# 高级数据库技术实验二实验报告

姓名：杨扬 学号：SA13226226

1. 实验目的

本次实验是设计并且实现一个数据库缓冲区以及存储管理器。在整个数据库系统实现的五层架构中，缓冲区与存储管理器是处于第一层以及第二层，是上层记录管理器、执行引擎和查询编译器的低层支持，位于整个数据库的底层。

学习并且掌握数据库缓冲区以及存储管理器的设计对今后继续学习数据库底层结构、数据库性能分析，进而参与设计下一代数据库的设计有很大帮助。

1. 实验环境

|  |  |  |  |
| --- | --- | --- | --- |
| 硬件环境 | | 软件环境 | |
| CPU | Intel Pentium E5300 | 操作系统 | Windows 7 Ultimate |
| 内存 | DDR2 800MHz 4GB | 开发工具 | Visual Studio Ultimate 2012 |
| 硬盘 | Seagate 7200r/m 320G \* 2 | 开发语言 | C++ |
| 磁盘阵列 | RAID 0 |  |  |

1. 实验设计
2. 数据库缓冲区设计

* 相关常量设定

缓冲区帧的容量：4096

缓冲区帧的数量：1024（默认值）

* 主要数据结构

缓冲区LRU链表节点：LRUElement。该数据结构包含两个bool型的变量isHead和isTrail，分别表示该节点是否为该节点的头结点或者尾节点。int型变量frameID表示该LRU节点所指代的缓冲区的帧的帧号。int型变量pageID表示该LRU节点所指代的缓冲区的帧所对应的块的块号。另外设定两个节点类型指针用来遍历LRU链表。

isHead

isTrail

frameID

pageID

More Recent

Less Recent

LRU链表：LRU链表是执行LRU替换算法所必需的数据结构。在本次实现中，为了提高代码的可重用性，该链表被定义为具有头节点和尾节点的双线链表。除了头结点和尾节点有特定的bool型标志位标识以外，其他各个数据节点均为LRUElement的普通数据模式。

**……**

Head

LRUElemen

Trail

LRUElemen

缓冲区控制块：BCB。其主要数据成员与LRUElement基本相同，只是额外增添了一个bool型变量isWrite作为数据是否被修改的标志位，如若数据在程序执行过程中被修改，那么在执行缺页调度的时候需要将该BCB所对应的数据块写回磁盘。

isHead

isTrail

frameID

pageID

isWrite

next

previous

P2F哈希映射表：P2F哈希映射表是一个与缓冲区的帧的数量一致的静态哈希数组，每一个数组指代一个特定的哈希桶，并且存放指向该桶头结点地址的指针。在桶内部实现的是以BCB为节点的，带有头结点和尾节点的双向链表。在哈希映射表内部采用静态数组以方便随机存取，在哈希桶内部采用带有头结点和尾节点的双向链表结构以方便增删改查的操作。假设当前测试缓冲区的帧的数量为N：

……

 BCB

Hash\_P2F

[0]

Hash\_P2F

[1]

Hash\_P2F

[2]

Hash\_P2F

[N-2]

Hash\_P2F

[N-1]

Head

Head

Head

Head

Head

Trail

BCB

BCB

Trail

Trail

BCB

BCB

Trail

BCB

BCB

Trail

BCB

BCB

BCB

* 算法流程描述

首先从页面请求文件中读出一条新的页号指令，将该页号经过哈希函数处理之后，通过索引进入特定的哈希桶，并且对该哈希桶进行遍历。如若寻找到页号相符合的BCB即说明缓冲区命中。

如若命中，则直接对缓冲区中的特定块进行读写操作，并且进行LRU链表的调整，将特定的LRU节点插入队尾。

如果未命中，则开始调用磁盘IO流程，如若缓冲区依旧还未满，则创建新的缓冲区的帧，同时从空LRU队列摘下一个节点并插入到满LRU队列尾部。如果缓冲区已经被填满，则通过LRU算法指定将要被置换的帧，通过IO接口从数据文件读入新的数据，并且维护修改之后的LRU链表，通过哈希函数访问特定的哈希桶，遍历搜索桶以便对数据的写标记位进行维护，最后执行最该数据的读写操作。如果缓冲区的帧在被置换之前已经被修改过，则需要在读入新的数据之前将原数据写回磁盘。

当最后一条指令被执行完毕之后，再次遍历所有哈希桶，将所有被修改过的数据全部写入磁盘。

读入请求并哈希

哈希桶命中？

调整LRU链表

LRU有空位？

调整LRU链表

插入P2F哈希表

调入所需页

读写操作

执行LRU替换

调整LRU链表

修改P2F哈希表

有后续请求？

写回所有脏数据

存储器管理器设计

* 相关常量设定

磁盘块的容量：4096

磁盘块的数量：501000（500000个磁盘块用于存放数据块，冗余的1000个磁盘块用于存放磁盘索引块）

* 主要数据结构

磁盘数据文件data.dbf内部实现：磁盘块逻辑上被定义为基准块和数据块两类，其中基准索引块位于数据文件的头部，作为后续全部磁盘块的索引块，采用密集索引结构。

索引块定义为<页号，文件内部偏移地址>的键值对形式存储，每个项都采用int型变量，在32位计算机中该键值对占用8Byte，在一个索引块中可以存放512个索引项，共占用977个索引块，索引块之间采用连续存储形式。

数据块是按照4096Byte分块的连续存储空间，数据块只存储数据内容，不涉及页号等维护信息。数据块之间也采用连续存储形式。

数据块

空白块

索引块

0

1

…

511

1000

1001

…

1511

* 算法流程描述

由程序中的存储管理器负责数据的存取工作，存储管理器在程序初始化时被实例化，并且与指定的数据文件以及日志文件建立fstream关联。在程序运行过程中，存储管理器通过消息接口等待缺页调用，并且根据实际需要进行磁盘块数据读写以及日志的记录。

分析缺页调用

遍历查找索引块

读/写数据块

写入日志

1. 设计实现

* 类设计

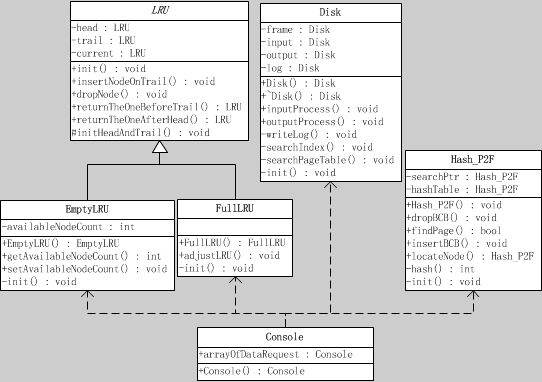
本次实验一共设计并实现了6个类，其中缓冲区管理器共涉及4个类，存储管理器仅涉及一个类。

在缓冲区管理器中，由LRU类、EmptyLRU类以及FullLRU类共同组成了LRU链表；其中LRU类为抽象基类，内部声明了对应不同适用范围的纯虚函数。EmptyLRU类继承自LRU类，实现了装载有空LRU节点的LRU链表。FullLRU类继承自LRU类，实现了装载有实体数据的LRU节点的LRU链表。Hash\_P2F类实现了由页号经哈希函数转换成帧号的功能。

在存储管理器中，设计了Disk类，实现了有关索引块搜索、数据块读写以及日志写入的功能。

此外，还设计并实现了Console类，作为整个项目页号请求的临时存储模块，实现了页号请求的写入以及读取操作。同时Console类作为整个项目的控制台，负责实例化其他各个类，并且通过消息接口调用其他对象的功能。

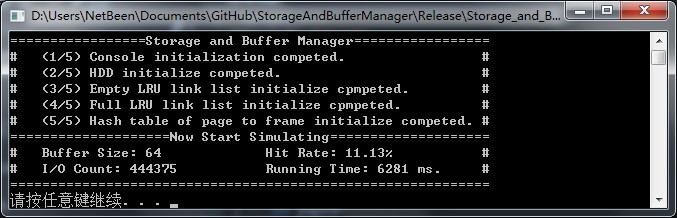
* 类图



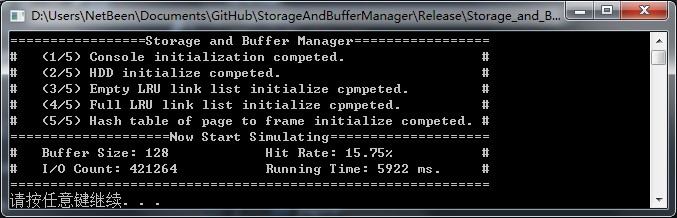
* 优化细节
  1. 在初始化的时候一次性为缓冲区分配全部的LRU节点，以避免在程序运行过程中频繁分配内存并创建LRU节点，可以提高前N次页请求的效率。
  2. 在初始化的时候一次性读入全部的页请求，并将其缓存于Colsole类的成员数组中，以避免程序每处理一个页请求都要启动一次IO，可以提高程序运行速度。

1. 运行结果及分析

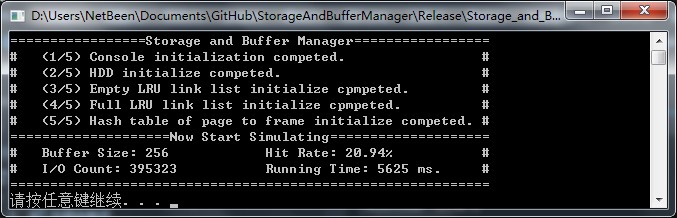
当缓冲区大小为64帧时的运行结果：



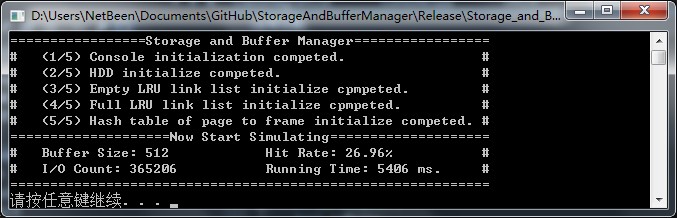
当缓冲区大小为128帧时运行的结果：



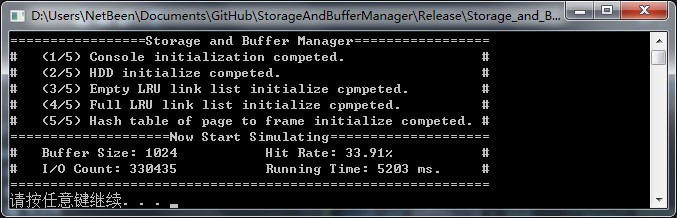
当缓冲区大小为256帧时运行的结果：



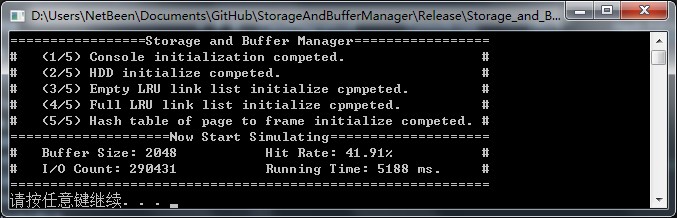
当缓冲区大小为512帧时运行的结果：



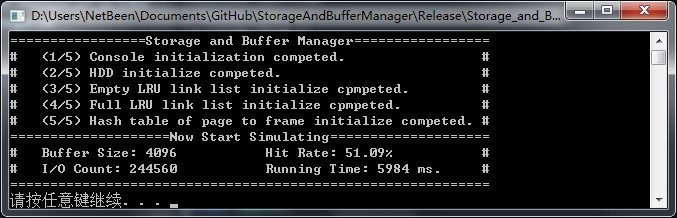
当缓冲区大小为1024帧时运行的结果：



当缓冲区大小为2048帧时运行的结果：



当缓冲区大小为4096帧时运行的结果：



* 结果分析

在缓冲区较小的时候，通过增大缓冲区容量能够有效提升缓冲区命中率，极大程度上降低程序运行时间。

由于测试平台采用RAID-0磁盘阵列，磁盘的读写速度均获得了较大程度上的提升，这在一定程度上降低了存储管理器的IO代价占总代价的比重；并且，缓冲区也是需要依赖于大量数据结构以及内存中开辟的大量空间，维护缓冲区的代价会随着缓冲区容量的不断提升而持续增加。因此，当缓冲区进一步增大直至超过2048Byte时，尽管缓冲区命中率环比依旧不断上升，但程序运行时间反而有了小幅增加。

通过实际测得的数据分析，最短程序运行时间出现在缓冲区大小为2048帧之时。经过统计数据中分析出的结果得出，适用于本平台的最佳缓冲区大小约为1636帧。

* 总结

通过本次实验，我对于数据库系统的底层实现有了更加深刻的理解。从仅仅了解如何使用关系数据库搭建各种应用到真正掌握数据库底层数据结构以及相关实现算法，进一步深化了我对数据库以及操作系统对于缓冲区的控制的了解。这对于我未来的学习有着莫大的帮助。