# 工科创-课程大作业四

519021910050, 曾航, nidhogg@sjtu.edu.cn

2021 年 11 月 25 日

## 1 简介

## 2 实验准备

### 2.1 在 Linux 系统中开启大页

本次项目的主机操作系统为 CentOS8，我们将在上面配置大页以及虚拟机。

在 Linux 系统上，我们可以通过 hugetlbfs 来使用大页。hugetlbfs 是一个虚拟文件系统，在使用之前我们需要确保内核编译时将 $CONFIG\_HUGETLB\_PAGE$ 和 $CONFIG\_HUGETLBFS$ 选项勾选，同时也要将 hugetlbfs 挂载到特定的目录。

我们可以在 $/boot 目录下查看到内核的 config 文件。

```
CONFIG_HUGETLBFS=y
CONFIG_HUGETLB_PAGE=y
```

使用 mount 对 hugetlbfs 进行挂载，在本机上需挂载到 /dev/hugepages。

```
[zenghang@localhost ~]$ sudo mount -t hugetlbfs hugetlbfs /dev/hugepages/
```

使用 df -a 可以查看挂载的文件系统。

```
[zenghang@localhost ~]$ df -a | grep hugetlbfs
hugetlbfs              0        0         0    - /dev/hugepages
```

可以注意到 hugetlbfs 已经被挂载到/dev/hugepages.

之后，我们需要修改 /etc/sysctl.conf，修改 $vm.nr\_hugepages$ 的值，该值表示大页的数量。在修改之前，我们在 /proc/meminfo 中可以看到大页大小为 2MB，但大页数量为 0。

```
[zenghang@localhost ~]$ cat /proc/meminfo | grep Huge
AnonHugePages:    172032 kB
ShmemHugePages:        0 kB
FileHugePages:         0 kB
HugePages_Total:       0
HugePages_Free:        0
HugePages_Rsvd:        0
HugePages_Surp:        0
Hugepagesize:       2048 kB
Hugetlb:               0 kB
```

通过 vi 打开配置文件，加入 $vm.nr\_hugepages = 1024$，设定大页数量为 1024。

```
[root@localhost zenghang]# vi /etc/sysctl.conf
# sysctl settings are defined through files in
# /usr/lib/sysctl.d/, /run/sysctl.d/, and /etc/sysctl.d/.
#
# Vendors settings live in /usr/lib/sysctl.d/.
# To override a whole file, create a new file with the same in
# /etc/sysctl.d/ and put new settings there. To override
# only specific settings, add a file with a lexically later
# name in /etc/sysctl.d/ and put new settings there.
#
# For more information, see sysctl.conf(5) and sysctl.d(5).

vm.nr_hugepages=1024
```

修改完成后重启，再次查看 /proc/meminfo 便可以发现大页分配完成。

```
[zenghang@localhost ~]$ cat /proc/meminfo | grep Huge
AnonHugePages:    124928 kB
ShmemHugePages:        0 kB
FileHugePages:         0 kB
HugePages_Total:    1024
HugePages_Free:     1024
HugePages_Rsvd:        0
HugePages_Surp:        0
Hugepagesize:       2048 kB
Hugetlb:         2097152 kB
```

注:图中 *AnonHugePages* 表示的是 *Transparent HugePages* 的统计值,之后我们将禁用 *Transparent HugePages*。*HugePages Total* 表示的是 *HugePage* 池中大小。*HugePages Free* 表示的是 *HugePage* 池中未被分配 *HugePage* 数量。*HugePages Rsvd* 表示的是 *HugePage* 池中承诺被分配但还未执行分配操作的 *HugePage* 数量。

为避免虚拟机使用透明大页，我们需要禁用透明大页，通过 cat 相关配置文件查看透明大页的启用状态。

```
[zenghang@localhost ~]$ cat /sys/kernel/mm/transparent_hugepage/enabled
[always] madvise never
```

输出有三种透明大页的状态，[always] 表示透明大页启用，[never] 表示透明大页禁用，而 [madvise] 表示只在 $MADV\_HUGEPAGE$ 标志的 VMA 中使用透明大页。

我们需要通过调整一些配置文件禁用透明大页。

① 暂时关闭

将 never 写入 */sys/kernel/mm/transparent_hugepage/defrag* & *enabled* 两个文件。

```
[root@localhost transparent_hugepage]# ls
defrag  enabled  hpage_pmd_size  khugepaged  shmem_enabled  use_zero_page
[root@localhost transparent_hugepage]# echo 'never' | sudo tee /sys/kernel/mm/transparent_hugepage/defrag
never
[root@localhost transparent_hugepage]# echo 'never' | sudo tee /sys/kernel/mm/transparent_hugepage/enabled
never
[root@localhost transparent_hugepage]# cat /sys/kernel/mm/transparent_hugepage/enabled
always madvise [never]
```

② 永久关闭

修改 /etc/default/grub 文件，将 $transparent\_hugepage = never$ 添加到 $GRUB\_CMDLINE\_LINUX$ 变量当中。通过 vi 打开配置文件。

```
[root@localhost zenghang]# vi /etc/default/grub
```

```
GRUB_TIMEOUT=5
GRUB_DISTRIBUTOR="$(sed 's, release .*$,,g' /etc/system-release)"
GRUB_DEFAULT=saved
GRUB_DISABLE_SUBMENU=true
GRUB_TERMINAL_OUTPUT="console"
GRUB_CMDLINE_LINUX="crashkernel=auto resume=UUID=ebd74071-8e03-4804-a49d-bc44889bdef1 rhgb quiet transparent_hugepage=never"
GRUB_DISABLE_RECOVERY="true"
GRUB_ENABLE_BLSCFG=true
```

执行生效命令后重启。

```
[root@localhost zenghang]# grub2-mkconfig -o /boot/grub2/grub.cfg
Generating grub configuration file ...
done
```

打印 */sys/kernel/mm/transparent_hugepage/enabled* 可以发现透明大页状态为 [never]，配置生效。

```
[root@localhost zenghang]# cat /sys/kernel/mm/transparent_hugepage/enabled
always madvise [never]
```

同时在 /proc/meminfo 中的 AnonHugePages 一项变为 0。

```
[root@localhost zenghang]# cat /proc/meminfo | grep Huge
AnonHugePages:         0 kB
ShmemHugePages:        0 kB
FileHugePages:         0 kB
HugePages_Total:    1024
HugePages_Free:     1024
HugePages_Rsvd:        0
HugePages_Surp:        0
Hugepagesize:       2048 kB
Hugetlb:         2097152 kB
```

## 2.2 使用大页启动虚拟机

在原有的启动虚拟机命令下，添加 -mem-prealloc -mem-path /dev/hugepages/libvirt/qemu 来使用大页启动虚拟机。

此参数其实是调用的 *file_ram_alloc* 方法实现的大页分配，在 mem-path 下使用 mkstemp 申请了固定容量的空间而 mem-path 下挂载的是 hugelbfs 虚拟机文件系统，即在预占的大页中分配容量。

```
[root@localhost centos7]# qemu-system-x86_64 -m 1024 -smp 2 -enable-kvm -drive if=virtio,file=centos_test.qcow2,cache=none
-vnc :1 -netdev tap,id=mynet0,ifname=tap0,script=./net_ifup,downscript=no -device e1000,netdev=mynet0,mac=00:0c:29:fa:d8:01
 -mem-prealloc -mem-path /dev/hugepages/libvirt/qemu
```

启动虚拟机之后，再次查看 /proc/meminfo 中 HugePages 的信息，可以发现空闲大页数量从 1024 变为 512。我们给虚拟机分配的内存为 1024MB，即为 512 张 2kB 大小的大页所占的空间。

```
[zenghang@localhost ~]$ cat /proc/meminfo | grep Huge
AnonHugePages:         0 kB
ShmemHugePages:        0 kB
FileHugePages:         0 kB
HugePages_Total:    1024
HugePages_Free:      512
HugePages_Rsvd:        0
HugePages_Surp:        0
Hugepagesize:       2048 kB
Hugetlb:         2097152 kB
```

### 2.3 sysbench 安装

在 CentOS8 上安装 sysbench 和安装其他程序流程类似，下载源码、编译、编译安装。首先从 github 上下载 sysbench 源码。

```
[root@localhost sysbench]# git clone https://github.com/akopytov/sysbench.git
```

执行./autogen.sh 脚本，生成配置文件。

```
[root@localhost sysbench]# ./autogen.sh
autoreconf: Entering directory `.'
autoreconf: configure.ac: not using Gettext
autoreconf: running: aclocal -I m4
```

执行./configure，进行预编译，参数–prefix 指定安装目录在/usr 下，同时为 root 用户安装该命令，也使其获得更多权限。

```
[root@localhost sysbench]# ./configure --prefix=/usr
configure: loading site script /usr/share/config.site
checking build system type... x86_64-pc-linux-gnu
checking host system type... x86_64-pc-linux-gnu
checking target system type... x86_64-pc-linux-gnu
checking for a BSD-compatible install... /bin/install -c
checking whether build environment is sane... yes
checking for a thread-safe mkdir -p... /bin/mkdir -p
```

执行 make 进行编译，-j8 表示同时使用 8 个核进行编译。

```
[root@localhost sysbench]# make -j8
```

使用 make install 进行安装，也可以使用 make uninstall 进行卸载。

```
[root@localhost sysbench]# make install
Making install in third_party/luajit
```

```
[root@localhost sysbench]# make uninstall
Making uninstall in third_party/luajit
```

除了从源码编译之外，我们还可以使用 yum install sysbench 直接安装适合的版本。
首先添加相关仓库。

```
[zenghang@localhost ~]$ curl -s https://packagecloud.io/install/repositories/akopytov/sysbench/script.rpm.sh | sudo bash
```

通过 yum install 安装 sysbench。

```
[root@localhost sysbench]# yum install sysbench
```

通过 sysbench –version 查看安装的 sysbench 版本。

```
[root@localhost sysbench]# sysbench --version
sysbench 1.0.20
```

通过 sysbench memory help 查看我们将使用的内存测试的相关参数，在本项目中，关键是–memory-hugetlb 是否打开。

```
[root@localhost centos7]# sysbench memory help
sysbench 1.0.20 (using bundled LuaJIT 2.1.0-beta2)

memory options:
  --memory-block-size=SIZE    size of memory block for test [1K]
  --memory-total-size=SIZE    total size of data to transfer [100G]
  --memory-scope=STRING       memory access scope {global,local} [global]
  --memory-hugetlb[=on|off]   allocate memory from HugeTLB pool [off]
  --memory-oper=STRING        type of memory operations {read, write, none} [write]
  --memory-access-mode=STRING memory access mode {seq,rnd} [seq]
```

# 3 测试–memory test

在该测试中，我们的测试环境为 VMware 所构建的 CentOS8 虚拟机上使用 Qemu 嵌套虚拟化，创建的 CentOS7 虚拟机。在嵌套的情况下，出现了与非嵌套不同的状况。我们是否开启 host 大页对于虚拟机大内存的访问提升十分有限，host 大页的提升效果不如虚拟机自身大页提升的效果。

在嵌套虚拟化的情况下，我们分 4 个场景进行测试，分别为是否使用大页启动虚拟机、是否在虚拟机中开启大页（大页为 2M）的组合。

同样，我们需要进行无关变量的控制，我们设定 memory-block-size、memory-total-size、memory-access-mode 以及线程数等参数在测试过程中保持一致。

同时设置 memory-total-size 为 20G，memory-access-mode 分别为顺序访问以及随机访问，同时比较 memory-block-size 分别为 4M、512M、4G 时各个场景的表现。

同非嵌套情况一致，使用 hugeadm 更加灵活地配置大页数量。

hugeadm –pool-pages-min 2MB:10，设置最小的大页数量为 10。

hugeadm –pool-pages-max 2MB:4100，设置最大的大页数量为 4100，由于我们需要在 8G 的场景下进行测试，故最大的大页总大小需要大于 8G。

4 个场景详细定义如下：

**场景 1**，在该场景下，使用 2M 大页启动虚拟机，禁用透明大页，打开虚拟机上的大页，使用 2MB-2MB 表示。

**场景 2**，在该场景下，使用 2M 大页启动虚拟机，我们未在虚拟机上设置大页，使用 2MB-None 表示。

**场景 3**，我们不使用主机上的大页启动虚拟机，并且仅在虚拟机中使用 2M 大页，使用 None-2MB 表示。

**场景 4**，在该场景下，不使用大页启动虚拟机，且我们未在虚拟机上设置大页，使用 None-None 表示。

### 3.0.1 memory-block-size=4M

我们设定 memory-block-size=4M，得到下表结果。

| 场景 | 平均速率 (MiB/s) | 平均时间 (s) | 平均延迟 (ms) | 访问模式 |
|---|---|---|---|---|
| 2MB-2MB | 11316.37 | 1.8080 | 1801.29 | seq |
| 2MB-None | 12009.98 | 1.7037 | 1697.73 | seq |
| None-2MB | 10916.87 | 1.8739 | 1866.84 | seq |
| None-None | 11451.05 | 1.7856 | 1779.40 | seq |
| 2MB-2MB | 1288.06 | 15.9107 | 15896.45 | rnd |
| 2MB-None | 1279.96 | 16.0188 | 16006.57 | rnd |
| None-2MB | 1172.72 | 17.4806 | 17460.70 | rnd |
| None-None | 1237.91 | 16.5511 | 16535.64 | rnd |

表 1: memory-block-size=4M 时，各个场景的表现

#### 3.0.2 memory-block-size=512M

我们设定 memory-block-size=512M，得到下表结果。

| 场景 | 平均速率 (MiB/s) | 平均时间 (s) | 平均延迟 (ms) | 访问模式 |
|---|---|---|---|---|
| 2MB-2MB | 9620.56 | 2.1300 | 2129.13 | seq |
| 2MB-None | 9087.91 | 2.2521 | 2251.27 | seq |
| None-2MB | 8128.01 | 2.5167 | 2515.72 | seq |
| None-None | 8561.17 | 2.3824 | 2380.25 | seq |
| 2MB-2MB | 453.35 | 45.1798 | 45178.93 | rnd |
| 2MB-None | 258.96 | 79.1285 | 79127.41 | rnd |
| None-2MB | 375.73 | 54.5201 | 54519.08 | rnd |
| None-None | 233.78 | 87.6066 | 87605.84 | rnd |

表 2: memory-block-size=512M 时，各个场景的表现

# 4 结果与原因分析

## 4.1 VMware 嵌套虚拟化测试的结果分析

若总是以 2MB-2MB 为标准值，我们可以得到各个场景的比例值：

| 场景 | 2MB-2MB | 2MB-None | None-2MB | None-None |
|---|---|---|---|---|
| 4M | 1.0000 | 1.0068 | 1.0987 | 1.0403 |
| 512M | 1.0000 | 1.7514 | 1.2067 | 1.9391 |

表 3: 随机访问下，嵌套虚拟机各个场景的平均访问时间比例

| 场景 | 2MB-2MB | 2MB-None | None-2MB | None-None |
|---|---|---|---|---|
| 4M | 1.0000 | 0.9423 | 1.0364 | 0.9876 |
| 512M | 1.0000 | 1.0573 | 1.1815 | 1.1185 |

表 4: 顺序访问下，嵌套虚拟机各个场景的平均访问时间比例

### 4.2 原因分析

## 5 总结

### 5.1 实验结论

在本项目中，实现了对 Libvirt 工具的编译与安装，使用 *.xml 文件定义 Qemu 虚拟机，并利用 C 与 Python 的接口函数获取虚拟机的基本信息，如 ID、名字、memory 信息以及虚拟 CPU 个数等。同时也借助 Python API 获取了虚拟机网络与磁盘相关的详情信息。本项目也使用了 virt-manager 这类图形化管理系统，加深了对虚拟机的底层与管理的理解。除此之外，本项目中也借助 Python API 实现了对虚拟机开机、暂停、定义、关机等操作的控制实现。