View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by The University of North Carolina at Greensboro

Batched Searching in Database Organizations

By: Prashant Palvia

Palvia, P. "Batched Searching in Database Organizations," Information Sciences, June 1988, Vol. 45(1), pp. 23-37.

Made available courtesy of Elsevier: http://www.elsevier.com/

*****Reprinted with permission. No further reproduction is authorized without written permission from Elsevier. This version of the document is not the version of record. Figures and/or pictures may be missing from this format of the document.*****

Abstract:

Savings in the number of page accesses due to batching on sequential, tree-structured, and random files are well known and have been reported in the literature. This paper asserts that substantial savings can also be obtained in database organizations by batching the requests for records (in queries), and also by batching intermediate processing requests while traversing the database. A simple database having two interrelated files is used to demonstrate such savings. For the simple database, three variations on batching are reported and compared with the case of unbatched requests. New mathematical expressions have been developed for the batched cases as well as for the unbatched case, and the savings are demonstrated with some example problems. As an extension, larger databases will enjoy even greater savings due to batching. The paper also discusses several strategies for applying the batching approach to current databases, and the advantages of emerging very large main memories for the batching approach.

Article:

1. INTRODUCTION

Several file structures and access paths are used to satisfy user queries from a database. Some models for file organization and database organization along with the accompanying access paths have been presented in [2, 10, 15, 18]. In addition several researchers have proposed and developed tools for selection of access paths to satisfy user queries [1, 3, 8, 9, 11, 17].

An overall strategy to satisfy user queries is to batch the requests for *records* from a file or a database. The desirability of batching queries is well known, as it reduces the total number of page (block) accesses from secondary memory. The cost savings due to batching have been theoretically and empirically demonstrated in [2, 13, 16]. Further direct expressions for batched searching of sequential and hierarchical files are reported in [13, 16]; and expressions for batched searching of random files are reported in [5, 14, 19, 20].

Just as batching yields significant savings in block accesses in file organizations, it has the potential of further savings in database organizations. The savings can be substantial, as they will be cumulative across several files in the database. This paper develops mathematical expressions for page accesses due to batching in database systems and then demonstrates savings due to batching. This demonstration is made for a simple database containing two interrelated files. Larger databases will enjoy greater savings which is also established in the paper.

The remainder of the paper is organized as follows. Section 2 presents characteristics of the database considered in the paper. Different combinations (called cases, later) of batching and "unbatching" are included in the paper. The typical query considered is one which requires traversing records from one file to the second file of the two-file database. Section 3 develops expressions when there is no batching in the first file, and Section 4 develops expressions when batching is also allowed in the first file. Section 5 reports and discusses results of experimental comparison of blocks accessed in the various cases. In this section, the savings in the different cases of batching are reported for several values of database parameters. Section 6 discusses the overall applicability and relevance of the batching approach in today's databases. Technology supporting the batching concept is identified, and alternative ways of supporting batching or limited batching are discussed. Section 7 concludes the paper.

2. A SIMPLE DATABASE

Consider two entity classes E_1 and E_2 having N_1 and N_2 instances respectively. Let the two entity classes be related by a relationship with degrees R_1 and R_2, i.e., each instance of E_1 is related to R_1 instances of E_2 and each instance of E_2 is related to R_2 instances of E_1 (see Figure 1). Note that R_1 is called the outdegree of E_1 and the indegree of E_2, and by the same token R_2 is called the outdegree of E_2 and the indegree of E_1.

We consider two cases: first where $R_2 =1$ and $R_1 >1$ (i.e. a 1: *M* relationship), and second where both R_2, $R_1 >1$ (i.e. an *M: N* relationship). An example of the first case is a department-employee relationship; an example of the second case is an employee-project relationship. Note that R_1 and R_2 need not be constants; there might be some variation in them for specific instances of E_1 and E_2. We assume R_1 and R_2 to be average outdegrees.[1]

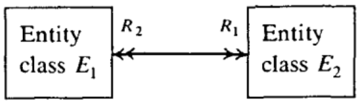


Fig. 1. A simple database.

Let the database be physically represented as two interrelated files F_1 and F_2. File F_1 contains N_1 records corresponding to E_1's instances, and file F_2 contains N_2 records corresponding to E_2's instances. To indicate the relationship between the records of files and F_2 let there be pointers in each F_1 record to related F_2 records and (if necessary) pointers in each F_2 record to related F_1 records. We are not concerned here with the details of pointer arrangement. Further, let file F_1 be organized in M_1 pages (blocks) with a blocking factor of $P_1 = N_1/M_1$, and file F_2 be organized in M_2 pages with a blocking factor of P_2. Such a physical design is available in many commercial DBMSs (e.g. IMS and DBTG based systems) as well as postulated in physical design models [2, 10].

Further, let there be a query which requests K records from file F_1 and also (some of) the records of file F_2 which are related to these K records of file (For example, the query may ask for some information about certain departments and about employees who work in those departments.)

For the simple database and the given query, we derive several expressions in the unbatched mode and the batched mode. For convenience and ease of presentation, the results are divided into two sections: one where there is no batching in the first target file F_1 , and the other where batching is used in F_1. We assume throughout that direct access paths are available on both files for accessing individual records, which is fairly common in commercial DBMSs. It is straightforward to extend these results to sequential search of files.

3. NO BATCHING IN FILE F_1

With no batching in file F_1, the K records of file F_1 are individually retrieved by direct access. Each record retrieval requires one page access, so the total number of page accesses of file F_1 is K. Each record of file F_1 is related to R_1 records of file F_2. Therefore R_1 records from file F_2 have to be retrieved K times, once for each of the file F_1 records.

Two cases arise. Each time, the R_1 records from file F_2 can be retrieved individually or can be retrieved collectively as a batch. It perhaps makes more sense to batch these, as they will most likely be required at the same time. (However, applications may not batch them, if each of the records requires substantial subsequent processing.) In any case, if they are retrieved individually, then the number of page accesses is R_1 each time. Since R_1 records have to be retrieved K times, the number of total page (block) accesses $B(u,u)$ are given by the following expression:

$$B(u, u) = K + KR_1. \tag{1}$$

(Note that in our notation, B refers to blocks and it has two arguments. The first argument is for the first file F_1, and the second is for the second file F_2. The argument value can be u or b, implying unbatching or batching in that particular file.)

The first term in the expression represents page accesses of file F_1 and the second term represents page accesses of file F_2.

In the second case, the R_1 records of file F_2 are retrieved as a batch. As is known, blocking will pay off in a reduced number of page accesses when the blocking factor is high [13, 14, 19]. The number of pages accessed for randomly retrieving k records from a file with n records blocked into m blocks is a function of k, n, and m, i.e. $F(k, n, m)$. Different exact and approximate expressions have been developed for this function [5, 14, 19, 20]. We use the expression in [14], as it has been demonstrated to be an overall better estimator than other approximate estimators and computationally more efficient than the exact expressions. According to [14],

$$F(k, n, m) = m\left[1 - \left(1 - \frac{k}{n}\right)^{n/m}\right] \tag{2}$$

Since R_1 records have to be searched K times, the number of total page accesses $B(u, b)$ is given by

$$B(u, b) = K + KF(R_1, N_2, M_2) \tag{3}$$

4. BATCHING IN FILE F_1

The K records from file F_1 may be collectively retrieved as a batch. The number of page accesses of file F_1 is then $F(K, N_1, M_1)$. Once all K records of file F_1 have been retrieved, then all pointers from the K records of file F_1 to the file-F_2 records are available in main memory. Only unique pointers from file F_1 to file F_2 need to be considered, as only the unique records will have to be brought into main memory. This is so because once a record is brought into main memory, it stays there until the whole batch is processed. As is assumed in all past works [5, 14, 19, 20], we assume that the main memory/buffer space is large enough to accommodate the complete batch of records. Note that batching still pays off when the memory is limited, as shown in [12]; although not to its fullest extent. We discuss more completely the implications of main memory sizes in section 6 of this paper.

Let

L= the number of unique records of file F_2 that are related to K records of file F_1.

An expression needs to be developed for L. Expressions for L, and consequently for the number of pages accessed, are heavily dependent on the indegree of entity E_1 (or file F_1). Two cases arise: one that the relationship between entities E_1 and E_2 (or equivalently, files F_1 and F_2) is 1: M or 1:1, and the other that the relationship is $M : N$.

4.1. RELATIONSHIP BETWEEN ENTITIES IS 1: M OR 1:1

The relationship between files F_1 and F_2 is such that each record of F_1 is related to one or more records of file F_2 ; however, each record of file F_2 is related to exactly one record of file F_1. If such is the case, then each record in file F_1 is related to unique records in file F_2. Consequently all pointers in file F_1 are also unique.

Since each file F_1 record has R_1 pointers to file F_2 and there are K target records in file F_1, the total number of unique pointers from file F_1 to file F_2 are KR_1. The unique pointers lead to unique records in file F_2 ; thus

$$L = KR_1. \tag{4}$$

If these records are individually retrieved, then

$$B(b,u)_{1:M} = F(K,N_1,M_1) + L, \tag{5}$$

where the subscript on B indicates that the relationship between files F_1 and F_2 is 1 : M (or 1:1). If the L records in file F_2 are retrieved as a batch, then

$$B(b,b)_{1:M} = F(K,N_1,M_1) + F(L,N_2,M_2). \tag{6}$$

Note that the value of L in Equations (5) and (6) is given by Equation (4).

4.2. RELATIONSHIP BETWEEN ENTITIES IS M:N

In this case, each instance of E_1 (or record of F_1) is related to several instances of E_2 (or records of F_2); similarly each instance of E_2 is related to several instances of E_1. The following discussion is made in terms of entities, to allow wide applicability. However, the discussion is equally applicable to the two files of the database, as in the current implementation the files correspond exactly to the entities.

In an $M : N$ relationship, several E_1 instances may be related to the same E_2 instances and vice versa. Thus if we used $L = KR_1$ as the number of E_2 instances related to the K instances of E_1 , several of these instances will be the same or nonunique. However, we are interested in the number of unique E_2 instances *(L)* that are related to K (unique) instances of E_1. The expression for L in the M : N case is developed from the following assumptions:

ASSUMPTION A. All instances of E_1 fully exhaust all instances of E_2 via the relationship (i.e. the property of complete coverage or exhaustibility).

ASSUMPTION B. For each instance of E_1, the R_1 related instances of E_2 are uniformly distributed over all of the N_2 instances of E_2. This assumption of uniform distribution is a common one in database organizations [13, 14, 19].

Implicit in the second assumption is the assumption of sampling with replacement. In sampling with replacement, any instance of E_1 can hit (i.e. be related to) any instance of E_2 ; thus it is possible for different instances of E_1 to hit the same instance of E_2.

Consider the second assumption. Then, with any instance of entity E_1, the probability of relating to or "hitting" an instance of E_2 is

$$P(\text{an } E_2 \text{ instance hit by one } E_1 \text{ instance}) = \frac{R_1}{N_2}.$$

Then,

$$P(\text{an } E_2 \text{ instance not hit by one instance of } E_1) = 1 - \frac{R_1}{N_2}$$

and

$$P(\text{an } E_2 \text{ instance not hit by } K \text{ instances of } E_1) = \left(1 - \frac{R_1}{N_2}\right)^K.$$

Then,

$$P(\text{an instance of } E_2 \text{ hit with } K \text{ instances of } E_1) = 1 - \left(1 - \frac{R_1}{N_2}\right)^K.$$

Since there are N_2 instances of E_2, the expected number of instances hit of entity E_2, i.e. L is

$$L = N_2\left[1-\left(1-\frac{R_1}{N_2}\right)^K\right]. \tag{7}$$

Note that the number of instances of E_2 hit is binomially distributed with parameters $N = N_2$ and $p = 1 - (1 - R_1/N_2)^K$.

Evaluating from equation (7),

$$\text{when } K = 0, \quad L = 0; \qquad \text{when } K = 1, \quad L = R_1;$$

$$\text{when } \quad K = N_1, \qquad L = N_2\left[1-\left(1-\frac{R_1}{N_2}\right)^{N_1}\right].$$

The last expression indicates that when $K = N_1$ and R_1 is less than N_2, then L is less than N_2. In other words, all instances of entity E_1 collectively do not address all instances of entity E_2. This contradicts the exhaustibility assumption stated earlier. From Equation (7), for $K = N_1$, L will approach N_2 only if $R_1 = N_2$ or if N_1 is very large, neither of which may be true. Thus we have shown that:

If the R_1 instances of E_2 related to one instance of E_1 are uniformally distributed over all instances of E_2 (sampling with replacement), then a complete coverage of all instances of E_2 is not possible with all instances of E_1.

To satisfy the "exhaustibility requirements," the sampling-with-replacement assumption has to be relaxed. In sampling without replacement, each instance of entity E_1 will hit R_1 unique instances of entity E_2. This would work in 1: *M* relationships, but is not possible in *M : N* relationships, since there simply are not that many entity-E_2 instances to hit on. Thus, in *M : N* relationships, some element of sampling with replacement is required out of necessity. The following assumption allows part sampling with replacement and part sampling without replacement, and also achieves the exhaustibility requirement. The assumption completely reverts to sampling without replacement for 1 : *M* relationships.

NEW ASSUMPTION B. Let each instance of entity E_1 hit N_2/N_1 distinct instances of entity E_2 (sampling without replacement). The remaining $R_1 — N_2/N_1$ instances are uniformly distributed over the remaining $N_2 - N_2/N_1$ instances of entity E_2 (sampling with replacement).

Note that R_1 has to be greater than or equal to N_2/N_1 for exhaustibility. (R_1 will equal N_2/N_1 in a 1: *M* relationship; then the assumption reverts completely to sampling without replacement.) Further, with this assumption, N_1 instances of entity E_1 will hit $N_1N_2/N_1 = N_2$ instances of entity E_2, thus meeting the exhaustibility requirement.

With the new Assumption B, L can be derived in the following two parts:

(i) Considering the "sampling without replacement" part of the assumption, the number of distinct instances of E_2 hit with K instances of E_1 is KN_2/N_1.

In addition, more instances of the remaining $N_2 - KN_2/N_1$ instances of E_2 will be hit (due to the "sampling with replacement" part of the above assumption). This is addressed next.

(ii) With the uniform-distribution assumption, with one instance of $E1$, the probability of hitting any of the remaining $N_2 - N_2 / N_1$ of E_2 is given by

$$P(\text{a remaining instance of } E_2 \text{ hit by one instance of } E_1) = \frac{R_1 - N_2/N_1}{N_2 - N_2/N_1}$$

and

$$P(\text{a remaining instance of } E_2 \text{ not hit by one instance of } E_1) = 1 - \frac{R_1 N_1 - N_2}{N_2 N_1 - N_2}.$$

Since there are K instances of E_1, the probability of an instance of E_2 not being hit is

$$P(\text{a remaining instance of } E_2 \text{ not hit by } K \text{ instances of } E_1) = \left(1 - \frac{R_1 N_1 - N_2}{N_2 N_1 - N_2}\right)^K.$$

and then,

$$P(\text{a remaining instance of } E_2 \text{ hit with } K \text{ instances of } E_1) = 1 - \left(1 - \frac{R_1 N_1 - N_2}{N_2 N_1 - N_2}\right)^K.$$

Since only the instances of E_2 that were not hit in part (i) are being considered, the additional instances of E_2 hit are

$$\left(N_2 - \frac{KN_2}{N_1}\right)\left[1 - \left(1 - \frac{R_1 N_1 - N_2}{N_2 N_1 - N_2}\right)^K\right].$$

Adding the entity-E_2 instances hit from part (i) and part (ii), the total entity-E_2 instances hit are seen to be

$$L = \frac{KN_2}{N_1} + \left(N_2 - \frac{KN_2}{N_1}\right)\left[1 - \left(1 - \frac{R_1 N_1 - N_2}{N_2 N_1 - N_2}\right)^K\right]. \qquad (8)$$

Note that the first term in this expression is linear and deterministic, while the second term is a binomial variable with parameters $n = N_2 - KN_2 / N_1$ and

$$p = 1 - \left(1 - \frac{R_1 N_1 - N_2}{N_2 N_1 - N_2}\right)^K.$$

Evaluating from Equation (8),

when $K = 0$, $L = 0$;

when $K = N_1$, $L = N_2$;

when $K = 1$, after much simplification, $L = R_1$.

Thus the expression in Equation (8) meets the stated requirements. Given the expression for L, the numbers of page accesses required to satisfy the query are written as

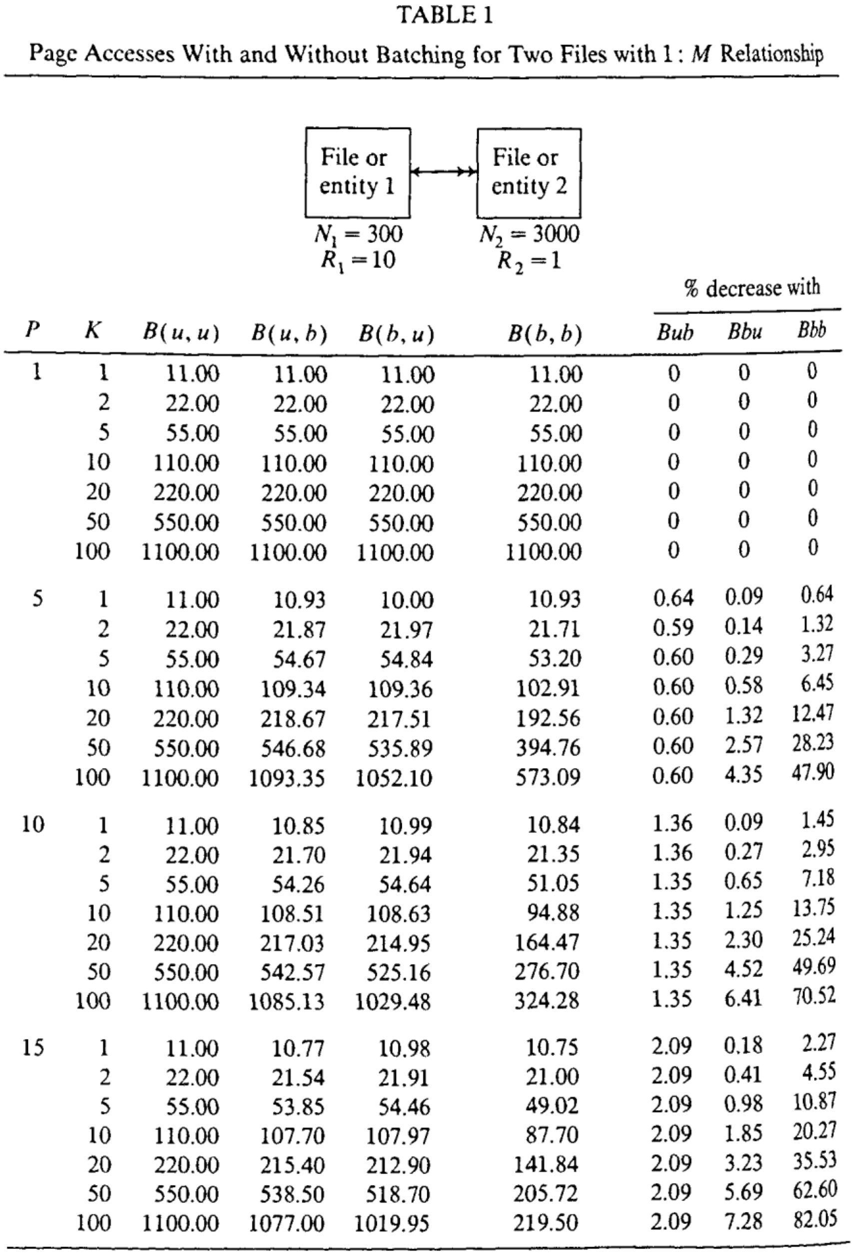
$$B(b,u)_{M:N} = F(K, N_1, M_1) + L, \qquad (9)$$

$$B(b,b)_{M:N} = F(K, N_1, M_2) + F(L, N_2, M_2). \qquad (10)$$

The value of L in the above two equations is given by Equation (8). Equation (9) is for batching in file F_1 and unbatching in file F_2 ; and Equation (10) is for batching in both files. Note that Equations (9) and (10) are generalizations of Equations (5) and (6), as the value of L in Equation (8) is a generalization of the value of L in Equation (4).

TABLE 1

Page Accesses With and Without Batching for Two Files with 1 : *M* Relationship



$N_1 = 300$
$R_1 = 10$

$N_2 = 3000$
$R_2 = 1$

P	*K*	*B*(*u*, *u*)	*B*(*u*, *b*)	*B*(*b*, *u*)	*B*(*b*, *b*)	% decrease with *Bub*	*Bbu*	*Bbb*
1	1	11.00	11.00	11.00	11.00	0	0	0
	2	22.00	22.00	22.00	22.00	0	0	0
	5	55.00	55.00	55.00	55.00	0	0	0
	10	110.00	110.00	110.00	110.00	0	0	0
	20	220.00	220.00	220.00	220.00	0	0	0
	50	550.00	550.00	550.00	550.00	0	0	0
	100	1100.00	1100.00	1100.00	1100.00	0	0	0
5	1	11.00	10.93	10.00	10.93	0.64	0.09	0.64
	2	22.00	21.87	21.97	21.71	0.59	0.14	1.32
	5	55.00	54.67	54.84	53.20	0.60	0.29	3.27
	10	110.00	109.34	109.36	102.91	0.60	0.58	6.45
	20	220.00	218.67	217.51	192.56	0.60	1.32	12.47
	50	550.00	546.68	535.89	394.76	0.60	2.57	28.23
	100	1100.00	1093.35	1052.10	573.09	0.60	4.35	47.90
10	1	11.00	10.85	10.99	10.84	1.36	0.09	1.45
	2	22.00	21.70	21.94	21.35	1.36	0.27	2.95
	5	55.00	54.26	54.64	51.05	1.35	0.65	7.18
	10	110.00	108.51	108.63	94.88	1.35	1.25	13.75
	20	220.00	217.03	214.95	164.47	1.35	2.30	25.24
	50	550.00	542.57	525.16	276.70	1.35	4.52	49.69
	100	1100.00	1085.13	1029.48	324.28	1.35	6.41	70.52
15	1	11.00	10.77	10.98	10.75	2.09	0.18	2.27
	2	22.00	21.54	21.91	21.00	2.09	0.41	4.55
	5	55.00	53.85	54.46	49.02	2.09	0.98	10.87
	10	110.00	107.70	107.97	87.70	2.09	1.85	20.27
	20	220.00	215.40	212.90	141.84	2.09	3.23	35.53
	50	550.00	538.50	518.70	205.72	2.09	5.69	62.60
	100	1100.00	1077.00	1019.95	219.50	2.09	7.28	82.05

5. EXPERIMENTAL COMPARISON

In order to gain an appreciation for the savings due to batching, the values of $B(u,u)$, $B(u,b)$, $B(b, u)_{1: M}$, $B(b, b)_{1 : M}$ are reported in Table 1 for a 1: *M* relationship. In this database, file F_1 has 300 records and file F_2 has 3000 records, and the relationship between F_1 and F_2 is 1 : 10. The same data for an *M* : *N* relationship, i.e. $B(u,u)$, $B(u,b)$, $B(b,u)_{M : N}$, $B(b,b)_{M : N}$, are reported in Table 2 for a database with file F_1 having 300 records and file F_2 having 120 records. The *M* : *N* relationship between files F_1 and F_2 is 10:4. All identical blocking factor is used for the two files. The *B* values are computed for blocking factors of 1, 5, 10, and 15, and *K* values of 1, 2, 5, 10, 20, 50, and 100. The last three columns in Tables 1 and 2 show the percentage savings of $B(u, b)$, $B(b, u)$, and $B(b, b)$ with respect to $B(u, u)$. Some comments based on the data in the two tables are in order, and are made below:

(A) Batching pays off only when the number of records to be retrieved from either file is more than one; the higher the number, the greater is the saving. The percentage savings of $B(u, b)$, $B(b, u)$, and $B(b, b)$ are only modest for low values of K and start increasing as *K* increases. However, even when $K = 1$ (i.e. only one record retrieved from file F_1), there are still some savings in the number of database accesses. This is because the number of related records accessed from file F_2 is still more than one (e.g. 10 in Table 1 and 4 in Table 2).

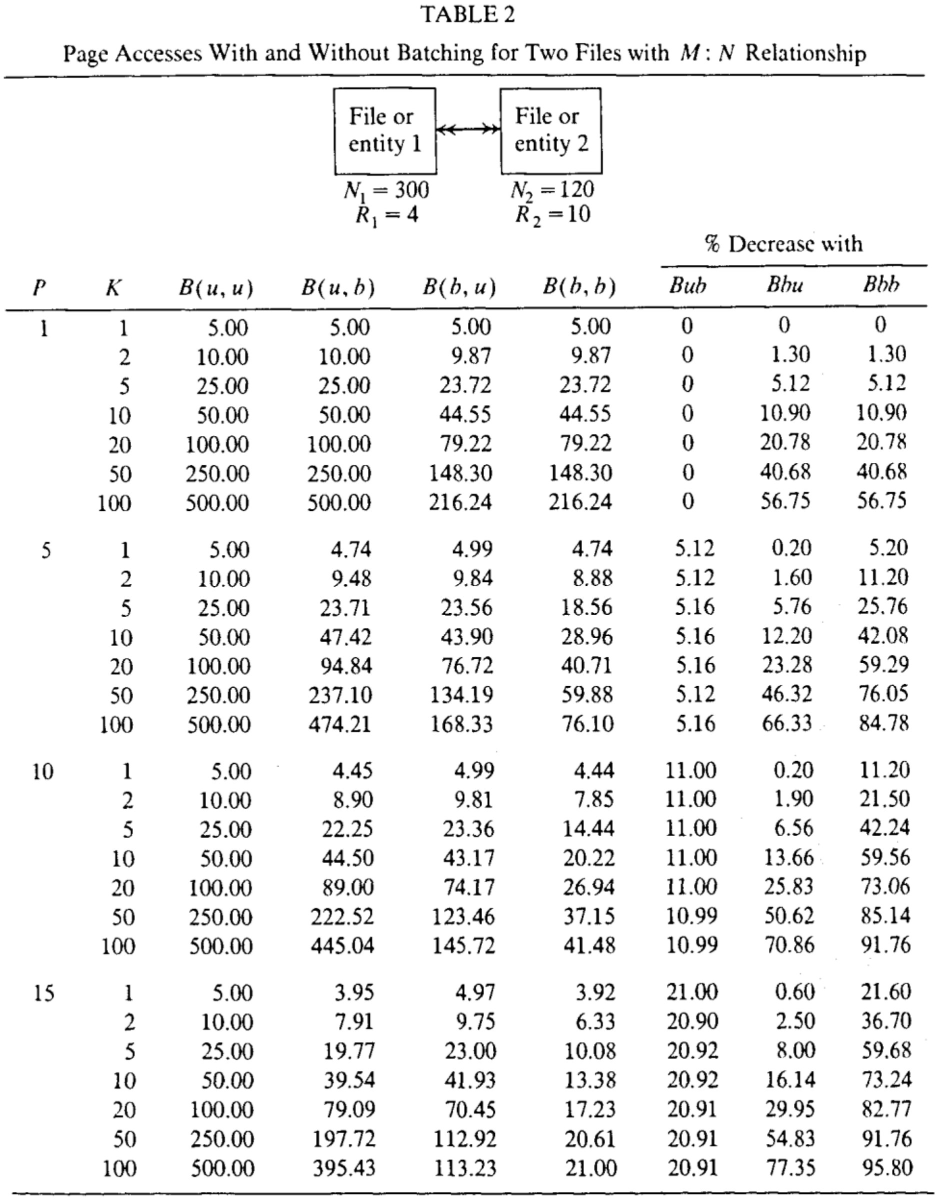


TABLE 2

Page Accesses With and Without Batching for Two Files with $M:N$ Relationship

File or entity 1 ↔ File or entity 2

$N_1 = 300$, $R_1 = 4$

$N_2 = 120$, $R_2 = 10$

						% Decrease with		
P	K	$B(u,u)$	$B(u,b)$	$B(b,u)$	$B(b,b)$	Bub	Bhu	Bbb
1	1	5.00	5.00	5.00	5.00	0	0	0
	2	10.00	10.00	9.87	9.87	0	1.30	1.30
	5	25.00	25.00	23.72	23.72	0	5.12	5.12
	10	50.00	50.00	44.55	44.55	0	10.90	10.90
	20	100.00	100.00	79.22	79.22	0	20.78	20.78
	50	250.00	250.00	148.30	148.30	0	40.68	40.68
	100	500.00	500.00	216.24	216.24	0	56.75	56.75
5	1	5.00	4.74	4.99	4.74	5.12	0.20	5.20
	2	10.00	9.48	9.84	8.88	5.12	1.60	11.20
	5	25.00	23.71	23.56	18.56	5.16	5.76	25.76
	10	50.00	47.42	43.90	28.96	5.16	12.20	42.08
	20	100.00	94.84	76.72	40.71	5.16	23.28	59.29
	50	250.00	237.10	134.19	59.88	5.12	46.32	76.05
	100	500.00	474.21	168.33	76.10	5.16	66.33	84.78
10	1	5.00	4.45	4.99	4.44	11.00	0.20	11.20
	2	10.00	8.90	9.81	7.85	11.00	1.90	21.50
	5	25.00	22.25	23.36	14.44	11.00	6.56	42.24
	10	50.00	44.50	43.17	20.22	11.00	13.66	59.56
	20	100.00	89.00	74.17	26.94	11.00	25.83	73.06
	50	250.00	222.52	123.46	37.15	10.99	50.62	85.14
	100	500.00	445.04	145.72	41.48	10.99	70.86	91.76
15	1	5.00	3.95	4.97	3.92	21.00	0.60	21.60
	2	10.00	7.91	9.75	6.33	20.90	2.50	36.70
	5	25.00	19.77	23.00	10.08	20.92	8.00	59.68
	10	50.00	39.54	41.93	13.38	20.92	16.14	73.24
	20	100.00	79.09	70.45	17.23	20.91	29.95	82.77
	50	250.00	197.72	112.92	20.61	20.91	54.83	91.76
	100	500.00	395.43	113.23	21.00	20.91	77.35	95.80

(B) The advantages of batching are obtained only with high blocking factors. As is seen in Table 1, the savings are nil with a blocking factor of 1 and start multiplying with higher blocking factors. This is due to each page containing only one record when the blocking factor is one; consequently the same number of pages as the required number of records have to be retrieved from the file to satisfy a query. Note that for an $M : N$ relationship, as in Table 2, there are still savings with a blocking factor of one. This is because many of the required file-F_2 records are nonunique and only unique records require new page accesses.

(C) Comparing the data in Tables 1 and 2, it is clear that the savings due to batching are higher for databases with $M : N$ relationships between files than for databases with $1 : M$ relationships. This is because, as indicated earlier, batching helps in two ways. In both $1 : M$ and $M : N$ relationships, hatching helps with high blocking factors, as the same page may have several required records and that page has to be retrieved only once. Second, in files with $M : N$ relationships, the records of one file related to a given number of records of another file may be replicated. Batching these nonunique records will require accessing only the unique records, which will generally be a smaller number.

(D) The most savings are obtained when batching is used on both files. This strategy should be adopted whenever possible. However, this may not be possible given the users' requirements. For example, the requests for records of file F_1 may come at different times, and users may want immediate action. Even if the K requests on file F_1 may be unbatched, the related records in file F_2 should still be batched to achieve some savings. It would be generally possible, within the programming constructs, to batch the related records in file F_2 .

(E) This example is for a simple database with only two interrelated files. Many real databases will contain several interrelated files. Batching would propagate similar savings for more complex queries involving retrieval of records from these several files. Especially for files having many $M : N$ relationships, such savings would multiply.

We now put our results in perspective, by discussing them in the wider context of technology and the constraints imposed by batching.

6. DISCUSSION AND APPLICABILITY OF RESULTS

A major implicit assumption made in prior works on batching [5, 13, 14, 16, 19, 20] as well as in ours is that the number of pages accessed into main memory is equal to the number of distinct pages on which the desired records are found. However, this will hold true for batched searching of random files only in two cases. First, if the desired records are searched in their physical page order, then a page which has been accessed once need not be accessed subsequently. If this requirement is met, then batching will naturally pay. (Note that this requirement is easily met in batched searching of sequential and hierarchical files.) However, this requirement may not be easily attainable in random files, as the physical ordering of the records will be randomized and the randomizing function may be system controlled and not easily available.

If the above (first case) requirement is not met, then for batching to pay (i.e. the second case), the main memory/buffer sizes has to be large enough to accommodate all the required pages for a given query all at once. This will ensure that once a page has been retrieved into main memory, it will not have to be retrieved again. We discuss the implications of memory sizes below.

(A) The main memory poses restrictions only when a lot of data have to be brought into it. 'Therefore, the batching approach merits consideration when the data requirements are not large. This is true for small databases and many ad hoc queries. Such queries generally address a very small fraction of records of the interrelated files. In such a case, the necessary records and pointers required would fit into the buffer spaces of current main memories.

(B) The large main memory is required to take full advantage of batching. However, limited main memory will still offer batching advantages, although less than the full advantage. Two approaches are possible with limited main memory. One is to keep the batch size as it is; this will have the effect of generating additional page accesses, although there will still be savings com-pared to the unbatched case. These savings are documented in [12]. Another approach to study, which has potential for savings, is to split the batch into subbatches of smaller sizes. The size of the subbatch will depend on the memory size, and the subbatch will be processed all at once.

(C) Main-memory sizes have been steadily increasing in the past years, and the future for very large main memories looks bright. The main-memory sizes are increasing at geometric rates (approximately by ten times every five years). For example, at the time of this writing, it is not uncommon to see personal computers with a million bytes of memory and mainframes with tens or hundreds of millions of bytes. Several recent papers [4, 6, 7] have convincingly argued that main memories of gigabyte capacity are now feasible. Such large memories are even permitting exploration of main-memory prototype database systems [4]. With such large main memories, batching can be used successfully and the main memory will no longer be a bottleneck. While main-memory databases may be distant for commercial application, batching may be used as an intermediate strategy to exploit the availability of large main memories.

7. CONCLUSIONS

Past research has recommended batching in a file organization as a means of reducing total number of secondary memory accesses. Similar savings can be obtained in a database organization where the potential for savings is even greater because of the multitude of interrelated files existing in the database. A simple database consisting of two interrelated files was used in this paper to demonstrate such savings. Several equations were developed for computing the total number of page accesses for various combinations of batched and un. batched records. Experimental data were generated for the various combinations, and the potential savings due to batching were reported. In larger databases, savings due to batching could be substantial, depending on the characteristics of the database, especially the relationships between the interrelated files of the database. Further, the availability of very large main memories makes the batching strategy more attractive.

Notes:

1 If R_1 and R_2 are average outdegrees, then the identity $N_1R_1 = N_2R_2$ must hold.

REFERENCES

1. H. D. Anderson and P. B. Berra, Minimum cost selection of secondary indexes for formatted files, *ACM Trans. Database Systems* 2, No. 1 (1977).
2. D. S. Batory and C. C. Gotlieb, A unifying model of physical databases, *ACM Trans Database Systems* 7:4 (1982).
3. D. S. Batory, On searching transposed files, *ACM Trans. Database Systems* 4, No 4 (1979).
4. Dina Bitton, The effect of large main memory on database systems (panel report), in *Proceedings ACM-SIGMOD*, 1986.
5. A. F. Cardenas, Analysis and performance of inverted database structures, *Comm.* ACV 18, No. 5 (May 1975).
6. D. DeWitt, R. Katz, F. Olken, L. Shapiro, M. Stonebraker, and D. Wood, Implementation techniques for main memory database systems, in *Proceedings ACM-SIGMOD*, 1984.
7. Hector Garcia-Molina, A massive main memory machine, *IEEE Trans. Comput.* No. 5 (May 1984).
8. M. Hammer and A. Chan, Index selection in a self-adaptive data base management. In *Proceedings of the ACM 1976 SIGMOD International Conference on Management of Data.* Washington, 1976.
9. J. Hoffer, Database design practices for inverted files, *Inform. and Management* 3 (1980)
10. R. H. Katz and E. Wong, Resolving conflicts in global storage design through replication. *ACM Trans. Database Systems* 8, No. 1 (Mar. 1983).
11. S. T. March, A mathematical programming approach to the selection of access paths for large multiuser databases, *Decision Sci.* 14, No. 4, 1983.
12. P. Palvia, The Effect of Buffer Size on Pages Accessed in Random Files, Working Papa Dept. of Management and MIS, Memphis State Univ., 1986.
13. P. Palvia, Expressions for batched searching of sequential and hierarchical files,,401 *Trans. Database Systems* 10, No. 1 (Mar. 1985).
14. P. Palvia and S. T. March, Approximation block accesses in database organizations *Inform. Process. Lett.* 19, Aug. 1984.
15. D. G. Severance. A parametric model of alternative file structures, *Inform. Systems 1,* No 2 (1975).
16. B. Shneiderman and V. Goodman, Batched searching of sequential and tree structure files, ACM *Trans. Database Systems* 1, No. 3 (Sept. 1976).
17. S. B. Yao and D. Deicing, Evaluation of database access cost analysis, in *Proceedings of the ACM International Conference on Management of Data,* Austin, Tex., 1978.
18. S. B. Yao, An attribute based model for database access cost analysis, *ACM Trans Database Systems* 2, No. 1 (1977).
19. S B. Yao, Approximating block accesses in database organizations, *Comm. ACM 20,* No, 4 (Apr. 1977).
20. P.C. Yue and C. K. Wong, Storage cost considerations in secondary index selection, *Internat. J. Comput. Inform. Sci.* 4, No. 4 (1975).