

# Revisiting LRU-K Page Replacement Algorithm Performance Experiments

Victor Bittorf, Igor Canadi, Larry Li, Seth Pollen  
University of Wisconsin-Madison  
{bittorf, canadi, xli99, pollen}@cs.wisc.edu

## ABSTRACT

This paper revisits the LRU-K page replacement algorithm and shows that LRU-K is more adaptive and efficient with  $K > 1$ . The LRU-K method selects pages for replacement in a database buffer by keeping track of the last  $k$  references to popular database pages. It adapts to real-world workloads by introducing the idea of correlated reference period. We implemented LRU-1, LRU-2 and LRU-3 with three experiments. The first two experiments, we replicated the two-pool and zipfian workloads from O’Neil et al. [1] and ran these three algorithms on them, our results are consistent with theirs in that LRU-2 outperforms LRU-1 and the results of LRU-2 and LRU-3 converge with increasing buffer sizes. In the third experiment, we simulated LRU-K on a trace from PostgreSQL and reached the same conclusion as the first two with correlated reference period set properly.

## 1. INTRODUCTION

Database systems use memory buffers to retain some number of pages for a while after they’ve been read from the disk. That way, if the DBMS needs the same page again and it is in memory buffer, it can save one access to disk. The question is, when the buffer is full and we need to put another page in, which page should we drop from the buffer? One popular algorithm to determine the page to drop is Least Recently Used (LRU). With LRU, the buffer drops the page that was used least recently.

In 1993, O’Neil et al. [1] developed an extension to LRU algorithm called LRU-K. The basic idea of LRU-K is to use last  $K$  references of the page and drop the page with least recent  $K$ th reference in the past. With  $K = 1$ , the algorithm is equivalent to the classic LRU algorithm.

To better adapt LRU-K to real-world database workloads and address some of the unrealistic assumptions made by the initial algorithm, O’Neil et al. propose the Correlated Reference Period (CRP), which is a time-out period that dealt with two significant performance issues caused by LRU-K (known as the *early page replacement* and *correlated references*). Early page replacement occurs when a page has been accessed less than  $k$  times and thus has the largest backward distance at the time of choosing a replacement victim; the CRP method automatically retains this page if its last access time is within the time-out period. The problem of correlated references occurs when a page was accessed multiple times by the same query or process in a short span of time, the CRP method collapses all references to a single page within the time-out period into one reference.

In their paper, O’Neil et al. propose a hypothesis that

LRU-2 and LRU-3 outperform LRU-1. They also hypothesize that LRU-3 is only slightly better than LRU-2 and propose using LRU-2 because of the slower adaptation of LRU-3 to evolving access patterns. As a proxy to prove their hypotheses, they evaluated LRU-K on three different workloads - two pool, zipfian and OLTP trace.

We recreate their first two experiments - two pool and zipfian. As a third experiment, we use a trace from instrumented PostgreSQL running `pgbench`. Our results in all three experiments support both of their hypotheses.

We organize this paper as follows. In section 2 we discuss our experiment protocol and methodology. In section 3 we present our results and in section 4 we conclude.

## 2. PROTOCOL

At the beginning of an experiment, when the buffer is empty we’ll inevitably see a lot of cache misses until the buffer fills up. This is why in all our experiments we don’t account for first 1000 buffer accesses. After we process 1000 buffer accesses, we reset the counters and report hit ratio only on the remaining accesses.

As a result of our experiments, we measure how hit ratio depends on buffer size. If there is a total of  $T$  buffer references and the number of references that find the page in the buffer is  $h$ , we define the hit ratio as  $C = \frac{h}{T}$ .

Unless otherwise noted, we report the average of four runs of the experiment. We don’t plot error bars because they are too small. In all cases, they were smaller than 0.01.

O’Neil et al. [1] didn’t specify what Correlated Reference Period (CRP) they used in their experiments. When replicating their experiments, we found that we get results closest to theirs if we use  $CRP = 0$ .

We wrote the LRU-K simulation and evaluation in Python and plotted all the figures with R.

## 3. RESULTS

With a goal of recreating findings of O’Neil et al. [1] we evaluated LRU-1, LRU-2 and LRU-3 algorithms on three workloads:

- a synthetic workload simulating alternating references to two pools’ pages with different frequencies,
- a synthetic workload simulating random accesses according to Zipfian 80-20 distribution, and
- buffer accesses captured with instrumented PostgreSQL running `pgbench`.

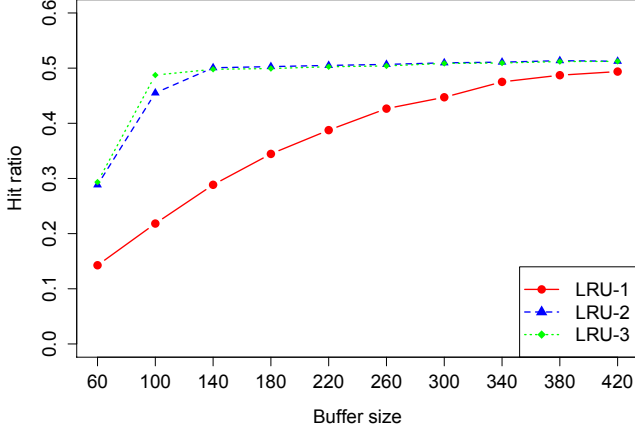


Figure 1: Simulation results of two-pool experiment with disk page pools of  $N_1 = 100$  pages and  $N_2 = 10000$  pages. The horizontal axis shows the simulate buffer size and the vertical axis shows measured hit ratio. All measurements were evaluated with Correlated Reference Period set to 0.

To prove our hypotheses that LRU-2 and LRU-3 outperform LRU-1 on database workloads, we expect our results to show higher hit ratio for all buffer sizes in LRU-2 and LRU-3 case. We also expect to see the difference in performance decrease as buffer size grows, since in the limit of infinite buffer size, they should all perform the same.

Our second hypothesis is that LRU-2 and LRU-3 performances are similar. We don't expect to see high differences in cache hit ratio in any workload.

We present results for these workloads in the following sections.

### 3.1 Two Pool Experiment

As our first experiment, we replicated the two-pool experiment conducted by O'Neil et al. [1]. They considered two pools of pages, Pool 1 with size  $N_1$  and Pool 2 with  $N_2$  pages and simulated random alternating accesses to those pools. With  $N_1 \ll N_2$ , the goal of this experiment was to simulate alternating accesses to index and record pages. Specifically, on every even access they uniformly choose one page from Pool 1 and on every odd access they choose one page from Pool 2. Thus, every page from Pool 1 has probability of reference  $\frac{1}{2N_1}$  and every page from Pool 2  $\frac{1}{2N_2}$ .

We replicated the experiment exactly, using  $N_1 = 100$  and  $N_2 = 10000$  pages. Figure 1 shows measured hit rates for specific buffer sizes. Our results are consistent with O'Neil et al. [1], where they found that LRU-1 has lower hit rates than LRU-2 and LRU-3 for a given buffer size. We also see that LRU-1 performance approaches LRU-2 and LRU-3 on larger buffer sizes. LRU-2 and LRU-3 performances are similar, with LRU-3 slightly outperforming LRU-2. However, O'Neil et al. argue that the difference is too small to be traded for the slower responsiveness which LRU-3 exhibits on evolving and dynamic access patterns.

### 3.2 Zipfian Random Access Experiment

Next, we evaluated LRU-K on workloads that access pages

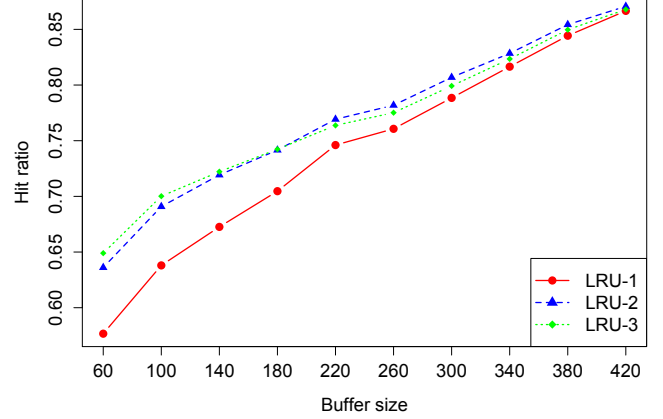


Figure 2: Simulation results of cache hit ratio for random access with Zipfian 80-20 distribution to  $N = 1000$  pages in the pool. The horizontal axis shows the simulated buffer size and the vertical axis shows measured hit ratio. All measurements were evaluated with Correlated Reference Period set to 0.

according to Zipfian distribution, replicating the experiment conducted by O'Neil et al. [1]. They used  $N = 1000$  pages numbered 1 through  $N$ ; the probability for referencing a page with page number less than or equal to  $i$  was  $\left(\frac{i}{N}\right)^{\frac{\log a}{\log b}}$  with constants  $a = 0.8$  and  $b = 0.2$ .

Their findings were that LRU-2 gains are lower than in the two-pool test, since the skew of Zipfian 80-20 distribution is milder than skew in the two-pool test. Figure 2 shows the hit ratios we measured for different buffer sizes. There are three main points we can take from this figure. First, LRU-2 and LRU-3 clearly outperform LRU-1, more so for smaller buffer sizes. Second, LRU-2 and LRU-3 performances are similar. Third, gains for LRU-2 are smaller than in two pool experiment.

Our first and third findings are consistent with O'Neil et al. They didn't evaluate LRU-3 against a Zipfian distribution, so we cannot compare our second finding.

### 3.3 PostgreSQL Trace Experiment

For their third experiment, O'Neil et al. [1] used a trace from the OLTP system of a large bank. We did not have access to this trace, so we could not replicate the experiment exactly. Instead, we instrumented PostgreSQL to capture all buffer accesses and ran `pgbench` benchmark with 400 clients, each issuing 10 transactions. This workload generated a total of 89692 accesses. However, because PostgreSQL's `elog` function isn't thread-safe, some parts of our trace were corrupted. As a result, we captured only 87373 accesses, which is 97.4% of total buffer accesses. We tried to make `elog` thread-safe, but that resulted in big slowdowns of the system, causing some transactions to be aborted.

We evaluated the LRU-1, LRU-2 and LRU-3 algorithms on the generated trace. For LRU-2 and LRU-3, we ran two experiments, one with Correlated Reference Period (CRP) set to be 0 and another with  $CRP = 20$ . With  $CRP = 20$ , we collapse any two consecutive references into one page ref-

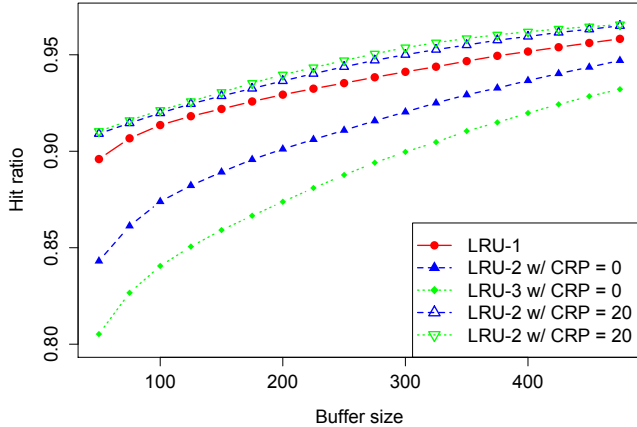


Figure 3: Simulation results of different buffer management strategies using a trace collected from PostgreSQL running **pgbench** benchmark. The horizontal axis shows the simulated buffer size and the vertical axis shows measured hit ratio. LRU-2 and LRU-3 were evaluated with Correlated Reference Period set to 0 and 20.

erence when there are no more than 19 other page references between them. Details of the Correlated Reference Period are discussed in [1]. Since we had only one trace, we ran the experiment only once and report the hit ratio of that run (unlike in previous experiments, where we reported average of four runs). Figure 3 shows the hit rates we measured for different buffer sizes. It is interesting to note is that LRU-2 and LRU-3 are inferior to LRU-1 with  $CRP = 0$ . However, with  $CRP = 20$  they both outperform LRU-1. Unfortunately, O’Neil et al. [1] never stated which CRP they used in their experiments nor suggested how to set suitable CRP under different workloads.

## 4. CONCLUSION

We evaluated LRU-K algorithm on three workloads: two-pool, zipfian and **pgbench** running on PostgreSQL.

We replicated two-pool and zipfian experiment from O’Neil et al. [1]. Our results are consistent with theirs, showing that LRU-2 outperforms LRU-1 and that there is no big difference between LRU-2 and LRU-3.

As an addition, we also evaluated LRU-K on a trace from PostgreSQL. Interestingly, we show LRU-2 and LRU-3 perform worse than LRU-1 with correlated reference period disabled. However, when we use the correlated reference period and set it to 20, we see much higher performance in LRU-2 and LRU-3 case. These results also prove that the correlated reference period is needed in real system, which O’Neil et al. [1] don’t show in their experiments.

Overall, we proved our hypothesis that performance benefits can be achieved using LRU-K for  $K > 1$ . We also showed that there is no big difference in performance for LRU-2 and LRU-3 for the workloads we measured. Since LRU-3 by its nature adapts to changes slower than LRU-2, we argue that LRU-2 should be preferred as buffer management algorithm.

## 5. REFERENCES

- [1] Elizabeth J. O’Neil, Patrick E. O’Neil, and Gerhard Weikum. The lru-k page replacement algorithm for database disk buffering. *SIGMOD Rec.*, 22(2):297–306, June 1993.