

National University of Singapore
School of Computing
CS3223: Database Systems Implementation
Semester 2, 2015/16
Assignment 1: Buffer Manager (8 marks)
Due: 11:59pm, February 21, 2016 (Sunday)

1 Introduction

In the previous assignment, we learned how to install `PostgreSQL` and use its client program `psql`. Now, we start with some actual C code hacking! `PostgreSQL` 9.4.5 uses a variant of the Clock replacement policy as the default replacement policy. In this assignment, you will modify the buffer manager component of `PostgreSQL` to implement the LRU replacement policy. The actual coding for this assignment is minimal, given the highly organized and modular `PostgreSQL` code (the code for the buffer manager policy is cleanly separated from the rest of the code base). However, you have to figure out what code to modify, which is non-trivial!

Late submission penalty: There will be a late submission penalty of 1 mark per day up to a maximum of 3 days. If your assignment is late by more than 3 days, it will not be graded and your team will receive 0 credit for the assignment. Do start working on your assignment early!

2 Overview of PostgreSQL

The following is a brief introduction to key aspects of `PostgreSQL` that are relevant for this assignment.

2.1 Shared-memory Data Structures

In general, there could be multiple backend processes accessing a database at the same time. Therefore, access to shared-memory structures (e.g., buffer pool data structures) needs to be controlled to ensure consistent access and updates. To achieve this, `PostgreSQL` uses *locks* to control access to shared-memory structures by following a locking protocol: before accessing a shared-memory structure, a process needs to acquire a lock for that structure; and upon completion of the access, the process needs to release the acquired lock. There are two basic types of locks available: *read/shared locks* and *write/exclusive locks*. A read lock should be used if read-only access is required; otherwise, a write lock should be used.

2.2 Buffer Manager

There are two types of buffers used in `PostgreSQL`: a *shared buffer* is used for holding a page from a globally accessible relation, while a *local buffer* is used for holding a page from a temporary relation that is locally accessible to a specific process. This assignment is about the management of `PostgreSQL`'s shared buffers.

Initially, all the shared buffers managed by PostgreSQL are maintained in a *free list*. A buffer is in the free list if its contents are invalid and is therefore of no use to any client process. Whenever a new buffer is needed, PostgreSQL will first check if a buffer is available from its free list. If so, a buffer from the free list is returned to satisfy the buffer request; otherwise, PostgreSQL will use its buffer replacement policy to select a victim buffer for eviction to make room for the new request. PostgreSQL 9.4.5 implements a variant of the Clock algorithm as its default replacement policy.

In PostgreSQL, when a record is deleted (or modified), it is not physically removed (or changed) immediately; instead, PostgreSQL maintains multiple versions of a record to support a *multiversion concurrency control* protocol (to be covered in the later part of the course). Periodically, a *vacuuming* process will be run to remove obsolete versions of records that can be safely deleted from relations. If it happens that an entire page of records have been removed as part of this vacuuming procedure, the buffer holding this page becomes invalid and is returned to the free list.

PostgreSQL also runs a background writer process (called **bgwriter**) that writes out dirty shared buffers to partly help speed up buffer replacement.

3 Getting Started

Each team will be using a reserved Compute Cluster compute node (with prefix compg) for this assignment. Refer to Appendix A (on the last two pages) for the compg node allocated to your team. Each team should use only the assigned compg node.

To access your assigned compg node, you will first need to ssh to an access node (ie., angšana or tembusu) and then ssh to your assigned compg node from there. The following illustrates how to ssh to the compg99 node for the user alice.

```
$ ssh alice@angšana.comp.nus.edu.sg
alice@sma4:~$ ssh compg99
alice@compg99:~$
```

Now, run the following set up for this assignment.

```
$ cd $HOME
$ wget http://www.comp.nus.edu.sg/~cs3223/assign/assign1.zip
$ unzip assign1
$ cd assign1
$ bash ./setup.sh
$ source ~/.bash.profile
```

The **setup.sh** script installs two additional modules, **pgbench** and **test_bufmgr** extension, that will be used for performance benchmarking and testing, respectively. The script also sets up your database cluster in the local disk located at **/temp** and updates the **~/.bash.profile** file.

The **assign1** sub-directory created in your home directory contains the following directories and files:

- **freelist-lru.c**: This is a modified version of the original file in **src/backend/storage/buffer/freelist.c**. You will need to make further modifications to this file to implement the LRU replacement policy.

- **bufmgr.c**: This is a modified version of the original file in `src/backend/storage/buffer/bufmgr.c`. It is not necessary to modify this file for the assignment.
- **freelist.original.c** and **bufmgr.original.c**: These are the original versions of `freelist.c` and `bufmgr.c` that can be used to restore the default PostgreSQL installation¹.
- **setup.sh**: This is a script to install additional programs used in this assignment.
- **test.sh**: This is script to run the test cases.
- **benchmark.sh**: This is a script for performance benchmarking.
- **test.bufmgr/**: This is a directory containing the `test.bufmgr` extension to be installed by `setup.sh`.
- **testdata/**: This is a directory containing the test data used by `test.sh` and its data loading script.
- **testresults-lru-solution/**: This is a directory containing the result files for testing LRU
- **Makefile**.
- **freelist-lru2.c**, **testresults-lru2-solution/**: These files are for the implementation and testing of LRU2 (optional).

4 What to do

This assignment consists of three parts. The first part (2 marks) is on the performance benchmarking of PostgreSQL’s Clock replacement policy. The second part (6 marks) is on the implementation of the LRU replacement policy and its performance benchmarking. The third part is an optional “bonus” part (2 marks) on the implementation and benchmarking of a variant of the LRU replacement policy.

4.1 Benchmarking PostgreSQL’s Clock Replacement Strategy (2 marks)

In the first part of the assignment, you will run a simple benchmark to measure the performance of PostgreSQL’s Clock replacement strategy.

1. Start the database server to use only 32 buffer frames with the “-B” option.

```
$ postgres -B 32 2>> ~/log.txt &
```

2. Create a new database named *assign1* for this assignment.

```
$ createdb assign1
```

3. Benchmark the performance of the Clock policy using the script `benchmark.sh`.

¹ To restore the default PostgreSQL installation, run the command “`make clock`”.

```
$ cd ~/assign1
$ ./benchmark.sh clock.txt
```

The results of the benchmarking will be written to a file named “clock.txt”.

The script `benchmark.sh` creates a four-relation database using the program `pgbench` (installed by `setup.sh`) and then executes a workload of transactions from 10 clients for a duration of 3 minutes. The following performance metrics are reported by the benchmarking:

- average transaction latency (in milliseconds),
- system throughput (in number of transactions processed per second), and
- buffer hit ratio which measures how often a requested data page is found in the buffer pool without incurring disk I/O to read the page from the disk; i.e., it is the ratio of the total number of page requests that are found in the buffer pool to the total number of page requests.

For more information on `pgbench`, refer to PostgreSQL 9.4.5 documentation or its man page.

4. Stop the database server.

```
$ pg_ctl stop
```

4.2 Implementing & Benchmarking LRU Strategy (6 marks)

In the second part of the assignment, you will implement the *LRU buffer replacement policy* and benchmark its performance on PostgreSQL.

A simple approach to implement the LRU policy is to use a doubly-linked list to link up the buffer pages such that a page that is closer to the front of the list is more recently used than a page that is closer to the tail of the list. Thus, whenever a buffer page is referenced, it is moved to the front of the list; and whenever a replacement page is sought from the list, the unpinned buffer page that is closest to the tail of the list is selected for eviction. This implementation approach has been referred to as the *Stack LRU* method in some operating systems textbooks; here, the top and bottom of the stack correspond to the front and tail of the linked list, respectively. Whenever a buffer is accessed, its position within the stack needs to be adjusted. This stack adjustment can be classified into four cases:

- (C1) If an accessed page is already in the buffer pool, then the containing buffer needs to be moved to the top of the stack.
- (C2) If an accessed page is not in the buffer pool and a free buffer is available to hold this page, then the selected buffer from the free list needs to be inserted onto the top of the stack.
- (C3) If an accessed page is not in the buffer pool and the free list is empty, then the selected victim buffer needs to be moved from its current stack position to the top of the stack.
- (C4) If a buffer in the buffer pool is returned to the free list, then the buffer needs to be removed from the stack.

4.2.1 Implementation

The following are some guidelines on how you can go about implementing the LRU replacement policy in PostgreSQL.

1. Before you begin making changes to PostgreSQL, you should examine the existing code to understand how the buffer manager (specifically its Clock policy replacement) is being implemented. The existing code is not extremely clear, but it is understandable. It may take you a few hours (or more) to digest it. Since understanding the existing code is a significant part of the assignment, the TA and Professor will not assist you in your understanding of the code base (beyond what we discuss here). The actual buffer manager code is neatly separated from the rest of the code base. Its files are located in the `src/backend/storage/buffer/` and `src/include/storage/` directories. The two main files of interest for this assignment are `bufmgr.c` and `freelist.c`. Modified versions of these two files (to be used for this assignment) are given in `assign1/bufmgr.c` and `assign1/freelist-lru.c`. While `bufmgr.c` contains the implementation of the buffer manager, we are only interested in `freelist-lru.c`, which defines the buffer replacement policy.

To identify the parts in `assign1/bufmgr.c` and `assign1/freelist-lru.c` that have been modified, search for the string “`cs3223`”.

The following is a brief description of some of the relevant files for this assignment:

- `assign1/freelist-lru.c`: Contains functions that implement the replacement strategy. This is the only file you need to modify.
 - `assign1/bufmgr.c`: Implements the buffer manager.
 - `src/include/storage/buf_internals.h`: Contains the definition of each buffer frame (called *BufferDesc*). Most of the fields in *BufferDesc* are managed by other parts of the code.
 - `src/backend/storage/buffer/buf_init.c`: Some initialization of the buffer frames occur in this file. However, you should do all your initialization in `freelist-lru.c` (see `StrategyInitialize` routine in `freelist-lru.c`).
 - `src/backend/storage/buffer/README`: Useful description of the *Strategy* interface implemented by `freelist-lru.c`.
2. For this assignment, the main file that you need to modify is `assign1/freelist-lru.c`. The given file is a slightly modified version of the original file `src/backend/storage/buffer/freelist.c`. You will need to make further modifications to this file to implement LRU (instead of Clock) replacement policy. To help you with the LRU implementation, we have defined a new function in `freelist-lru.c`:

```
void StrategyUpdateAccessedBuffer (int buf_id, bool delete)
```

This function is used to update the LRU stack when a buffer, which is uniquely identified by its buffer index number given by `buf_id`, is accessed. The `delete` parameter is used to distinguish case C4 from the other three cases: the value of `delete` is *true* for handling case C4; and *false*,

otherwise. This new function is used in two files: `bufmgr.c`, and `freelist-lru.c`. You will need to implement this new function. Refer to the `BufferAlloc` function in `assign1/bufmgr.c` for an example of how `StrategyUpdateAccessedBuffer` is being used to update the LRU stack to handle case C1 (i.e., when the accessed page is already in the buffer pool).

3. Besides implementing the `StrategyUpdateAccessedBuffer` function, you will need to make other necessary changes to `freelist-lru.c`. You will probably need to modify the following functions in `freelist-lru.c`:

- `StrategyInitialize` (to allocate and initialize shared memory for your LRU-related data structures using `ShmemInitStruct` function),
- `StrategyShmemSize` (to account for any additional shared memory used by your LRU-related data structures using functions such as `add_size`, `mul_size`, etc.),
- `StrategyGetBuffer` (to handle cases C2 and C3), and
- `StrategyFreeBuffer` (to handle case C4).

You can use the `assign1/Makefile` to compile your `freelist-lru.c` and install the modified files into the PostgreSQL source tree. You will need to first edit the Makefile to ensure that the `PSQLPATH` variable is correctly set to the path of your PostgreSQL source tree. To compile your `freelist-lru.c`, use the command “`make freelist-lru.o`”.

4.2.2 Testing

Once your `freelist-lru.c` compiles correctly, you are now ready to install your LRU implementation into the PostgreSQL source tree and test your implementation.

To install your changes, you need to replace `src/backend/storage/buffer/freelist.c` and `src/backend/storage/buffer/bufmgr.c` with `assign1/freelist-lru.c` and `assign1/bufmgr-lru.c`, respectively, and re-install PostgreSQL. This can be done by simply using the Makefile².

```
$ cd ~/assign1
$ make lru
```

Once your LRU implementation is installed successfully, test your implementation with the script `test.sh` by starting the server with 16 buffer pages.

```
$ postgres -B 16 2>> ~/log.txt &
$ cd ~/assign1
$ ./test.sh
$ pg_ctl stop
```

The testing comprises of 10 test cases (`~/postgresql-9.4.5/contrib/test_bufmgr/testcases`) that is part of the `test_bufmgr` extension installed by the `setup.sh` script. Each test case is a sequence of write/unpin requests to pages of a relation named `movies` (`~/assign1/testdata`) which occupies 43 8KB-pages (numbered from 0 to 42). Specifically, there are three types of page requests in the test cases:

² To restore the original PostgreSQL installation, run the command “`make clock`”.

- **write_pin_block(blkno)** modifies the page with block number **blkno** and the page remains pinned;
- **write_unpin_block(blkno)** modifies the page with block number **blkno** and unpins the page; and
- **unpin_block(blkno)** unpins the page with block number **blkno**.

Each test case is executed via the SQL extension function `test_bufmgr` which reports the identifier of the buffer frame (given by `bufid` with values ranging from 0 to 15) and its pin count (given by `refcount`) for each processed page request (with block number given by `blkno`). The results of running the test cases are stored in the directory `~/assign1/testresults`. You can compare your results against the provided files in `testresults-lru-solution/`.

4.2.3 Benchmarking

After testing and debugging your implementation of LRU policy, benchmark its performance and save the results in a file named “`lru.txt`”.

```
$ postgres -B 32 2 >> ~/log.txt &
$ cd ~/assign1
$ ./benchmark.sh lru.txt
$ pg_ctl stop
```

4.3 Implementing & Benchmarking LRU-2 Strategy (Optional)

In the third optional part of the assignment (bonus 2 marks), you will implement the *LRU-2 buffer replacement policy* and benchmark its performance on PostgreSQL.

4.3.1 LRU-2 Buffer Replacement Policy

The LRU-2 policy is a variant of LRU: instead of choosing a replacement page based on the recency of its last access time, the LRU-2 policy chooses a replacement page based on the recency of its second last access time. More specifically, let $S = S_1 \cup S_2$ denote the set of unpinned buffer pages that can be selected for replacement, where S_1 is the set of buffer pages that have been accessed only once and S_2 is the set of buffer pages that have been accessed at least twice. The LRU-2 policy selects a replacement page from S for replacement as follows. If S_1 is non-empty, choose the least recently accessed page in S_1 as the replacement page; i.e., we apply the conventional LRU policy to select the replacement page from S_1 . Otherwise, if S_1 is empty, we consider the second-last access times for the pages in S_2 and choose the page in S_2 with the smallest second-last access time as the replacement page.

The following example illustrates the working of LRU-2 on a buffer pool consisting of three frames for a sequence of pin/unpin commands.

Step	Commands	Contents of Buffer Pool
1	write_unpin_block(10)	10
2	write_unpin_block(20)	10, 20
3	write_unpin_block(20)	10, 20
4	write_unpin_block(10)	10, 20
5	write_unpin_block(30)	10, 20, 30
6	write_pin_block(40)	10, 20, 40
7	write_unpin_block(10)	10, 20, 40
8	write_unpin_block(50)	10, 50, 40

The retrieval of page 40 replaces page 30 as page 30 is the only page that is accessed only once. The retrieval of page 50 replaces page 20 as the second-last access time for page 20 (step 2) is less recent than the second-last access time for page 10 (step 4).

4.3.2 Implementation

Similar to the guidelines for implementing LRU policy, you would need to implement the LRU-2 policy by coding the function `StrategyUpdateAccessedBuffer` and modifying other required functions in `assign1/freelist-lru2.c`.

1. You can compile `freelist-lru2.c` as follows: `make freelist-lru2.o`.
2. Once your `freelist-lru2.c` compiles correctly, install your LRU-2 implementation in the actual PostgreSQL source tree by running `"make lru2"` within the `assign1/` directory.
3. You can test your implementation by running `./test.sh` and comparing your test results against the provided files in `testresults-lru2-solution/`.

4.3.3 Benchmarking

After installing your implementation of LRU-2 policy into PostgreSQL, benchmark its performance and save the results in a file named `"lru2.txt"`.

```
$ postgres -B 32 2>> ~/log.txt &
$ cd ~/assign1
$ ./benchmark.sh lru2.txt
$ pg_ctl stop
```


5 What & How to Submit

For this assignment, you will need to submit the following files:

1. **README.txt**: This is a short text file providing the following information:
 - (a) The names of the team members
 - (b) A concise description of the key changes that you have made to implement LRU. You should highlight any new structures used and any subtle points about your code.
 - (c) A summary of your observations from the benchmarking results.
 - (d) Include items (b) and (c) for LRU-2 if you have also completed the optional part of the assignment.
2. **freelist-lru.c**: This is your implementation of LRU policy.
3. **clock.txt** and **lru.txt**: Benchmark results for Clock & LRU.
4. **freelist-lru2.c** and **lru2.txt**: Your implementation and benchmarking of LRU-2 if you have completed the optional part of the assignment.

If you have modified any other additional file (e.g., bufmgr.c), submit all the additional modified files as well and explain the modifications in **README.txt**.

The above files should be submitted in a .zip file containing a directory named after the student number of one the team members.

```
$ mkdir a0123456x
$ cp README.txt freelist-lru.c clock.txt lru.txt freelist-lru2.c lru2.txt a0123456x
$ zip -r a0123456x a0123456x
```

Upload the zip file into the module's IVLE **Submission-Assignment-1** workbin.

A Allocation of Compute Nodes

The following table lists the team account and compute cluster node assigned to each team.

Team	Name	Email	Assigned Node
a	Lin Che Hao Sim Thiam Guan Calvin	a0112918@u.nus.edu a0111947@u.nus.edu	compg10
b	Daryl Ng Wee Kiat Tay Shuan Siang	a0111546@u.nus.edu a0108543@u.nus.edu	compg11
c	Nguyen Viet Dung Tran Manh	a0112068@u.nus.edu a0112058@u.nus.edu	compg12
d	Bui Do Hiep Vu Dinh Quang Dat	a0126502@u.nus.edu a0126517@u.nus.edu	compg13
e	Nicholas Ng Nuo Yang Tan Ruyi	a0112224@u.nus.edu a0112856@u.nus.edu	compg14
f	Huang Li Stefano Chiesa Suryanto	a0112508@u.nus.edu a0126509@u.nus.edu	compg15
g	Daryl Tan Zhi Wei Li Fengshuang	a0111660@u.nus.edu e0014551@u.nus.edu	compg16
h	Sharon Lynn Tjong Anthony	a0112171@u.nus.edu a0126539@u.nus.edu	compg17
i	Mi Chengyi Wang Kunzhen	a0105494@u.nus.edu a0105514@u.nus.edu	compg18
j	Nguyen An Ha Nguyen Van The	a0113038@u.nus.edu a0098068@u.nus.edu	compg19
k	Chng Hui Yie Xu Gelin	a0127357@u.nus.edu a0105508@u.nus.edu	compg25
l	Chi Cheng Pranav Kashyap Narasimhan	a0083782@u.nus.edu e0010100@u.nus.edu	compg26
m	Sherlyn Goh Sze Hui Zhang Wei	a0101651@u.nus.edu a0101349@u.nus.edu	compg27
n	Ho Jia Yi Koh Wen Yao	a0101360@u.nus.edu a0097210@u.nus.edu	compg28
o	Chan Wai Hoong Alvin Tee Jia Min Michelle	a0097379@u.nus.edu a0101222@u.nus.edu	compg29

Team	Name	Email	Assigned Node
p	Le Ba Hieu Giang Yong Shan Xian	a0127828@u.nus.edu a0132763@u.nus.edu	compg30
q	Cao Shengze Feng Junhan	a0119403@u.nus.edu a0125489@u.nus.edu	compg31
r	Ji Yahui Ong Teck Sheng	a0101973@u.nus.edu a0097706@u.nus.edu	compg32
s	Lai Hoang Dung Tham Zheng Yi	a0131125@u.nus.edu a0108232@u.nus.edu	compg33
t	Guo Xiang Li Mengying	a0112659@u.nus.edu a0113594@u.nus.edu	compg34
u	Qua Zi Xian Zheng Weihao	a0110781@u.nus.edu a0097582@u.nus.edu	compg38
v	Pranav Murugappan Spatika Narayanan	e0009229@u.nus.edu a0088416@u.nus.edu	compg39
w	Bai Chuan Huang Bohan	a0133948@u.nus.edu a0133907@u.nus.edu	compg40
x	Lai Mun Keat Lin Jiahao	a0129561@u.nus.edu a0108235@u.nus.edu	compg41
y	Luah Bao Jun Moazzam Ali Khan	a0126258@u.nus.edu a0110924@u.nus.edu	compg42
z	Li Yumeng Xu Haiyan	a0133911@u.nus.edu e0013685@u.nus.edu	compg43
aa	Nguyen Tan Sy Nguyen Olof Erik Hoejvall	a0099429@u.nus.edu e0009404@u.nus.edu	compg44
ab	Padmagirisan Sai Subramanian Rizki Ahmad Firdaus	a0101938@u.nus.edu a0099566@u.nus.edu	compg45
ac	Sherlyn Toh Hui Hsian Tamana Anna Tharakan	a0126288@u.nus.edu a0099584@u.nus.edu	compg46