

LFU- K : An Effective Buffer Management Replacement Algorithm¹⁾

Leonid B. Sokolinsky

Chelyabinsk State University
Br. Kashirinykh St. 129, Chelyabinsk 454021, Russia
sokolinsky@acm.org
<http://www.csu.ru/~sok/>

Abstract. This paper introduces a new approach to database disk buffering, called the **LFU- K** method. The **LFU- K** page replacement algorithm is an improvement to the Least Frequently Used (**LFU**) algorithm. The paper proposes a theoretical-probability model for formal description of **LFU- K** algorithm. Using this model we evaluate estimations for the **LFU- K** parameters. This paper also describes an implementation of **LFU-2** policy. As we demonstrate by trace-driven simulation experiments, the **LFU-2** algorithm provides significant improvement over conventional buffering algorithms for the shared-nothing database systems.

1 Introduction

Until the early 80's, **LRU** (*Least Recently Used*) buffer replacement algorithm was used almost in all cases. **LRU** algorithm replaces the page that was least recently accessed or used. It has been shown in Sleator and Tarjan [17] that **LRU** never replaces more than a factor k as many elements as an optimal clairvoyant algorithm **OPT**, where k is the size of the buffer. However, the further investigations showed, that **LRU** strategy does not always adequately take into account specificity of database systems [3, 16, 22]. As a result, intensive efforts have been undertaken in search of the replacement algorithms, which meet the requirements of database systems better [3, 8, 11, 13, 15, 16, 18].

One of the most impressive achievements in this direction is **LRU- K** algorithm proposed by Elizabeth O'Neil, Patrick O'Neil and Gerhard Weikum [13]. This algorithm is a generalization of **LRU** algorithm in the sense that **LRU-1** algorithm coincides with algorithm **LRU**. However, for $K \geq 2$, **LRU- K** provides significant improvement (up to 40%) over **LRU**, for the access patterns being peculiar to the relational database systems.

The basic idea of **LRU- K** is to keep track of the times of the last K references to popular database pages, using this information to statistically estimate the interarrival times of references on a page by the history basis. If P_1 page's K th most recent access is more recent than P_2 page's, then P_1 will be replaced after P_2 . For practical use the

¹⁾ Supported by the Russian Foundation for Basic Research under Grant 03-07-90031.

authors advocate the **LRU-2** method. For $K > 2$, the **LRU-K** algorithm provides a somewhat improved performance over **LRU-2** algorithm for stable patterns of access, but it is less responsive to changes in access patterns, which is an important consideration for database applications.

In [14], the authors proved optimality of **LRU-K** algorithm among all replacement algorithms that have the limited knowledge of the K most recent references to a page and no knowledge of future access. In [8], Johnson and Shasha proposed the **2Q** replacement algorithm that uses the same principles as **LRU-K**, but which is simpler and much faster to execute.

Algorithm **LRU-K** has initially been developed for the multitask environment providing simultaneous execution of multiple transactions in time-sliced mode. In such environment, distribution of the page reference probabilities has dynamic and hardly predicted character. Parallel database systems with shared memory and disks have the same access patterns. It occurs because all processors share the common buffer pool (the approach based on splitting the buffer pool on a number of processors leads to inefficient memory use [13]). However, quite different access patterns are peculiar for shared-nothing database systems. In such systems the workload profile is optimal when on each node of the system during each moment of the time, a few relational operators belonging to the same query are being executed. At such a workload, the probability distribution of page references will have static character during some time interval, which is equal to the execution period of the *query fragment* scheduled to this processor node. Then the new query fragment will be scheduled to this processor node that will cause instant replacement of one probability distribution by another. After that there again will be some period of stability. Under such conditions, **LRU-K** algorithm doesn't work quite adequately. Indeed, after changing the current probability distribution, **LRU-K** algorithm will be for a while determining the ratings of the pages by means of the old distribution. If the period of **LRU-K** adaptation to the new distribution is approximately equal to the period of query fragment execution, effectiveness of algorithm **LRU-K** will be low.

In this paper, we introduce a new replacement algorithm, called **LFU-K**. This algorithm is destined, first of all, for the parallel shared-nothing database systems. The **LFU-K** algorithm is a generalization of the **LFU** algorithm. In the case when reference probabilities remain constants, the **LFU** algorithm demonstrates the outstanding effectiveness approaching the optimal. However, the **LFU** algorithm becomes inadequate in the case when these probabilities vary with time. Indeed, if the page had a lot of references in the past, **LFU** will keep it in the buffer for some time irrespective of the presence of future references to the page. In contrast to **LFU**, **LFU-K** algorithm counts amount of references to the page in the reference string segment intercepted by the fixed parameter m (**LFU-0**). The counted value can be refined by taking into account the "*speed*" (**LFU-1**) and the "*acceleration*" (**LFU-2**) of growing the amount of references to the page. Simulation experiments will demonstrate in this article that the **LFU-K** algorithm does not concede on effectiveness to known algorithms on conventional traces and considerably outperforms them on traces like those produced by parallel shared-nothing database systems.

The remainder of this paper has the following outline. In Section 2, we present the basic concepts of the **LFU-K** approach to page replacement. In Section 3, we construct an analytical model, which provides a theoretical setting for parameter m of the

LFU-K algorithm. In Section 4, we present simulation performance results for **LFU-2** in comparison with **LFU**, **LRU**, and **LRU-2**. Section 5 deals with setting the **LFU-2** parameters. Finally, in Section 6 we summarize the major results of the paper and identify some directions for further research.

2 LFU-K Algorithm Definition

Assume we are given a set $\mathbf{N} = \{1, 2, \dots, N\}$ of disk pages, denoted by positive integers, and that the database system under study makes a succession of references to these pages specified by the *reference string*:

$$r_1, r_2, \dots, r_M, \quad (1)$$

where $r_t = i$ ($i \in \mathbf{N}$) means that the term numbered t in the reference string refers to disk page i . In the following discussion, we shall measure all *time* in terms of counts of successive page accesses in the reference string (t for r_t) rather than clock time. Moreover, we assume the time flowing *from larger indices to lesser* ones. So the next current moment of time could be denoted by zero index and the future moments of time could be denoted by negative indices in the reference string. In this context, M denotes the time interval elapsed from the instant of system start.

Let m be an integer number such that $1 \leq m \leq M$. Given a page $i \in \mathbf{N}$, we consider sequence

$$k_{i1}, k_{i2}, \dots, k_{im}. \quad (2)$$

Here $k_{ij} = \delta_{ir_j}$ for all j such that $1 \leq j \leq m$ (δ_{ir_j} is Kronecker's symbol [9]). Let $F_{il}(z)$ be the generating function for the subsequence k_{il}, \dots, k_{im} of the sequence (2) where $1 \leq l \leq m$. By the definition of generating function we have

$$F_{il}(z) = \sum_{j=l}^m k_{ij} z^j \quad (3)$$

Given an integer number h such that $1 \leq h \leq m$, we define

$$F_{il}^{[0]}(z) = F_{il}(z),$$

$$F_{il}^{[n]}(z) = F_{il}^{[n-1]}(z) - F_{i,l+h}^{[n-1]}(z)$$

for an arbitrary integer $n > 0$. Let us set

$$F_i^{[n]}(z) = F_{il}^{[n]}(z).$$

Let us designate $t = m/h$ ¹⁾. We define

¹⁾ Without loss of generality, we can suppose that m is always multiple of h .

$$\mathbf{R}_{\text{LFU-K}}(i) = \sum_{n=0}^K F_i^{[n]}(1) \frac{t^n}{n!} \quad (4)$$

for an arbitrary integer $K \geq 0$. In particular, we have

$$\mathbf{R}_{\text{LFU-0}}(i) = F_i(1) = F_i^{[0]}(1), \quad (5)$$

$$\mathbf{R}_{\text{LFU-1}}(i) = F_i^{[0]}(1) + F_i^{[1]}(1)t, \quad (6)$$

$$\mathbf{R}_{\text{LFU-2}}(i) = F_i^{[0]}(1) + F_i^{[1]}(1)t + F_i^{[2]}(1)\frac{t^2}{2}. \quad (7)$$

It is obvious that formula (5) counts the number of occurrences of page i in the sequence

$$r_1, r_2, \dots, r_m. \quad (8)$$

If we assume that the page reference probabilities are stable, formula (5) can be used for estimating the number of page i occurrences in the future sequence of references

$$r_{-m+1}, \dots, r_{-1}, r_0. \quad (9)$$

If the probability of reference to page i varies with time, formula (6) can give a more precise estimation for the number of page i occurrences in sequence (9) due to the second term, which includes as a factor the *velocity* of varying the reference frequency for page i . Respectively, formula (7) can provide even a more exact estimation due to the factor of *acceleration* in the third term if the factor of speed itself varies in time. Of course, the accuracy of estimation will be dependent on the parameters m and h .

In such a way, we are led to the generalized formula (4), which gives the following definition of the algorithm **LFU-K**.

Definition of LFU-K Algorithm. The **LFU-K** Algorithm specifies a page replacement policy when a buffer is needed for a new page being read in from disk: page i to be dropped (i.e., selected as a replacement victim) is the one whose $\mathbf{R}_{\text{LFU-K}}$ value is the minimum of all pages in the buffer pool. The only time the choice is ambiguous is when more than one page has the same $\mathbf{R}_{\text{LFU-K}}$ value. In this case, a subsidiary policy may be used to select a replacement victim among the pages with equal $\mathbf{R}_{\text{LFU-K}}$ value.

Note that **LFU-0** corresponds to the classical **LFU** algorithm in the case of $m = M$. However, for $K \geq 1$, **LFU-K** provides significant improvement (up to 45%) over **LFU** and other conventional algorithms for the access patterns being peculiar to the parallel shared-nothing database systems.

Effectiveness of the algorithm **LFU-K** depends strongly on values of the parameters m and h . In the general case, obtaining the analytical estimations for an optimal choice of the specified parameters is not a trivial problem. In the following section, we provide an analytical estimation of parameter m with various page reference probability distributions. The problem of selecting the optimal values for parameter h will be considered in Section 5.

3 Theoretical Setting for Parameter m

In the current section, we construct a probability model of page reference behavior in some abstract database system. Our model is based on the assumption that the access probabilities at different points in the reference string are independent. These assumptions are known as the independent reference model [4].

3.1 Probability Model

Let N be a number of the caching pages (i.e., $N = |\mathbf{N}|$). Define p_i as a reference probability of page i ($i \in \mathbf{N}$).

By the definition, we have $\sum_{i=1}^N p_i = 1$.

Let us assume that all probabilities p_i are stationary and are distributed according to the following law [10]:

$$p_i = \frac{1}{i^{\Theta} H_N^{(\Theta)}} \quad (10)$$

where $H_N^{(s)}$ is a harmonic number of order s (i.e., $H_N^{(s)} = 1^{-s} + 2^{-s} + \dots + N^{-s}$) and Θ is the *skew factor*, $0 \leq \Theta \leq 1$. Under $\Theta = 0$, we have $p_i = 1/N$ that corresponds to the uniform distribution (i.e., there is no skew). The case $\Theta = 1 - \log 0.80 / \log 0.20$ corresponds to the 80-20 distribution [7]. Under $\Theta = 1$, we have

$$p_i = \frac{1}{i^1 H_N^{(1)}} = \frac{1/H_N}{i}$$

that corresponds to the Zipf's distribution [24].

For harmonic numbers of order r , the following approximating formula is known [23]:

$$H_n^{(r)} = \zeta(r) + \frac{n^{1-r} - 1}{1-r} + \frac{1}{2n^r} - \sum_{k=1}^m \frac{B_{2k}}{2kn^{r+2k-1}} \binom{-r}{2k-1} + \mathcal{O}\left(\frac{1}{n^{r+2m+1}}\right) \quad (11)$$

where $\zeta(r)$ is Riemann's zeta function, and B_j is the Bernoulli number [9]. By formula (11), we can obtain the following simplified formula:

$$H_n^{(r)} = \zeta(r) + \frac{n^{1-r} - 1}{1-r} + \mathcal{O}\left(\frac{1}{n^r}\right). \quad (12)$$

Using (12), we can transform (10) to

$$p_i \approx \frac{1}{i^{\Theta} \left(\zeta(\Theta) + \frac{N^{1-\Theta} - 1}{1-\Theta} \right)}. \quad (13)$$

In particular, we have the following approximation for reference probability of the most “popular”, in our model, page 1:

$$p_1 \approx \frac{1}{\zeta(\Theta) + \frac{N^{1-\Theta} - 1}{1-\Theta}}. \quad (14)$$

Now we explore the influence of parameter m on the effectiveness of **LFU-K** in the frame of our model. First of all, note that $\lim_{m \rightarrow \infty} F_i^{[m]} = 0$ with $n > 0$ and $h = m$. Therefore, we can limit our exploration to the case of **LFU-0**. We have

$$\lim_{m \rightarrow \infty} \frac{R_{\text{LFU-0}}(i)}{m} = p_i.$$

This means that, as the m value increases, the effectiveness of **LFU-0** algorithm will approach the effectiveness of **A0** algorithm that replaces the page that is the one whose reference probability is the minimum of all pages in the buffer pool. It has been shown in [1] that the **A0** algorithm is optimal for stable probability distributions. It follows that increasing the value m will result in increasing the effectiveness of **LFU-0** algorithm under static probability distribution. However, for practical use, we need a measure, which would give us the relationship between parameter m and **LFU-K** effectiveness.

3.2 Measure for Parameter m

Using the proposed probability model and apparatus of generating functions we constructed in [21] the measure $\mathbf{M}(m)$, which gives us a relationship between parameter m and **LFU-K** effectiveness:

$$\mathbf{M}(m) = \int_0^{1/p} v_m(x) dx, \quad (15)$$

where $p = p_1$, $v_{im}(x) = e^{-(mp_i x + 1/2) \ln x + (m(1-p_i x) + 1/2) \ln \frac{1-p_i}{1-p_i x}}$, $v_{im}(0) = v_{im}(1/p_i) = 0$.

The smaller value \mathbf{M} we have, the greater accuracy of calculating the value of p_i we can attain by the formula

$$p_i = k/m, \quad (16)$$

where k is the number of page i occurrences in the sequence r_1, r_2, \dots, r_m .

However, the measure determined by formula (15) cannot be used in practice because it seems difficult to find the function that is inverse to $\mathbf{M}(m)$. We solved this problem by constructing the Taylor series representation of normal function $v_{im}(x)$ to obtain the rougher measure $\hat{\mathbf{M}}(m)$ that could be used in practice:

$$m > 2\Delta \left(\zeta(\Theta) + \frac{N^{1-\Theta} - 1}{1-\Theta} - 1 \right), \text{ where } \Delta = 700. \quad (17)$$

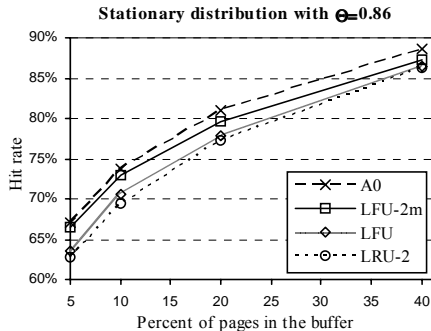


Fig. 1. Comparison of different replacement algorithms using static distribution with $\Theta=0.86$

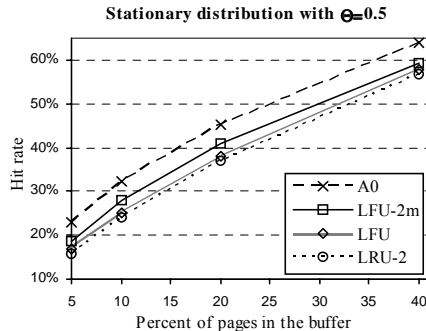


Fig. 2. Comparison of different replacement algorithms using static distribution with $\Theta=0.5$

4 Experimental Results

This section describes the trace-driven simulation of page buffering and the results obtained from our analysis. A program was written to simulate the buffer manager. The simulator reads a trace file consisting of trace records, and simulates the actions of a page buffer according to the trace record.

Besides the **LFU-2** algorithm, four other replacement algorithms, namely **LFU**, **LRU** [5, 12], **LRU-2** [13] and **2Q** [8], are simulated for comparison purposes. There is no **2Q** algorithm in figures below because **2Q** demonstrates the effectiveness, which is very close to that of **LRU-2**, in all the experiments. As a reference point, we use the **OPT** algorithm [2, 12], which replaces the page with the longest future reference distance, and the **A0** algorithm, which replaces the page with the lowest reference probability [4].

4.1 Stationary Reference Probability Distribution

The first set of experiments investigates the effectiveness of **LFU-2** on artificial traces with static reference probability distribution defined by formula (10) with $\Theta=0.86$ (80-20 rule) and $\Theta=0.5$ (45-20 rule). In both cases, we generated 1000,000 references to $N=32000$ pages (numbered 1 through N). The results of these experiments are presented in **Fig. 1** and **Fig. 2**. As a reference point, we included the hit rate of the **A0** algorithm which is known to be optimal for stable probability distributions [4]. The **LRU** algorithm is missing in these experiments because it is known (see, e.g., [8]) that **LRU** always performs worse than **LRU-2** on the traces with the static distribution (10).

The first experiment (**Fig. 1**) exploits the parameter $\Theta=0.86$ that approximately corresponds to the 80-20 rule (80% of the references accesses 20% of the database).

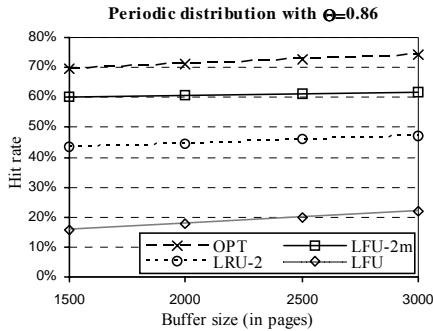


Fig. 3. Comparison of different replacement algorithms using periodic distribution with $\Theta=0.86$

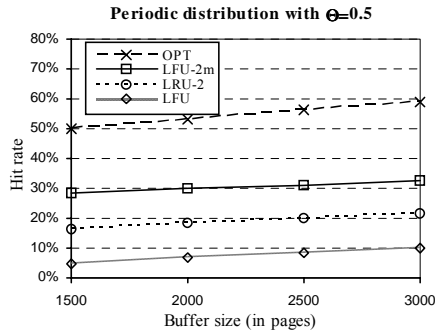


Fig. 4. Comparison of different replacement algorithms using periodic distribution with $\Theta=0.5$

The chart shows that **LFU-2** provides substantial improvement over **LFU** and **LRU-2**.

The second experiment (Fig. 2) exploits the parameter $\Theta = 0.5$ that approximately corresponds to the 45-20 rule (45% of the references accesses 20% of the database). The chart shows that **LFU-2** provides perceptible improvement over **LFU** and **LRU-2**. At the same time we have to notice that this improvement is less on a less skewed distribution. These experiments show that the hit rate of **LFU-2** is close to one of **A0** for small values of the buffer size with the static reference probability distributions.

4.2 Periodic Reference Probability Distribution

The second set of experiments investigates the effectiveness of **LFU-2** on the traces, which are typical of a shared-nothing database system. As mentioned in Section 1, shared-nothing processor node generates a trace consisting of intervals with relatively stable page reference probabilities, which are interleaved by very short intervals of a momentary changing of the probability distribution.

In accordance to this, we generated an artificial trace with the *periodic distribution*, which consists of 1000 equal-sized segments. Each segment consists of 1000 references which obey (10). Each page has a single stationary reference probability within the segment at any point in time. However, the reference probability of the same disk page is varied from one segment to another. As a reference point, we included the hit rate of the clairvoyant algorithm **OPT**, which is known to be theoretically optimal [2, 12].

We ran two experiments, each of which had a database size of 32000. In the first experiment (Fig. 3), we used the distribution (10) with the parameter $\Theta = 0.86$. In the second experiment (Fig. 4) we used the distribution (10) with the parameter $\Theta = 0.5$. These charts show that **LFU-2** provides significant improvement over **LRU-2** (up to 15%). The performance improvement is highest when the buffer pool is small. The hit rate of **LFU-2** is close to the optimal for small values of the buffer size. We also

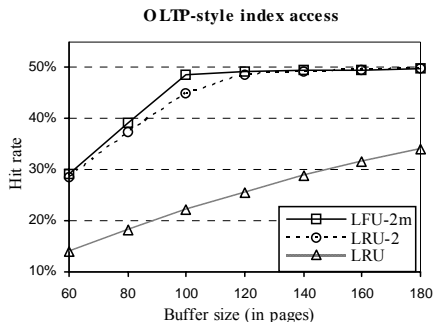


Fig. 5. Comparison of different replacement algorithms on index access to data pages

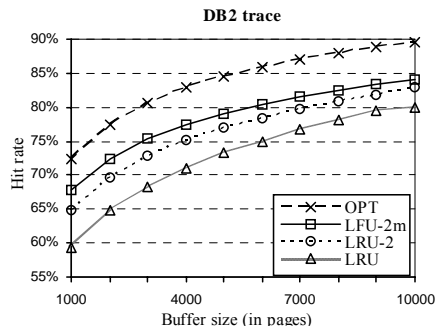


Fig. 6. Comparison of different replacement algorithms on DB2 trace

note from these experiments that the **LFU** algorithm demonstrates much worse results with periodic distribution, compared to the experiments with static distribution.

4.3 OLTP-style Index Accesses

An important observation of paper [13] is that **LRU** is unable to differentiate between pages that have relatively frequent references and pages that have very infrequent references until the system has wasted a lot of resources keeping infrequently referenced pages in memory for an extended period. The authors posited an example where there are 100 index pages and 10000 data pages. References come in pairs, one page selected uniformly at random from the set of index pages and one selected uniformly at random from the set of data pages. They showed that **LRU-2** gives priority to the index pages as it should, since their hit rate with 100 page slots is almost 50%. We ran a simulation with identical parameters, and we found that **LFU-2** also gives preference to index pages (**Fig. 5**).

4.4 Experiments on Real Data

The final experiment of this section was based on the real trace that came from a commercial application of DB2. We received a trace file containing 1000,000 references to 52701 unique pages. The file was fed to our buffer management simulator. As a reference point, we included the hit rate of the theoretically optimal algorithm **OPT**. The results in **Fig. 6** show that **LFU-2** outperforms both the **LRU** and **LRU-2** algorithms. The hit rate of **LFU-2** is close to the optimal for small values of the buffer size.

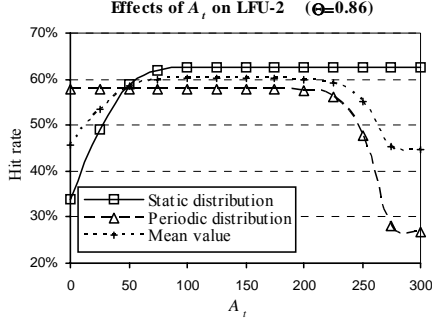


Fig. 7. Effects of A_t on the LFU-2 algorithm using periodic (with period of 1000) and static distributions with $\Theta=0.86$ and buffer of 1000 pages

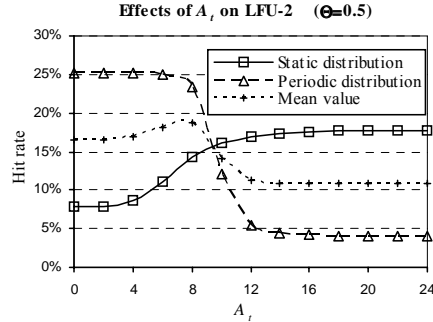


Fig. 8. Effects of A_t on the LFU-2 algorithm using periodic (with period of 1000) and static distributions with $\Theta=0.5$ and buffer of 1000 pages

5 Setting the Parameters

In Section 3, we obtained a theoretical result which shows, that the value $m = 30,000$ is acceptable for the 80-20 distribution, while the value $m = 500,000$ is acceptable for the 45-20 distribution. Obtaining theoretical estimation for the other parameters of **LFU-2** is a non-trivial problem. To gain intuition about the **LFU-2** sensitivity to various values of h and A_t , we performed the following experiments.

To investigate the responsiveness of **LFU-2** to variations of the A_t value, we ran this algorithm on the stationary and periodic artificial traces described in sections 4.1 and 4.2. The results of these experiments are shown in **Fig. 7** and **Fig. 8**. For the stationary distribution, the hit rate of **LFU-2** goes up as the A_t value increases. For the periodic distribution, the hit rate of **LFU-2** goes down as the A_t value increases. This is to be expected because, increasing the A_t value, we smooth the *stochastic disturbances* which decrease the **LFU-2** efficiency for a stationary distribution. However, in the case of periodic distribution, the increase of A_t value results in smoothing even the *non-stochastic* fluctuations caused by the change of probability distribution. As a result, the **LFU-2** performance degrades. In accordance with this, we have to find a trade-off. For $\Theta = 0.86$ (**Fig. 7**), we can see that the trade-off is $A_t = 100$, while for $\Theta = 0.5$ (**Fig. 8**) trade-off is $A_t = 8$.

In the next experiment, we investigated the responsiveness of **LFU-2** to variations of the h value. The results of this experiment are shown in **Fig. 9**. For the stationary distribution, the hit rate of **LFU-2** goes down as the h value increases. This takes place because, increasing h value, we increase the number of stochastic disturbances overcoming the boundary A_t of non-zero accelerations, which decrease the effectiveness of the **LFU-2** algorithm. For the periodic distribution, the hit rate of **LFU-2** initially increases to its maximum at $h = 3000$ and then decreases slowly as the h value increases. The analysis of the chart shows that trade-off is $h = 3000$.

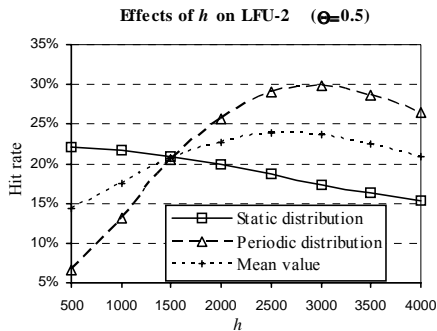


Fig. 9. Effects of h on the LFU-2 algorithm using periodic (with period of 1000) and static distributions with $\Theta=0.5$ and buffer of 1600 pages

6 Conclusion

In this paper we introduced a new database buffering algorithm named **LFU-K**. Our simulation results provide evidence that the **LFU-K** algorithm has significant performance advantages over conventional algorithms for the shared-nothing database systems. The **LFU-2** algorithm was integrated into Omega parallel database system running on the MBC multi-processor [19, 20]. We believe, however, that the potential usage of our algorithm may be even wider. It is interesting to investigate an applicability of the **LFU-K** algorithm to the Web caching.

We also developed an analytic reference model related to the **LFU-K** algorithm. Using this model, we derived an analytical estimation of parameter m with various page reference probability distributions. There are at least two directions of future research suggested by this work. First, it is useful to derive the analytical estimations of the parameters h и A_i with various page reference probability distributions. Second, a proof of the optimality of **LFU-K** among all replacement algorithms which have not more knowledge of a trace than **LFU-K** has seems to be a solvable problem.

References

1. Aho A.V., Denning P.J., Ullman J.D. Principles of Optimal Page Replacement // Journal of the ACM. 1971. Vol. 18. No. 1. P. 80-93.
2. Belady L.A. A Study of Replacement Algorithms for Virtual-Storage Computer // IBM Systems Journal. 1966. Vol. 5. No. 2. P. 78-101.
3. Chou H.-T., DeWitt D.J. An Evaluation of Buffer Management Strategies for Relational Database Systems // VLDB'85, Proceedings of 11th International Conference on Very Large Data Bases, August 21-23, 1985, Stockholm, Sweden. Morgan Kaufmann. 1985. P. 127-141.
4. Coffman E.G., Denning P.J. Operating Systems Theory. Prentice-Hall, 1973.
5. Effelsberg W., Haerder T. Principles of Database Buffer Management // ACM Trans. on Database Systems. Dec. 1984. Vol. 9. No. 4. P. 560-595.
6. Gray J., Graefe G. The Five-Minute Rule Ten Years Later, and Other Computer Storage Rules of Thumb // SIGMOD Record. 1997. Vol. 26. No. 4. P. 63-68.
7. Heising W.P. Note on Random Addressing Techniques // IBM Systems Journal. 1963. Vol. 2. No. 2. P. 112-116.
8. Johnson T., Shasha D. 2Q: A Low Overhead High Performance Buffer Management Replacement Algorithm // VLDB'94, Proceedings of 20th International Conference on Very Large Data Bases, September 12-15, 1994, Santiago de Chile, Chile. Morgan Kaufmann. 1994. P. 439-450.

9. *Knuth D.E.* Art of Computer Programming, Volume 1: Fundamental Algorithms (3rd Edition). Addison-Wesley, 1997. 700 p.
10. *Knuth D.E.* Art of Computer Programming, Volume 3: Sorting and Searching (2nd Edition). Addison-Wesley, 1998. 780 p.
11. *Lee D., Choi J., Kim J.-H., Noh S.H., Min S.L., Cho Y., Kim C.-S.* On the existence of a spectrum of policies that subsumes the least recently used (LRU) and least frequently used (LFU) policies // SIGMETRICS '99, International Conference on Measurement and Modeling of Computer Systems, May 1-4, 1999, Atlanta, Georgia, USA, Proceedings. 1999. P. 134-143.
12. *Mattson R.L., et al.* Evaluation techniques for storage hierarchies // IBM Systems Journal. 1970. Vol. 9. No. 2. P. 78-117.
13. *O'Neil E.J., O'Neil P.E., Weikum G.* The LRU-K Page Replacement Algorithm For Database Disk Buffering // Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, Washington, D.C., May 26-28, 1993. ACM Press. 1993. P. 297-306.
14. *O'Neil E.J., O'Neil P.E., Weikum G.* An optimality proof of the LRU-K page replacement algorithm // Journal of the ACM. 1999. Vol. 46. No. 1. P. 92-112.
15. *Robinson J.T., Devarakonda M.V.* Data Cache Management Using Frequency-Based Replacement // 1990 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems, University of Colorado, Boulder, Colorado, USA, May 22-25, 1990, Proceedings. 1990. P. 134-142.
16. *Sacco G. M., Schkolnick M.* Buffer management in relational database systems // ACM Transactions on Database Systems (TODS). Dec. 1986. Vol. 11. No. 4. P. 473-498
17. *Sleator D.D., Tarjan R.E.* Amortized efficiency of list update and paging rules // Communications of the ACM. 1985. Vol. 28. No. 2. P. 202-208.
18. *Smaragdakis Y., Kaplan S., Wilson P.R.* EELRU: Simple and Effective Adaptive Page Replacement // SIGMETRICS '99, International Conference on Measurement and Modeling of Computer Systems, May 1-4, 1999, Atlanta, Georgia, USA, Proceedings. 1999. P. 122-133.
19. *Sokolinsky L.B.* Design and Evaluation of Database Multiprocessor Architecture with High Data Availability // 12th International DEXA Workshop, Munich, Germany, 3-7 September, 2001, Proceedings. IEEE Computer Society. 2001. P. 115-120.
20. *Sokolinsky L.B.* Organization of Parallel Query Processing in Multiprocessor Database Machines with Hierarchical Architecture // Programming and Computer Software. 2001. Vol. 27. No. 6. P. 297-308.
21. *Sokolinsky L.B.* Page Replacement Algorithm for Buffer Management in the Omega Parallel Database System. Technical report OMEGA08. Chelyabinsk State University, 2003. [<http://www.csu.ru/~sok/papers/sources/omega08.pdf>]
22. *Stonebraker M.* Operating System Support for Database Management // Communications of the ACM. 1981. Vol. 24. No. 7. P. 412-418.
23. *Titchmarsh E.C.* The theory of the Riemann zeta-function; second edition. Oxford University Press, 1951. 346 p.
24. *Zipf G.K.* Human Behavior and the Principle of Least Effort: an Introduction to Human Ecology. Reading, MA, Addison-Wesley, 1949.