论文阅读与前期工作总结

姓名：陈童菲 陈至渲 高东育 郭朓

学号：17343013，17343020，17343028,17343035

前期工作

使用示意图展示普通文件IO方式(fwrite等)的流程，即进程与系统内核，磁盘之间的数据交换如何进行？为什么写入完成后要调用fsync?

1. 用文件映射的方式操作文件的工作流程是:

   (1) 使用open()系统调用打开文件，并返回文件描述符

   ```
   #include <sys/types.h>
   #include <sys/stat.h>
   #include <fcntl.h>
   int fd = open(const char *name, int flags, mode_t mode);
   ```

   (2) 使用mmap()建立内存映射，并返回映射首地址指针

   ```
   #include <sys/mman.h>
   void* mmap(void* addr, size_t len, int port, int flags, int fd, off_t offset);
   ```

   (3) 对映射文件进行各种操作，如printf、 sprintf、memcpy等

   ```
   #include <stdio.h>
   sprintf(char *string, char *format, arg_list);
   ```

   (4) 用munmap解除映射

   ```
     #include <sys/mman.h>
     int munmap(void* addr, size_t len);
   ```

   (5) 调用msync实现磁盘上文件内容与共享内存区的内容一致

   ```
   #incldue <sys/mman.h>
   int msync(void* addr, size_t len, int flags);
   ```

   (6)调用close关闭文件fd

```
#include <unistd.h>
int close(int fd);
```

2. 文件映射与普通I/O有什么区别?

   普通I/O是首先用open系统调用打开文件，然后使用read, write, lseek等调用进行顺序或者随机的I/O，每一次I/O都需要一次系统调用。而文件映射使得进程间通过映射同一个普通文件实现内存共享，或者使用特殊文件提供匿名内存映射。对于普通文件，它减少了数据的复制，对于特殊文件，它用于进程之间的通信。

3. 为什么写入完成后要调用msync?

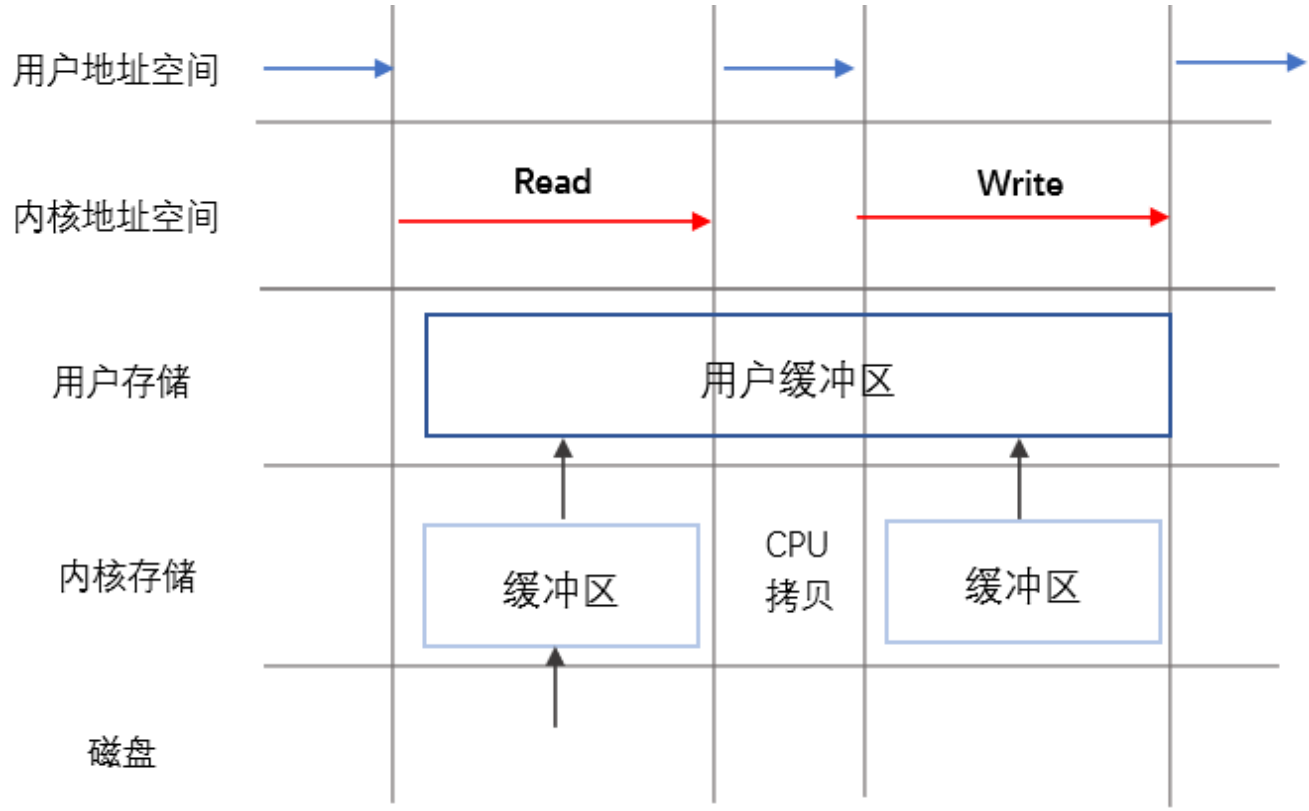
   调用msync实现磁盘上文件内容与共享内存区的内容一致。

4. 文件内容什么时候被载入内存?

   建立文件映射的过程中，并没有实际的拷贝数据，文件没有被载入内存，真正的数据拷贝是在缺页中断处理时进行的。

简述文件映射的方式如何操作文件。与普通IO区别？为什么写入完成后要调用msync？文件内容什么时候被载入内存?

(使用什么函数，函数的工作流程)



大多数 unix 系统为了减少磁盘 IO，采用了“延迟写”技术，也就是说当我们执行完 write 调用后，数据并不一定立马被写入磁盘（可能还是保留在系统的 buffer cache 或者 page cache 中），这样当主机突然断电，这些我们本以为已经写入到磁盘文件的数据可能就 会丢失；所以当我们需要确保数据被完整正确的写入磁盘（譬如数据库的持久化），则 需要调用同步函数 fsync，它会一直阻塞直到数据全部被写入到硬盘。

参考Intel的NVM模拟教程模拟NVM环境，用fio等工具测试模拟NVM的性能并与磁盘对比（关键步骤结果截图）。

(推荐Ubuntu 18.04LTS下配置，跳过内核配置，编译和安装步骤)

1. 配置grub

- 修改etc/grub文件

  `sudo gedit /etc/default/grub`

  把grub文件的GRUB_CMDLINE_LINUX_DEFAULT="quiet”里加 memmap=xxG!yyG

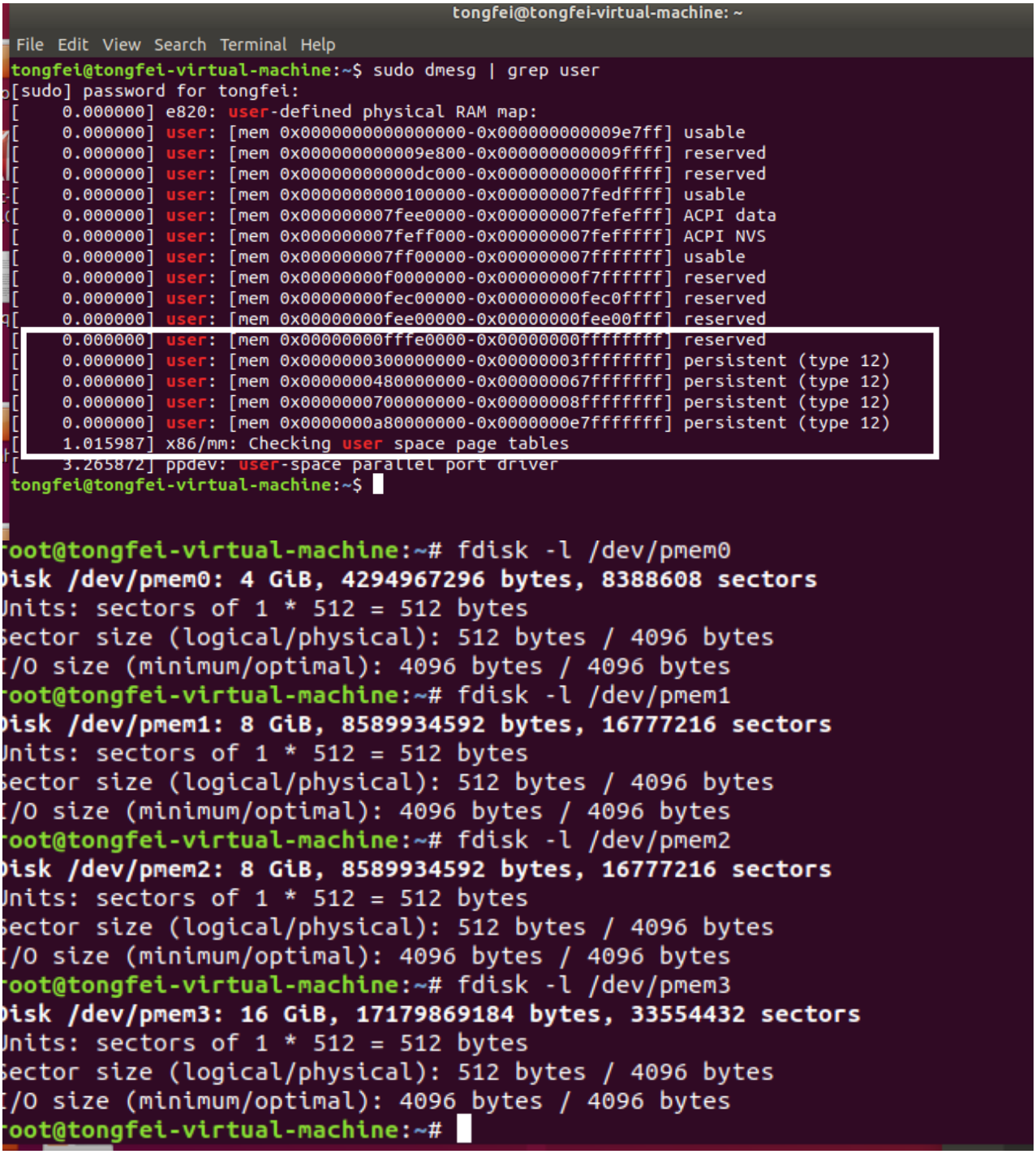
  变成

  `GRUB_CMDLINE_LINUX_DEFAULT="quiet memmap=4G!12G memmap=8G!18G memmap=8G!28G memmap=16G!42G"`

- 更新grub.cfg文件

  `sudo update-grub`

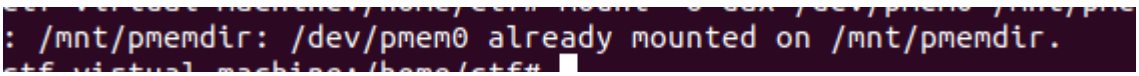
- 将虚拟机重启后，应该能够看到仿真设备呈现 /dev/pmem0...pmem3。尝试获取保留的内存区支持持久性内存仿真，定义持久性 (type 12) 区域的各个内存范围将下图所示。



可以看到系统已经识别出pmem0m持久化设备，我们可以像操作磁盘设备一样对其操作。

2. DAX - Direct Access 扩展

- '# egrep '(DAX|PMEM)' /boot/config-`uname -r`'
- ' # mkdir /mnt/pmemdir'
- '# mkfs.ext4 /dev/pmem3'
- '# mount -o dax /dev/pmem3 /mnt/pmemdir' 现在可以在新加载的分区上创建文件，并作为输入提供给 NVML 池。

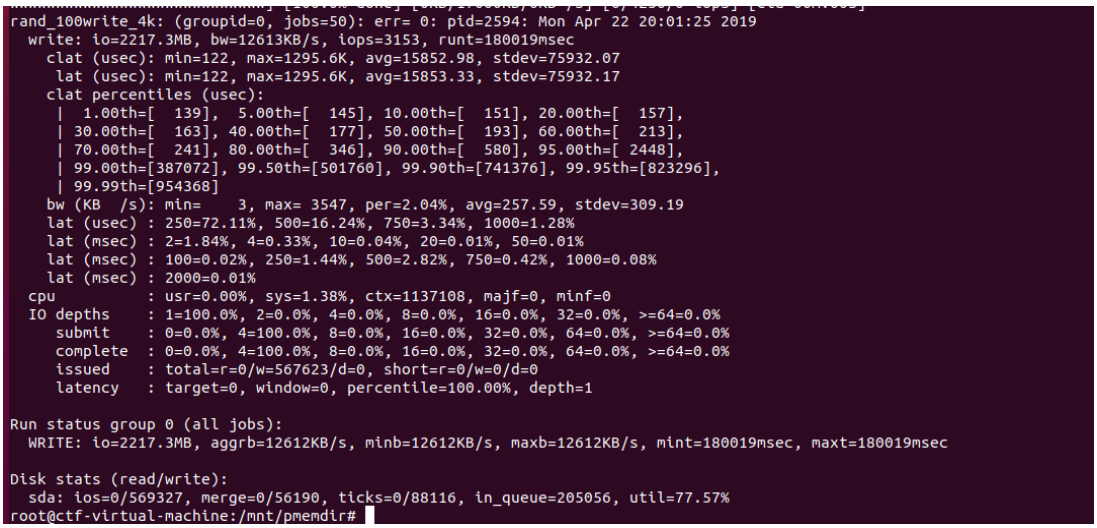


文件夹/mnt/pmemdir已挂载，接下来可以进行fio压力测试。

3. 用fio等工具测试模拟NVM的性能并与磁盘对比

- 安装fio
  - 官网下载解压
  - `./configure` 安装fio依赖
  - `make` 编译
  - `make install` 安装
- 测试
  - 随机写压力测试
    - `fio -filename=/mnt/pmemdir/a -direct=1 -iodepth 1 -thread -rw=randwrite -ioengine=psync -bs=4k -size=4G -numjobs=50 -runtime=180 -group_reporting -name=rand_100write_4k`

    向文件夹中写入a文件，大小4G，每次写入4k ，线程，创建50个jobs。

    - 结果
      - 

        - 分析:

          io=2217.3MB,bw=12613KB/s,iops= 3153(每秒写次数)，runt=180019msec。 可以看出模拟NVM的写过程比较慢。
  - 随机读压力测试
    - `fio -filename=/mnt/pmemdir/a -direct=1 -iodepth 1 -thread -rw=randread -ioengine=psync -bs=4k -size=4G -numjobs=50 -runtime=180 -group_reporting -name=rand_100read_4k`

    随机读取刚刚写入的文件a，每次读4k

    - 结果

-
```
root@ctf-virtual-machine:/mnt/pmemdir# fio -filename=/mnt/pmemdir/a -direct=1 -iodepth 1 -thread -rw=randread -ioengine=psync -bs=4k -size=4G
-numjobs=50 -runtime=180 -group_reporting -name=rand_100read_4k
rand_100read_4k: (g=0): rw=randread, bs=4K-4K/4K-4K/4K-4K, ioengine=psync, iodepth=1
...
fio-2.1.10
Starting 50 threads
Jobs: 50 (f=50): [rrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrr] [100.0% done] [120.1MB/0KB/0KB /s] [30.1K/0/0 iops] [eta 00m:00s]
rand_100read_4k: (groupid=0, jobs=50): err= 0: pid=2676: Mon Apr 22 20:11:39 2019
  read : io=23892MB, bw=135911KB/s, iops=33977, runt=180009msec
    clat (usec): min=1, max=396691, avg=1469.11, stdev=3009.08
     lat (usec): min=1, max=396692, avg=1469.28, stdev=3009.31
    clat percentiles (usec):
     |  1.00th=[    2],  5.00th=[    2], 10.00th=[    2], 20.00th=[    3],
     | 30.00th=[    4], 40.00th=[    5], 50.00th=[    7], 60.00th=[ 1544],
     | 70.00th=[ 2008], 80.00th=[ 2352], 90.00th=[ 3504], 95.00th=[ 4960],
     | 99.00th=[12096], 99.50th=[18816], 99.90th=[34048], 99.95th=[41216],
     | 99.99th=[74240]
    bw (KB  /s): min=  222, max= 6720, per=2.00%, avg=2722.70, stdev=1092.07
    lat (usec) : 2=0.23%, 4=26.35%, 10=24.49%, 20=0.40%, 50=0.89%
    lat (usec) : 100=0.35%, 250=0.12%, 500=0.26%, 750=0.26%, 1000=0.51%
    lat (msec) : 2=15.85%, 4=22.59%, 10=6.37%, 20=0.87%, 50=0.42%
    lat (msec) : 100=0.02%, 250=0.01%, 500=0.01%
  cpu          : usr=0.06%, sys=3.10%, ctx=2882878, majf=0, minf=48
  IO depths    : 1=100.0%, 2=0.0%, 4=0.0%, 8=0.0%, 16=0.0%, 32=0.0%, >=64=0.0%
     submit    : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
     complete  : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
     issued    : total=r=6116320/w=0/d=0, short=r=0/w=0/d=0
     latency   : target=0, window=0, percentile=100.00%, depth=1

Run status group 0 (all jobs):
   READ: io=23892MB, aggrb=135911KB/s, minb=135911KB/s, maxb=135911KB/s, mint=180009msec, maxt=180009msec

Disk stats (read/write):
  sda: ios=2880731/57, merge=0/77, ticks=1049916/740956, in_queue=4340784, util=100.00%
```
  - 分析： io=23892MB, bw=135911KB/s, iops=33977, runt=180009msec
    可以看出模拟NVM的读速度比较快(读的iops为33977，是写的将近10倍)，大大快于自身的写速度，可以看出模拟NVM的读写不平衡。
- 顺序写压力测试
  - `fio -filename=/mnt/pmemdir/b -direct=1 -iodepth 1 -thread -rw=write -ioengine=psync -bs=4k -size=4G -numjobs=50 -runtime=180 -group_reporting -name=sqe_100write_4k`

    顺序写文件b，其他配置与随机写相同
  - 结果
    -
```
njobs=50 -runtime=180 -group_reporting -name=sqe_100write_4k
sqe_100write_4k: (g=0): rw=write, bs=4K-4K/4K-4K/4K-4K, ioengine=psync, iodepth=1
...
fio-2.1.10
Starting 50 threads
sqe_100write_4k: Laying out IO file(s) (1 file(s) / 4096MB)
Jobs: 50 (f=50): [WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW] [100.0% done] [0KB/12083KB/0KB /s] [0/3020/0 iops] [eta 00m:00s]
sqe_100write_4k: (groupid=0, jobs=50): err= 0: pid=2756: Mon Apr 22 20:21:14 2019
  write: io=2722.9MB, bw=15487KB/s, iops=3871, runt=180031msec
    clat (usec): min=118, max=795835, avg=12910.95, stdev=75842.26
     lat (usec): min=118, max=795836, avg=12911.24, stdev=75842.30
    clat percentiles (usec):
     |  1.00th=[  131],  5.00th=[  137], 10.00th=[  137], 20.00th=[  139],
     | 30.00th=[  145], 40.00th=[  147], 50.00th=[  149], 60.00th=[  155],
     | 70.00th=[  171], 80.00th=[  203], 90.00th=[  322], 95.00th=[  572],
     | 99.00th=[489472], 99.50th=[536576], 99.90th=[626688], 99.95th=[667648],
     | 99.99th=[724992]
    bw (KB  /s): min=    5, max= 4910, per=2.02%, avg=313.60, stdev=355.05
    lat (usec) : 250=86.64%, 500=7.46%, 750=2.00%, 1000=0.68%
    lat (msec) : 2=0.23%, 4=0.05%, 10=0.01%, 20=0.01%, 50=0.01%
    lat (msec) : 100=0.01%, 250=0.15%, 500=1.88%, 750=0.87%, 1000=0.01%
  cpu          : usr=0.00%, sys=1.03%, ctx=1398347, majf=0, minf=0
  IO depths    : 1=100.0%, 2=0.0%, 4=0.0%, 8=0.0%, 16=0.0%, 32=0.0%, >=64=0.0%
     submit    : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
     complete  : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
     issued    : total=r=0/w=697043/d=0, short=r=0/w=0/d=0
     latency   : target=0, window=0, percentile=100.00%, depth=1

Run status group 0 (all jobs):
  WRITE: io=2722.9MB, aggrb=15487KB/s, minb=15487KB/s, maxb=15487KB/s, mint=180031msec, maxt=180031msec

Disk stats (read/write):
  sda: ios=4/697094, merge=0/239, ticks=88/44916, in_queue=145660, util=79.73%
```
    - 分析

      io=2722.9MB, bw=15487KB/s, iops=3871, runt=180031msec

      与随机写相似
- 顺序读压力测试
  - `fio -filename=/mnt/pmemdir/b -direct=1 -iodepth 1 -thread -rw=read -ioengine=psync -bs=4k -size=4G -numjobs=50 -runtime=180 -group_reporting -name=sqe_100read_4k`
  - 结果
    -

```
root@ctf-virtual-machine:/mnt/pmemdir# fio -filename=/mnt/pmemdir/b -direct=1 -iodepth 1 -thread -rw=read -ioengine=psync -bs=4k -size=4G -num
jobs=50 -runtime=180 -group_reporting -name=sqe_100read_4k
sqe_100read_4k: (g=0): rw=read, bs=4K-4K/4K-4K/4K-4K, ioengine=psync, iodepth=1
...
fio-2.1.10
Starting 50 threads
Jobs: 11 (f=11): [_______________R___R_____R__R_R__R____R_R_RR_R] [92.3% done] [3073MB/0KB/0KB /s] [787K/0/0 iops] [eta 00m:14s]
sqe_100read_4k: (groupid=0, jobs=50): err= 0: pid=2854: Mon Apr 22 20:30:19 2019
  read : io=204800MB, bw=1212.5MB/s, iops=310372, runt=168922msec
    clat (usec): min=0, max=1339.7K, avg=128.82, stdev=4826.57
     lat (usec): min=1, max=1339.7K, avg=129.29, stdev=4828.84
    clat percentiles (usec):
     |  1.00th=[    1],  5.00th=[    1], 10.00th=[    1], 20.00th=[    1],
     | 30.00th=[    1], 40.00th=[    1], 50.00th=[    2], 60.00th=[    2],
     | 70.00th=[    2], 80.00th=[    2], 90.00th=[    2], 95.00th=[    2],
     | 99.00th=[  209], 99.50th=[  386], 99.90th=[16768], 99.95th=[75264],
     | 99.99th=[259072]
    bw (KB  /s): min=    3, max=2372552, per=2.62%, avg=32582.97, stdev=135176.16
    lat (usec) : 2=47.62%, 4=49.64%, 10=0.57%, 20=0.12%, 50=0.26%
    lat (usec) : 100=0.12%, 250=0.84%, 500=0.46%, 750=0.12%, 1000=0.04%
    lat (msec) : 2=0.05%, 4=0.03%, 10=0.02%, 20=0.01%, 50=0.02%
    lat (msec) : 100=0.03%, 250=0.03%, 500=0.01%, 750=0.01%, 1000=0.01%
    lat (msec) : 2000=0.01%
  cpu          : usr=0.60%, sys=2.32%, ctx=899987, majf=0, minf=49
  IO depths    : 1=100.0%, 2=0.0%, 4=0.0%, 8=0.0%, 16=0.0%, 32=0.0%, >=64=0.0%
     submit    : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
     complete  : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
     issued    : total=r=52428800/w=0/d=0, short=r=0/w=0/d=0
     latency   : target=0, window=0, percentile=100.00%, depth=1

Run status group 0 (all jobs):
   READ: io=204800MB, aggrb=1212.5MB/s, minb=1212.5MB/s, maxb=1212.5MB/s, mint=168922msec, maxt=168922msec

Disk stats (read/write):
  sda: ios=784821/22, merge=5209/36, ticks=5357624/92960, in_queue=5747200, util=80.66%
```

- 分析 io=204800MB, bw=1212.5MB/s, iops=310372, runt=168922msec

  顺序读的速度也大于顺序写的速度

- 随机读写测试
  - `fio -filename=/mnt/pmendir/a -direct=1 -iodepth 1 -thread -rw=randrw -rwmixread=70 -ioengine=psync -bs=4k -size=1G -numjobs=10 -runtime=300 -group_reporting -name=iotest -iosscheduler=deadline`

    对文件a进行70%的读和30%的写
  - 结果

```
Starting 10 threads
Jobs: 10 (f=10): [mmmmmmmmm] [100.0% done] [10781KB/5474KB/0KB /s] [2695/1368/0 iops] [eta 00m:00s]
iotest: (groupid=0, jobs=10): err= 0: pid=2951: Mon Apr 22 20:45:42 2019
  read : io=3489.1MB, bw=11912KB/s, iops=2978, runt=300004msec
    clat (usec): min=1, max=410166, avg=2482.09, stdev=5223.76
     lat (usec): min=1, max=410166, avg=2482.29, stdev=5223.77
    clat percentiles (usec):
     |  1.00th=[    4],  5.00th=[  430], 10.00th=[  724], 20.00th=[ 1176],
     | 30.00th=[ 1528], 40.00th=[ 1832], 50.00th=[ 2160], 60.00th=[ 2448],
     | 70.00th=[ 2832], 80.00th=[ 3344], 90.00th=[ 4128], 95.00th=[ 4896],
     | 99.00th=[ 6944], 99.50th=[ 8256], 99.90th=[43776], 99.95th=[109056],
     | 99.99th=[197632]
    bw (KB  /s): min=   80, max= 1840, per=10.04%, avg=1196.52, stdev=246.42
  write: io=1496.5MB, bw=5107.8KB/s, iops=1276, runt=300004msec
    clat (usec): min=124, max=410452, avg=2033.38, stdev=4741.34
     lat (usec): min=124, max=410452, avg=2033.66, stdev=4741.35
    clat percentiles (usec):
     |  1.00th=[  137],  5.00th=[  149], 10.00th=[  159], 20.00th=[  211],
     | 30.00th=[  708], 40.00th=[ 1400], 50.00th=[ 1816], 60.00th=[ 2224],
     | 70.00th=[ 2608], 80.00th=[ 3088], 90.00th=[ 3952], 95.00th=[ 4704],
     | 99.00th=[ 6688], 99.50th=[ 7968], 99.90th=[17280], 99.95th=[98816],
     | 99.99th=[197632]
    bw (KB  /s): min=   15, max=  822, per=10.04%, avg=512.86, stdev=113.40
    lat (usec) : 2=0.01%, 4=0.38%, 10=1.30%, 20=0.10%, 50=0.03%
    lat (usec) : 100=0.01%, 250=7.34%, 500=3.86%, 750=3.40%, 1000=4.57%
    lat (msec) : 2=26.94%, 4=41.26%, 10=10.54%, 20=0.16%, 50=0.02%
    lat (msec) : 100=0.04%, 250=0.05%, 500=0.01%
  cpu          : usr=0.03%, sys=4.60%, ctx=2213965, majf=0, minf=0
  IO depths    : 1=100.0%, 2=0.0%, 4=0.0%, 8=0.0%, 16=0.0%, 32=0.0%, >=64=0.0%
     submit    : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
     complete  : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
     issued    : total=r=893416/w=383086/d=0, short=r=0/w=0/d=0
     latency   : target=0, window=0, percentile=100.00%, depth=1

Run status group 0 (all jobs):
   READ: io=3489.1MB, aggrb=11912KB/s, minb=11912KB/s, maxb=11912KB/s, mint=300004msec, maxt=300004msec
  WRITE: io=1496.5MB, aggrb=5107KB/s, minb=5107KB/s, maxb=5107KB/s, mint=300004msec, maxt=300004msec

Disk stats (read/write):
  sda: ios=621683/385281, merge=0/38126, ticks=44032/10872, in_queue=607156, util=86.86%
root@ctf-virtual-machine:/mnt/pmemdir#
```

  - 分析

    read : io=3489.1MB, bw=11912KB/s, iops=2978, runt=300004msec

    write: io=1496.5MB, bw=5107.8KB/s, iops=1276, runt=300004msec

    READ: io=3489.1MB, aggrb=11912KB/s, minb=11912KB/s, maxb=11912KB/s, mint=300004msec, maxt=300004msec WRITE: io=1496.5MB, aggrb=5107KB/s, minb=5107KB/s, maxb=5107KB/s, mint=300004msec, maxt=300004msec

- 顺序读写测试
  - ``
  - 结果
    - ```
      Starting 10 threads
      Jobs: 10 (f=10): [MMMMMMMMMM] [100.0% done] [14337KB/5782KB/0KB /s] [3584/1445/0 iops] [eta 00m:00s]
      iotest: (groupid=0, jobs=10): err= 0: pid=2972: Mon Apr 22 20:52:45 2019
        read : io=3812.2MB, bw=13012KB/s, iops=3252, runt=300002msec
          clat (usec): min=1, max=398290, avg=2304.48, stdev=3158.91
           lat (usec): min=1, max=398291, avg=2304.69, stdev=3158.93
          clat percentiles (usec):
           |  1.00th=[    6],  5.00th=[  434], 10.00th=[  740], 20.00th=[ 1176],
           | 30.00th=[ 1496], 40.00th=[ 1784], 50.00th=[ 2096], 60.00th=[ 2352],
           | 70.00th=[ 2672], 80.00th=[ 3152], 90.00th=[ 3952], 95.00th=[ 4640],
           | 99.00th=[ 6368], 99.50th=[ 7392], 99.90th=[12608], 99.95th=[38144],
           | 99.99th=[175104]
          bw (KB  /s): min=  386, max= 1760, per=10.02%, avg=1303.53, stdev=178.75
        write: io=1634.5MB, bw=5578.8KB/s, iops=1394, runt=300002msec
          clat (usec): min=122, max=305925, avg=1787.99, stdev=2715.33
           lat (usec): min=122, max=305926, avg=1788.31, stdev=2715.36
          clat percentiles (usec):
           |  1.00th=[  135],  5.00th=[  143], 10.00th=[  153], 20.00th=[  189],
           | 30.00th=[  498], 40.00th=[ 1224], 50.00th=[ 1640], 60.00th=[ 2008],
           | 70.00th=[ 2416], 80.00th=[ 2864], 90.00th=[ 3664], 95.00th=[ 4320],
           | 99.00th=[ 6048], 99.50th=[ 6944], 99.90th=[11200], 99.95th=[17280],
           | 99.99th=[168960]
          bw (KB  /s): min=   84, max=  864, per=10.02%, avg=558.64, stdev=87.52
          lat (usec) : 2=0.05%, 4=0.20%, 10=0.56%, 20=0.05%, 50=0.02%
          lat (usec) : 100=0.01%, 250=8.04%, 500=4.52%, 750=3.38%, 1000=5.18%
          lat (msec) : 2=28.53%, 4=40.76%, 10=8.54%, 20=0.10%, 50=0.01%
          lat (msec) : 100=0.03%, 250=0.02%, 500=0.01%
        cpu          : usr=0.04%, sys=5.25%, ctx=2572803, majf=0, minf=0
        IO depths    : 1=100.0%, 2=0.0%, 4=0.0%, 8=0.0%, 16=0.0%, 32=0.0%, >=64=0.0%
           submit    : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
           complete  : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
           issued    : total=r=975875/w=418409/d=0, short=r=0/w=0/d=0
           latency   : target=0, window=0, percentile=100.00%, depth=1

      Run status group 0 (all jobs):
         READ: io=3812.2MB, aggrb=13011KB/s, minb=13011KB/s, maxb=13011KB/s, mint=300002msec, maxt=300002msec
        WRITE: io=1634.5MB, aggrb=5578KB/s, minb=5578KB/s, maxb=5578KB/s, mint=300002msec, maxt=300002msec

      Disk stats (read/write):
        sda: ios=839692/418819, merge=0/719, ticks=20896/2052, in_queue=643692, util=86.63%
      ```
    - 分析

      read : io=3812.2MB, bw=13012KB/s, iops=3252, runt=300002msec

      write: io=1634.5MB, bw=5578.8KB/s, iops=1394, runt=300002msec

      READ: io=3812.2MB, aggrb=13011KB/s, minb=13011KB/s, maxb=13011KB/s, mint=300002msec, maxt=300002msec WRITE: io=1634.5MB, aggrb=5578KB/s, minb=5578KB/s, maxb=5578KB/s, mint=300002msec, maxt=300002msec
- 对比顺序读写和随机读写，可以看出的顺序读写比随机读写性能更好。
- 磁盘的随机读写 对磁盘中的dev/sda1文件进行随机读取过程
  - `fio -filename=/dev/sda1 -direct=1 -iodepth 1 -thread -rw=randread -ioengine=psync -bs=4k -size=10G -numjobs=10 -runtime=300 -group_reporting -name=iotest`
  - 结果
      ```

```
root@tongfei-virtual-machine:/home/tongfei/Desktop/fio-2.2.5# fio -filename=/dev/sda1 -direct=1 -iodepth 1 -t
hread -rw=randread -ioengine=psync -bs=4k -size=10G -numjobs=10 -runtime=300 -group_reporting -name=iotest
iotest: (g=0): rw=randread, bs=4K-4K/4K-4K/4K-4K, ioengine=psync, iodepth=1
...
fio-2.2.5
Starting 10 threads
Jobs: 10 (f=10): [r(10)] [100.0% done] [43000KB/0KB/0KB /s] [10.8K/0/0 iops] [eta 00m:00s]
iotest: (groupid=0, jobs=10): err= 0: pid=5308: Sun Apr 21 13:03:00 2019
  read : io=11620MB, bw=39661KB/s, iops=9915, runt=300001msec
    clat (usec): min=45, max=266972, avg=1005.13, stdev=739.36
     lat (usec): min=45, max=266973, avg=1005.40, stdev=739.46
    clat percentiles (usec):
     |  1.00th=[   67],  5.00th=[  402], 10.00th=[  494], 20.00th=[  628],
     | 30.00th=[  764], 40.00th=[  860], 50.00th=[  932], 60.00th=[ 1004],
     | 70.00th=[ 1096], 80.00th=[ 1240], 90.00th=[ 1496], 95.00th=[ 1880],
     | 99.00th=[ 3344], 99.50th=[ 3824], 99.90th=[ 5280], 99.95th=[ 6624],
     | 99.99th=[19584]
    bw (KB  /s): min= 1064, max= 4680, per=10.01%, avg=3969.62, stdev=751.08
    lat (usec) : 50=0.01%, 100=1.84%, 250=0.26%, 500=8.19%, 750=18.87%
    lat (usec) : 1000=29.96%
    lat (msec) : 2=36.70%, 4=3.80%, 10=0.33%, 20=0.01%, 50=0.01%
    lat (msec) : 100=0.01%, 250=0.01%, 500=0.01%
  cpu          : usr=0.13%, sys=9.10%, ctx=2924002, majf=0, minf=10
  IO depths    : 1=100.0%, 2=0.0%, 4=0.0%, 8=0.0%, 16=0.0%, 32=0.0%, >=64=0.0%
     submit    : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
     complete  : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
     issued    : total=r=2974621/w=0/d=0, short=r=0/w=0/d=0, drop=r=0/w=0/d=0
     latency   : target=0, window=0, percentile=100.00%, depth=1

Run status group 0 (all jobs):
   READ: io=11620MB, aggrb=39661KB/s, minb=39661KB/s, maxb=39661KB/s, mint=300001msec, maxt=300001msec

Disk stats (read/write):
  sda: ios=2974706/158, merge=140/13, ticks=1119956/22884, in_queue=1142508, util=99.45%
root@tongfei-virtual-machine:/home/tongfei/Desktop/fio-2.2.5#
```

- 通过对比iops(这里是每秒读的次数)，读磁盘的iops为9915，而模拟NVM的为33977，可以看出模拟NVM的访问速度大大快过于对磁盘的访问速度。通过带宽bw等等也可以看出模拟NVM的读的速度比 磁盘块很多。

结论 ：
模拟NVM的读写速度不对称，随机读写和顺序读写也有一定区别
与磁盘对比，性能比较好，但由于再这里是使用DRMA模拟的NVM所以这些区别可能由于是内存模拟而导致的。

使用PMDK的libpmem库编写样例程序操作模拟NVM（关键实验结果截图，附上编译命令和简单样例程序）。

（样例程序使用教程的即可，主要工作是编译安装并链接PMDK库）

1. 编译安装并链接PMDK库 (1)安装好各种依赖包。 apt-get install autoconf asciidoc xmlto automake libtool systemd pkg-config apt-get install libkmod-dev libudev-dev bash-completion-dev (2)安装ndctl git clone https://github.com/pmem/ndctl.git cd ndctl ./autogen.sh ./configure 如下图，make check指令输入后显示没有error，输入make install

```
SKIP: pfn-meta-errors.sh
SKIP: security.sh
============================================================================
Testsuite summary for ndctl 64.1
============================================================================
# TOTAL: 26
# PASS:  0
# SKIP:  26
# XFAIL: 0
# FAIL:  0
# XPASS: 0
# ERROR: 0
============================================================================
root@chenzhx-virtual-machine:/home/chenzhx/ndctl# make install
```


（3）安装pmdk

git clone https://github.com/pmem/pmdk.git cd pmdk make make install

```
chenzhx@chenzhx-virtual-machine:~$ cd pmdk
chenzhx@chenzhx-virtual-machine:~/pmdk$ make
make -C src all
make[1]: Entering directory '/home/chenzhx/pmdk/src'
make -C libpmem
make[2]: Entering directory '/home/chenzhx/pmdk/src/libpmem'
make[2]: Nothing to be done for 'all'.
make[2]: Leaving directory '/home/chenzhx/pmdk/src/libpmem'
make -C libpmem DEBUG=1
make[2]: Entering directory '/home/chenzhx/pmdk/src/libpmem'
make[2]: Nothing to be done for 'all'.
make[2]: Leaving directory '/home/chenzhx/pmdk/src/libpmem'
make -C jemalloc -f Makefile.libvmem     jemalloc_EXTRA_CFLAGS=" -I/home/chenzhx
```


2. 编译运行样例程序

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <errno.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <libpmem.h>

/* using 4k of pmem for this example */
#define PMEM_LEN 4096

#define PATH "/pmem-fs/myfile"

int main(int argc, char *argv[])
{
    char *pmemaddr;
    size_t mapped_len;
    int is_pmem;

    /* create a pmem file and memory map it */

    if ((pmemaddr = pmem_map_file(PATH, PMEM_LEN, PMEM_FILE_CREATE,
```

```
            0666, &mapped_len, &is_pmem)) == NULL) {
        perror("pmem_map_file");
        exit(1);
    }

    /* store a string to the persistent memory */
    strcpy(pmemaddr, "hello, persistent memory");

    /* flush above strcpy to persistence */
    if (is_pmem)
        pmem_persist(pmemaddr, mapped_len);
    else
        pmem_msync(pmemaddr, mapped_len);

    /*
     * Delete the mappings. The region is also
     * automatically unmapped when the process is
     * terminated.
     */
    pmem_unmap(pmemaddr, mapped_len);
}
```

```
chenzhx@chenzhx-virtual-machine:~$ cd Desktop
chenzhx@chenzhx-virtual-machine:~/Desktop$ gcc main.cpp
```

论文阅读

总结一下本文的主要贡献和观点(500字以内)(不能翻译摘要)。

（回答本文工作的动机背景是什么，做了什么，有什么技术原理，解决了什么问题，其意义是什么）

各种SCM的迅速出现，它们具有非易失性的额外特性而SSD和DRAM并无这种特性，所以人们希望能够发展内存和存储合并这种数据结构。虽然说SCM与DRAM仍有相似特性，但是SCM延迟较慢且不对称，写操作明显慢于读操作。SCM需要索引，这种差异就导致了DRAM的传统B树无法满足SCM的要求。所以本文主要提出了fptree这种新的树。fptree是新颖的、持久的B-树，它利用了SCM的功能，同时表现出与传统瞬态B-树相似的性能。也有曾提出CDDS B-树[24]、WBtree[8]和NV-树[28]，但它们无法匹配基于DRAM优化的B-树的速度。

接下来提出了fptree四种设计原则，1. Selective Persistence， 2. Fingerprinting，3. Selective Concurrency，4. Amortized persistent memory allocations 。本文又讲述了fptree的用法，和实现这些函数的算法。主要有find，insert，delete，recorevy，update等几个基本操作。又进行了实验，与其他树进行对比。在实验中，这种树被应用于数据库。用被评估的树替换字典编码的字典索引，数据库的列存储引擎。字典索引只需要树的固定大小的键版本。然后进行实验测量持久树对数据库事务性能的影响。然后通过各个方面与NVTree和WbTree的对比，可以看出FPtre的开销更少，性能更好。

本文主要提出了fptree这种可以满足SCM需要的新树，同时也给出了如何实现这种树的算法。这对SCM等非易失性内存的发展提供了帮助，促进这个方面的发展。

SCM硬件有什么特性？与普通磁盘有什么区别？普通数据库以页的粒度读写磁盘的方式适合操作SCM吗？

可按位寻址，非易失，低延迟，读写速度快，写入不对称，写入慢于读取。 scm与磁盘存储容量相当，有比磁盘更高的存取速度。 适合。

操作SCM为什么要调用CLFLUSH等指令?

(写入后不调用，发生系统崩溃有什么后果)
因为要访问SCM要经过很长的过程，包括存储缓冲区，CPU缓存，内存控制器缓冲区，所有的软件都很少起到控制的作用。SCMaware文件系统使用mmap函数使得应用层能够直接访问SCM，因此SCM写的有序性和持久性不能得到保证，为了解决这个问题，要调用CLFLUSH等指令。如果写入后不调用，发生系统崩溃会导致缓存行的内容还没有写入到内存里。

FPTree的指纹技术有什么重要作用?

SCM中的无序叶进行线性扫描需要很大代价。指纹技术是无序的叶键的单字节散列，连续存储在叶的开头。在搜索的过程中先扫描，指纹技术起到过滤的作用，避免搜索那些有指纹但是跟搜索键不匹配的键，使用指纹技术，成功搜索到叶内键的期望次数等于1.这就大大提高了FPTree的性能。

为了保证指纹技术的数学证明成立，哈希函数应如何选取?

（哈希函数生成的哈希值具有什么特征，能简单对键值取模生成吗？）
指纹是叶键的一个字节散列，在叶的开头连续存储。通过在搜索过程中首先 对它们进行扫描，指纹可以起到过滤作用，以避免探测指纹与搜索密钥不匹配的 密钥。只考虑唯一键的情况，证明，使用指纹技术，成功搜索过程中预期的叶内 键探针数量等于 1。在下面计算这个期望值。假设一个哈希函数生成均匀分布的 指纹。设 m 为叶中的条目数， n 为可能的哈希值（对于一个字节的指纹，n=256）。 计算叶中指纹的预期出现次数，表示为 E[k]，相当于指纹数组中哈希冲突的次 数加上一个（因为我们假设搜索键存在）：

$$E[K] = \sum_{i=1}^{m} i \cdot P[K = i]$$

其中 p[k=i]是指搜索指纹至少出现一次的概率。 用二项分布计算，至少存在一个匹配指纹的概率，表示为不存在匹配指纹的互补 概率。 知道指纹点击的预期数量后，就可以确定叶内密钥探针的预期数量，表示为 EFPTree[T], 这是指纹点击指示的键上长度 e[k]的线性搜索中的预期键探针数 量。

$$\begin{aligned} E[T] &= \frac{1}{2}\left(1 + E[K]\right) = \frac{1}{2}\left(1 + \sum_{i=1}^{m} i \frac{\left(\frac{1}{n}\right)^{i}\left(1-\frac{1}{n}\right)^{m-i}\binom{m}{i}}{1-\left(1-\frac{1}{n}\right)^{m}}\right) \\ &= \frac{1}{2}\left(1 + \frac{\left(\frac{n-1}{n}\right)^{m}}{1-\left(\frac{n-1}{n}\right)^{m}} \sum_{i=1}^{m} \frac{i\binom{m}{i}}{(n-1)^{i}}\right) \\ &= \frac{1}{2}\left(1 + \frac{\left(\frac{n-1}{n}\right)^{m}}{1-\left(\frac{n-1}{n}\right)^{m}} \left(\frac{m}{n-1}\right) \sum_{i=0}^{m-1} \frac{\binom{m-1}{i}}{(n-1)^{i}}\right) \end{aligned}$$

通过对和应用二项式定理，我们得到：计算得期望值为1。

$$E[T] = \frac{1}{2}\left(1 + \frac{\left(\frac{n-1}{n}\right)^m}{1-\left(\frac{n-1}{n}\right)^m}\left(\frac{m}{n-1}\right)\left(\frac{n}{n-1}\right)^{m-1}\right)$$
$$= \frac{1}{2}\left(1 + \frac{m}{n\left(1-\left(\frac{n-1}{n}\right)^m\right)}\right)$$

持久化指针的作用是什么？与课上学到的什么类似？

当程序重新启动时，它会使用一个新的地址空间来执行此操作，该地址空间会使所有存储的虚拟指针无效。 持久分配器提供持久指针和可变指针之间的双向转换。由于持久指针在失败时保持有效，因此它们用于在重新启动时刷新易失性指针。类似于LSN。