1 测试动态快照稳定性

1.1 测试的思路

首先我介绍一下当前环境：libvirt 中定义了两个虚拟机 overlay1 和 overlay2。overlay1 的磁盘文件是 overlay1.qcow2，overlay2 的磁盘文件也是 overlay1.qcow2。overlay1.qcow2 的后端镜像是 base.qcow2。

overlay1.qcow2 上存放这 test_in_vm.sh、monitor.sh 和 judge.sh 这三个脚本文件。这个三脚本文件的功能分别是：

```
1    test_in_vm.sh：将1～4000000这些数字写入num.txt文件中，每个数字一行。
2    monitor.sh：这个文件用于监测test_in_vm.sh这个进程是否运行结束。
3    judge.sh：这个文件用于判断num.txt是否有4000000，用于判断数据的完整性。
```

接下来我说一下测试方案：

1. 首先启动虚拟机 overlay1, 注意它的磁盘文件为 overlay1.qcow2。
2. 在宿主机环境中，ssh 登录到 overlay1 上，执行上面的 test_in_vm.sh 脚本。
3. 动态创建快照，此时虚拟机 overlay1 的磁盘文件就变成了 overlay2.qcow2。
4. 查看 overlay1 上的数据是否完整，也就是查看 overlay2.qcow2 磁盘上的数据是否完整。
5. 启动虚拟机 overlay2, 注意它的磁盘文件为 overlay2.qcow2。随后查看它上面的数据是否不完整，也就是查看 overlay1.qcow2 磁盘上的数据是否不完整。
6. 关闭虚拟机 overlay1 和 overlay2。
7. 清除之前的改动，返回到最初的环境，也就是要把 overlay2.qcow2 这个增量文件删除，将虚拟机 overlay1 的磁盘文件更换为原来的 overlay1.qcow2，将虚拟机 overlay1 的磁盘文件更换为 overlay1.qcow2，并且删除虚拟机 overlay1 的快照数据。

1.2 测试需要的环境

首先创建一个 base.qcow2 镜像：

```
1    qemu-img create -f qcow2 base.qcow2 50G
```

随后在 base.qcow2 镜像上安装 ubuntu 系统，并且在这个镜像的基础上创建增量镜像 overlay1.qcow2：

```
qemu-img create -f qcow2 -b base.qcow2 overlay1.qcow2
```

在 overlay1.qcow2 上启动虚拟机 overlay1,overlay1 的 xml 配置文件内容如下:

```
<domain type='kvm'>
    <name>overlay1</name>
    <memory>1048576</memory>
    <currentMemory>1048576</currentMemory>
    <vcpu>4</vcpu>
    <os>
        <type arch='x86_64' machine='pc'>hvm</type>
        <boot dev='cdrom'/>
    </os>
    <features>
        <acpi/>
        <apic/>
        <pae/>
    </features>
    <clock offset='localtime'/>
    <on_poweroff>destroy</on_poweroff>
    <on_reboot>restart</on_reboot>
    <on_crash>destroy</on_crash>
    <devices>
        <emulator>/usr/bin/qemu-system-x86_64</emulator>
        <disk type='file' device='disk'>
            <driver name='qemu' type='qcow2'/>
            <source file='/home/pengsida/kvm/openstack/overlay1.qcow2'/>
            <target dev='hda' bus='ide'/>
        </disk>
        <interface type='network'>
            <source network='default'/>
        </interface>
        <interface type='network'>
            <source network='default'/>
        </interface>
        <input type='mouse' bus='ps2'/>
        <graphics type='vnc' port='-1' autoport='yes' listen='0.0.0.0' keymap='
            en-us'/>
        <channel type='unix'>
            <source mode='bind' path='/var/lib/libvirt/qemu/f16x86_64.agent'/>
            <target type='virtio' name='org.qemu.guest_agent.0'/>
        </channel>
    </devices>
</domain>
```

在宿主机上安装 qemu-guest-agent:

```
sudo apt install qemu-guest-agent
```

随后启动虚拟机 overlay1:

```
# 在overlay1.xml目录下
sudo virsh define overlay1.xml
sudo virsh start overlay1
```

在虚拟机中安装 qemu-guest-agent：

```
sudo apt install qemu-guest-agent
```

然后在 overlay1 上创建两个 test_in_vm.sh、monitor.sh 和 judge.sh 这三个脚本文件。

test_in_vm.sh 脚本内容如下：

```
#!/bin/bash

times=2000

rm num.txt

num=1

for((i=0;i<=times;i++))
do
    for((j=0;j<=times;j++))
    do
        echo $num >> num.txt
        num=$[$num+1]
    done
done
```

monitor.sh 脚本内容如下：

```
#!/bin/bash

test=`ps aux | grep ./test.sh`
num=`echo $test | awk -v RS='/bin/bash ./test.sh' 'END {print --NR}'`

while (( $num >= 1 ))
do
    echo "./test.sh is running"
    sleep 5
    test=`ps aux | grep ./test.sh`
    num=`echo $test | awk -v RS='/bin/bash ./test.sh' 'END {print --NR}'`
done

echo "./test.sh is done"
```

judge.sh 脚本内容如下：

```
#!/bin/bash

num=`awk 'END {print NR}' num.txt`

if (( num == 4000000))
then
    echo "data is complete"
```

```
    else
        echo "data is not complete"
    fi
```

随后在宿主机上创建 address.sh 脚本文件，这个脚本文件可以根据虚拟机的名字返回虚拟机的 ip 地址：

```
#!/bin/bash

MAC=`awk '/virbr0/ { print $4 }' /proc/net/arp`
VM=$1

line=`virsh dumpxml $VM | grep "mac address"`
address=`echo $line | awk -F "'" '
{
    for(i=1; i<=NF; i++)
    {
        if($i ~ /'\''/)
            break
    }
    i++
    str=""
    for(;i<=NF;i++)
    {
        if($i ~ /'\''/)
            break
        temp=$i
        str=(str temp)
    }
    print str
}'`

ip=`awk '/'$address'/ {print $1}' /proc/net/arp`

echo $ip
```

首先 ssh 登录到虚拟机 overlay1:

```
# 使用address.sh脚本获得overlay1虚拟机的ip地址
overlay1=`./address.sh overlay1`
echo $overlay1 # 如果输出为空，需要重复上面的命令
ssh pengsida@$overlay1 # 假设虚拟机的用户名是pengsida
```

随后关闭虚拟机 overlay1：

```
sudo virsh destroy overlay1
```

随后开启虚拟机 overlay2：

```
sudo virsh define overlay2.xml
sudo virsh start overlay2
```

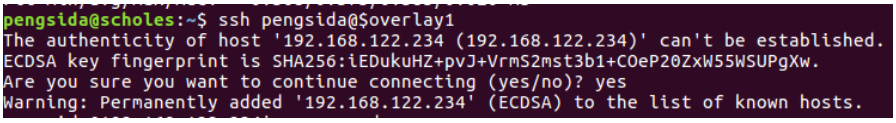
同样是 ssh 登录到虚拟机 overlay2：

```
overlay2=`./address.sh overlay2`
echo $overlay2 # 如果输出为空，需要重复上面的命令，直到获得overlay2的ip
ssh pengsida@$overlay2
```

随后关闭虚拟机 overlay2：

```
sudo virsh destroy overlay2
```

这里之所以要 ssh 登录到虚拟机 overlay1 和 overlay2 的原因是需要信任未连接过的 ip 地址，如下图所示：



1.3 测试的代码

在宿主机上创建 test.sh 脚本，内容如下：

```
#!/bin/bash

# 步骤一
# 创建虚拟机overlay1，磁盘文件为overlay1.qcow2
sudo virsh start overlay1
# 等待虚拟机启动
sleep 30


# 步骤二
# 获取虚拟机ip地址
overlay1=`./address.sh overlay1`
# ssh登录到overlay1上，执行上面的test.sh脚本
sshpass -p p1111111 ssh pengsida@$overlay1 ./test_in_vm.sh &

# 先让./test.sh脚本运行5秒
sleep 5


# 步骤三
# 动态创建快照，此时虚拟机overlay1的磁盘文件变为overlay2.qcow2
sudo virsh snapshot-create-as overlay1 snap snap-desc --disk-only --diskspec
    hda,snapshot=external,file=/home/pengsida/kvm/openstack/overlay2.qcow2 --
    atomic

sleep 5


# 步骤四
# 等待test.sh脚本执行结束
sshpass -p p1111111 ssh pengsida@$overlay1 ./monitor.sh
# 判断overlay1.qcow2上的数据是否完整
```

```
31      sshpass -p p1111111 ssh pengsida@$overlay1 ./judge.sh
32
33
34      # 步骤五
35      # 创建虚拟机overlay2，磁盘文件为overlay1.qcow2
36      sudo virsh start overlay2
37      # 等待虚拟机启动
38      sleep 60
39      # 获取虚拟机ip地址
40      overlay2=`./address.sh overlay2`
41      # 判断overlay1.qcow2上的数据是否完整
42      sshpass -p p1111111 ssh pengsida@$overlay2 ./judge.sh
43
44
45      # 步骤六
46      sudo virsh destroy overlay1
47      sudo virsh destroy overlay2
48
49
50      # 步骤七
51      sudo virsh snapshot-delete overlay1 snap --metadata
52      sudo rm /home/pengsida/kvm/openstack/overlay2.qcow2
53
54      # 将虚拟机overlay1的磁盘文件更换为overlay1.qcow2
55      sudo virsh detach-disk overlay1 hda --persistent
56      sudo virsh attach-disk overlay1 /home/pengsida/kvm/openstack/overlay1.qcow2 hda
            --subdriver qcow2 --persistent
```