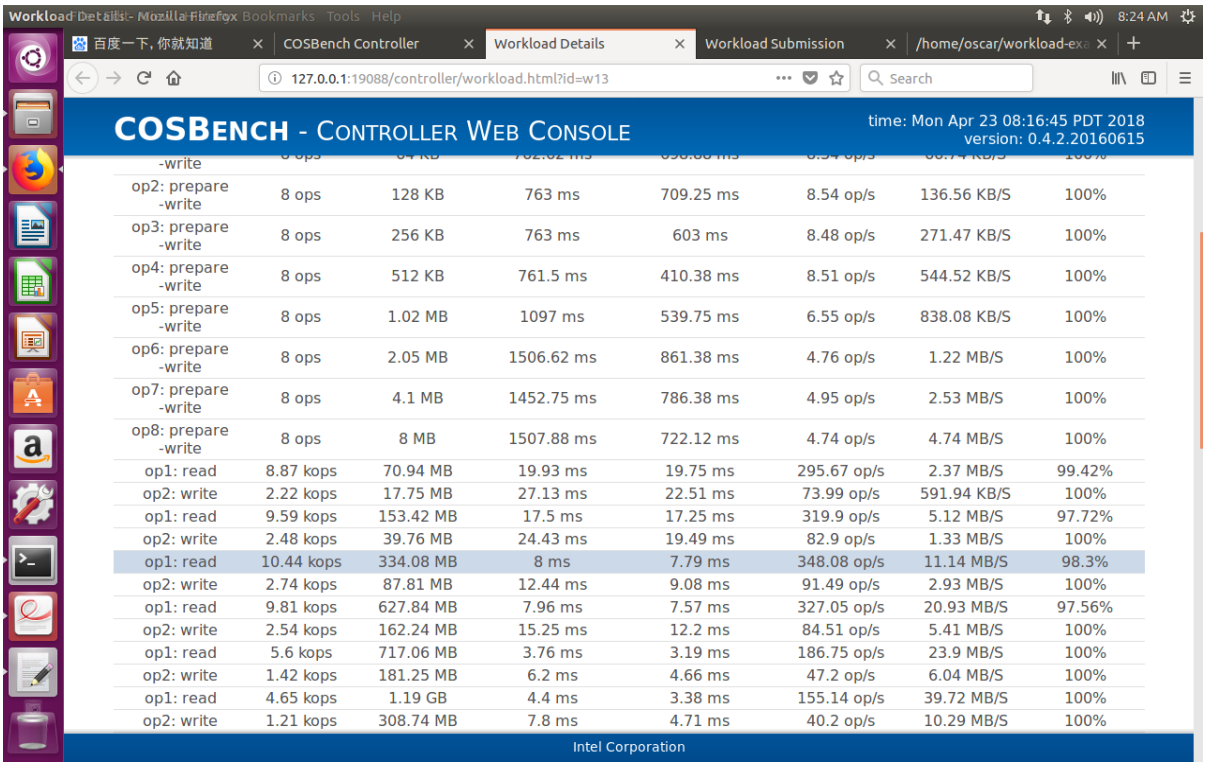

图 5-3 Cosbench 测评

4.学习配置 Cosbench 的方法并理解其测试程序的构成

Cosbench 的 workload 使用 xml 文件提交，提供了网页端作为用户界面。而其和存储服务器的交互则是通过 HTTP 实现，打开 archive 文件可以查看 Cosbench 每一步 GET 和 PUT 方法。

Cosbench 的 Workload 由四层结构组成：workload,workstage,work,operation。而最重要的 work 则是由 4 种特殊任务和常规任务组成，4 种特殊任务中的 init 用于创建容器；prepare 用于创建容器中的对象；cleanup 用于清除对象；dispose 用于清除容器。

分析 s3 的 demo 我可以得到关于测试程序如下信息：

测试程序首先创建 8 个容器，然后在 8 个容器中分别创建 8 个对象，大小从 8KB 到 1MB 不等。接着进入测试阶段，创建 8 个常规 work，每个 work 的读操作占 80%而写操作占 20%。读操作其次对 1 到 8 个容器中的对象进行读操作测试，而写操作始终对 1 号容器中的 9 到 16 号对象进行写操作，由于创建时只有 1 到 8 号对象，所以我认为这里的写操作会创建新的对象。

可以观察到线程数的设定是和任务的参数有关联的，按照 Cosbench 的 guidance 所言，容器数应该是 init 和 dispose 的线程数的整数倍，而容器中的对象

数应该是 prepare 和 cleanup 的线程数的整数倍。而常规任务的线程数我认为也应当参照这样的思路。至于究竟多少线程数合适，应当通过测试得出，具体方法是不断增加线程数直到得到最好的性能。

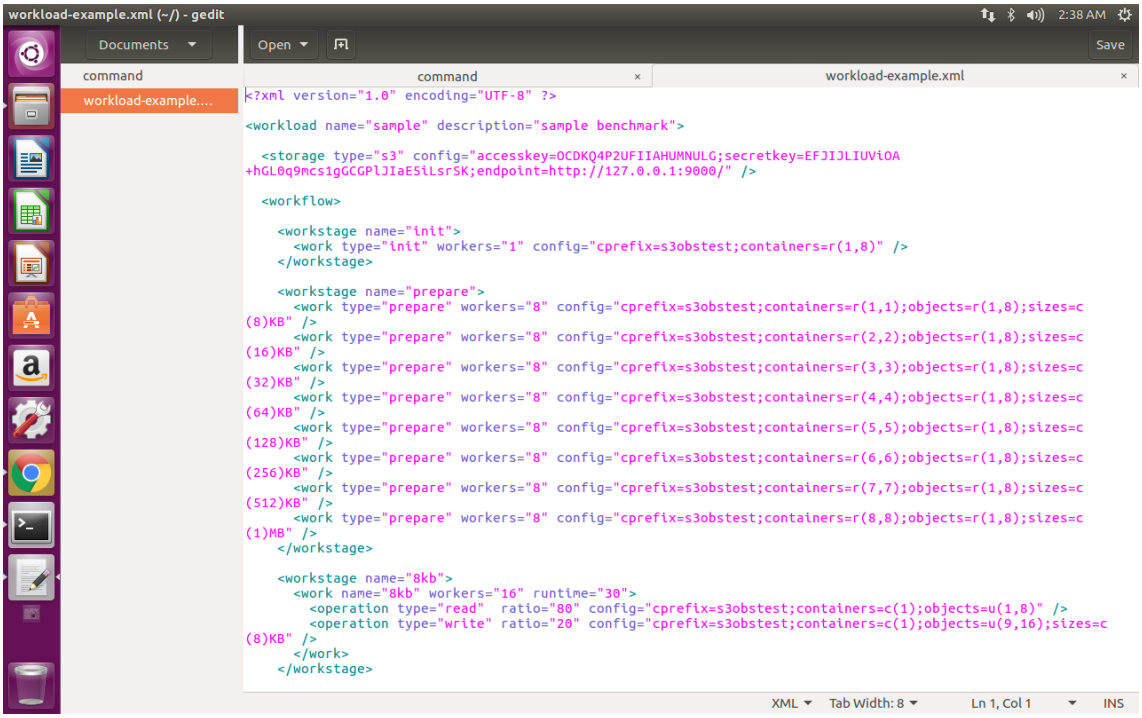

图 5-4 demo workload 内容解析

5.安装 Mock-s3 并使用 COSBench 评测

下载预编译包，内含自动完成部署的脚本，部署完成后运行即可。

为了方便，我将 Mock-s3 的端口和地址设定为和 Minio 一致，运行 COSBench 测评即可。

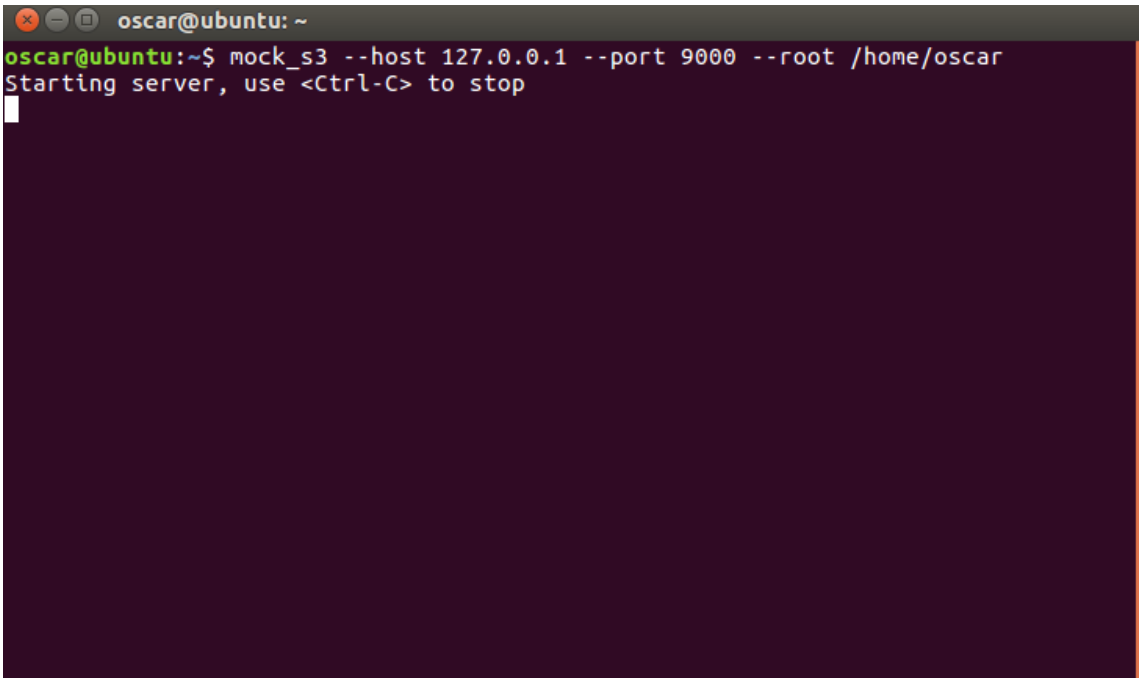

图 5-5 Mock-s3 启动

Final Result

General Report

Op-Type	Op-Count	Byte-Count	Avg-ResTime	Avg-ProcTime	Throughput	Bandwidth	Succ-Ratio
init -write	0 ops	0 B	N/A	N/A	0 op/s	0 B/S	N/A
prepare -write	8 ops	8.05 MB	3083.75 ms	2144.75 ms	2.95 op/s	2.96 MB/S	100%
read	132 ops	132 MB	165.15 ms	142.99 ms	4.42 op/s	4.42 MB/S	100%
write	43 ops	43 MB	184.91 ms	170.84 ms	1.44 op/s	1.44 MB/S	100%
cleanup -delete	128 ops	0 B	74.8 ms	74.8 ms	12.83 op/s	0 B/S	100%
dispose -delete	0 ops	0 B	N/A	N/A	0 op/s	0 B/S	N/A

图 5-6 Mock-s3 测评

6.关于部署多个服务器节点

在同一台 PC 上部署多个服务器节点的方法是在不同地址和端口打开服务器程序。由于不是在不同的物理机上所以很难达到分布式存储的效果。

7.关于编程实现测试程序

S3 提供了一整套 API 编程接口，使用 JAVA 编写自己的程序是可以完成 COSBench 提供的各项服务内容的。由于时间关系，我未能完成此项实验内容。

六、实验总结

对象存储的概念我第一接触，本次实验目前为止并没有太多地接触对象存储的理论知识，而是围绕着对象存储服务器的部署、配置以及测试来展开。

实验的过程还是比较考验我对 linux 系统的熟悉程度以及查阅资料的能力，

Minio 的部署还是比较简单的。Cosbench 则比较复杂一些，由于其为开源软件我前几次下载的都是它的源代码版本。废了好大劲才找到已经配置好的版本，安装运行后 Workload 的配置也是一大难题。我按照 OG 一步步地仔细阅读，再参考已经给出了 example 完成对 Minio 的基本测试。

Cosbench 是个基于 WEB 平台的存储服务器测试工具，利用它服务商可以更好地了解自己产品的性能，而用户可以更好地选择服务器产品。

我的计划是尝试利用 JAVA 自己编程实现 COSbench，为此我还向老师协商晚交一会报告，但是由于时间关系最终未能实现。在此说声抱歉。

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.