Expressions for Batched Searching of Sequential and Hierarchical Files

By: Prashant Palvia

Palvia, P. "Expressions for Batched Searching of Sequential and Hierarchical Files," ACM Transactions on Database Systems, March, 1985, Vol. 10, No. 1, pp. 97-106.

Abstract:
Batching yields significant savings in access costs in sequential, tree-structured, and random files. A direct and simple expression is developed for computing the average number of records/pages accessed to satisfy a batched query of a sequential file. The advantages of batching for sequential and random files are discussed. A direct equation is provided for the number of nodes accessed in unbatched queries of hierarchical files. An exact recursive expression is developed for node accesses in batched queries of hierarchical files. In addition to the recursive relationship, good, closed-form upper- and lower-bound approximations are provided for the case of batched queries of hierarchical files.
Categories and Subject Descriptors: H.3.3 [Information Storage and Retrieval]: Information Search and Retrieval—*search process;* H.2.2 [Database Management]: Physical *Design—access methods*
General Terms: Performance, Theory
Additional Key Words and Phrases: Database design, physical design, files, block accesses, sequential search, number of records, access paths

Article:

1. INTRODUCTION

Techniques of file organization and access methods have been widely discussed in the literature [4, 7, 8, 11]. Different organization structures and access methods are relevant, depending on the usage requirements of the data stored in the files and in the database. In today's proliferation of on-line, fast-response systems, it is very common to have random (hash-based) and indexed file organizations with fast, direct access to individual records in the file. However, in batch applications and some online applications, it may be desirable to sequentially search a batch of records in the file. Shneiderman and Goodman [9] have shown the desirability of batched searches in sequential and tree structure organizations.

This paper refines and extends the expressions and results reported by Shneiderman and Goodman [9] and later by Batory and Gotlieb [1]. Shneiderman and Goodman developed expressions to show the savings due to batching in sequential and tree organizations. They did not, however, find exact closed-form expressions. Their expressions were complex recursive relations; they did, however, find a closed-form lower-bound estimate for sequential files. Batory and Gotlieb speculated on the form of the expression for the number of node accesses in a sequential search on the basis of the work in [9].

A direct approach is taken here. Rather than obtaining savings due to batching, explicit and accurate expressions are derived for the cost of batching. Then, the cost of batching can be compared to the cost of any other type of search. (Note that the above authors [1,9] compare the cost of batched k requests to the cost of k individual searches.) The benefits of these expressions threefold: first, the new equations are exact and closed-form (nonrecursive, noniterative) in the sequential case; second, closed-form equations are easier and simpler to use in any further or related work; and, finally, savings due to batching can be obtained in comparison with any other search technique.

Expressions are developed first for the sequential files and then for hierarchically structured files.

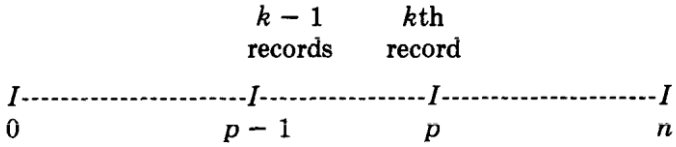
2. SEQUENTIAL FILES

Let the sequential file contain n records organized in m pages (blocks) with a blocking factor of $p = n/m$. Let there be k distinct records requested from this file. Note that these k records are distinct, so that once a record has been requested it may not be requested again (i.e., sampling without replacement). This requirement is especially sensible in batched requests.

The work of Shneiderman and Goodman [9] is based on the assumption of nondistinct k records. However, the number of nondistinct k records can be converted to the corresponding number of distinct records (e.g., see [3]). Thus the work described here can be applied to the retrieval of distinct as well as nondistinct records.

The commonly used approximation when sequentially retrieving records from a file is a complete scan of the file. This approximation is fairly good when a large number of records is being requested. However, as can be seen below, a complete scan is not required when the number of records requested *(k)* is not large (i.e., starting from the beginning of the file, the k records may be found without going through the whole file).

To find the k records in the n-record file, at least the first k records of the file have to be searched. The probability of finding k records in the first p records of the file is derived in the following manner:



For the first p records to be searched, the first p — 1 records should contain k — 1 of the k records, and the pth record should contain the kth record. Also, the remaining n — p records should not contain any of the desired k records. Using a combinatorial argument, there are $C(p - 1, k - 1)$ ways of having k — 1 desired records in the first p — 1 records. There is only one way of having the pth record as the kth record and only one way of having zero desired records in the remaining n — p records. Thus the total number of ways in which the records may be arranged so as to require searching only the first p records is $C(p - 1, k - 1)$.

On the other hand, the total number of ways of arranging k desired records in the n records of the file is $C(n, k)$. Therefore, the probability of searching the first p records is

$$= \frac{C(p-1,\, k-1)}{C(n,\, k)}.$$

Since p can range from k to n, the expected number of records $E(p)$ to be searched is

$$= \sum_{p=k}^{n} p \cdot \frac{C(p-1,\, k-1)}{C(n,\, k)}.$$

The summation in the above expression could be begun from 1, as the additional terms introduced are all 0s. On simplification,

$$E(p) = \frac{k \cdot (n-k)!}{n!} \sum_{p=1}^{n} \frac{p!}{(p-k)!}.$$

This can be further rewritten as

$$E(p) = \frac{k \cdot k! \cdot (n-k)!}{n!} \sum_{p=1}^{n} C(p,\, k).$$

Now, it can be easily proven that

$$\sum_{p=1}^{n} C(p,\, k) = C(n+1,\, k+1).$$

This is based on the recursive application of the following identity:

$$C(n+1,\, k+1) = C(n,\, k) + C(n,\, k+1).$$

Making the above substitution leads to

$$E(p) = \frac{k \cdot k! \cdot (n-k)!}{n!} \cdot C(n+1,\, k+1).$$

On further simplification, the expected records accessed are

$$E(p) = \frac{k}{k+1} \cdot (n+1). \tag{1}$$

Since there are a total of n records in the file:

$$\text{Expected file proportion accessed} = \frac{k}{k+1} \cdot \frac{(n+1)}{n}. \tag{2}$$

The same proportion of pages have to be accessed. Since there are m pages in the file:

$$\text{Expected pages accessed} = \frac{k}{k+1} \cdot \frac{(n+1)}{n} \cdot m \quad . \tag{3}$$

According to Eq. (3), about one-half of the file has to be searched to retrieve one record, two-thirds to retrieve two records, three-fourths to retrieve three records, and so on. More than 90 percent of the file has to be searched when searching nine or more records. Figure 1 (column A) shows the expected number of pages accessed for a file with 300 records; blocking factor, $p = n/m$ of 1, 5, 10, and 15; and number of records required, k = 2, 5, 10, 20, 50, and 100. As can be seen, with k getting larger, almost the whole file has to be scanned. It should be noted that both papers ([1] and [9]) have speculated on the form of the above equation, without providing a formal proof. [1] also suggests that the proportion of the file that is accessed is $k/(k + 1)$, and not as in Eq. (2).

k and p values		Column A. Batch of k – sequential search	Column B. k individual sequential searches	Column C. Batch of k – random search	Column D. k individual random searches
$k = 2$	$p = 1$	200.67	301.0	2	2
	$p = 5$	40.13	60.2	1.973	2
	$p = 10$	20.07	30.1	1.941	2
	$p = 15$	13.38	20.07	1.909	2
$k = 5$	$p = 1$	250.83	752.5	5	5
	$p = 5$	50.17	150.5	4.836	5
	$p = 10$	25.08	75.25	4.641	5
	$p = 15$	16.72	50.17	4.457	5
$k = 10$	$p = 1$	273.64	1505.0	10	10
	$p = 5$	54.73	301.0	9.355	10
	$p = 10$	27.36	150.5	8.626	10
	$p = 15$	18.24	100.33	7.972	10
$k = 20$	$p = 1$	286.67	3010.0	20	20
	$p = 5$	57.33	602.0	17.505	20
	$p = 10$	28.67	301.0	14.952	20
	$p = 15$	19.11	200.67	12.895	20
$k = 50$	$p = 1$	295.10	7525.0	50	50
	$p = 5$	59.02	1505.0	35.887	50
	$p = 10$	29.51	752.5	25.155	50
	$p = 15$	19.67	501.67	18.702	50
$k = 100$	$p = 1$	298.02	15050.0	100	100
	$p = 5$	59.60	3010.0	52.099	100
	$p = 10$	29.80	1505.0	29.48	100
	$p = 15$	19.87	1003.33	19.954	100

Fig. 1. Expected pages accessed. k = number of records required; p = blocking factor = n/m; n = number of records = 300.

To put the batched sequential method in perspective, the data in column A are compared with data in columns B, C, and D. Column B is the number of page accesses with k individual sequential searches. Each individual sequential search requires accessing 1/2 • (n + 1) records, so that the total number of records accessed is k/2 (n + 1), and the total number of pages accessed is $k/2p \bullet (n + 1)$. The third column shows the number of page accesses for k batched requests on a randomly accessed file. The following expression evaluates the pages accessed for this case:

$$\text{Pages } accessed = m \bullet (1 - (1 - (k/n))^P).$$

This expression was shown by Palvia and March [6] to be an overall better estimator than the prevalently used approximation by Cardenas [2], and computationally more efficient than the exact expression by Yao [10].

The last column shows the number of pages accessed if each record request was searched individually using a random access path. A random access path requires accessing exactly one page to find one record. Thus, to find *k* records individually, exactly *k* pages will be required.

Some comments corroborating intuition are in order, based on the data in Figure 1.

(1) When there are many records requested from the file, it is indeed better to batch them and then retrieve them collectively in one search of the file. Looking at columns A and C, it is clear that batching using a random access path yields fewer *accesses* than batching with a sequential *access* path. Only when the number of records required becomes very high (on the order of 50% or higher of the total file size) do the two methods yield about the same number of accesses. However, maintaining a direct access path incurs additional maintenance and/ or storage costs. A guideline may be to consider using the sequential access path only when the proportion of records to be retrieved is relatively high, (e.g., 10 percent or higher of the entire file size [5]).

(2) It is also apparent from the figure that using blocking can be especially helpful if the record retrieval requests are batched and randomly searched. Thus, a higher blocking factor may be considered when batching queries and using a random access path.

(3) When only one or a few records are required from the file, it is definitely advantageous to have a direct access path.

(4) Even if using a random access path, it is preferable to batch the queries (if possible); doing so results in a much lower number of page accesses. In essence, it is always best to batch queries—and not to batch them only if it is not possible because of user requirements.

(5) Column B shows the most number of page accesses using *k* individual sequential searches. It makes no sense at all to satisfy *k* queries by *k* individual sequential searches; and doing so should definitely be discouraged. The only exception may be when the sequential access path is the only access mechanism available and the specific user requirement dictates individual sequential searches.

The next section proceeds with an analysis of hierarchical file structures.

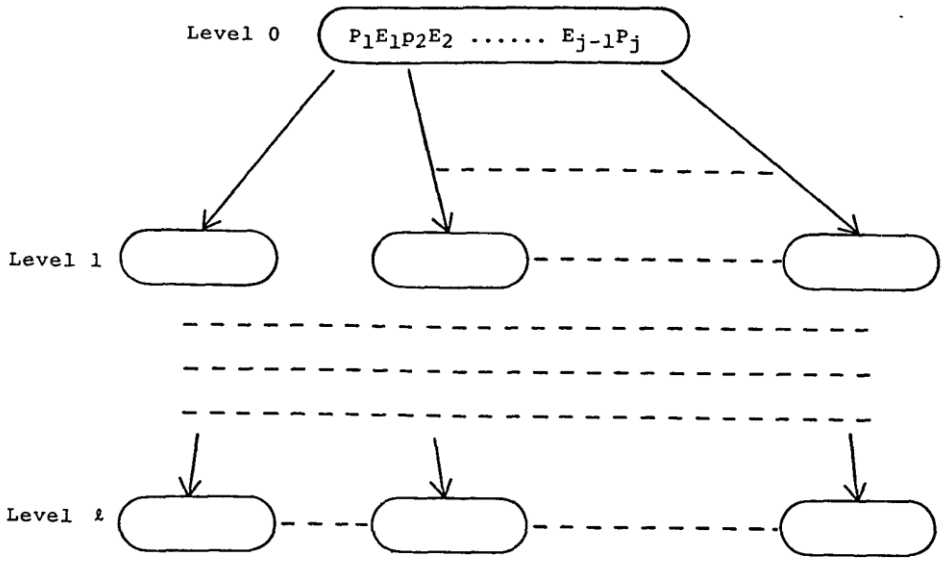


Fig. 2. A *j*-ary tree.

3. HIERARCHICAL SEARCH TREES

Consider a j-ary tree (shown in Figure 2) with j — 1 elements (keys or records) at each node and j branches emerging via pointers from each node. The levels of the tree are numbered 0 through l, the height of the tree being l + 1. Let the tree also be completely filled; that is, each possible node exists and contains j — 1 elements. (The same assumptions were used by Shneiderman and Goodman in [9].) Again using a direct approach, expressions are developed for the number of node accesses for unbatched queries as well as for batched queries. It should be noted that the expression due to Shneiderman and Goodman gives savings due to batching, while the expressions due to Batory and Gotlieb are upper- and lower-bound estimates based on somewhat different assumptions. Their assumptions in [1] always require traversing to the highest level in the tree.

3.1 Unbatched Queries

Let the query require the searching of k elements. In unbatched searching, each element will be found one at a time. Thus the entire query will require accesses equal to k times the average number of accesses required in finding one element in the tree.

With l + 1 levels, the number of elements in the tree equal $(j - 1) \cdot \sum_{i=0}^{l} j^i = j^{l+1} - 1$. Assuming a uniform distribution of elements among the nodes, and the query element to be randomly selected from all of the elements, then any of these elements has an equal probability of being selected. Since each node has the same number of elements, it follows that each node also has an equal probability of containing the required element.

Since the total number of elements = j^{l+1} and the number of elements in any node = j — 1, the probability of any node containing the required element, or being selected, is given by $(j - 1)/(j^{l+1} 1)$. However, the 0^{th}-(root) level node requires only one access, the l^{st}-level nodes require two accesses, 2^{nd}-level nodes require three accesses, and so on. Also, there is one node at the root level, j nodes at the 1^{st} level, j^2 nodes at the 2^{nd} level, j^3 nodes at the 3^{rd} level, and so on.

The expected number of accesses is then determined by the summation:

$$= \frac{1(j-1)}{(j^{l+1}-1)} + \frac{2(j-1)j}{(j^{l+1}-1)} + \frac{3(j-1)j^2}{(j^{l+1}-1)} + \cdots + \frac{(l+1)(j-1)j^l}{(j^{l+1}-1)}$$

$$= \frac{(j-1)}{(j^{l+1}-1)} \cdot (1 + 2j + 3j^2 + \cdots + (l+1)j^l).$$

On further simplification, this reduces to:

$$\text{Expected node accesses for searching one element} \left\{ = \frac{(l+1)j^{l+2} - (l+2)j^{l+1} + 1}{(j^{l+1}-1)(j-1)}. \right. \quad (4)$$

And expected node accesses to find k unbatched elements

$$= k \cdot \frac{(l+1)j^{l+2} - (l+2)j^{l+1} + 1}{(j^{l+1}-1)(j-1)}. \quad (5)$$

3.2 Batched Queries

Two expressions are developed for a query that batches k elements. The first expression is an exact recursive relationship and the second expression a closed-form approximate result.

3.2.1 Exact Recursive Relationship. For the exact expression, define $A(k, l + 1)$ as the number of node accesses to retrieve k elements from a j-ary tree with l + levels. There are a total of j^{l+1} — 1 elements in the complete tree. Consider one of the j level —1 subtrees: each one of them has j^l — 1 elements, and anywhere from 0 to k desired elements. Define $P(i, l, k)$ as the probability of having i elements of the total k elements in a subtree of height l. This probability is given by the hypergeometric distribution as

$$P(i, l, k) = \frac{C(j^l - 1, i) \cdot C(j^{l+1} - j^l, k - i)}{C(j^{l+1} - 1, k)}.$$

It takes one access to reach the root node; the next level subtrees may be accessed depending on the $P(i, l, k)$ probabilities. Since there are j subtrees of height l, the following recursive equation is defined:

$$A(k, l+1) = 1 + j \cdot \sum_{i=1}^{k} P(i, l, k) \cdot A(i, 1). \qquad (6)$$

The summation on i can be started from 1, since $A(0, l)$ is always equal to 0 for all l. Also, $A(i, 1) = 1$ for all i. Thus the accesses for a batched request can be obtained by using Eq. (6). For example, for a binary tree (i.e., $j = 2$) with $l = 2$ and $k = 2$, the number of accesses is given by $A(2, 3)$, and is obtained as

$$\begin{aligned} A(2,3) &= 1 + 2\cdot[P(1,2,2)\cdot A(1,2) + P(2,2,2)\cdot A(2,2)] \\ &= 1 + 2\cdot[4/7\cdot A(1,2) + 1/7\cdot A(2,2)] \\ &= 1 + 2\cdot[4/7\cdot(1 + 2(P(1,1,1)\cdot A(1,1))) \\ &\qquad + 1/7\cdot(1 + 2(P(1,1,2)\cdot A(1,1)))] \\ &= 1 + 2\cdot[4/7\cdot(1 + 2/3) + 1/7\cdot(1 + 4/3)] \\ &= 25/7 = 3.57 \text{ accesses.} \end{aligned}$$

Although the equation can be recursively applied, its complexity does not easily allow a closed-form solution. One should note that in the special case when $k = 1$, Eq. (6) does reduce to the closed-form solution of Eq. (4). The next section develops approximate closed-form solutions for batched queries.

3.2.2 Approximate Closed-Form Results. As has been assumed earlier, the k elements are randomly distributed among the total $j^{l+1} - 1$ keys. The probability of selection of any element can then be approximated as $k/(j^{l+1} - 1)$, and the probability of an element not being selected is one minus the above probability. Since there are j — 1 elements in a node, the probability Q of a node not being selected is given by

$$Q = \left(1 - \frac{k}{j^{l+1} - 1}\right)^{j-1}. \qquad (7)$$

Then, the probability of any node being selected is given by 1 — Q, A lower bound on the node accesses is found as follows: Assume that each node that is selected requires only one node access. This implies that if a node is selected at a higher level, all nodes in the branch connecting this node with the root have also been selected. This is not very realistic. but does yield a lower bound.

$$\begin{aligned} \text{Nodes accessed} &= (1 - Q)(1 + j + j^2 + j^3 + \cdots + j^l) \\ &= (1 - Q)(j^{l+1} - 1)/(j - 1). \end{aligned} \qquad (8)$$

An upper bound on the node accesses can be obtained by assuming that each node access requires the maximum number of node accesses (i.e., equal to the height of the tree up to that node). This of course assumes that if a node is selected at a higher level, then all nodes in the branch connecting this node with the root have previously not been selected. Again, this is not realistic, but does yield an upper bound. Then, nodes accessed

$$\begin{aligned} &= (1 - Q)(1 + 2\cdot j + 3\cdot j^2 + \cdots + (l+1)\cdot j^l) \\ &= (1 - Q)\cdot\frac{(l+1)\cdot j^{l+2} - (l+2)\cdot j^{l+1} + 1}{(j-1)^2}. \end{aligned} \qquad (9)$$

Using the previous example with $j = 2$, $l = 2$, and $k = 2$, the value of Q is 5/7, and Eq. (8) gives 2 accesses and Eq. (9) gives 4.858 accesses.

An accurate (but more complex) expression based on the estimated Q probability is derived using the following arguments. The root node is always required, which is one access. Consider any of the subtrees at level 1. Each subtree contains $(j^l - 1)/(j - 1)$ nodes. The root node of this subtree will be accessed only if the nodes in the entire subtree contain any of the desired elements. Since Q is the probability of a node not being selected, the probability of a subtree rooted at level 1 not being selected is given by

$$Q^{(jl - 1)/(j - 1)}$$

One minus this probability gives the probability of this subtree being selected. Also, there are j subtrees at level 1 (j^2 subtrees at the next level, and so on).

Applying this argument successively to higher levels of the tree yields:

$$\text{Node accesses} = 1 + j(1 - Q^{(j^l-1)/(j-1)}) + j^2(1 - Q^{(j^{l-1}-1)/(j-1)}) + \cdots + j^{l-1}(1 - Q^{(j^2-1)/(j-1)}) + j^l(1 - Q^{(j-1)/(j-1)}).$$

On simplification, this becomes

$$= (j^{l+1} - 1)/(j - 1) - \sum_{i=1}^{l} j^{l-i+1} \cdot Q^{(j^i-1)/(j-1)}. \qquad (10)$$

A summation of the series in the second part of the expression will give a "nice" closed-form estimate for the nodes accessed in a batched query. However, note that since Q is less than or equal to 1 and j is greater than or equal to 2, the successive terms in the summation get progressively very small, and can be ignored. Thus, very good upper-bound estimates can be obtained.

Considering only the first two terms of the series,

$$\text{Nodes accessed} = (j^{l+1} - 1)/(j - 1) - j^l \cdot Q - j^{l-1} \cdot Q^{j+1} \qquad (11)$$

Considering the first three terms of the series,

$$\text{Nodes accessed} = (j^{l+1} - 1)/(j - 1) - j^l \cdot Q - j^{l-1} \cdot Q^{j+1} - j^{l-2} \cdot Q^{j2+j+1} \qquad (12)$$

Again, using the previous binary tree example, with $j = 2$, $l = 2$, and $k = 2$, expression (11) yields 3.414 node accesses and expression (12) yields 3.319, both very close to the accurate values given by Eq. (6).

Savings due to batching are again apparent. For example, with a binary tree of height 3 (the example used in our discussion), a batched query of two elements requires 3.57 accesses. The same query, when unbatched, will require 2 x 2.429 or 4.858 node accesses. Higher batch sizes will yield higher savings. Savings due to batching in j-ary trees have been reported in [9].

4. SUMMARY

Batching generates considerable savings in sequential access on linear files as well as direct access on random files. It also brings savings in hierarchical files. The conclusion is to always batch queries if it is permissible, given the nature of the immediacy of user requirements.

This paper provides exact expressions for node (block or page) accesses for batched sequential accessing of a linear file as well as batching on a hierarchical j-ary tree file. The latter expression being a recursive one, good closed-form approximations are provided.

REFERENCES

1. **BATORY, D. S., AND GOTLIEB, C. C.** A unifying model of physical databases. *ACM Trans. Database Syst.* 7, 4 (Dec. 1982).
2. **CARDENAS, A. F**. Analysis and performance of inverted database structures. *Commun. ACM 18,* 5 (May 1975).
3. **CHEUNG**, **T.-Y.** Estimating block accesses and number of records in file management. *Commun. ACM 25,* 7 (July 1982).
4. **HSAIO, D**., **AND HARARY, F**. A formal system for information retrieval from files. *Commun. ACM 13,* 2 (Feb. 1970; Corrigendum, *Commun. ACM 13,* 4 (April 1970).
5. **MARCH, S. T**. A mathematical programming approach to the selection of access paths for large multiuser databases. *Decis. Sci. 14,* 4 (Fall 1983).

6. PALVIA, P., **AND MARCH, S. T**. Approximating block accesses in database organizations. *Inf. Process. Lett. 19* (Aug. 1984).
7. SEVERENCE, D. G. Some generalized modeling structures for use in design of file organizations. Ph.D. dissertation, Univ. of Michigan, 1972.
8. SEVERENCE, D. G., AND CARLIS, J. V. A practical approach to selecting record access paths. *ACM Comput. Surv. 9,* 4 (Dec. 1977).
9. SHNEIDERMAN, B., AND GOODMAN, V. Batched searching of sequential and tree structure files. ACM *Trans.* Database *Syst. 1,* 3 (Sept. 1976).
10. YAO, S. B. Approximating block accesses in database organizations. *Commun. ACM 20,* 4 (April 1977).
11. **YAO, S. B**. An attribute-based model for database access cost analysis. ACM Trans. Database Syst. 2, 1 (Mar. 1977).