本文是内存热度感知项目的测试说明，其中内容包括各项测试的测试流程、测试结果和结果的分析，为了方便说明，下文中以RUNMAP（Reverse-Unmap）或者ours来指代我们的方案。

# 一、基础环境说明

表 1 基础环境简要说明

|  |  |  |
| --- | --- | --- |
| **名称** | **说明** | **备注** |
| 内核版本 | Linux 5.15.114 |  |
| 服务器型号 | Dell PowerEdge R750 |  |
| socket | 2个 |  |
| CPU型号 | Intel(R) Xeon(R) Gold 6330 CPU @ 2.00GHz |  |
| DRAM | 每个socket配置16G\*6 DDR4 | 实际实验基于memmap进行了容量限制 |
| PM | 每个socket配置256G PM | 实际实验基于ndctl进行了容量限制 |

如上表所示，我们基于Linux 5.15.114版本内核实现了RUNMAP。测试的服务器配置为Intel(R) Xeon(R) Gold 6330 CPU @ 2.00GHz，2个socket，每个socket上均配置了96G DRAM（16G\*6 DDR4）以及256G的Intel persistent memory。因此，在ss逻辑上，整个系统中共有4个内存节点：socket0上的DRAM与PM，socket1上的DRAM与PM。

为了便于测试，实验中根据需要限制了DRAM和PM的容量。其中DRAM容量是通过内核的memmap参数进行限制，PM容量是基于ndctl进行限制。

# 二、测试流程说明

## 2.1 应用负载

本文选择了四个工作负载作为测试，分别为graph500、SPEC-ACCEL、redis数据库以及GUPS。其中，GUPS基于HeMem论文发布的版本（<https://github.com/cuhk-mass/hemem/tree/main/microbenchmarks>）。各个工作负载运行时的基本情况如下表所示，表中列举了各个工作负载时运行的参数以及内存配置。其中，内存配置一列分别标明了4个内存节点的容量配置，例如“24G (4G:4G:8G:8G)”表示：系统总内存大小为24G，两个DRAM容量均为4G，两个PM容量均为8G。值得一提地是，实验中的内存容量配置与工作负载的工作集大小非常接近，以模拟内存紧张的场景。

表 2 应用负载简要说明

|  |  |  |
| --- | --- | --- |
| **工作负载** | **参数配置** | **内存配置（DRAM0:DRAM1:PM0:PM1）** |
| Graph500 | 参数“25 25” | 24G (4G:4G:8G:8G) |
| SPEC-ACCEL | 404.lbm | 12G (4G:4G:4G:4G) |
| Redis | 10个客户端, 每个客户端10个线程，每个读写1亿次 | 24G (4G:4G:8G:8G) |
| GUPS | 16个线程读写16G数据共10亿次，其中4G为热数据 | 24G (4G:4G:8G:8G) |

具体地，

1. 对于Graph500，实验中采用“25 25”的参数，即边数和角数均为2^25，在此参数下，应用程序的工作集大约22.8G。

2. 对于SPEC-ACCLE，实验中使用404.lbm进行测试，由于其工作集大约10G，因此我们限制单个PM容量为4G（实验中发现基于memmap设置的DRAM容量无法小于4G，否则会出错）。

3. 对于Redis，实验中先是使用redis-benchmark进行大量写操作，构造了一个大约19.4G的数据库（为了避免干扰，关闭了rdb功能）；然后再使用redis-benchmark进行大量读操作，来测试数据库性能。具体地，写操作时，设置了50个客户端，每个客户端开50个线程，每个线程操作16\*1024\*1024次，每个请求的数据块大小为1KB；读操作时，设置了10个客户端，每个客户端开10个线程，每个线程操作8\*16\*1024\*1024次，每个请求的数据块大小为1KB。

4. 对于GUPS，实验中分配了16G的数据，同时设定分散的4G热区域。然后让16个线程读写这16G数据共10亿次。但是为了构造热数据，修改了各部分数据的读写概率，使得90%的请求去读写4G热数据，另外10%请求随机读写。

## 2.2 对比实验说明

为了验证RUNMAP方案的有效性，实验中选取了tiering-0.8以及memtis作为对比实验。其中：

1. tiering-0.8是由intel针对tieried memory开发的内存动态布局方案，基于Linux 5.3版本，方案基于AutoNUMA（也被称为numa balancing）技术。需要注意的是，在测试误判率的实验中，我们对tiering代码进行了一些统计代码的添加；

2. memtis是SOSP’23中的一篇文章提出的动态阈值优化方案，基于Linux 5.15.19，方案基于PEBS技术。需要注意的是，（1）由于memtis在提出时，只考虑了单socket场景，而我们的实验环境是双socket场景，为了不损失memtis的性能，我们仅为memtis配置了一个socket，同时使得“该socket上的dram size”等于“双socket上的local dram size + remote dram size”，因此理论上memtis的配置更好一些（因为DRAM容量相同，但是没有跨socket的访问）。（2）memtis原本是通过cgroup来限制DRAM容量的，但是为了统一同时保证环境尽量一致，我们使用memmap来限制memtis当然DRAM容量。相应地我们修改了memtis的cgroup配置脚本，取消了此处的内存限制。（3）在memtis环境下运行工作负载时，要通过一个launch程序间接运行。launch程序的作用是，在工作负载运行开始后，调用一个系统调用，向内核传递工作负载的pid信息，使得memtis内核可以追踪目标进程。

3. RUNMAP是本文提出的优化方案。需要注意的是，（1）由于RUNMAP需要扫描lru链表，而5.15内核的lru链表默认与cgroup绑定，因此为了只扫描目标进程的lru链表，需要为目标进程单独配置一个cgroup。当然，如果一个cgroup中有多个进程，也是可以的，只是这种情况下，RUNMAP会扫描多个进程的页面。（2）与memtis类似，RUNMAP也需要一个launch程序间接运行工作负载。

## 2.3 测试步骤说明

1. 测试步骤简要说明如下：

（1）修改内核配置，编译与安装内核；

（2）切换内核，可以使用kexec快速切换，或者在grub界面手动切换。切换时，需要设置好memmap参数来限制socket0和sockt1的dram大小。

（3）使用ndctl配置PM。配置PM时，需要使用“--size”来限制PM容量。

（4）[可选] 对于memtis与RUNMAP，需要配置cgroup。

（5）[可选] 对于memtis与RUNMAP，需要通过launch程序运行工作负载。

（6）[可选] 根据需要修改控制参数，例如sysctl变量。

（7）运行工作负载。

（8）[可选] 对于redis程序，除了需要启动redis-server外，还要启动客户端进程。

（9）程序运行完毕，保存运行日志与统计信息。

2. 为了方便测试，可以使用提写好的脚本进行测试。

如下面运行graph500的脚本：

|  |
| --- |
| // 设置运行参数  ………….  sudo sysctl page\_hotness.page\_unmap\_cold\_threshold\_sec=20  sudo sysctl page\_hotness.page\_recently\_accessed\_threshold\_sec=1  // 设置输出的脚本目录  script\_path="my\_scripts/temp.sh"  // 设置日志输出路径（默认为./out）  out\_dir\_prefix=""  run\_gen\_graph500()  {  num=$1  script\_absolute\_path=$(realpath "$script\_path")  caller\_script\_absolute\_path=$(realpath $0)  echo -e "\n=== generating script...."  cd /home/dell/lmy/my\_scripts/gen\_cmd  python3 gen\_cmd.py\  --add\_out\_dir\_time\_suffix\  --out\_script\_path="$script\_absolute\_path"\  --caller\_script\_path="$caller\_script\_absolute\_path"\  --out\_dir\_prefix="$out\_dir\_prefix"\  --benchmark\_args "$num $num"\  --method\_type="pebs"\  --log\_my\_stat\  --log\_numa\_maps\  --log\_vmstat\  --log\_sysctl\  --log\_dmesg\  --quiet  cd –  ……………..  }  run\_graph500()  {  name=$1  num=$2  out\_dir\_prefix=$name  script\_path="my\_scripts/temp.sh"  run\_gen\_graph500 $num  }  # check cgroup  if [[ ! -d "/sys/fs/cgroup/mygroup" ]]; then  sudo cgcreate -g cpu,memory:mygroup  sudo chown $USER:$USER -R /sys/fs/cgroup  fi  // 设置PM容量  bash /home/dell/lmy/my\_scripts/gen\_cmd/common\_scripts/init.sh "8G"  // 传入目录  run\_graph500 "test-graph25" 25 |

其中，gen\_cmd.py是一个python脚本，作用类似一个脚本模板，即读取并解析输入的参数，然后输出一个实际的shell脚本。

# 三、测试结果说明

## 3.1 测试指标

为了验证方案的有效性，实验中分别测试了应用的整体性能、扫描及预测开销以及页面误判率三个指标。其中：

1. 对于整体性能，即统计应用程序的整体运行时间；

2. 对于扫描开销，由于tiering-0.8是基于页表扫描pte（page table entry，页表项），而本实验的RUNMAP是基于struct page进行扫描，且二者的扫描规模并不相同。因此为了对比开销，测试中对开销进行了归一化：针对tiering-0.8，统计扫描pte的平均开销，即“扫描总时长/扫描到的pte总数”；针对RUNMAP，统计扫描page的平均开销，即“扫描总时长/扫描到的page总数”。此处，对于预测开销，由于tiering-0.8和RUNMAP均基于hint page fault以及时间戳比较进行预测，因此这部分开销相同。

3. 对于页面误判率，实验中采取在线分析的方式，由于GUPS的热区是提前设定好的，因此根据这个先验知识我们可以知道各个页面是否为热集。基于此，可以扫描内存节点中的各个地址区域，扫描出各个节点上热页的数量，计算“1-local DRAM中热页数目/总的热页数目”来得到热页成功迁移到DRAM的比例，以此作为页面误判率的参考指标。

## 3.2 整体性能结果

整体性能结果如下图所示，针对tiering-0.8、memtis以及RUNMAP（即图中的ours），共测试了4个工作负载。为了便于比较，将运行时间以tiering为标准进行了归一化处理。不过由于memtis代码有点问题，不能成功运行redis（OSDI’24的MATRYOSHKA论文中也有相关说明），因此跳过了memtis在redis上的结果。

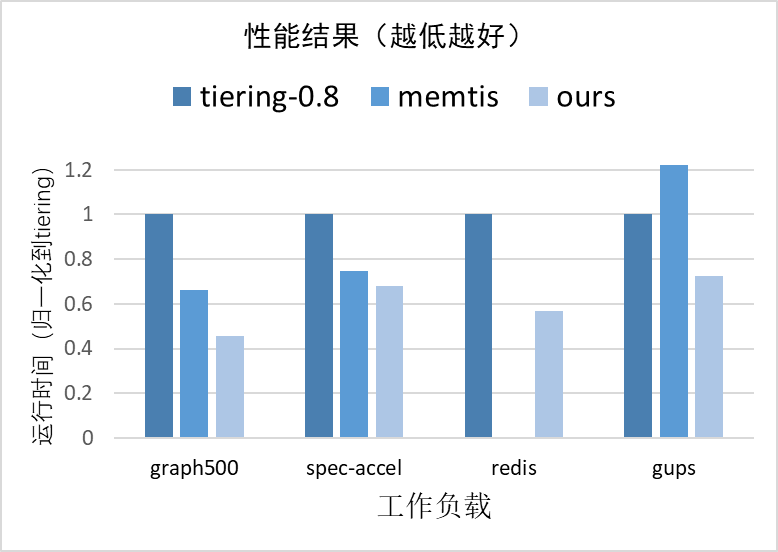


图 1 整体性能结果

实验结果表明：

（1）相比于tiering-0.8，RUNMAP在各个benchmark上均有性能提升，性能平均提升36.77%。其中：对于graph500，加速比达到54.50%，说明在内存压力紧张的场景下，RUNMAP能够更有效地进行页面的动态布局；对于graph500与spec-accel两个工作负载，平均提升

（2）memtis在graph500和SPEC-ACCEL上性能较好，但在gups场景下性能较差。根据实验日志，gups在GUPS程序开始时能够很快地捕捉到热页面并进行迁移，但是后续就无法进行更加灵活的布局调整，同时由于PEBS一直在采集事件，也造成了一些开销。

（3）而相比于memtis，RUNMAP在各个benchmark上均有性能提升，即使在gups场景下也能降低27.52%的运行时间。

## 3.3 扫描开销结果

扫描开销结果如下图所示，相比于tiering-0.8，RUNMAP增加了28.42%开销。这个主要是由于RUNMAP在扫描时，需要获取lru lock，同时需要遍历lru list，以及需要检查page的多个flag信息。但是该扫描过程为异步执行，因此对于应用程序本身来说，不会直接造成太大影响。为了提高热度信息的准确度，这个开销是比较值得的。

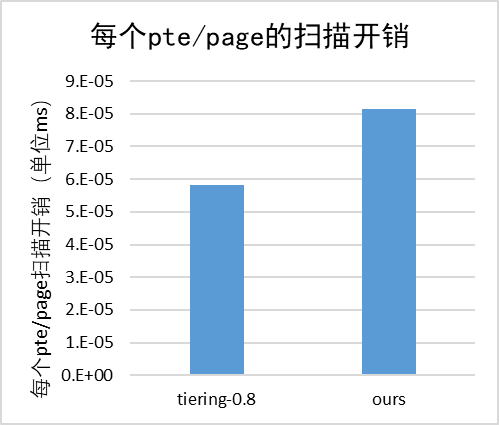


图 2 扫描开销结果对比

## 3.4 页面误判率结果

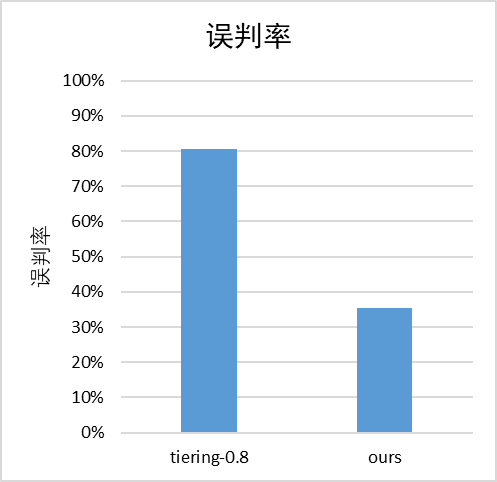


图 3 页面误判率结果

页面误判率结果如上图所示，相比于tiering-0.8，RUNMAP减少了45.04%的误判率。这主要得益于频率更高、范围更广的页面扫描，使得许多热页面可以及时被感知到。