实验 存储与缓存管理器

李清伟 **SA23011033**
qingweili2357@mail.ustc.edu.cn

1 摘要

本项目实现了数据库管理系统（DBMS）中不考虑并发情况下的存储管理器、缓冲管理器。在缓冲替换策略上，本项目实现了 naive 版本、LRU、LRU-K、Clock 算法，并在通过 `FixNewPage` 构建好的包含 50000 个页的堆文件上实现按照 trace 文件操作数据库堆文件进行了磁盘 IO 次数、缓冲命中率、运行时间、被替换的数量、数据结构维护时间 等性能数据的测量与对比。本项目有以下发现：1) LRU-2 策略相比于其他版本的替换策略在磁盘 IO 次数、缓冲命中率、被替换的数量 指标上效果更好。2) LRU-k 中 k 的设置不是越大越好。3) LRU-k(k=2,3,4) 策略在磁盘 IO 次数、缓冲命中率、被替换的数量 上效果比 LRU 好。4) Clock 算法在运行时间 和数据结构维护时间 上比 LRU 系算法好，但是在磁盘 IO 次数、缓冲命中率、被替换的数量 指标上效果较差。

2 管理器设计

本项目主要有 4 部分组成，包含存储管理器、缓冲管理器、性能评估器、驱动程序。

2.1 存储管理器

存储管理器 使用 page ID 作为索引方式。一个 page 的大小为 FRAMESIZE = 4KB。一个 page 的从文件开始的偏移为[pageID * pageSize, (pageID+1)* pageSize)。存储管理器 主要提供了 `ReadPage` 和 `WritePage` 接口供缓冲管理器 调用，分别对应读、写操作。每次读写操作的大小为 1 个 page 大小。其他接口功能比较简单直接，因此不再赘述。

2.2 缓冲管理器

一个事务通过 `FixPage` 接口请求一个已经存在的 page。当上层事务使用完这个 page 之后，会通过 `UnfixPage` 来提示缓冲管理器 对该 page 的使用结束，可以将记录使用该 page 的数量减少。当上层事务要求新 page 时，会通过 `FixNewPage` 请求获得新 page。

缓冲管理器 主要是因为存在加速上层索引、文件、记录管理器请求响应的需求而出现的。为了达到这一目的，缓冲管理器 将频繁访问的 page 放到内存中（这里为 `GlobalBuf`），并且 `BCB` 来存储这个 page 的状态信息，包括是否 dirty，当前使用该 page 的事务数量，frame 和 page 的映射、是否加锁等。

由于内存资源有限，因此当缓冲被填满时，需要将一些替换代价低的页替换出去，并从存储管理器 中获得请求所需要的页，并加入到缓冲中，维护相关信息。在选择受害者时，存在替换策略的问题，本项目实现了常见的替换策略 LRU、LRU-K、Clock，并且还实现了一个非常简单的 naive 版本。

2.2.1 page 和 frame 对应关系设计

本项目中 page 和 frame 的关系如图 1 所示。只要有一个 frame 是空闲的，就会将其分配给没有 frame 对应的新 page。分配完成之后，会将 frame → page 的映射存入 `f2p` 中。而 page → frame 的关系则通过键为 page，值为 `BCB` 链表的哈希表 `p2bcb` 进行存储。具体来说，要从一个 page 找到 frame ID，需要：

1. 通过哈希函数得到 pageHash = Hash(page ID)

2. 通过 pageHash 和 p2bcb 得到 BCB 链表头指针
3. 在 BCB 链表中比对 BCB 中存储的 page ID，得到 BCB 之后返回 frame ID

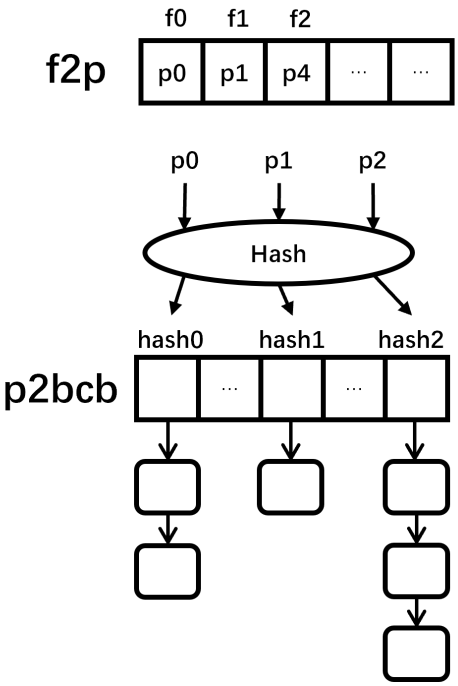

图 1　f2p 和 p2bcb 结构

2.2.2 BCB 链表

BCB 结构定义如代码 1 所示。

```
class BCB {
public:
    BCB(int pageid, int frameid)
    : pageID(pageid), frameID(frameid), latch(0), count(0), dirty(0), next(nullptr) {}
    int pageID;
    int frameID;
    int latch;
    int count;
    int dirty;
    BCB* next;
};
```

代码 1　BCB 结构定义

2.2.3 FixPage 操作流程

操作流程如图 2 所示。

2.2.4 替换流程

在图 2 的操作流程中，需要通过 SelectVictim 选择合适的受害者并进行替换，同时还需要维护 BCB, f2p, p2bcb 以及替换策略实现者内部的数据结构。SelectVictim 的流程如图 3 所示。

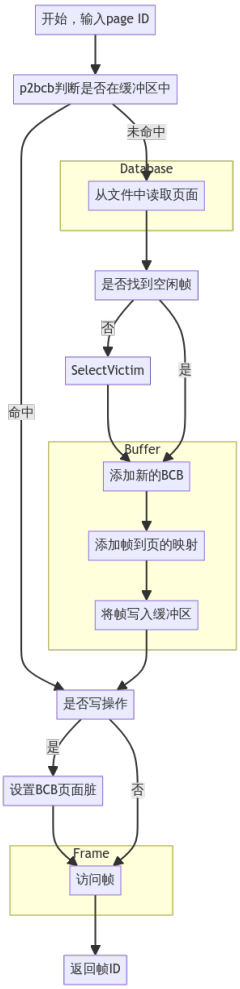

图 2　FixPage 操作流程图

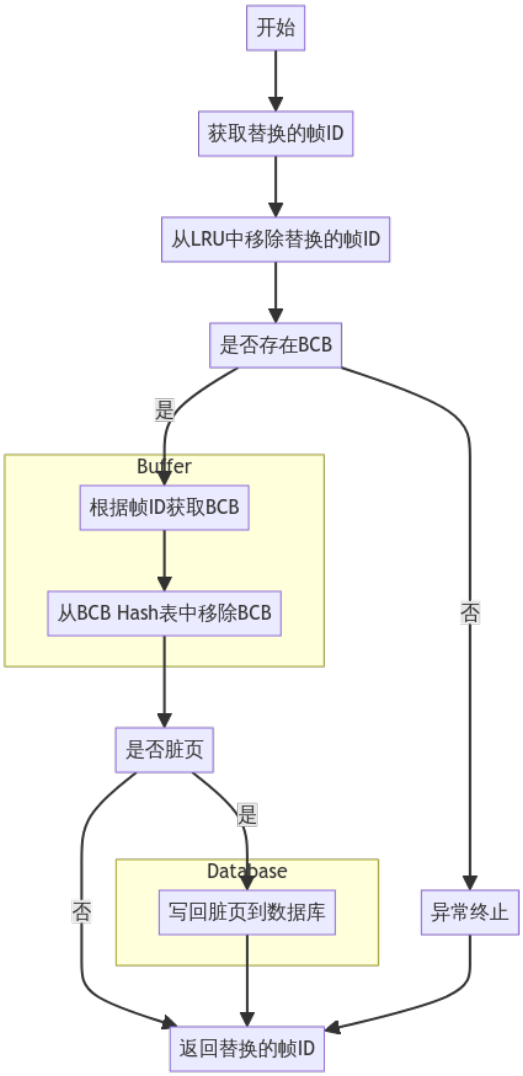

图 3　SelectVictim 操作流程图

2.2.5 替换策略

2.2.5.1 naive

每次只选择 0 号 frame 进行替换。见../src/BMgr.cc

2.2.5.2 LRU

每次选择队尾作为 victim。每次访问都放到队头（如果在 LRU 链表中存在，则先删除，后放到队头），见../src/LRUBMgr.cc 50 行以上。

2.2.5.3 LRU-K

维护两个列表，访问次数 ≥ K 的放到一个热表，否则放到另一个冷表。优先替换冷表。见../src/LRUBMgr.cc 50 行以下。

2.2.5.4 Clock

当 clock 未满的时候，不断增加 current 并填充新访问的 frameID，设置 referenced 为 true。当 clock 满的时候，如果要访问的 frame 在 clock 中存在，则将相应项中的 referenced 设置为 true。否则表示已经替换了受害者页，需要设置新的 frame ID。此时直接在 current 指针下设置 frame ID 并设置 referenced 为 true。

在选择受害者时，不断将 referenced == true 的项设置为 referenced = false。一旦找到一个 referenced 为 false 的项，则作为受害者返回 frame ID。

2.3 性能评估器

性能评估器主要用于评估磁盘 IO 次数、缓冲命中率、运行时间、被替换的数量、数据结构维护时间。其中：

1. 磁盘 IO 次数 次数通过存储管理器 的 ReadPage 和 WritePage 接口的调用次数确定。
2. 缓冲命中率 通过 FixPage 中是否找到 BCB 确定。如果找到，则为 hit，否则为 miss。
3. 运行时间 为所有 Read, Write 以及最后的 WriteDirtys 操作的时间求和组成。
4. 被替换的数量 为 SelectVictim 对被替换页的数量，包括被替换的干净页、脏页。干净和脏通过 BCB 的 dirty 位进行区分。
5. 数据结构维护时间 主要通过插桩得到，是不同替换策略维护数据结构所花费的时间累计和。

源代码见../include/Evaluator.h

2.4 驱动程序

驱动器主要是用于解析待测负载文件并转换为对缓冲管理器 的操作序列。同时驱动器还用于驱动生成性能测试数据。源代码见../main.cpp

3 运行结果

运行方式见../README.md。

统计指标见小节 2.3 的说明。

3.1 不同替换策略的性能对比

不同策略的性能对比如图 4 - 图 8 所示。可以发现 LRU-2 策略相比于其他版本的替换策略在磁盘 IO 次数、缓冲命中率、被替换的数量 指标上效果更好。LRU-k(k=2,3,4)策略在磁盘 IO 次数、缓冲命中率、被替换的数量 上效果比 LRU 好。LRU 算法的磁盘 IO 次数 甚至比不过 naive 版本的实现。

在 LRU-k 系列算法中，随着 K 增大，磁盘 IO 次数 并没有减少，缓冲命中率 也有所降低，被替换的数量 也有所减少。说明 k 不是越大越好。

同时 Clock 算法相比于 LRU 系算法在运行时间 和数据结构维护时间 上效果较好，但是磁盘 IO 次数、缓冲命中率、被替换的数量 指标上效果甚至比 naive 算法还要差。

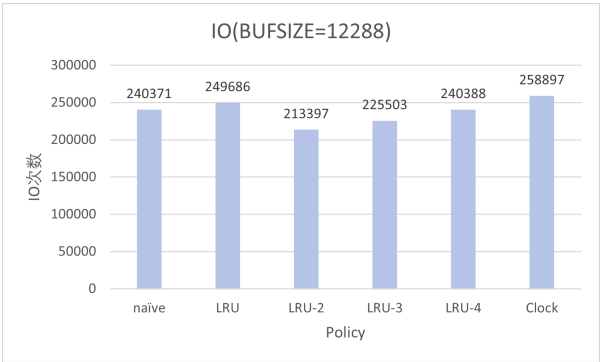

图 4　BUFSIZE=12288 时不同策略的磁盘 IO 次数

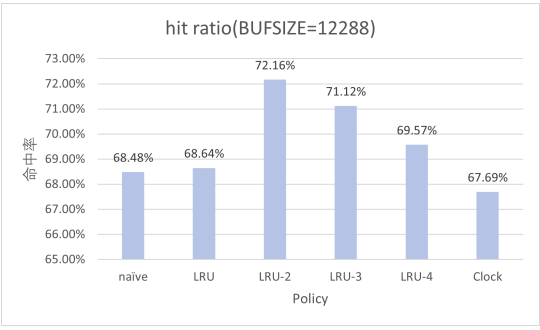

图 5　BUFSIZE=12288 时不同策略的缓冲命中率

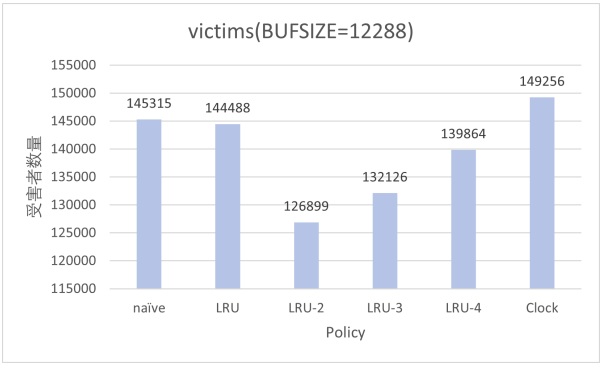

图 6　BUFSIZE=12288 时不同策略的被替换的数量

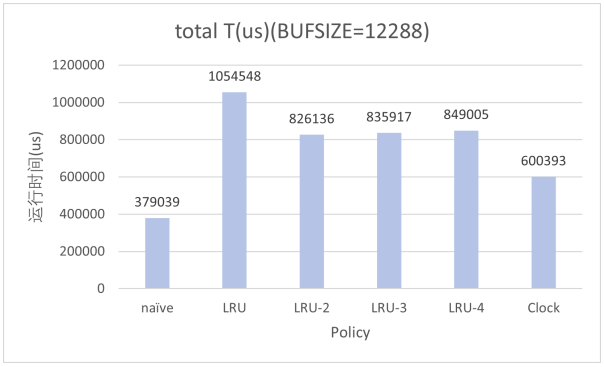

图 7　BUFSIZE=12288 时不同策略的运行时间

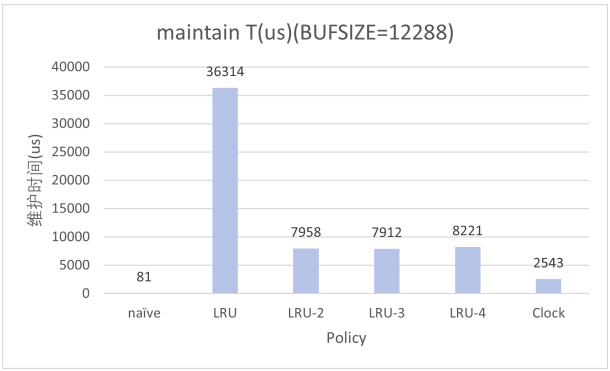

图 8　BUFSIZE=12288 时不同策略的数据结构维护时间

3.2 LRU-2 算法在不同 BUFSIZE 下的性能变化

不同 BUFSIZE 对于替换策略的效果也有影响。对于 LRU-2 算法来说，其磁盘 IO 次数、缓冲命中率、运行时间 的变化如图 9 - 图 11 所示。当 BUFSIZE 增大时，磁盘 IO 次数、缓冲命中率、运行时间 效果都显著变好。但是当 BUFSIZE 进一步扩大时，运行时间 并没有进一步减少，主要是因为 BUFSIZE 太大导致维护代价变高，如图 12 所示。

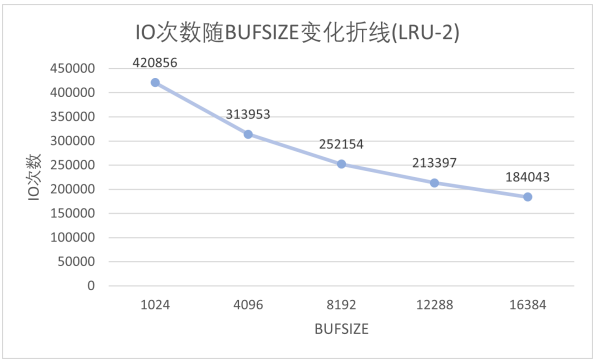

图 9　LRU-2 磁盘 IO 次数 随 BUFSIZE 变化折线图

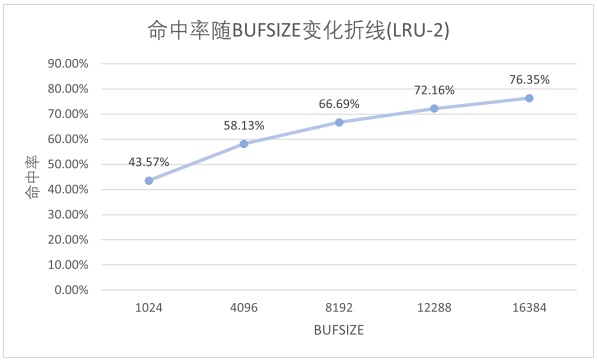

图 10　LRU-2 缓冲命中率 随 BUFSIZE 变化折线图

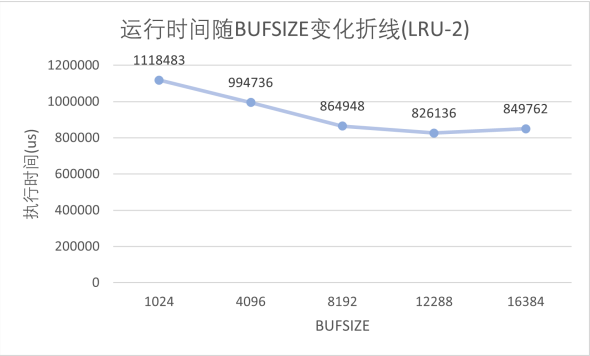

图 11　LRU-2 运行时间 随 BUFSIZE 变化折线图

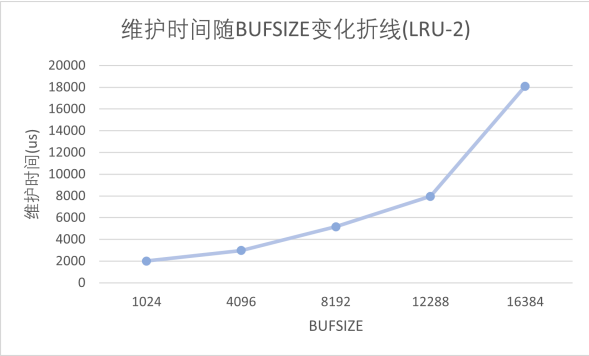

图 12　LRU-2 数据结构维护时间 随 BUFSIZE 变化折线图

4 总结

本项目实现了存储管理器、缓冲管理器 以及多种替换策略。在替换策略的对比中发现 LRU-2 的效果较好。LRU-k 中 k 的设置不是越大越好。Clock 算法在运行时间 和数据结构维护时间 上比 LRU 系算法好，但是在磁盘 IO 次数、缓冲命中率、被替换的数量 指标上效果较差。