分布式系统作业--2

姓名：郭晓龙　　学号：**18340047**　　班级：大数据

1.实验内容

- 使用CRIU实现进程的热迁移
- 使用CRIU实现容器（Docker）的热迁移
- 观察web服务迁移的性能损耗

2.实验环境选择和配置

- 主机：两个 Ubuntu 18.04虚拟机；互相可以ssh免密登录；使用NFS挂载共享目录，在 MPI 实验中已经配置过，可参考[1]；此外，对下面提到的环境都进行相同配置
- 热迁移工具：CRIU 3.11，配置过程参考[2]
- 容器：Docker 17.06，配置过程参考[3]，开启experimental功能
    - 选择原因：高版本Docker 19.03无法正常恢复断点（会从头开始）；卸载参考[4]

3.实验过程及结果分析

3.1 进程的热迁移

3.11 样例程序：helloer-c

一个简单的每0.1秒hello一次的程序如下，在每次输出中可以看到实际hello经过了多久

```c
#include <stdio.h>
#include <unistd.h>
#include <sys/time.h>

int main(){
    int start = 0;
    struct timeval t0;
    gettimeofday(&t0,NULL);
    while (1) {
        usleep(100000);
        struct timeval t1;
        gettimeofday(&t1,NULL);
        float dt = t1.tv_sec-t0.tv_sec+(float)(t1.tv_usec-t0.tv_usec)/1000000;
        printf("+ %f s; counting:%d\n",dt,start);
        start = (start+1)%3600;
        t0=t1;
    }
    return 0;
}
```

3.12 进程热迁移脚本：prc-migrate.sh

```
# node1：在一个终端运行helloer-c后在另一个终端运行此脚本
#!/bin/bash

# /home/mpi_share为node1和node2的nfs共享目录
rm -rf /home/mpi_share/img
mkdir /home/mpi_share/img

# 查看进程helloer-c的当前运行主机和PID
pid=`ps -ef | grep helloer-c | grep -v grep | awk '{print $2}'`
echo "host=$(hostname) pid=${pid}"

# 使用CRIU将进程helloer-c停止并将断点保存在共享目录中
criu dump -D /home/mpi_share/img -j -t $pid

# ssh node2远程执行命令 CRIU恢复断点 查看进程所在主机
ssh -t root@node2 "
    criu restore -D /home/mpi_share/img -j;
    hostname;"
```

3.13 测试结果与分析

- helloer-c所在终端：

```
root@ubuntu1:/home/mpi_share# ./helloer-c
+ 0.100729 s; counting:0
+ 0.101071 s; counting:1
+ 0.101327 s; counting:2
+ 0.100383 s; counting:3
+ 0.100990 s; counting:4
+ 0.100667 s; counting:5
+ 0.100457 s; counting:6
+ 0.100691 s; counting:7
+ 0.101022 s; counting:8
+ 0.101001 s; counting:9
+ 0.101234 s; counting:10
+ 0.101105 s; counting:11
+ 0.100666 s; counting:12
Killed
root@ubuntu1:/home/mpi_share#
```

- CRIU进程迁移脚本所在执行终端：

```
root@ubuntu1:/home/gggxxl/Desktop# bash prc_migrate.sh
host=ubuntu1 pid=3383
Warn  (criu/image.c:134): Failed to open parent directory
Warn  (compel/arch/x86/src/lib/infect.c:273): Will restore 3383 with interrupted
 system call
+ 0.581724 s; counting:13
+ 0.100981 s; counting:14
+ 0.101503 s; counting:15
+ 0.100722 s; counting:16
+ 0.100603 s; counting:17
+ 0.100646 s; counting:18
+ 0.101182 s; counting:19
^Cubuntu2
Connection to node2 closed.
root@ubuntu1:/home/gggxxl/Desktop#
```

- 结果分析：进程成功从ubuntu1热迁移到ubuntu2，从第13次counting可以看出热迁移所需时间大概为0.5s

3.14 PID变化和其他实验细节

- **PID变化**

  上面的脚本无法看到PID的变化，所以另外进行实验观察

```
root@ubuntu1:/home/gggxxl/Desktop# bash prc_migrate.sh
host=ubuntu1 pid=3398
```

```
root@ubuntu2:/home/gggxxl/Desktop# ps -ef|grep helloer-c
root        3398    3470  0 09:23 pts/1    00:00:00 ./helloer-c
```

  可见**PID**也没有变化

- 共享可执行文件

  实验过程发现，当进程依赖的可执行文件不在共享目录中时会迁移失败：

```
root@ubuntu2:~# criu restore -D /home/mpi_share/img -j;
  2625: Error (criu/files-reg.c:1720): Can't open file home/gggxxl/Desktop/hello
er-c on restore: No such file or directory
  2625: Error (criu/files-reg.c:1654): Can't open file home/gggxxl/Desktop/hello
er-c: No such file or directory
  2625: Error (criu/mem.c:1279): `- Can't open vma
Error (criu/cr-restore.c:2300): Restoring FAILED.
```

  当把依赖文件放入共享目录后，该问题得到解决。（为什么呢？）

- 脚本中的`ssh -t root@node2 "command"`：

  -t是指给ssh分配一个伪终端，使得ssh以交互模式工作，这样可以避免以下问题：

```
root@ubuntu1:/home/gggxxl/Desktop#   2679: Error (criu/tty.c:1016): tty: Don't h
ave tty to inherit session from, aborting
  2679: Error (criu/files.c:1203): Unable to open fd=0 id=0x4
Error (criu/cr-restore.c:2300): Restoring FAILED.
```

  即CRIU恢复的shell-job找不到可以运行的终端。但是这也带来ssh占用终端的问题，需要结束ssh才能继续执行后续脚本语句，这也是为什么脚本3.12不能获取迁移后的PID。

3.2 Docker容器的热迁移

配置好Docker环境后，按照CRIU官方文档[5]进行Docker容器的热迁移测试：

3.21 样例Docker容器的热迁移脚本

```
#!/bin/bash
# set -v on
# 镜像的迁移
image-migrate(){
    docker save busybox:latest > /home/mpi_share/busybox.tar
    ssh root@node2 "cd /home/mpi_share/; docker load -i busybox.tar"
    #克隆容器
    ssh root@node2 "docker create --name looper-clone --security-opt
seccomp:unconfined busybox:latest /bin/sh -c 'i=0; while true; do echo $i;
i=$(expr $i + 1); sleep 1; done'"
}

# 在node1和node2上删除有关容器 在node1开始运行一个容器 CID记录容器ID
reset(){
    ssh root@node2 "docker stop looper-clone; docker rm looper-clone"
    docker stop looper
    docker rm looper
    #提前复制镜像
    #image-migrate
```

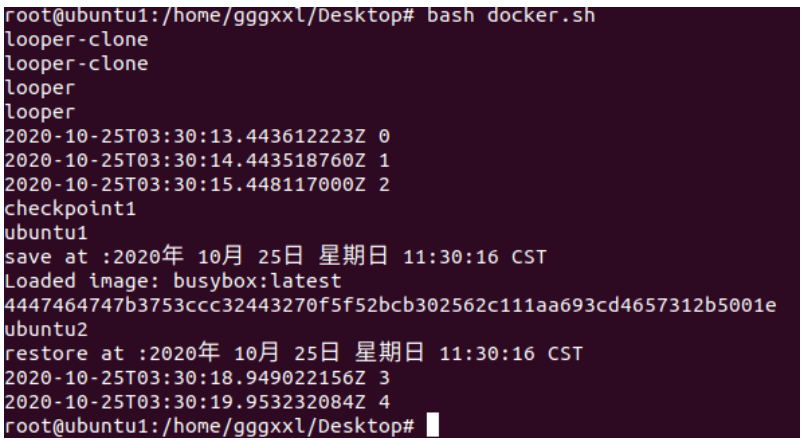
```
    CID=`docker run -d --name looper --security-opt seccomp:unconfined
busybox:latest  \
         /bin/sh -c 'i=0; while true; do echo $i; i=$(expr $i + 1); sleep 1;
done'`
    sleep 2
}

# 在node1将断点保存到共享目录 在node2克隆容器并从断点恢复运行
migrate(){
    docker logs looper -t
    rm -rf /home/mpi_share/*
    docker checkpoint create --checkpoint-dir=/home/mpi_share looper
checkpoint1
    hostname
    echo "save at :`date`"
    #迁移中复制镜像
    image-migrate
    ssh root@node2 "
        docker start --checkpoint-dir=/home/mpi_share/${CID}/checkpoints \
                --checkpoint=checkpoint1 looper-clone
        hostname
        echo 'restore at :`date`'
        sleep 2
        docker logs looper-clone -t"
}

reset
migrate
```

3.22 运行结果

在node1上运行脚本3.21，结果如下：



可以看到容器成功从ubuntu1热迁移到ubuntu2，从logs的时间戳可以看出热迁移所需时间大概为3.5s

3.23 镜像复制与其他实验细节

- **是否需要迁移时复制镜像和创建容器**：实际上，在脚本3.21中我们不需要在迁移时才进行镜像的复制，因为镜像本身不包含运行状态，所以可以在原进程保存断点并停止服务前就复制好镜像和创建容器，因为一些镜像比较大，所以可以节省很多时间。下面是提前进行镜像和容器复制的运行结果：

```
2020-10-25T03:52:18.509216791Z 2
checkpoint1
ubuntu1
save at :2020年 10月 25日 星期日 11:52:19 CST
ubuntu2
restore at :2020年 10月 25日 星期日 11:52:19 CST
2020-10-25T03:52:21.374627359Z 3
```

可以看到时间缩减为3s以下，且热迁移仍然正常进行

- 恢复断点时的断点文件夹定位：

  如果按照文档[5]的做法使用--checkpoint-dir选项进行restore：

```
docker start --checkpoint-dir=/home/mpi_share \
             --checkpoint=checkpoint1 looper-clone
```

  会出现以下错误：

```
Error response from daemon: shim error: open /home/mpi_share/checkpoint1/config.json: no such file or dir
ectory
```

  这是因为CRIU找不到断点保存的文件夹。打开/home/mpi_share可以发现：

```
/home/mpi_share/
└── cf0e531a83b26a3e31de3f5da9c4aa064454aaa0d4f76c1b5dd62760c3c46a59
    └── checkpoints
        └── checkpoint1
            ├── cgroup.img
            ├── config.json
```

  断点文件夹所在文件夹为/home/mpi_share/容器ID/checkpoints

  所以--checkpoint-dir需要指定这一文件夹，其中的容器ID可以从容器运行命令的返回信息中获取，见3.21中的$CID

3.3 观察web服务迁移的性能损耗

3.31 一个web server样例

Docker本身是一个应用容器，很多应用可以通过Docker部署，其中也包括各种web服务应用，这里以一个测速网站Librespeed为例进行迁移测试[6]，部署过程参考[7]

3.32 web服务热迁移脚本

```
#!/bin/bash
# set -v on
image-migrate(){
    docker save adolfintel/speedtest:latest > /home/mpi_share/test.tar
    ssh root@node2 "cd /home/mpi_share/; docker load -i test.tar;"
}

container-clone(){
    ssh root@node2 "docker create --name test -p 8080:80 --security-opt
seccomp:unconfined adolfintel/speedtest:latest;"
}


reset(){
    ssh root@node2 "docker stop test; docker rm test"
    killall ssh
    docker stop test
    docker rm test
```

```
    image-migrate
    container-clone
    CID=`docker run -d --name test -p 8080:80 --security-opt
seccomp:unconfined adolfintel/speedtest:latest`
}

migrate(){
    echo "migrate begin at `date`"
    rm -rf /home/mpi_share/*
    docker checkpoint create --checkpoint-dir=/home/mpi_share test checkpoint
    # 端口转发
    ssh -N -f -L node1:8080:node2:8080 node2
    ssh root@node2 "cd /home/mpi_share;
        docker start --checkpoint-dir=/home/mpi_share/${CID}/checkpoints --
checkpoint=checkpoint test"
    echo "migrate finish at `date`"
}

reset
migrate
```

3.33 简单的web benchmark脚本

```
# webbench.sh 每0.01秒请求一次web服务页面 共100次
#!/bin/bash
for var in {1..100..1}
do
    curl -I -s http://node1:8080/
    sleep 0.01
done
```

```
# autobench.sh 自动执行webbench 统计每轮webbench中成功请求次数和时间
#!/bin/bash

D0=`date +%s`
for var in {1..100..1}
do
    N=`bash webbench.sh |grep -c "HTTP"`
    D1=`date +%s`
    let dd=($D1 - $D0)
    D0=$D1
    echo "`date`:speed : $N/${dd}s"
done
```

3.34 热迁移测试

在node1上执行迁移脚本：

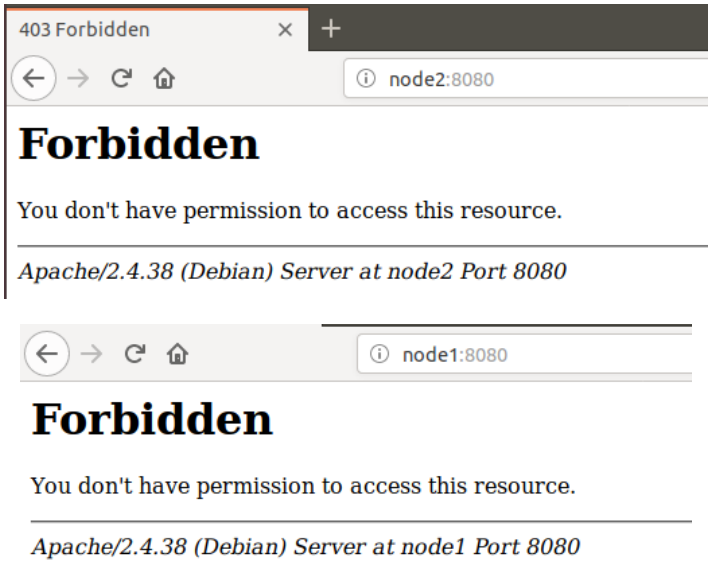
```
root@ubuntu1:/home/gggxxl/Desktop# bash migrate.sh
test
test
test
test
c71c411094967506be5c01be5455210cfedb51024c2fed3bc1b4216ba6de990c
migrate begin at 2020年 10月 25日 星期日 13:42:56 CST
checkpoint
channel 1: open failed: connect failed: Connection refused
channel 1: open failed: connect failed: Connection refused
channel 1: open failed: connect failed: Connection refused
migrate finish at 2020年 10月 25日 星期日 13:42:58 CST
root@ubuntu1:/home/gggxxl/Desktop#
```

在node2上同时执行bench脚本：

```
root@ubuntu2:/home/gggxxl/Desktop# bash autobench.sh
Sun Oct 25 13:42:53 CST 2020:speed : 100/3s
Sun Oct 25 13:42:56 CST 2020:speed : 69/3s
Sun Oct 25 13:43:01 CST 2020:speed : 87/5s
Sun Oct 25 13:43:04 CST 2020:speed : 100/3s
Sun Oct 25 13:43:06 CST 2020:speed : 100/2s
Sun Oct 25 13:43:09 CST 2020:speed : 100/3s
Sun Oct 25 13:43:12 CST 2020:speed : 100/3s
Sun Oct 25 13:43:15 CST 2020:speed : 100/3s
Sun Oct 25 13:43:17 CST 2020:speed : 100/2s
Sun Oct 25 13:43:20 CST 2020:speed : 100/3s
^C
root@ubuntu2:/home/gggxxl/Desktop#
```

- 性能损耗：可以看到13:42:56开始迁移，13:42:58结束。同时autobench结果中记录了13:42:56开始的速度下降为87/5s，即5s内完成了87次成功请求，所以在该测试下的峰值性能（以测试方来看）损耗大概为$\frac{100/3-87/5}{100/3}$ =47%。然而该数值只是在1秒内连续请求100次的测试条件下有参考价值。
- 实际上，该性能下降只是因为迁移时的服务不可用，当请求方在服务迁移期间请求的越多，看上去性能就下降的越厉害。所以用服务不可用时间反应性能损耗应该更合适，比如这里的性能损耗为2s的服务中断时间。

测试迁移后服务是否正常：



- 服务不正常：迁移后服务不正常，无论是从原IP地址 node1 通过端口转发访问，还是直接访问node2的端口，都出现403 Forbidden的问题。可能是因为web服务所需的网络条件发生了变化。

3·35 容器迁移重启测试

在迁移脚本中将restore语句修改为：

```
docker start --checkpoint-dir=/home/mpi_share/checkpoint test
```

会让Docker找不到正确的checkpoint文件，所以相当于迁移后重启容器，丢弃了运行状态。

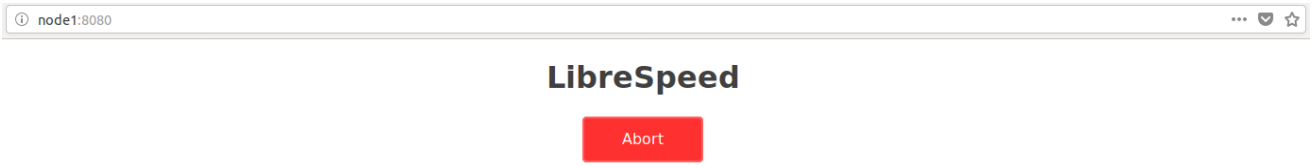
测试结果如下：

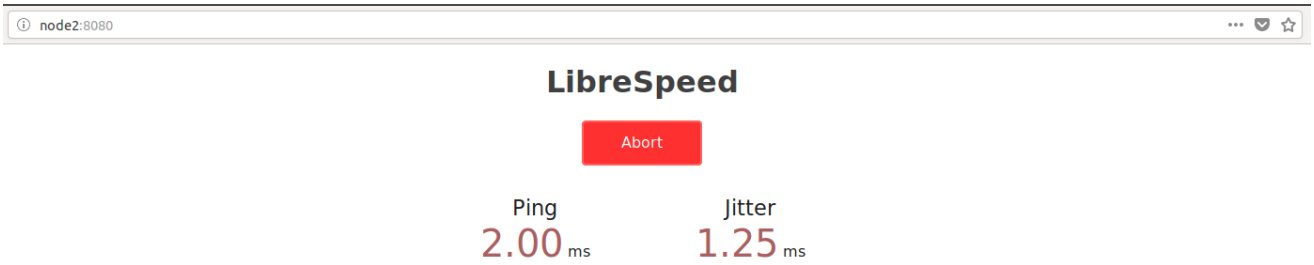
```
migrate begin at 2020年 10月 25日 星期日 13:48:07 CST
checkpoint
channel 1: open failed: connect failed: Connection refused
channel 1: open failed: connect failed: Connection refused
test
migrate finish at 2020年 10月 25日 星期日 13:48:08 CST
```

```
root@ubuntu2:/home/gggxxl/Desktop# bash autobench.sh
Sun Oct 25 13:47:55 CST 2020:speed : 100/2s
Sun Oct 25 13:47:58 CST 2020:speed : 100/3s
Sun Oct 25 13:48:01 CST 2020:speed : 100/3s
Sun Oct 25 13:48:04 CST 2020:speed : 100/3s
Sun Oct 25 13:48:06 CST 2020:speed : 67/2s
Sun Oct 25 13:48:11 CST 2020:speed : 91/5s
Sun Oct 25 13:48:14 CST 2020:speed : 100/3s
Sun Oct 25 13:48:17 CST 2020:speed : 100/3s
^C
root@ubuntu2:/home/gggxxl/Desktop#
```

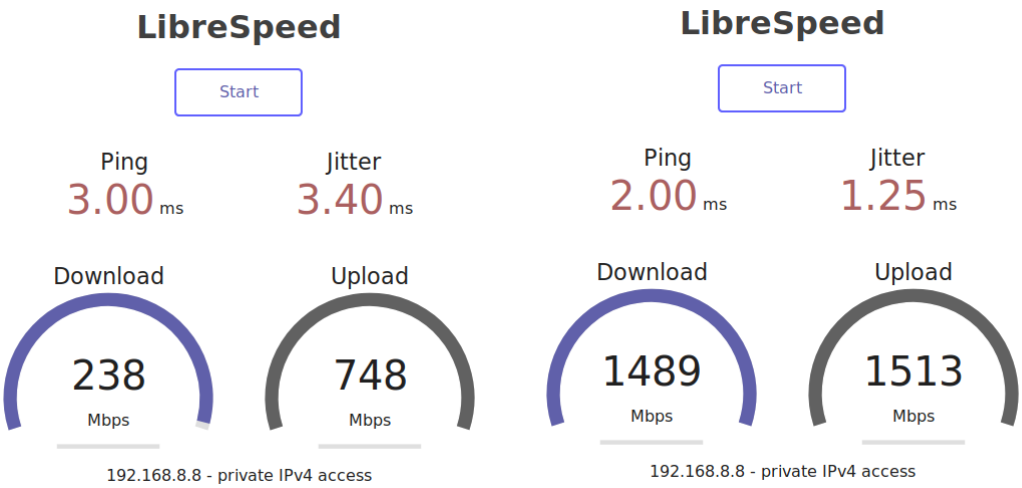
- 性能损耗：与热迁移类似

观察服务是否正常：





- 服务正常：可以看到可以从原地址node1访问，也可以从新地址node2访问
- 在迁移过程中使用服务、在迁移后使用服务的对比：分别如下图所示，可以看到迁移对测速功能的影响



3.36 端口转发和其他实验细节

- 端口转发：

  脚本中`ssh -N -f -L node1:8080:node2:8080 node2`是让ssh进行端口转发：使得其他主机可以通过node1的8080端口访问node2的8080端口，这就解决了web服务从node1迁移到node2后的访问问题，如果不设置端口转发，原来使用node1地址访问服务的客户的所请求的服务就会在迁移后不可用，端口转发可以解决这个问题。同时我们也可以从端口转发中观察web服务请求消息，比如`channel 1: open failed: connect failed: Connection refused`，这也是性能损耗的一种反映

- 迁移之前的性能损耗：

  在autobench结果中可以观察到迁移之前的性能损耗，这只是脚本停止之前服务导致的，与迁移本身无关

- **RPC引起的checkpoint错误**：

  如果在服务重启后过一段时间再进行checkpoint操作，且有autobench正在运行，那么会出现以下错误：

  ```
  migrate begin at 2020年 10月 25日 星期日 13:41:42 CST
  Error response from daemon: Cannot checkpoint container test: rpc error: code =
  2 desc = exit status 1: "criu failed: type NOTIFY errno 0\nlog file: /home/mpi_s
  hare/78dea09fb4e1fd2564787d829efad82161ffbe5d0b032c01d02f4703d3d12507/checkpoint
  s/checkpoint/criu.work/dump.log\n"
  ```

  可能是因为服务正在被使用，所以无法停止服务进行来进行checkpoint操作。一种可能的解决方案是`--leave-running`选项，但没有试过。

4.实验总结

使用CRIU和DOCKER可以方便地进行进程和容器的在线迁移。在容器的迁移中，提前进行镜像和容器的克隆可以减少迁移所需的时间和性能损耗。在对web服务的迁移及其性能损耗的测试中，发现通过端口转发和容器迁移后的重启可以应付部分场景的需求，除此以外在实验中并没有得到比较好的结果，对web服务的了解不够。

5.参考资料

[1]. 虚拟机集群搭建https://blog.csdn.net/secyb/article/details/78697976

[2]. CRIU 3.11安装和使用https://www.jianshu.com/p/2b288415896c?

[3]. Docker 17.06安装https://blog.csdn.net/NelsonCheung/article/details/109164437

[4]. Docker 卸载https://www.cnblogs.com/shmily3929/p/12085163.html

[5]. Docker 热迁移https://criu.org/Docker

[6]. Librespeedhttps://github.com/librespeed/speedtest

[7]. Librespeed Docker部署https://www.jianshu.com/p/00e8ae89224d