数据库课程project实验报告

基于NVM的线性哈希索引

陈梓桂

程宇航 15352066

1. 实验内容

线性哈希

索引是数据库系统中重要的数据结构，数据库的数据存储依靠索引来进行组织，可以理解为一种存放协议。有了存放的规则，我们就可以依据存放的规则进行相应的查找，怎么放就怎么找。有了查找功能，我们也可以相应实现修改和删除功能。所以索引具有四大基本操作：\*\*增删改查\*\*，而操作的数据对象都是\*\*键值对\*\*的形式。

常见的索引有B+Tree和哈希索引，线性哈希属于一种哈希索引。线性哈希的核心是会将所有哈希表维护成一种连续的数组的模式，这样我们就可以方便地直接通过下标来访问相应的哈希桶。本次课程设计的线性哈希实现参考课本《数据库管理系统原理与设计》第三版的283页部分对线性哈希的描述来实现其增删改查操作。

非易失性内存NVM

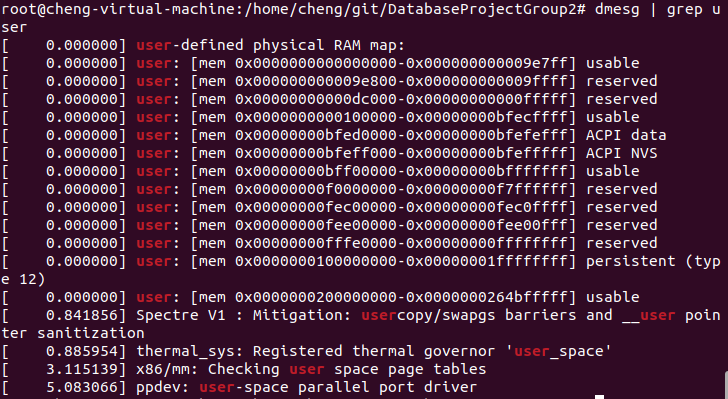
Non-Volatile Memory(NVM)也即非易失性内存，是一种新型存储硬件。其不仅具有传统内存的字节寻址特性，同时也具有磁盘的数据持久化的特性，将数据存放在NVM上面可以实现持久化。这次课程设计中的线性哈希的键值数据等就是要存放到NVM上，以达到持久化和可恢复的目的。要实现持久化，需要进行相应的数据存储的设计，即解决如何存放持久的数据然后如何读取这些数据的问题，便于程序依靠这些数据恢复。

NVM编程要点

由于NVM属于一种特殊的存储硬件，所以编程上需要按照特定的模式进行以对NVM的持久数据按字节访问而不是IO流的形式。现在主流的NVM编程模型是依据SNIA的推荐的标准，使用mmap内存映射的方式进行NVM文件数据的访问，mmap相关背景请自行百度了解。具体来说，就是在NVM的设备上安装一个NVM-aware的文件系统如EXT4,然后开启DAX模式以供程序能够以直接访问模式访问文件数据，然后程序以mmap的形式打开NVM文件，以字节寻址的方式操作文件数据。

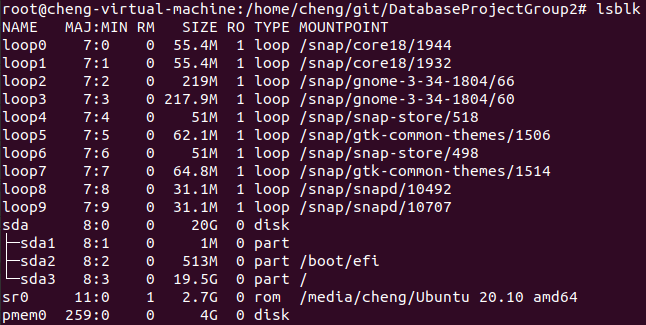
1. 实验步骤
2. 利用普通内存模拟NVM环境

使用memmap指令保留4gb内存，定义pmem区域谜语GRUB模板生成引导配置文件后，进行dmesg分析，结果如图所示：

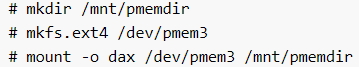


可见持久性存储区域突出显示为persisten（type 12）。

最后构建为持久性设备启用DAX的文件系统。通过lsblk指令我们发现，设备上已经创建好了名为pmem0的分区：



为其创建一个ext4类型的文件系统，然后使用dax选项将其挂载



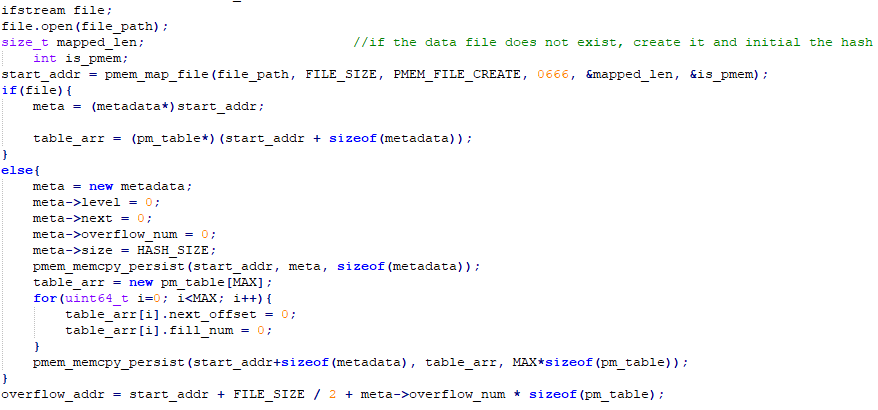
之后，我们便可以从/mnt/pmemdir文件夹下进行持久数据的开发。

1. 根据PMDK的README安装教程进行库安装

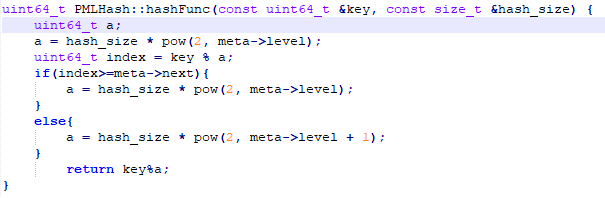
首先从github上clone pmdk的库，并进行库的安装，并配置相关环境。在完成库的安装后，编写测试代码测试本次实验需要用到的三个函数pmem\_map()、pmem\_persist()和pmem\_unmap()的功能。其中pmem\_map()将目标文件通过内存映射的方式打开并返回文件数据的虚拟地址，这样操作这个虚拟地址的数据即通过字节寻址的方式操作数据；pmem\_persist()函数调用CLFLUSH和SFENCE指令来显式持久化相应的数据，每次NVM数据的修改都应调用一次；pmem\_unmap(): 这个函数用于关闭pmem\_map打开的NVM文件。测试发现三个函数运行正常，且/mnt/pmemdir 文件夹中出现对应文件，即可确定PMDK库安装成功。

1. 根据项目框架和需求实现代码并运行

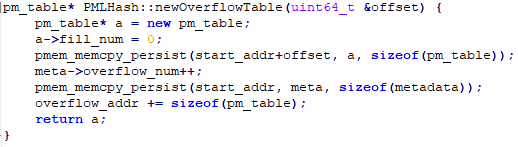
首先实现PMLHash()函数，该函数功能为初始化，输入为文件路径。当数据文件不存在时，创建文件，完成各项参数的初始化，并将其persist到数据文件中，若文件已存在，则从文件中读取项目所需的各项参数，如哈希桶的level、溢出桶个数、哈希桶分裂点等等。具体实现如下图所示：



之后实现哈希函数hashFunc()，输入为key与哈希桶的大小。这个函数的作用是通过输入的key返回保存这个kv对的哈希桶的编号，首先假设当前桶未分裂，计算当前kv对应该保存在哪个桶，然后将桶号与当前分裂点的桶号进行比较。若大于分裂点的桶号，说明对应的桶还未分裂，因此计算出的桶号即为保存这个kv对的桶；若小于分裂点的桶号，说明对应的桶已经分裂过，此时kv对可能保存在分裂后的旧桶或新桶中，应根据level值使用h(2)函数判断kv对具体保存在分裂后的旧桶还是新桶中。实现如下：

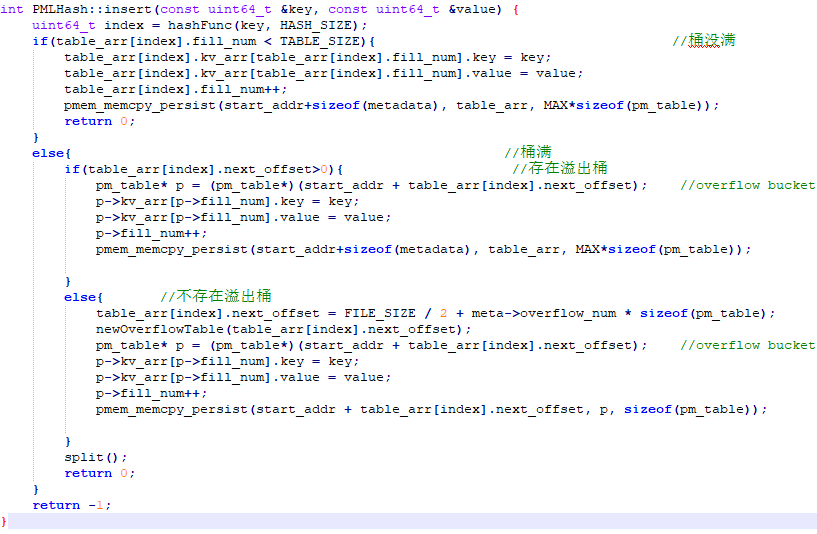


然后是创建溢出桶的函数newOverflowTable()。每当哈希桶发生溢出时，我们都要判断该哈希桶有无对应溢出桶，若不存在溢出桶则需要创建一个。为了创建溢出桶，我们需要获得当前溢出桶的地址，并根据偏移量为其分配空间，同时修改meta中保存的溢出桶的数量，和新建溢出桶的地址。实现如下图：



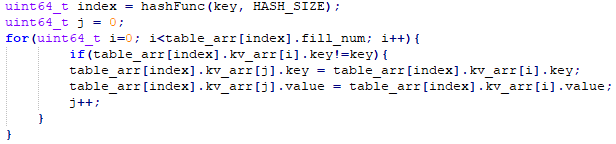
由于桶的分裂需要将旧桶的内容部分转移到分裂桶中，需要kv对的插入和删除功能，因此需实现这两个函数insert()和remove()。

insert()函数的输入为key和value，即要插入的kv对。首先通过key计算出需要插入的桶的桶号，之后判断桶是否已满。若桶未满，则直接将kv对复制到此桶的kv对结构体组的末尾并更新fill\_num参数即可；若当前桶已满，则需要将数据插入溢出桶。此时有两种可能，若溢出桶存在，需要根据哈希桶的next\_offset参数计算出哈希桶对应溢出桶的虚拟地址，然后对其进行插入，并更新溢出桶的fill\_num参数；若溢出桶不存在，则通过溢出桶的数量、桶结构体的大小、文件的起始地址计算出哈希桶的指向其溢出桶的地址，即next\_offset值，并更新，之后根据地址使用newOverflowTable()函数新建溢出桶，然后再进行kv对的插入，由于桶满时需触发分裂操作，因此还需调用split()函数进行桶的分裂。不管kv对被插入哪里，都需要调用persist函数来将数据持久化。实现如下：

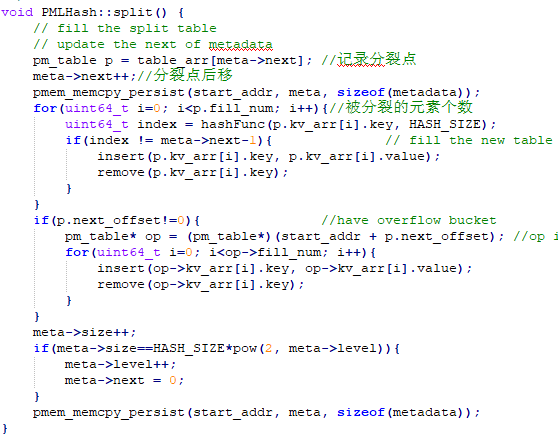


remove()函数的输入为key，首先根据hashFunc()函数计算出保存key的桶号，key可能保存在该哈希桶或其对应的溢出桶中。移除的方法是使用两个参数i、j进行同步，遍历哈希桶中的kv对结构体寻找key对应的value，i、j初始是相等的，若未到这个kv对，则使kv对[j]的key和value等于kv对[i]的value，由于此时i、j是相等的，因此在桶底的kv对数值不变，之后同时使i和j+1；当在kv对[j]处key与输入的key相等时时,不进行相等操作，且只将i+1，这样在下一个循环时kv对[j]的值将会被kv对[i]，也就是kv对[j+1]的值覆盖，这样就达到了移除kv对[j]的目的。

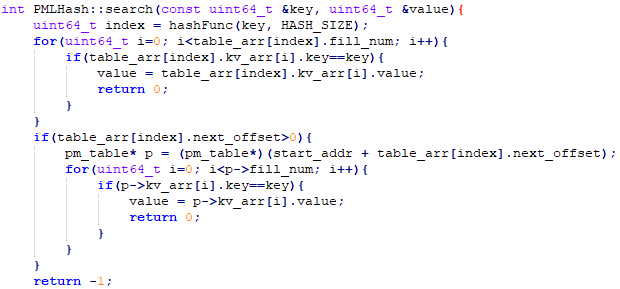
如果遍历哈希桶中的元素没有找到对应的key，说明该kv对一定保存在哈希桶对应的溢出桶中，只需对溢出桶进行相同操作即可。移除操作的代码如下图，只选择了移除哈希桶中元素的部分：

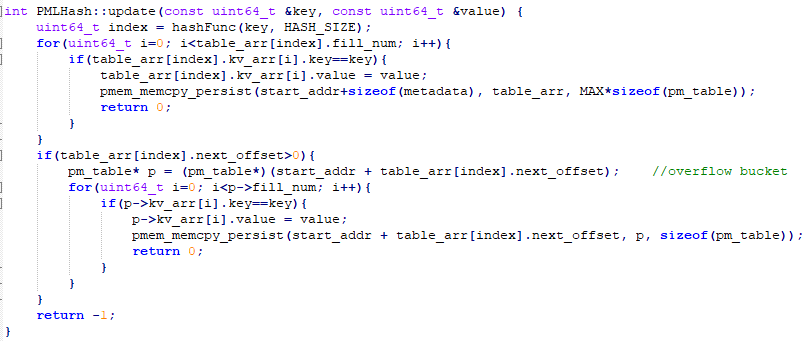


在insert()函数中，当一个kv对插入已满的哈希桶时，会触发桶的分裂，接下来就是分裂函数split()的实现过程。对于分裂操作，由于我们已经知道分裂点，每次触发分裂只需对处于分裂点的桶进行操作，因此该函数没有输入。首先记录下当前分裂的桶，然后将分裂点后移，这样我们就可以使用hashFunc()函数正确的计算出原kv对在分裂后应该插入的桶号。然后遍历桶中的kv对，对每一个kv对使用hashFunc()函数计算出分裂后应插入的桶，使用insert()函数将其插入到对应的桶，然后将原kv对使用remove()函数移除。如果分裂的桶拥有一个溢出桶，则对溢出桶的kv对进行相同操作。在分裂完成后，更新meta的size，即哈希桶的个数。如果哈希桶的个数已达到上限，则将分裂点移回第一个哈希桶，并将哈希桶的level+1。具体实现如下图：



最后是search()函数和update()函数，两个函数的功能和结构都类似，输入参数均为key和value，不同的是search()函数输入的value是用来接收找到kv对的value值，因此在输入时不需要赋值；而update()函数输入的value为找到的kv对需要更新的value值，因此在输入时需要赋值。对于search()函数，首先通过hashFunc()函数确定保存key的桶号，然后在对应的哈希桶和哈希桶的溢出桶中进行遍历，找到key对应的kv对，并将kv对的value值赋给输入的value参数即可；对于update()函数，同样通过hashFunc()函数确定保存key的桶号，然后在对应的哈希桶和哈希桶的溢出桶中进行遍历，找到key对应的kv对，然后将输入的value的值赋给kv对中的value。两个函数的实现如下：





至此，线性哈希的基本功能就已全部实现了。

1. 功能测试

为了验证设计的正确性，现进行以下测试：

* + 1. 插入测试
    2. 删除测试
    3. 更新测试
    4. 查找测试
    5. 综合测试