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

Abstract TODO Larry

1. INTRODUCTION

Database systems use memory buffer to retain some number of pages for a while after they've been read from the disk. That way, if they need the same page again and it is in memory buffer, they can save one access to the 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. Popular algorithm to determine the page to drop is Least Recently Used (LRU). With LRU, buffer drops the page that was used least recently.

In 1993, O'Niel 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 Kth reference in the past. With $K = 1$, the algorithm is equivalent to classic LRU algorithm.

As an extension to LRU-K, O'Niel et al. propose correlated reference period method. (TODO Larry - describe correlated reference period)

In their paper, O'Niel 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. They show that their hypotheses hold for three workloads - two pool, zipfian and OLTP trace.

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

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

2. METHODOLOGY

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 number of references that find the page in the buffer is h, we define hit ratio as $C = \frac{h}{T}$.

Unless otherwise noted, we report 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'Niel 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 LRU-K simulation and evaluation in Python and plotted all the figures with R.

3. RESULTS

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

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

We present results for these workloads in the following sections.

3.1 Two Pool Experiment

As our first experiment, we replicated two pool experiment conducted by O'Niel 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 $N_1 << N_2$, 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'Niel et al. [1], where they found that LRU-1 has lower hit rates than LRU-2 and LRU-3 for specific buffer size. We also see that LRU-1 performance approaches LRU-2 and LRU-3 on higher buffer sizes. LRU-2 and LRU-3 performances are similar, with LRU-3 slightly outperforming LRU-2. However, O'Niel et al. argues that the difference is too small to be traded for slower responsiveness that LRU-3 exhibits on evolving and dynamic access patterns.

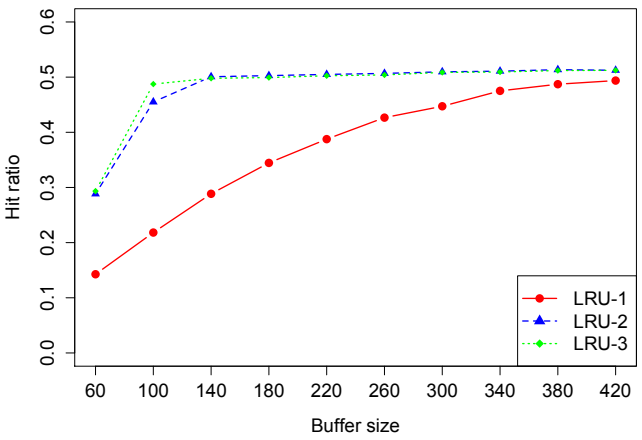

Figure 1: Simulation results of two pool experiment with disk page pools of $N_1 = 100$ pages and $N_2 = 10000$ pages. Horizontal axis shows the simulate buffer size and vertical axis shows measured hit ratio. All measurements were evaluated with Correlated Reference Period set to 0.

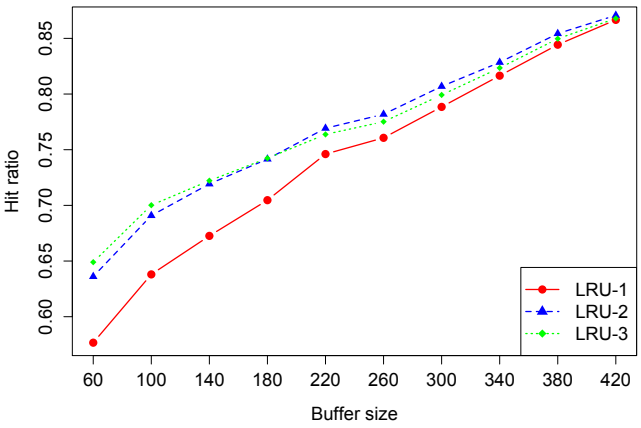

Figure 2: Simulation results of cache hit ratio for random access with Zipfian 80-20 distribution to $N = 1000$ pages in the pool. Horizontal axis shows the simulate buffer size and vertical axis shows measured hit ratio. All measurements were evaluated with Correlated Reference Period set to 0.

3.2 Zipfian Random Access Experiment

Next, we evaluated LRU-K on workloads that access pages according to Zipfian distribution, replicating the experiment conducted by O'Niel et al. [1]. They used $N = 1000$ pages numbered 1 through N; 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 two pool test, since the skew of Zipfian 80-20 distribution is milder than skew in two pool test. Figure 2 shows hit ratio 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.

First and third finding are consistent with O'Niel et al. They didn't evaluate LRU-3 on pool with Zipfian distribution, so we can not compare our second finding.

3.3 PostgreSQL Trace Experiment

For their third experiment, O'Niel et al. [1] used a trace from OLTP system of a large bank. We did not have access to the trace, so we could not replicate the experiment exactly. Instead, we instrumentalized 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 `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 caused big slowdowns of the system, which resulted in some of the transactions getting aborted.

We evaluated 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 mark two references as correlated when there are no more than 19 other page references between them. Details of 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 hit rates we measured for different buffer sizes. Interesting thing 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'Niel et al. [1] never stated which CRP they used in their experiments.

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'Niel 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 PosgreSQL. Interestingly, we show LRU-2 and LRU-3 perform worse than LRU-1 with correlated reference period disabled. However, when we use 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 correlated reference period is needed in real system, which O'Niel 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 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

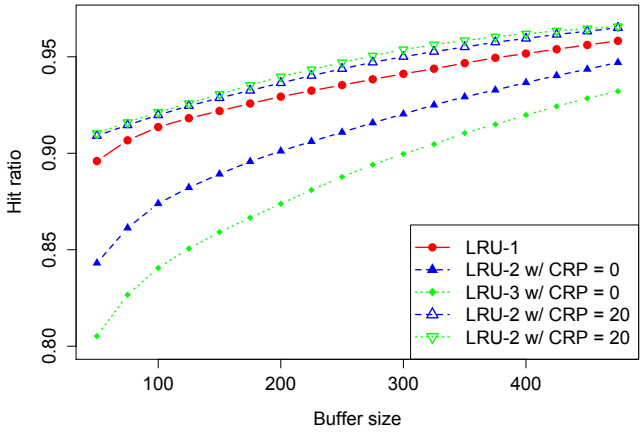

Figure 3: Simulation results of different buffer management strategies using a trace collected from PostgreSQL running `pgbench` benchmark. Horizontal axis shows the simulate buffer size and vertical axis shows measured hit ratio. LRU-2 and LRU-3 were evaluated with Correlated Reference Period set to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

[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.