

EI313 Project 4

游灏溢 (519030910193, yuri-you@sjtu.edu.cn)

工程实践与科技创新 III – D
计算机科学与技术系，上海交通大学

摘要 本次项目中，在 host 和 qemu 中分别使用巨页，以并比较使用前后内存读写性能。正常操作系统中页大小为 4k, 但可以允许用户设置巨页。巨页大小一般在 1m 以上，本项目中设置了 1G 的巨页。巨页可以减少内存查页的次数，从而提升内存性能。

1 Host 主机打开 Hugepage

1.1 挂载 Hugetlbfs

打开文件

```
sudo gedit /etc/libvirt/qemu.conf
```

取消挂载注释，将文件中 hugetlbfs_mount = "/dev/hugepages" 前的注释删除

1.2 Grub 中修改 Hugepage 设置

打开 grub 设置

```
sudo gedit /etc/default/grub
```

找到 GRUB_CMDLINE_LINUX= 这一行，改为

```
GRUB_CMDLINE_LINUX="transparent_hugepage=never default_hugepagesz=${hugepage_size} hugepagesz=${hugepage_size} hugepages=${hugepage_number}"
```

其中第一项是关闭透明大页，第二第三项设置巨页的大小，最后一项是设置巨页的数量。值得说明的是，巨页数量是尽量使得满足，如果无法打开这么多巨页，实际操作系统中巨页数量会小于设置的数量。

1.3 重启并查看实际设置

本人物理机是 16G 内存。我选择了使用 1G 大小的巨页，并且设置 8 张巨页，实际效果如下。

```
(base) yuri@yuri-Inspiron-7590:~/桌面$ cat /proc/meminfo | grep -i hugepage
AnonHugePages:      0 kB
ShmemHugePages:     0 kB
FileHugePages:      0 kB
HugePages_Total:    0
HugePages_Free:     0
HugePages_Rsvd:     0
HugePages_Surp:     0
Hugepagesize:       2048 kB
```

图 1. 打开巨页前

```
(base) yuri@yuri-Inspiron-7590:~$ cat /proc/meminfo | grep -i hugepage
AnonHugePages:      0 kB
ShmemHugePages:     0 kB
FileHugePages:      0 kB
HugePages_Total:    8
HugePages_Free:     8
HugePages_Rsvd:     0
HugePages_Surp:     0
Hugepagesize:       1048576 kB
```

图 2. 打开巨页后

2 创建虚拟机并使用巨页运行虚拟机

2.1 使用 virt-manager

按照 project3 的方式创建一个新的虚拟机。值得注意的是内存设置为 8G 并且要自定义设置。如图 ??



图 3. 内存分配 8G



图 4. 自定义安装配置

之后在配置版面，修改 xml 配置，其中添加使用巨页的配置，让虚拟机开机使用巨页。

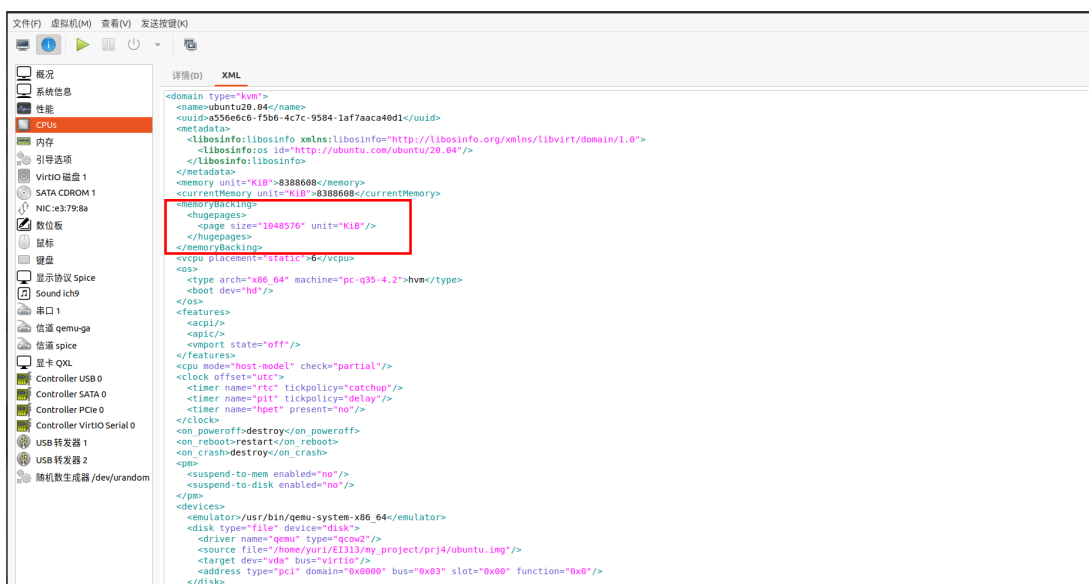


图 5. 虚拟机使用巨页

之后选择 host 是否使用巨页就只需要觉得是否添加该项配置。之后实验就通过对比添加和不添加巨页。

2.2 使用 virsh

首先使用 qemu 创建新的虚拟机

```
qemu-img create -f qcow2 ubuntu.img 30G
sudo ./qemu -m 8G --enable-kvm ubuntu.img -cdrom ./ubuntu-20.04.1-desktop-amd64.iso
```

之后创建 xml 文件

```
sudo virsh create ./ubuntu.xml
gedit ubuntu.xml
```

其中 xml 文件内容注意添加使用巨页的配置，如 2.1 类似。

创建后安装完操作系统后关机，之后就可以通过 **virsh** 用巨页打开虚拟机

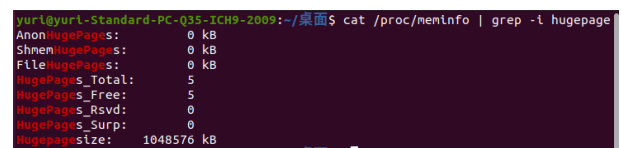
2.3 在虚拟机中打开巨页

创建打开虚拟机后，类似于 host 主机，同样可以选择在虚拟机中修改 grub 来选择是否打开虚拟机内部的巨页。



```
yuri@yuri-Standard-PC-Q35-ICH9-2009:~/桌面$ cat /proc/meminfo | grep -i hugepage
AnonHugePages:    0 kB
ShmemHugePages:   0 kB
FileHugePages:    0 kB
HugePages_Total:  0
HugePages_Free:   0
HugePages_Rsvd:   0
HugePages_Surp:   0
Hugepagesize:     2048 kB
```

图 6. 打开巨页前



```
yuri@yuri-Standard-PC-Q35-ICH9-2009:~/桌面$ cat /proc/meminfo | grep -i hugepage
AnonHugePages:    0 kB
ShmemHugePages:   0 kB
FileHugePages:    0 kB
HugePages_Total:  5
HugePages_Free:   5
HugePages_Rsvd:   0
HugePages_Surp:   0
Hugepagesize:     1048576 kB
```

图 7. 打开巨页后

可以发现，由于虚拟机分配了 8G 内存，打开的巨页个数只能至多为 5，其余内存以普通页形式分配着。

3 使用 sysbench 测试 memory 读写速度

sysbench 是 Linux 系统中用于测量系统性能的软件（可能最近也开始支持 windows），在本项目中使用其来测量内存。首先需要安装 **sysbench**，使用 apt 即可（注：这是于虚拟机中安装）。

```
sudo apt install sysbench
```

安装之后就可以使用 **sysbench -test=memory run** 来测量内存性能，如图 8 所示。而且测试可以添加很多参数，使用

```
sysbench --test=memory --help
```

以下介绍几个重要参数

```
sysbench 1.0.18 (using system LuaJIT 2.1.0-beta3)

Running the test with following options:
Number of threads: 1
Initializing random number generator from current time


Running memory speed test with the following options:
  block size: 1048576KiB
  total size: 10240MiB
  operation: write
  scope: global


Initializing worker threads...

Threads started!

Total operations: 10 ( 0.90 per second)
10240.00 MiB transferred (919.15 MiB/sec)


General statistics:
  total time:          11.1394s
  total number of events: 10


Latency (ms):
  min:                 1096.84
  avg:                 1113.92
  max:                 1191.63
  95th percentile:    1191.92
  sum:                 11139.17


Threads fairness:
  events (avg/stddev): 10.0000/0.00
  execution time (avg/stddev): 11.1392/0.00

yuri@yuri-Standard-PC-Q35-ICH9-2009:~/桌面$
```

图 8. 使用 Sysbench 测试内存

1. `-memory-block-size=`: 设置每次的内存测试块大小
2. `-memory-total-size=`: 设置总共的内存测试大小，当测试内存的量超过后，就结束测试。
3. `-threads=`: 设置测试内存使用的进程数。
4. `-memory-access-mode=`: 设置测试内存的方式，一共有两种参数，`[seq/rnd]`，分别代表顺序访问内存或随机访问内存。
5. `-memory-hugetlb=`: 设置是否使用巨页来测量，一共有两种参数 `[on/off]`。
6. `-time=`: 设置重复该项测试的次数。

值得说明的是，sysbench 测试方式是先向内存中申请 `memory-block-size` 的区域，再使用该区域进行测试。所以 `memory-block-size` 大小应该小于操作系统能分配的大小。并且当 `memory-hugetlb` 打开时，分配的是巨页，测试申请大小应该小于系统中巨页的总内存。

4 实验测试过程及结论

由于 sysbench 测试参数很多，而且很多参数可以选择的范围也很广，故本人先打算做预实验，先探究基本的比较测试模式。

4.1 预实验

如图 9 所示, 选择参数的时候 threads 个数选择了 1,2,4; blocksize 选择了 16k,4m,1G; 再比较在这些情况下 Host 开不开巨页的区别, Virtual Machine 开不开巨页的区别。在这些情况下使用参数 `-time=100` 以重复 100 次来保证实验结果的可靠性。

根据实验测试结果, 可以发现几个问题, 在此根据所查询的资料以及操作系统、体系结构的知识给出以下分析:

1. 顺序访问速度远大于随机访问。这是因为顺序访问基本上连续的区域上会有缓存, 切换页表次数也会远小于随机访问
2. 当使用多线程时候, 测试结果能迅速上升, 因为多个进程能提升测试效率。但 Threads 数量大于 2 时候, 开不开巨页的区别很小, 尤其是开 4 进程时候几乎可以忽略不计。可能是因为多个进程会导致访问的延迟被多个进程所均摊, 导致本来就不大的性能差异被掩盖。
3. 在 blocksize 很小时候 (16k), 开不开巨页基本上没有差别。这是因为缓存会影响测试结果, 导致也行根本不需要访问页表, 从而导致巨页开不开基本上没有用处。
4. 对于随机访问, blocksize 越大, 访问速度越慢。这是由于 size 越大, 缺页的可能性就越多。而对于顺序访问, blocksize 很小时候速度也不是很快, 这可能是由于速度太快, sysbench 在内存小于 1M 时候框架程序开销的影响占比较大。

4.2 正式实验

根据预实验的分析, 本人选择使用 1 线程, block_size 从 1m 逐步增长到 1G, 再对四种情况下做对比, 其中 YY 为均使用巨页, YN 为只使用 host 巨页, NY 为只是用 VM 巨页, NN 为均不使用。测试使用 `total_blocksize=100G`, `-time=100` 以增加实验重复次数来提升可靠性。每组测试记录速度与每一次的运行时间, 最后使用 python 进行可视化操作, 结果如图 10, 11

4.3 实验结果分析

顺序访问

根据图 12 与图 13 可知, 顺序访问时, 是否开启巨页区别不大。个人分析可能是因为顺序访问的速度太快, 巨页带来的影响不大, 被测试的误差所掩盖。但是总体看来, 均开启的速度会稍微大一些, 时间也稍微少一些, 这是因为两边巨页都使用会使得地址的映射是连续的, 而关闭任意一个都会破坏这个连续性, 从而带来更多 tlb miss 和 page fault。所以只有均开启时候速度会稍微大一些。

随机访问

根据图 10 与图 11 可知。当 blocksize 很小时候, 四种情况基本上没有什么区别, 因为这时候由于块很小, 无论是否打开巨页, page fault 和 tlb miss 的次数都不多, 所以区别不大。但是随着 blocksize 增大时候, 均开启巨页速度下降最慢, 其次是内外只开一个巨页, 都不开是最慢的。这是因为 blocksize 很大时候, page fault 和 tlb miss 的次数都会增多, 而开启巨页会导致 page 的大小变大, 从而这两个次数减小, 所以开启巨页减小的比例就变小。

4.4 附录

相关数据、绘图代码文件请参见 “<https://github.com/yuri-you/EI313.git>”。

Host Hugepage	VM Hugepage	threads	block_size	seq speed	rnd speed
0	0	1	16k	21489.3175	3398.0042
0	0	2	16k	19550.5233	1096.4619
0	0	4	16k	22299.1577	1260.6672
0	0	1	4m	24416.4787	756.8297
0	0	2	4m	48170.497	1021.4126
0	0	4	4m	80446.7172	1369.2616
0	0	1	1g	12334.0466	370.8526
0	0	2	1g	22650.4281	698.9611
0	0	4	1g	41829.3963	1343.2479
0	1	1	16k	21390.0861	3088.8954
0	1	2	16k	18950.203	1106.0864
0	1	4	16k	22988.7913	1251.0543
0	1	1	4m	24979.3254	786.3632
0	1	2	4m	48159.6749	1033.5148
0	1	4	4m	82044.9178	1374.4198
0	1	1	1g	12450.3414	392.0733
0	1	2	1g	20574.3717	719.3826
0	1	4	1g	40360.8792	1353.1473
1	0	1	16k	21312.607	3596.7497
1	0	2	16k	19535.9521	1116.4343
1	0	4	16k	22616.0329	1263.9831
1	0	1	4m	24672.3879	784.4767
1	0	2	4m	48468.879	1061.7184
1	0	4	4m	81259.7392	1390.7832
1	0	1	1g	12334.7664	472.7738
1	0	2	1g	20709.2176	949.1525
1	0	4	1g	41855.2364	1783.9034
1	1	1	16k	21112.1922	3492.8032
1	1	2	16k	19619.9609	1131.3092
1	1	4	16k	23284.8976	1258.6685
1	1	1	4m	24950.5921	778.8265
1	1	2	4m	48223.6215	1047.6738
1	1	4	4m	79982.3076	1365.4244
1	1	1	1g	12809.8116	478.7764
1	1	2	1g	23004.5716	978.4165
1	1	4	1g	40351.3271	1776.8106

图9. 速度与各个参数的关系

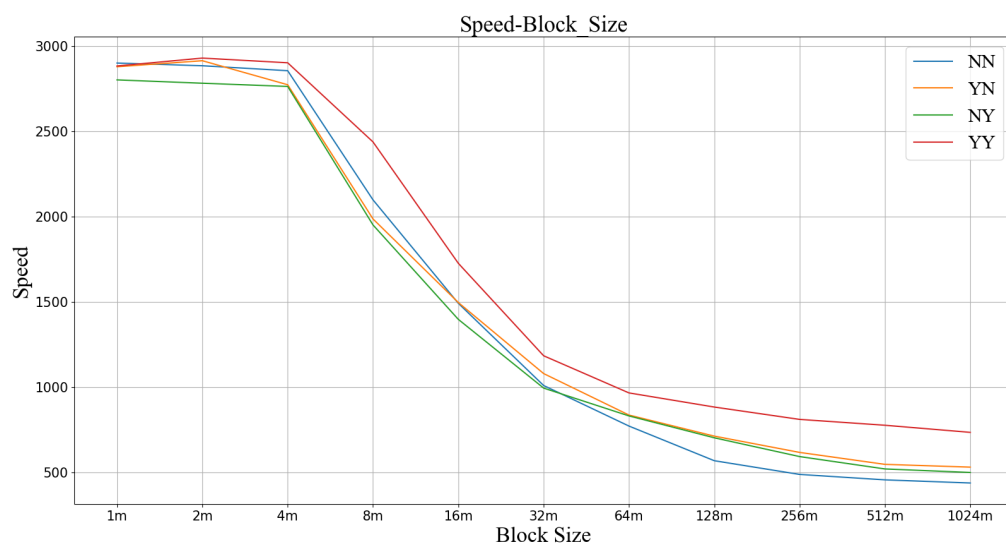


图 10. 随机访问下速度与 blocksize 的关系

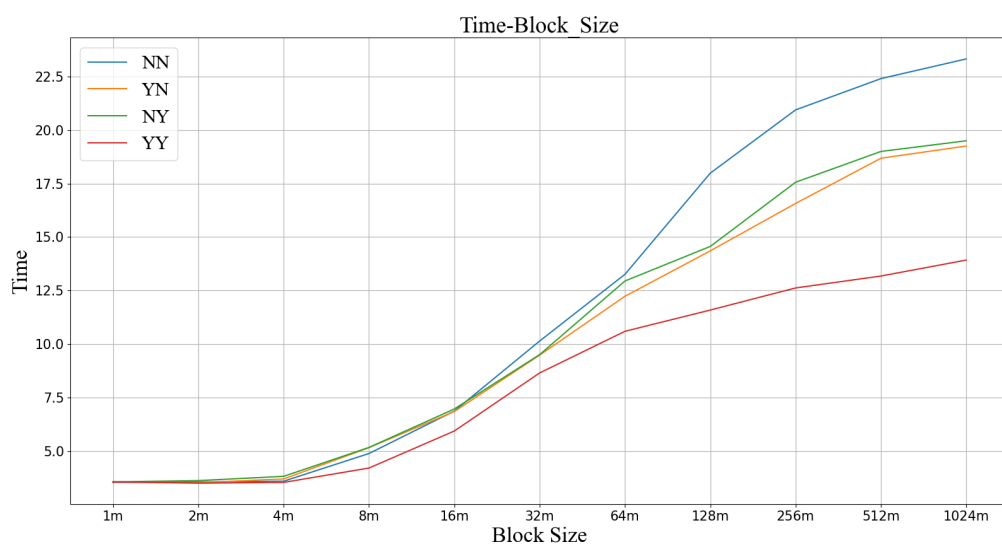


图 11. 时间与 blocksize 的关系

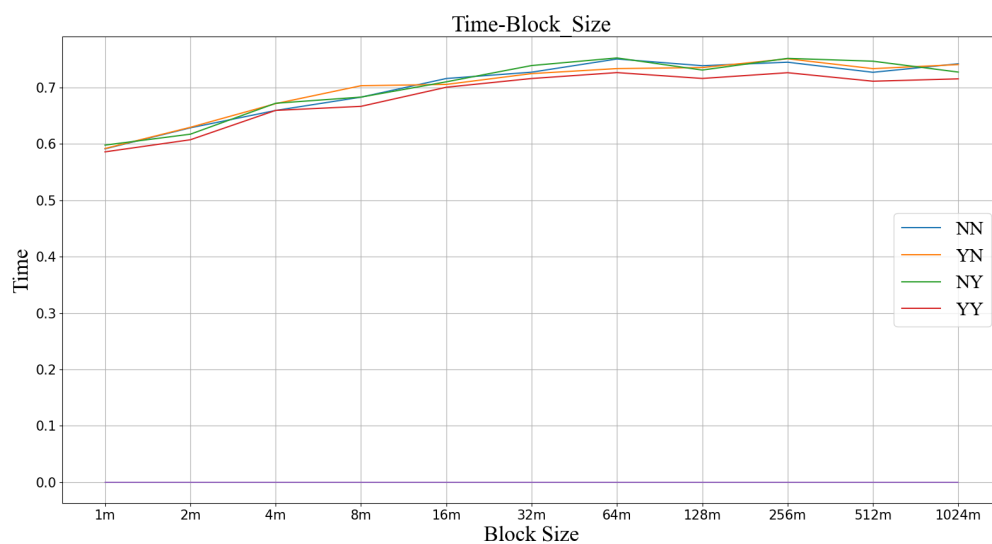


图 12. 顺序访问下速度与 blocksize 的关系

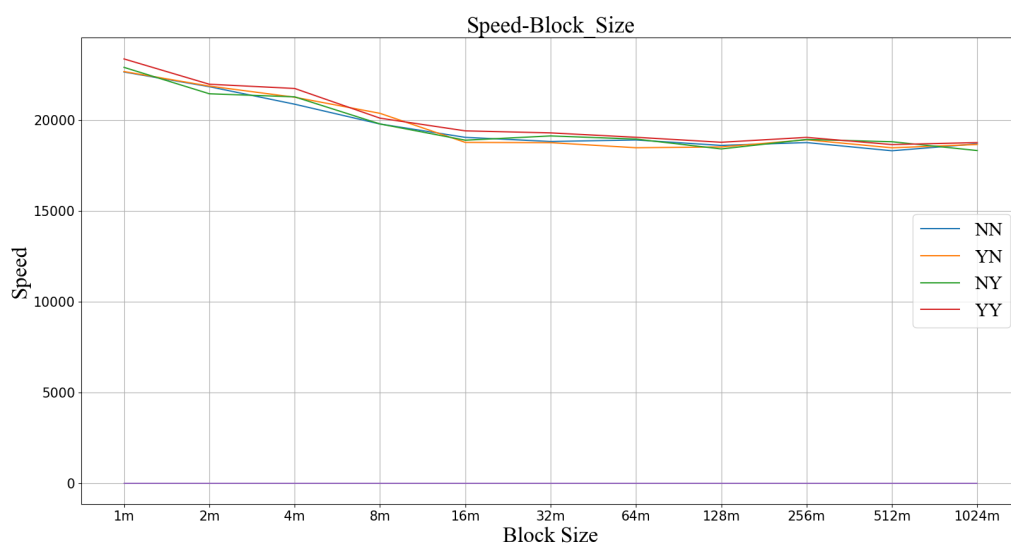


图 13. 顺序访问下时间与 blocksize 的关系