INDEX ACCESS WITH A FINITE BUFFER

Giovanni Maria Sacco
Dipartimento di Informatica
Universitá di Torino
Torino, Italy

ABSTRACT:

A buffer is a main-memory area used to reduce access to disks. The buffer holds pages from secondary storage files. A process requesting a page causes a fault if the page is not in the buffer: the requested page is read into the buffer. If no buffer space is available, a page in the buffer is replaced by the requested one.

The solution of many relational queries (e.g. joins) require the repeated access of a relation through a unique clustered index. The fault rate of such queries as a function of the available buffer size is analyzed. A B-tree structure is assumed, but the results presented here carry over to most other hierarchical index structures.

It is shown that the LRU replacement strategy, commonly used with this type of access, is not the best strategy. Two alternative strategies, ILRU and OLRU, are proposed. ILRU is shown to be always better than LRU, especially for small buffer sizes and independently of the probability of page references. OLRU is proved to be optimal under the assumption of a uniform distribution of page reference densities. The behaviour of LRU and OLRU under distributions that violate this assumption (such as Zipfian distributions) is discussed.

1. INTRODUCTION

Most database systems use a main-memory area as a cache buffer, to reduce accesses to disks. This area is subdivided into *frames*, and each frame can contain a *page* from a secondary storage file. A process requesting a page will cause a *fault* if the page is not in the buffer: the requested page is then read into an available buffer frame (demand paging). When no available frames exist, a frame is made available by a replacement policy. Its contents are copied, if required, back to the disk.

The problem of managing a buffer is analogous to the problem of managing the real memory in a virtual memory system, and, in fact, most of the replacement policies currently used for database systems are the same policies used for virtual memory. The most popular replacement policy is LRU (least recently used page), which replaces the page which has not been referenced for the longest time; other strategies are discussed in [EFFE84].

In buffered environments, the number of physical disk accesses is not a constant (as assumed by most previous analyses), but it is a function of the available buffer size. In practice, previous analyses have modelled logical reference strings. If buffering is used, the number of physical references (disk accesses) is generally significantly lower than the number of logical references.

Performance models based on buffering are important to design and tune applications, and especially so when a query optimizer is used to discriminate among different evaluation strategies: ignoring the effect of buffering can lead to seriously wrong choices [SACC86]. They are essential for predictive buffer management strategies, such as the hot set model [SACC82, SACC83, SACC86] and the query locality set model [CHOU85], which rely on buffering

models to predict the buffer size needed by a query before it is executed. These strategies are used to prevent thrashing in the buffer.

The problem addressed in this paper is the modelling of the fault rate of a process repeatedly accessing a relation through a clustered index. An index is called clustered when the data pages are physically ordered in the same order as the index entries. When this property does not hold, the index is called unclustered. Clustered indices are almost always used to cluster a relation over its primary key. For this reason, the present analysis assumes a *unique clustered index* (i.e. no duplication of index keys). A B+tree [BAYE72, COME79] index structure is assumed: however, the analysis can be easily extended to most other hierarchical index structures.

Repeated access to an index (called *index reusal* in [SACC86]) can arise in two basic cases:

(1) a join in which the indexed relation is inner: the relation is accessed through the index for each qualifying tuple in the outer relation;

(2) several users randomly accessing the indexed relation: examples of this situation are bank teller and point-of-sale applications.

The contributions of this paper are:

(1) a modification of LRU replacement (ILRU) which improves the fault rate especially for small buffer sizes, under any distribution of page reference densities;

(2) a strategy (OLRU) which is proved to be optimal under the assumption of a uniform reference density for data pages. The effect of non-uniform distributions is discussed.

This paper is organized as follows. In section 2 previous research is reviewed. In section 3 optimality principles are discussed. Section 4 shows the suboptimality of LRU, and introduces ILRU. Section 5 describes OLRU which is then proved to be optimal. OLRU and ILRU are compared in section 6; a discussion of non-uniform reference densities is given in section 7.

2. PREVIOUS RESEARCH

The fact that LRU replacement significantly reduces the number of faults for index reusal of B-tree structures has been known for some time [BAYE72, KNUT73]. However, the effect of buffering has been ignored by most previous research, and the I/O cost of an index access has been assumed to be equal to the number of levels in the index, including the data level.

The first systematic model of buffered access is the *hot set model* [SACC82, SACC83, SACC86]. The hot set model is based on the empirical observation that the fault rate of a query as a function of the available buffer space is a stable curve, with a small number of discontinuities. The interval in which a discontinuity occurs is called an unstable interval. The left extremum of an unstable interval (i.e. the one corresponding to the smaller buffer size) is called a *cold point*, the right extremum is called a *hot point*. The fault rate inside an unstable interval is estimated by linear interpolation between the fault rate at the cold point, and the fault rate at the hot point.

The optimal buffer size for a given query (called the *query hot set size*), is the largest hot point not exceeding the available buffer space. In the case in which the available buffer space falls within an unstable interval, the query hot set size is the available buffer space. This definition minimizes the number of faults of the query.

Access to a unique clustered index was discussed in [SACC86]: estimation formulas linearly interpolate the behaviour of LRU from the buffer size at which the root of the index is guaranteed to be preserved in memory, to the point in which all the touched pages can be guaranteed to be kept in the buffer. This usually results in a gross overestimation of the fault rate of the query in the interpolated region.

Although the hot set model was originally defined in terms of LRU replacement, it can be extended to most other replacement strategies, and even to the case in which each relation in the evaluation plan is independently managed [SACC86], as in the present paper. The latter case was studied in detail by Chou and DeWitt, in their query locality set model [CHOU85].

3. PRINCIPLES OF OPTIMALITY FOR BUFFER MANAGEMENT

Assume that a reference string S of length ω referencing a set of pages $\{p_i\}$ of cardinality P, and a buffer of b frames are given. When $b \geq P$, the buffer can contain all the pages to be referenced, so that no page needs replacement, and demand access is optimal.

When $b < P$, at least one page will need to be replaced. A general optimality principle for buffer management, called algorithm B_0, was found by Mattson et al. [MATT70] for virtual memory systems. Algorithm B_0 replaces the page in the buffer with the maximum forward reference distance. That is, when a page must be replaced, the reference string is scanned from the current point to the end, and page p_j in the buffer whose distance to the next reference is maximum is replaced (if p_j is not referenced again, its forward distance is infinity).

Algorithm B_0 requires the detailed knowledge of the entire reference string, and consequently is not practically implementable. It represents a lower bound on the fault rate of a process. The *independent reference model* (IRM), proposed by Franaszek and Wagner [FRAN74], does not require such a detailed knowledge of the reference string:

Let the reference string S $r_1, \ldots, r_j, \ldots$, be a sequence of independent random variables with a given common stationary reference density distribution $\{p(p_1), \ldots, p(p_P)\}$ such that $Pr[r_i=p_i] = p(p_i)$, for all $j>1$. The forward distance to p_i just after r_j, $d_j(p_i)$, is therefore a random variable with the stationary geometric distribution

$$Pr[d_j(p_i)=k] = p(p_i)\,(1-p(p_i))^{k-1}, \quad k=1,2,...$$

with mean values $1 / p(p_i)$, i.e. the expected distance between two consecutive references to p_i (the *interreference distance*) is maximum in S.

Given a buffer of $b < P$ frames, the following policy can be easily proved by induction to be optimal:

(1) order the elements of $\{p_i\}$ by decreasing reference densities $p(p_i)$. Ties are arbitrarily solved;

(2) partition the ordered set $\{p_i\}$ into two subsets:
HI: $\{p_i \mid 1 \le i \le b-1\}$ and LO: $\{p_i \mid b \le i \le P\}$.

(3) allocate $b-1$ frames to the set HI, i.e. one frame for each page in HI;

(4) allocate 1 frame to set LO;

A reference to a page in HI causes no replacement; a reference to a page in LO causes the replacement of the page contained in the frame allocated to LO (no page reusal is possible for any p_i in LO).

The independent reference model was originally formulated as a tool to obtain bounds on the deviations of replacement strategies from optimality in virtual memory systems, where reference densities are usually unknown. It will be shown in the following that, at least for hierarchical structures such as B+trees, the distribution of reference densities can be estimated under reasonable assumptions. Consequently, practically implementable strategies can be derived on this basis.

The LRU policy is now interpreted in the present framework. LRU has no knowledge of reference densities, and tries to estimate them on the basis of past history. LRU orders pages by increasing distance from the last reference. The idea is that this order models the order among average interreference distances in the entire reference string. By capturing the pages with the lowest distance from the last reference, LRU tries to capture the pages with the lowest interreference distance, hence the pages with the highest reference density.

4. INDEXED ACCESS

4.1. Assumptions

A unique clustered index is assumed. When the access to the indexed relation is performed within a join, the indexed relation is assumed to be inner. The outer relation is ignored in the following; in practice it will be characterized independently, and assigned an independent region of the buffer. This scenario also models the situation in which different users access the indexed relation.

The quantity K (number of key values to be accessed in the indexed relation) is assumed to be known. This quantity is usually estimated by the query optimizer. Note that $K = 1$ equals to a selection operation on the indexed relation. Thus, selections can be seen as a degenerate case of a join, and will not be explicitly considered.

The index is assumed to be stored as a hierarchical structure on L levels. Level 1 corresponds to the root of the index; data records are all stored at level L. In order to locate a data record, a complete index tree traversal must be performed, from a page at the root level to a page at level L.

Let P_i denote the number of pages at level i: it is assumed that $P_i \le P_j$, for all $i < j$.

Following a request for K data records, T_i pages out of P_i are accessed at each level i. T_i is estimated by Yao's function [YAO77]: $T_i=Y(K,P_i)$. Approximations to Yao's function can be found in [BERN81, WHAN83, IJBE85]. P denotes in the following the total number of pages in the index (leaf level included), T denotes the total number of pages accessed.

B+trees satisfy all the assumptions stated above, and the following discussion will focus on them.

4.2. LRU access to B+trees

The pattern of access is a traversal of the tree from the root level (*lowest level*), consisting of a single page, down to the data level (*highest level*): for each logical access to a data page, L logical page references are generated.

Let p be the father page in the index of a set of sons s. In order to access a son s_i in s, p must be accessed first: the sum of the reference densities for s is equal to the reference density of p. This implies that the reference density of a son s_i of p cannot exceed the reference density of p. Consequently it is always suboptimal to replace p, if any of its sons are still in the buffer.

It turns out that LRU violates this property. Consider a reference to a data page. This causes L logical references to pages; at the end, the first L positions in LRU stack will be ordered by the inverse order of reference, with the data page being at the top, its father in the second position, and the root at the L-th position. Pages will be scheduled for replacement in this order: consequently when $b < T$, at least 1 frame, and as many as $L-1$ frames will be wasted. In particular, the root is not kept in the buffer, unless $b \geq L$.

This specific problem can be easily corrected by **ILRU** ("inverse" LRU), a modified LRU policy. When a page $p_{i,j}$ at level i is accessed, it is not placed at the top of the LRU stack, but at the i-th position of the LRU stack: the root will be placed at position 1 (top of LRU), a data page at the L-th position. If the LRU stack has $b < L$ elements, the currently referenced page at level i ($i \geq b$) is placed at level b. Pages to be replaced are always taken from the last element (b) of the LRU stack.

ILRU simply reverses the order in which pages are scheduled for replacement, in such a way that sons are replaced before their fathers are. As a noteworthy consequence, a buffer $b \geq 2$ is sufficient to keep the root in the buffer, thereby decreasing the total number of faults by K.

ILRU is guaranteed to be always no worse than LRU, since no wasted frames occur. For higher levels of the tree, the advantages of ILRU over LRU tend to decrease. When $b \geq T$, the two policies are obviously equivalent.

5. OPTIMAL REPLACEMENT

Although ILRU gives an improvement over standard LRU replacement, it is not necessarily optimal under the independent reference model. Optimal buffer allocation under IRM is now discussed. A uniform reference density for leaf (i.e. data) pages is initially assumed.

The current assumptions imply that the reference density for any page $p_{i,j}$ at level j is uniform, and it is equal to

$$p(p_{i,j}) = \frac{1}{LT_j} = \frac{1}{LY(K,P_j)}$$

The number of pages at level j is by definition no smaller than the number of pages at level i ($i < j$). For B+trees it is given by $P_j \simeq F^{j-1}$ where F is the index fanout. By the definition of Yao's function,

$$T_i = Y(K,P_i) \leq T_{i+1} = Y(K,P_{i+1})$$

Consequently, the set $\{p_{i,j} \mid 1 \leq j \leq L, 1 \leq i \leq T_j\}$ of pages ordered by decreasing reference densities is obtained by ordering all the touched pages by level (with pages within a level being arbitrarily ordered).

The optimal strategy (**OLRU**) follows. The addressing space is logically partitioned into L independent regions, each managed by a local LRU chain, in such a way that region j contains pages from the j-th level in the index. The size of region j is T_j pages.

When $b \geq T$, all regions can be given a full allocation, and no replacement occurs. When $b < T$, they cannot. Buffer frames are allocated to a region i iff each region j ($1 \leq j < i$) has been allocated a number of frames equal to its size. If $j < L$, and the available buffer is $b' \leq T_j$ frames, j will be allocated no more than $b'-1$ frames. Otherwise, T_j frames will be allocated to j. This means that, unless region L can be allocated at least one frame, one unallocated frame exists in the buffer.

Figure 1 - Estimated faults per record access at different values of K. Index parameters: L=5, P_5=500, F=5

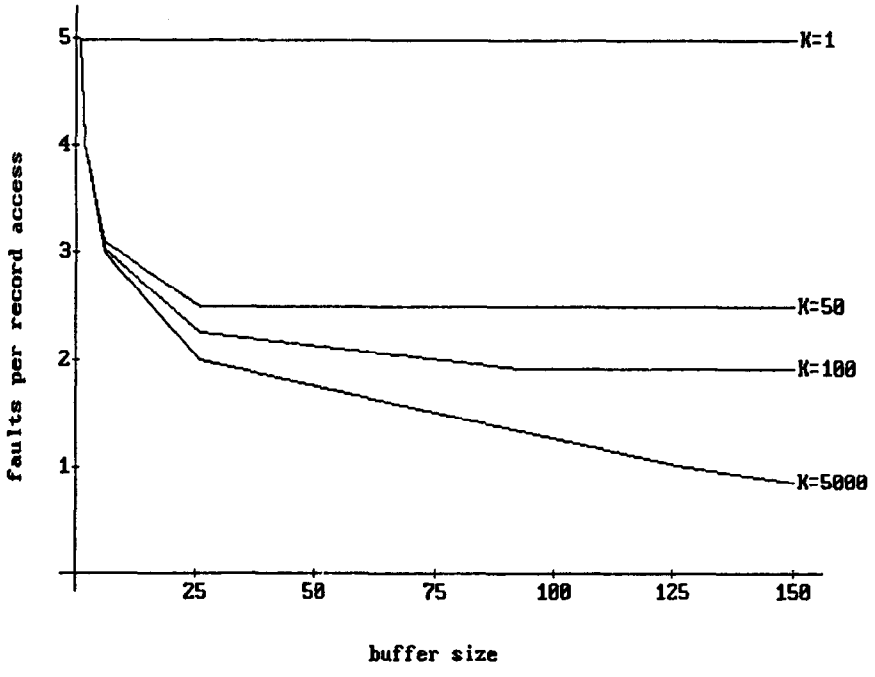
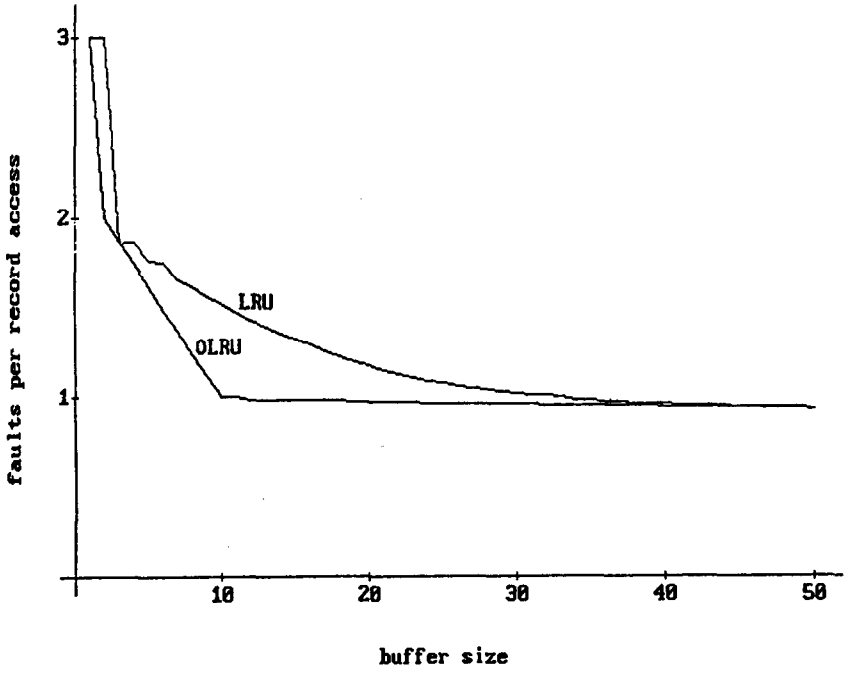


Figure 2 - LRU vs. OLRU. Uniform reference density at leaf level. Access parameters: K=5000, L=3, P_3=800, F=100



In general, for a given buffer size the first regions will be allocated their size (these regions are called *non-deficient*), no more than one region will be allocated a number of frames smaller than its size (this region is called *deficient*). Some regions might not be allocated any frame. These regions are called *cohalesced* and they share the single free frame in the buffer: that is, all the cohalesced regions will be independently managed on the single unallocated frame.

The following holds. Reusal of pages in non-deficient regions is total, reusal for deficient regions is partial, reusal for cohalesced regions is null. It is straightforward to show that, under the assumptions, this buffer allocation is indeed optimal, because frames will be allocated to pages according to their reference density.

In the optimal strategy for the independent reference model, described in section 3, buffer frames are assigned to specific pages. Under the current assumptions, the number of pages accessed at a given index level can be estimated, but the specific pages accessed are not known. OLRU therefore uses LRU replacement to manage pages. It is easy to show that the two strategies produce, under the current assumptions, the same number of faults.

LRU replacement for managing a level is not strictly necessary under the assumptions: a random replacement policy would perform equally well. LRU is chosen because, as discussed before, it is an estimator of reference densities, and consequently tends to decrease the fault rate when densities for pages at a given level deviate from uniformity. LRU also diminishes the effects of errors in the estimation of the number of accessed pages.

The characterization of the fault curve is straightforward. The curve has $L+1$ hot points and cold points:

$$hp_0 = cp_0 = 1, \quad faults(hp_0) = KL$$

$$hp_i = cp_i = \sum_{j=1}^{i} T_j + 1, \quad 1 \le i < L$$

$$hp_L = cp_L = \sum_{j=1}^{L} T_j = T$$

$$faults(hp_i) = \sum_{j=1}^{i} T_j + K(L-i), \quad i > 0$$

Hot points occur in the presence of a local minimum, i.e. when a level becomes non-deficient. At hot point i, levels j ($j \le 1$) are non-deficient: consequently all touched pages T_j are in the buffer and no further accesses are required. All other levels j ($j > i$) are generally cohalesced (no reusal is possible). For hot point hp_{L-1}, only level L is cohalesced, and is therefore deficient (with a single frame allocation): the probability of reusal is larger than 0, but it is, in practice, negligible.

Between cp_{i-1} and hp_i the curve exhibits a discontinuity. Under the independent reference model, which assumes a maximum interreference distance given by the stationary geometric distribution, the fault curve is stable from cp_{i-1} to hp_i-1 and the number of faults is equal to $faults(cp_{i-1})$. This estimate is a worst-case estimate, the best-case one being a fault curve stable between $cp_{i-1}+1$ and hp_i-1, with a number of faults equal to $faults(hp_i)$. In general, the probability of a page hit for region j may safely be assumed to be uniformly distributed: therefore the fault curve can be linearly interpolated between the cold point and the hot one.

Note that when $K \le T_j$, no reusal occurs at levels $k \ge j$, and by definition, $T_k = T_j$. Consequently, the fault curve is a constant for $b > hp_{j-1}$. Although hot points are defined (for convenience in the analysis) also for $b > hp_{j-1}$, it must be understood that the maximum hot point is, in this case, hp_{j-1}. In the limit case, $K=1$, the maximum hot point is hp_0. Figure 1 shows the fault rate in a specific case for different values of K.

The number of faults at the hot points is independent of the reference density distribution. It is also independent of orders in the reference string. More skewed distributions (such as Zipf's law [ZIPF49]), or ordered reference strings tend to produce a best-case interpolation between a cold point and a hot point (as shown in figure 3).

6. OLRU vs. LRU

The improvement over an ILRU replacement strategy will be characterized under the current assumptions. This represents an underestimate of the improvement over standard LRU replacement, because ILRU was shown to be never worse than LRU. ILRU tends to allocate frames *vertically*, i.e. depth-first by traversal stacks, rather than *horizontally*, i.e. breadth-first by levels.

In fact, when the reference density distribution for data pages is uniform and $b \geq L$, the ILRU stack will tend to contain entire traversal stacks of length $L-1$ (the root is locked in the buffer), up to the point in which T_2 traversal stacks can be entirely contained in the buffer. From this point on, the first two levels are locked, and excess frames are used to contain entire $L-2$ traversal stacks, and so on.

This characterization is a simplification of the real behaviour of ILRU because ILRU is sensitive to variations in the interreference distance between pages at the same level. When the vertical law is disattended by ILRU, allocation will suboptimally privilege higher levels of the tree. In fact, by construction, the distance in the ILRU stack between two pages at the same level i is no lower than $L-i+1$. If a page $p_{h,i}$ in the buffer is rereferenced n times in sequence with different sons, the distance between $p_{h,i}$ and the other most recently used page at level i increases to $n(L-i+1)$. This means that a page at level i can be replaced by a page at level $j > i$, which is suboptimal under the current assumptions.

6.1. Storage overhead

The following analysis assumes, conservatively, that levels j, $1 \leq j < i$, have been optimally allocated, and considers the difference caused by a vertical allocation of the frames allocated by OLRU for the i-th level. Assume that reusal occurs at every level of the tree, and that the tree is "bushy", i.e. $T_j = P_j = F^{j-1}$ for all levels j. The point at which level i is locked in memory occurs when

$$hp'_i = \sum_{j=1}^{i} T_j + T_i\,(L-i)$$

that is, $T_i\,(L-i)-1$ frames later than in OLRU. The fault rate at hp'_i is slightly lower for ILRU because there is a probability of reusal for higher levels of the tree. If the fanout is fairly large (as it is usually for index pages) the two fault rates tend to the same value. Between hot points, the fault curve can be linearly interpolated although this tends to overestimate the actual behaviour. When the fanout F tends to infinity, the buffer overhead of ILRU with respect to OLRU tends to the limiting value $(L-i)$, which decreases with increasing i's, and is no lower than 100% when $i < L$.

6.2. Fault rate overhead

From the characterization above, it appears that ILRU can be significantly worse than OLRU, especially when the parameter K is such that no reusal is expected at the highest levels of the tree. In this case, keeping entire traversal stacks in the buffer is useless, since only the reusable prefix of the stack lowers the fault rate.

The fault rate at hp_1 and hp_L is obviously the same for the two strategies: consequently, their behaviour is identical for $L=2$. When $L>2$, the approximate overhead of ILRU with respect to OLRU is computed as follows, under the assumption of a large fanout F (usually verified in practice), and the assumption of $K \gg T$. Assume that $b=hp_i$ for OLRU, i.e. levels 1 to i are non-deficient. Assume conservatively that with the same buffer size ILRU is able to lock levels 1 to $i-1$ in the buffer. The vertical allocation of the last P_i frames will be considered. Let $t \approx P_i / (L-i+1)$ denote the number of traversal stacks which can be maintained. The overhead of ILRU with respect to OLRU is:

$$\delta = \frac{\sum_{j=1}^{L} P_j + \sum_{j=1}^{L} (K-P_j)\left(1-\frac{t}{P_j}\right) - \sum_{j=1}^{L} P_j - \sum_{j=i+1}^{L} (K-P_j)}{\sum_{j=1}^{i} P_j + K(L-i-1)}$$

When F is large, the ratio t / P_j tends to 0 for all $j > i$. The limit

$$\lim_{K\to\infty} \delta \approx \lim_{K\to\infty} \frac{(K-F^{i-1})\left(1-\frac{1}{L-i+1}\right)}{\frac{F^i-1}{F-1} + K(L-i)} \approx \frac{1}{L-i+1}$$

is maximum when $L-i$ is minimum. Since $i < L$, the maximum relative overhead occurs at hp_{L-1} and is approximately 50%. Being LRU no better than ILRU, the relative overhead of LRU exceeds this figure.

In practice, most indices have no more than 5 levels. In this case the relative overhead is in the range 25-50%. Figure 2 shows the behaviour of OLRU and LRU in a sample case.

7. RELAXING THE ASSUMPTIONS

It was shown that storage by levels is optimal under the assumption of a uniform distribution of reference densities. Intuitively, the more the reference density distribution deviates from uniformity, the more a vertical allocation law becomes beneficial.

Let the pages accessed at level i be numbered by decreasing reference density, for all levels i. According to the independent reference model, optimality of storage by traversal stacks arises when

$$p(p_{j,i}) \leq p(p_{j-1,k}), \text{ for } all\ 1 \leq i < k \qquad (7.1)$$

In this case, allocating storage to $p_{j,i}$ (i.e. horizontally) is worse than allocating storage to $p_{j-1,k}$ (i.e. vertically). When equality holds, both allocation strategies are optimal.

The Zipf's law distribution is commonly assumed to be the most skewed distribution encountered in practice: it will be assumed now as the reference density distribution for data records. In practice, a random assignment of records to pages can be assumed: this means that the reference distribution for data pages and lower levels of the tree progressively tends to uniformity, because of the equalizing effect of the random assignment. Such an equalization will be currently ignored, and it will be conservatively assumed in the following that the reference density distribution is Zipfian *for all the levels of the tree.*

The Zipfian reference probability for the j-th most accessed page at level i, $p(p_{j,i})$, is

$$p(p_{j,i}) = \frac{1}{L\, j\, H(F^{i-1})}$$

where $H(n)$ is the n-th harmonic number, and F is the average fanout of the index. Disequality (7.1) can be written as:

$$(j-1)\, H(F^{k-1}) < j\, H(F^{i-1})$$

$H(n)$ can be approximated by $\ln n + \gamma$ (where $\gamma \approx 0.577$ is Euler's constant). Consequently:

$$j < \frac{(k-1)\ln F + \gamma}{(k-i)\ln F} \approx \frac{k-1}{k-i}$$

This result implies that, under very skewed distributions:

(1) a vertical law is beneficial only for very few traversal stacks. Consequently, a complete vertical allocation is optimal only under extreme deviations from uniformity, rarely if ever encountered in practice;

(2) the number of traversal stacks to be maintained is independent of the average fanout. Consequently, with the relatively high fanouts used in practice, the portion of the buffer to be vertically allocated is very small;

(3) horizontal allocation becomes increasingly beneficial for lower (i.e. closer to the root) levels of the tree, even when the randomic equalization of the Zipfian distribution is ignored. Thus, OLRU offers a good performance for the part of the index which is most likely to be contained in the buffer.

Figure 3 shows the behaviour of OLRU and LRU in a sample case. The fault rate of the two strategies is very close, with OLRU being at its best with very small buffer sizes, and at its worst at hp_{L-1} as expected from the analysis. For b<3 and b>10, OLRU is beneficial (LRU overhead is 33% at b=2, and 17% at b=15). The maximum OLRU overhead occurs at b=10, and is 12%.

8. CONCLUSIONS

Two new buffer management strategies for the access to a unique clustered index were proposed. The first strategy, ILRU, was shown to provide an improvement over LRU under any reference density distribution. This improvement is relevant for relatively small buffer sizes. The second strategy, OLRU, is optimal, according to the independent reference model, when the reference density distribution of data pages is uniform. When compared to OLRU, ILRU shows a storage overhead larger than 100%, and a fault overhead in the range 25% to 50%. Overheads for a standard LRU exceed these figures.

The fault rate of OLRU at the analyzed hot points is independent of page reference distributions. It is easy to show that, under the independent reference model, optimal policies for uniform distributions produce a higher (or no smaller) fault rate than optimal policies for any other distribution. Since OLRU is optimal under the uniform distribution assumption, it guarantees a lower bound on the maximum fault rate, and consequently a maximum lower bound on the system throughput.

When reference distributions severely deviate from uniformity, ILRU tends to become more efficient than OLRU. Preliminary results indicate that in these cases the fault rate of the two strategies is very close. It can be argued, on the basis of the analysis reported in section 7, that a strategy which initially allocates buffer frames to a limited number of traversal stacks, and subsequently resorts to OLRU, might provide a better overall performance under different reference distribution.

Both OLRU and ILRU can be used in connection with hot set or query locality set management.

Research on other types of access, including sequential scans, access by sorted relations, non-unique indices, and unclustered indices, is currently under way. Preliminary results indicate that optimal strategies for sequential scans and ordered access can be derived under very general assumptions. For unclustered indices, LRU appears to be the best choice since the bottleneck is given by the access to data pages, and index access costs play a secondary role.

1. REFERENCES

[BAYE72] Bayer, R., McCreight, C., "Organization and maintenance of large ordered indexes", *Acta Inf., 1*, 3, 1972, 173-189

[BERN81] Bernstein, P.A, et al., "Query processing in a system for distributed databases (SDD-1)", *ACM Trans. Database Syst., 6*, 4, Dec. 1981, 602-625

[CHOU85] Chou, H.T, DeWitt, D.J., "An evaluation of buffer management strategies for relational database systems", *Proc. 11th Conf. on Very Large Data Bases*, Stockholm, 1985

[COME79] Comer, D., "The ubiquitous B-tree", *ACM Comp. Surv., 11*, 2, 1979, 105-120

[EFFE84] Effelsberg, W., Harder, T., "Principles of database buffer management", *ACM Trans. Database Syst., 9*, 4, Dec. 1984, 560-595

[FRAN74] Franaszek, P.A., Wagner, T.J, "Some distribution-free aspects of paging algorithm performance", *ACM Journ., 21*, 1, Jan. 1974, 31-39

[KNUT73] Knuth, D.E., "*The art of computer programming - Volume 3: Sorting and searching*", Addison-Wesley, 1973

[IJBE85] Ijbema, A., Blanken, H., "Estimating bucket accesses: a practical approach", *Proc. 2nd IEEE Conf. on Data Engineering*, 1985

[MATT70] Mattson, R.L., et al., "Evaluation techniques for storage hierarchies", *IBM Syst. J. 9*, 2, 1970, 78-117

[SACC82] Sacco, G.M., Schkolnick, M., "A technique for managing the buffer pool in a relational system using the hot set model", *Proc. 8th Conf. on Very Large Data Bases*", Mexico City, 1982

[SACC83] Sacco, G.M., Schkolnick, M., "Thrashing reduction in demand accessing of a data base through an LRU paging buffer pool", US patent 4.422.145, Dec. 1983

[SACC86] Sacco, G.M., Schkolnick, M., "Buffer management in relational database systems", *ACM Trans. Database Syst., 11*, 4, Dec. 1986

[WHAN83] Whang, K., *et al.*, "Estimating block accesses in database organizations: a close noniterative formula", *Comm. of the ACM 26*, 11, Nov. 1983, 940-944

[YAO77] Yao, S.B., "Approximating block accesses to database organizations", *Comm. of the ACM 20*,, 1977, 260-261

[ZIPF49] Zipf, G.K., "*Human behaviour and the principle of least effort: an introduction to human ecology*", Addison-Wesley, 1949

Figure 3 - LRU vs. OLRU. Zipf's reference density at data record level. Random assignment of data records to pages. Access parameters: K=5000, L=3, P_3=800, F_i=100 ($1 \leq i < 3$), F_3=2.