



2019 级

《物联网数据存储与管理》课程

实 验 报 告

姓 名 王子义

学 号 201915085

班 号 物联网 1901

日 期 2022.04.19

目 录

一、实验目的.....	1
二、实验背景.....	1
三、实验环境.....	1
四、实验内容.....	2
4.1 对象存储技术实践.....	2
4.2 对象存储性能分析.....	2
五、实验过程.....	2
5.1 对象存储技术实践.....	2
5.2 对象存储性能分析.....	4
5.3 数据分析.....	5
六、实验总结.....	8
6.1 实验当中的问题.....	8
6.2 实验总结.....	9
参考文献.....	10

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

对象存储是面向对象/文件的、海量的互联网存储，它也可以直接被称为“云存储”。对象尽管是文件，它是已被封装的文件（编程中的对象就有封装性的特点），也就是说，在对象存储系统里，你不能直接打开/修改文件，但可以像 ftp 一样上传文件，下载文件等。另外对象存储没有像文件系统那样有一个很多层级的文件结构，而是只有一个“桶”（bucket）的概念（也就是存储空间），“桶”里面全部都是对象，是一种很扁平化的存储方式。其最大的特点就是它的对象名称就是一个域名地址，一旦对象被设置为“公开”，所有网民都可以访问到它；它的拥有者还可以通过 REST API 的方式访问其中的对象。因此，对象存储最主流的使用场景，就是存储网站、移动 app 等互联网/移动互联网应用的静态内容（视频、图片、文件、软件安装包等等）。

对象存储在很多重要方面与 SAN 和 NAS 迥然不同，对存储管理员而言最显著的区别在于对象存储没有 LUNs，卷以及 RAID 等要素。对象数据不是存储在固定的块，而是在大小可变的“容器”里。鉴于元数据（metadata）和数据本身可通过传统数据访问方法进行访问，对象存储允许数据被直接访问。此外，支持对象级和命令级的安全策略设置。

MinIO 是一个基于 Apache License v2.0 开源协议的对象存储服务。它兼容亚马逊 S3 云存储服务接口，非常适合于存储大容量非结构化的数据，例如图片、视频、日志文件、备份数据和容器/虚拟机镜像等，而一个对象文件可以是任意大小，从几 kb 到最大 5T 不等。

s3bench 是使用了 AWS Go SDK 实现的对象存储服务器测试程序，可以通过设置请求对象大小和数量、并行客户端数量等参数，根据吞吐率、延迟等结果评测对象存储系统的性能。

三、实验环境

本次对象存储测试实验通过虚拟机 VMware 在 Linux 系统中完成，具体的实验环境以及所用到的工具如表 1 所示。

表 1 实验环境

操作系统	ubuntu-20.04.1
虚拟机软件	VMware Workstation 14 Pro
CPU	Intel® Core™ i5-10210U CPU @ 1.60GHz
分配内存	2GB
程序语言环境	go-1.13.8,git-2.25.1
服务端	minio
客户端	minio client
测试程序	s3bench
服务器 IP	localhost:9000

四、实验内容

本次实验所做的主要内容如下：

- 1、进行系统搭建，配置文件运行所需要的环境，在 linux 上安装 git、go 等软件；
- 2、对对象存储系统进行实践，采用 minio/mc 配置服务器端和客户端，并进行简单的创建或删除 bucket、上传或删除文件等操作。
- 3、对对象存储系统进行测试，采用 s3bench 进行负载测试。

4.1 对象存储技术实践

- (1) 在 Linux 虚拟机中配置实验所需的 go 与 gopm 环境。
- (2) 通过 minio 官网教程获取 minio 与 minio client 可执行程序。
- (3) 使用 minio 创建服务器，然后通过 minio client 向服务器发送请求，通过服务器网页查看请求是否成功。
- (4) 在本地创建文件 qaz.txt，然后通过 mc 上传到 bucket 的 eeee 当中，并通过网页观察是否请求成功。

4.2 对象存储性能分析

- (1) 使用 go get 指令从 github 中获取并自动安装 s3bench。
- (2) 使用实验指导中的 run-s3bench.sh 对 minio 服务器进行样例测试，测试 minio 服务器的性能。
- (3) 修改 run-s3bench.sh，使其在多次循环中改变特定参数，并对 mock-s3 进行性能测试。
- (4) 收集性能测试结果数据，通过图表分析对象存储系统的性能影响因素。

五、实验过程

5.1 对象存储技术实践

- (1) 下载 minio 服务器：minio

首先根据网上的资料文档与 minio 官方文档，输入以下指令：

```
wget https://dl.min.io/server/minio/release/linux-amd64/minio
```

（wget 从官网获取程序文件）

- (2) 给 minio 可执行权限

```
chmod +x minio 或者 chmod 777 minio
```

- (3) 设置 minio 服务器的用户名、密码，以及监听端口

```
MINIO_ROOT_USER=U201915085 MINIO_ROOT_PASSWORD=wzywzy  
123 ./minio server ./data --console-address ":9099"
```

```
Access key length should be at least 3, and secret key length at least 8 characters
root@ubuntu:/opt/minio# MINIO_ROOT_USER=U201915085 MINIO_ROOT_PASSWORD=wzywzy123 ./minio server ./data --console-address ":9099"
API: http://192.168.240.132:9000 http://127.0.0.1:9000
RootUser: U201915085
RootPass: wzywzy123

Console: http://192.168.240.132:9099 http://127.0.0.1:9099
RootUser: U201915085
RootPass: wzywzy123

Command-line: https://docs.min.io/docs/minio-client-quickstart-guide
$ mc alias set myminio http://192.168.240.132:9000 U201915085 wzywzy123

Documentation: https://docs.min.io
```

图 1.设置 minio 服务器

(4) 下载 minio 客户端 mc, 并赋予其可执行权限

```
wget https://dl.min.io/client/mc/release/linux-amd64/mc
```

```
chmod +x mc
```

(5) 根据提示, 设置 IP、RootUser 和 RootPass。

```
root@ubuntu:/opt/minio# chmod +x mc
root@ubuntu:/opt/minio# ./mc alias set wzyminio http://192.168.240.132:9000 U201915085 wzywzy123
mc: Configuration written to '/root/.mc/config.json'. Please update your access credentials.
mc: Successfully created '/root/.mc/share'.
mc: Initialized share uploads '/root/.mc/share/uploads.json' file.
mc: Initialized share downloads '/root/.mc/share/downloads.json' file.
Added 'wzyminio' successfully.
root@ubuntu:/opt/minio#
```

图 2.将 mc 连接到 minio

(6) 运用 mc 指令, 创建一个名为 test1 的桶, 并通过浏览器访问 127.0.0.1:9000, 查看是否创建成功。

```
./mc mb data/test1
```

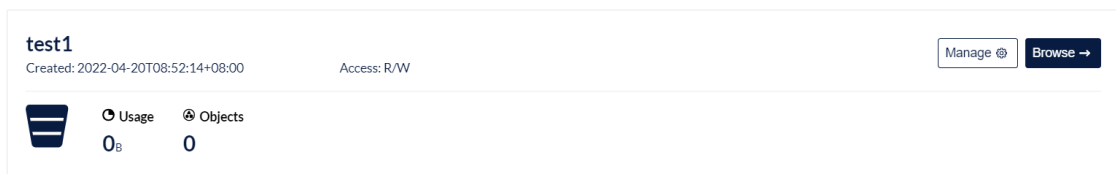


图 3.创建桶 test1

(7) 创建另外一个桶 eeee, 并在本地创建文件 qaz.txt, 然后将该文件上传到该桶当中, 并在浏览器里查看是否上传成功。

```
./mc mb minio/data/eeee
```

```
./mc cp qaz.txt minio/data/eeee
```

```
Bucket created successfully 'minio/data/eeee'.
实践\qaz.txt: 4 B / 4 B [=====] 6
```

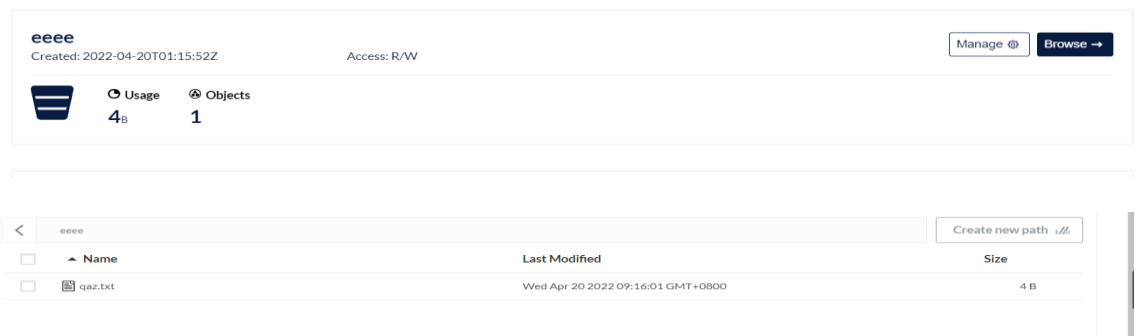


图 4.上传文件 qaz.txt

5.2 对象存储性能分析

(1) 下载安装 s3bench

```
go get -u github.com/igneous-systems/s3bench
```

注意：这个过程中遇到了一个问题：由于网络原因 go get 无法访问 GitHub，从而无法完成下载。可能会出现超时访问 timeout 或者是 github 拒绝访问的问题。

解决尝试 1：

将镜像地址换为阿里云，以解决 GitHub 无法访问的问题

```
go env -w GO111MODULE=on
```

```
go env -w GOPROXY=https://mirrors.aliyun.com/goproxy/,direct
```

解决尝试 2：

安装 gopm，使用 gopm 安装包

```
go get -u github.com/gpmgo/gopm
```

```
gopm get -g github.com/igneous-systems/s3bench
```

解决尝试 3：

由于墙等一系列原因，有可能会导导致 gopm 下载不下来，同时由于 go 版本及 GO111MODULE 问题，下载下来后会存放在 pkg/mod 文件夹内，所以采用以下办法：

(1) 配置 GOPROXY 以及关闭 GO111MODULE

1. `go env -w GOPROXY=https://goproxy.io,direct`

2. `go env -w GO111MODULE=off`

(2) 拉取编译 gopm

1. `go get -u github.com/gpmgo/gopm`

2.

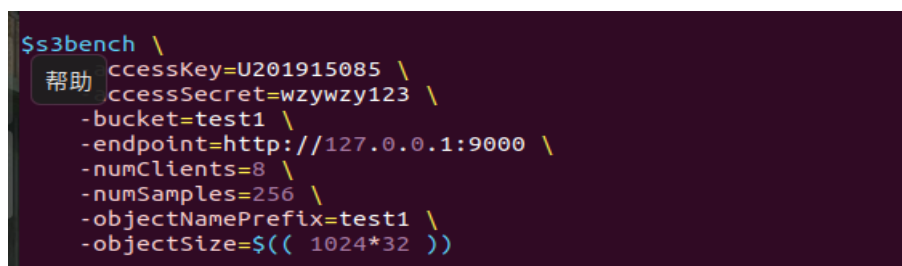
3. `go install github.com/gpmgo/gopm`

(3) 使用 `gopm -v` 查看版本

(4) 重新打开 GO111MODULE（按自己需要）

(2) 进行样例测试

下载实验指导中的 run-s3bench.sh 文件，修改参数如图 5 所示。其中用户名、密码、ip 与 minio 服务器一致，设置测试用的并行客户端数量为 8 个，数据对象大小为 32KB，对象数量为 256，使用 minio 服务器中的 bucket: test1 进行实验。



```
$s3bench \
帮助 accessKey=U201915085 \
accessSecret=wzywzy123 \
-bucket=test1 \
-endpoint=http://127.0.0.1:9000 \
-numClients=8 \
-numSamples=256 \
-objectNamePrefix=test1 \
-objectSize=$(( 1024*32 ))
```

图 5.修改测试文件 s3bench

S3bench 的运行结果如图 6:

```
Test parameters
endpoint(s):      [http://127.0.0.1:9000]
bucket:           test1
objectNamePrefix: test1
objectSize:       0.0010 MB
numClients:       8
numSamples:       256
verbose:          %!d(bool=false)

Generating in-memory sample data... Done (3.9908ms)

Running Write test...

Running Read test...

Test parameters
endpoint(s):      [http://127.0.0.1:9000]
bucket:           test1
objectNamePrefix: test1
objectSize:       0.0010 MB
numClients:       8
numSamples:       256
verbose:          %!d(bool=false)

Results Summary for Write Operation(s)
Total Transferred: 0.250 MB
Total Throughput:  0.10 MB/s
Total Duration:    2.426 s
Number of Errors:  0
-----
Write times Max:      0.172 s
Write times 99th %ile: 0.166 s
Write times 90th %ile: 0.102 s
Write times 75th %ile: 0.093 s
Write times 50th %ile: 0.077 s
Write times 25th %ile: 0.055 s
Write times Min:      0.017 s

Results Summary for Read Operation(s)
Total Transferred: 0.250 MB
Total Throughput:  1.02 MB/s
Total Duration:    0.244 s
Number of Errors:  0
-----
Read times Max:      0.023 s
Read times 99th %ile: 0.021 s
Read times 90th %ile: 0.015 s
Read times 75th %ile: 0.009 s
Read times 50th %ile: 0.006 s
Read times 25th %ile: 0.004 s
Read times Min:      0.002 s

Cleaning up 256 objects...
Deleting a batch of 256 objects in range {0, 255}... Succeeded
Successfully deleted 256/256 objects in 1.3288742s
```

图 6.测试结果

(3) 数据分析

1、对象大小对存储性能的影响

Numclients 为 8 固定, 通过更改脚本范例中的 objectSize 参数, 为 1024, 2048, 4096, 8192, 分析其运行结果。

Size	WRITE Throught	WRITE Time-90 th %ile	WRITE Time-99 th %ile	READ Throught	READ Time-90 th %ile	READ ime-90 th %il
1024	0.10	0.102	0.166	1.02	0.015	0.021
2048	0.20	0.110	0.181	5.02	0.004	0.006
4096	0.45	0.105	0.126	6.72	0.007	0.013
8192	0.91	0.097	0.018	13.64	0.006	0.008

对象大小对吞吐率的影响

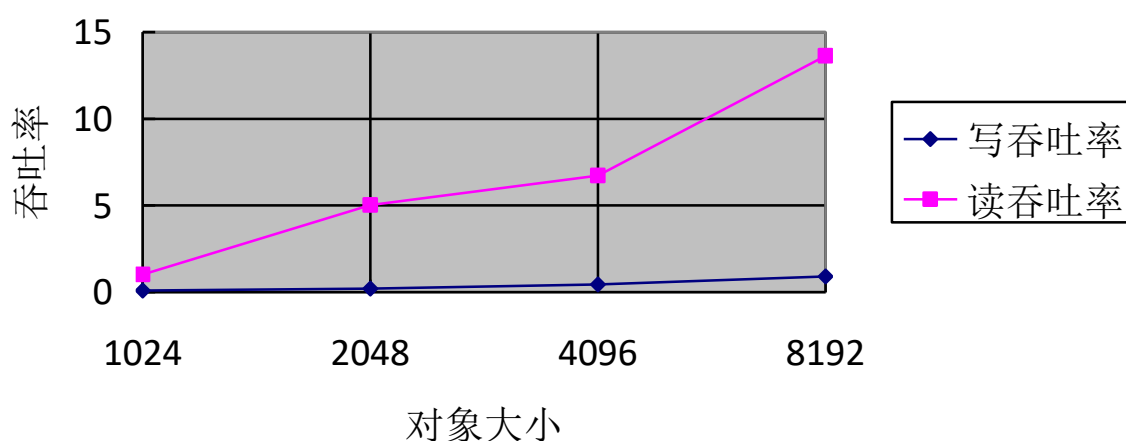


表 1

由该表可知：

- (1) 在我们的测试当中，随着对象 size 的逐渐增大，读和写吞吐率也是随之增大的，且基本上呈线性关系，所以可以推测，单位时间内，我们输入\输出的数目是一定的。
- (2) 写请求，90%延迟的时间比较稳定，而读请求 90%延迟的时间逐渐下降。

2、对象数量对存储性能的影响

NUMsamples	WRITE Throught	WRITE Time-90 th %ile	WRITE Time-99 th %ile	READ Throught	READ Time-90 th %ile	READ ime-90 th %il
256	0.10	0.102	0.166	1.02	0.015	0.021
512	0.13	0.092	0.113	1.46	0.011	0.022
1024	0.13	0.092	0.107	2.47	0.005	0.009
2048	0.12	0.098	0.122	1.25	0.011	0.021

数据数据量对性能的影响

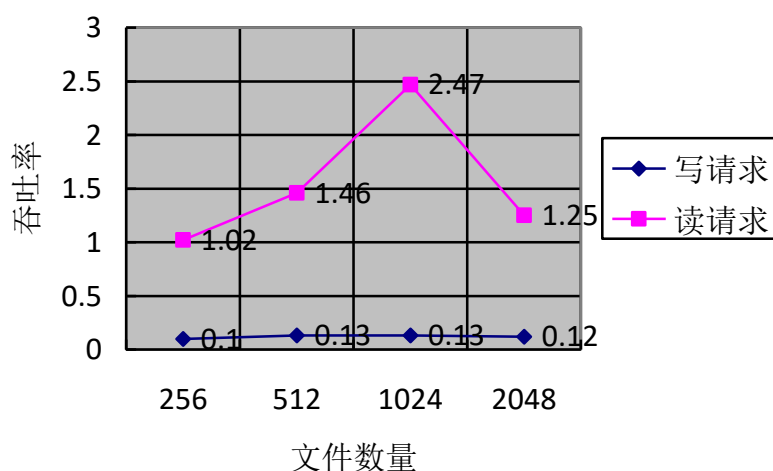


表 2

由该表可知：

(1) 随着对象数量的增大，读请求的吞吐量基本不变，所以写请求单位时间内输入的数目是一定的；而对于读请求，吞吐率是先增大然后减小，所以在一定范围内，单位时间内输出的文件不是固定的，且会受到读请求数目的影响。

(2) 然后观察延迟，可以发现，对于写请求，90%延迟与 99%延迟基本上都是随文件数量增加而略微下降，然后当数目过大时，略微上升；而对于读请求，90%延迟与 99%延迟基本上都是先下降然后再上升。

(3) 可得结论，当文件数目过大时，无论是读还是写，都会使系统性能下降。

3、客户端数量对存储性能的影响

设置对象大小为 128*1024B 也就是 128KB，分别设置并发数为 1，2，4，8，16 分析系统性能。

Numclients	WRITE Throught	WRITE Time-90 th ile	WRITE Time-99 th ile	READ Throught	READ Time-90 th ile	READ ime-90 th il
1	0.14	0.008	0.012	0.62	0.002	0.003
2	0.23	0.010	0.021	1.18	0.002	0.002
4	0.34	0.013	0.032	1.44	0.004	0.010
8	0.10	0.102	0.166	1.02	0.015	0.021
16	0.12	0.167	0.231	1.38	0.018	0.026

客户端数量对性能的影响

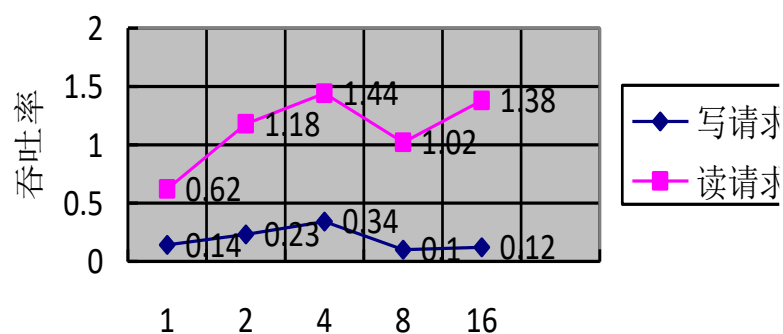


表 3

客户端数量对延迟的影响

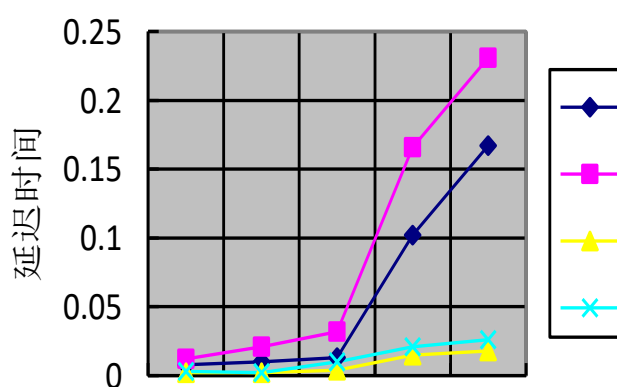


表 4

由上表可知：

(1) 随着客户端数量的增加，对于写请求和请求，吞吐率先上升，再下降，最后再上升，所以，二者并不是简单的线性关系，当数量过多时，本身的开销可能也会影响性能。

(2) 随着客户端数量的增加，读和写请求的延迟先是缓慢增加，然后快速增加，所以可得客户端数量多的话，请求沟通所需要的时间会增大从而影响性能。

六、实验总结

6.1、实验当中遇到的问题

(1) 当直接关掉终端窗口，重新创建 minio 服务器时，总是会提醒端口已经被占用。

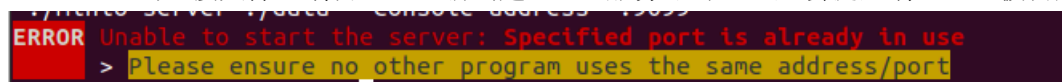


图 .端口占用

解决：

1. 查看端口是否被占用

```
netstat -anp | grep [端口号]
```

2. 查看占用的进程

```
lsof -i:[端口号]
```

3. 关闭进程

```
kill -9 [进程 PID]
```

```
root@ubuntu:/opt/minio# netstat -anp | grep 9000
tcp6      0      0 :::9000          :::*              LISTEN
4024/./minio
root@ubuntu:/opt/minio# lsof -i:9000
COMMAND  PID USER  FD  TYPE DEVICE SIZE/OFF NODE NAME
minio    4024 root  10u  IPv6  93934      0t0  TCP *:9000 (LISTEN)
root@ubuntu:/opt/minio# kill -9 4024
root@ubuntu:/opt/minio#
```

然后再重新运行 minio，就可以了：

```
root@ubuntu:/opt/minio# MINIO_ROOT_USER=U201915085 MINIO_ROOT_PASSWORD=wzywzy123 ./minio server ./data --console-address ":9099"
API: http://192.168.240.132:9000 http://127.0.0.1:9000
RootUser: U201915085
RootPass: wzywzy123

Ubuntu Software /192.168.240.132:9099 http://127.0.0.1:9099
RootUser: U201915085
RootPass: wzywzy123

Command-line: https://docs.min.io/docs/minio-client-quickstart-guide
$ mc alias set myminio http://192.168.240.132:9000 U201915085 wzywzy123
Documentation: https://docs.min.io

You are running an older version of MinIO released 2 weeks ago
Update: Run 'mc admin update'
```

(2) 下载 s3bench 的时候被墙，无法下载。

解决方案已在文中 5.2 (1) 说明

6.2、实验总结

通过本次实验，我对于对象存储系统有了更加深入的实践与认识，使用了 minio 成功建立了服务器，然后使用 minio client 作为客户端，最后使用了 s3bench 作为测试用例，最后又对于一些环境参数对于存储性能的影响做了分析，还认识到了尾延迟的原理，虽然都是采用的最基本的方案，但是还是觉得收获很大，而且因为是在 linux 上进行的实验，对于 linux 的环境配置与操作更加熟悉，还有当中遇到的那些问题，在解决的过程当中，也学习到了不少知识，以后有时间的话，还想尝试一些老师建议的其他方案进行对象存储。

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.