B+ Trees and Indexed Sequential Files: A Performance Comparison*

D.S. Batory
Computer and Information Sciences Department
University of Florida
Gainesville, Florida, USA

Abstract

An analytic method for comparing the performance of B+ trees and indexed sequential files is proposed. Preliminary results indicate that indexed sequential files may be more efficient than B+ trees in certain applications.

1. INTRODUCTION

A familiar problem in database and file design is choosing between a B+ tree and an indexed sequential implementation for a file. Over the past ten years, a variety of evidence has accumulated in favor of B+ trees. Support has come from surveys ([Com79]), experience ([Ston80]), simulation studies ([Rei76]), and analytic results ([Knu73],[Yao78]). Evidence in favor of indexed sequential files has been limited ([HeSt78]).

Some of the primary advantages cited for the popularity of B+ trees are that reorganizations are unnecessary, algorithms are simple, and performance is good even under adverse conditions. Yet for the exception of reorganizations, indexed sequential files also have simple algorithms and can have good performance. Merely listing the advantages and disadvantages of each structure is not sufficient to make a good decision. It would be more useful to compare the performance of both structures under the conditions of anticipated file usage in order to choose that structure which performs better. Unfortunately, such a comparison of B+ tree and indexed sequential performance has not been adequately addressed in the literature.

Recent techniques for the analytic modeling of file evolution and file reorganization enable such comparisons to be made (see [Bat80a-b]). Although comparisons could be based on simulation studies, analytic comparisons are much less expensive. In this paper, some of these techniques are reviewed and a methodology for comparing B+ tree and indexed sequential file performance is proposed.

*This work was completed while the author was at the Department of Computer Science, University of Toronto, Toronto, Ontario, Canada.

2. ESTIMATING PERFORMANCE OF EVOLVING FILES

The basic component of a file structure is a node which contains zero or more records. The records of a node are stored in a primary block and on a linear list called an overflow chain (Fig. 1). Nodes of B+ trees have overflow chains of zero length; nodes of indexed sequential files have chains of variable length.

File structures have been identified with uniform height directed trees where each vertex of a tree corresponds to a node ([Yao77]). The data level or leaf level, level 0, is where all data records reside; all higher levels constitute the cluster index of the structure (Fig. 2). Nodes that possess overflow records are usually confined to the data level. The topmost node, or root node, is main memory resident; all other nodes are secondary storage resident.

File structures are modeled by a collection of parameters such as those given in Table 1. A specific file is described by the values that are assigned to these parameters. This collection of values is called the file's descriptor. Figures 3 and 4 illustrate an indexed sequential and a B+ tree file with their descriptors.

File performance can be estimated by cost functions that accept a file descriptor as input and return estimates (or bounds on) the expected cost of executing designated file operations. For a file with descriptor $\mathcal{F}$, some common operations and their expected cost functions are:

cost function	file operation
RET($\mathcal{F}$,ONE)	retrieving a record given its identifier
RET($\mathcal{F}$,SCAN)	retrieving all records
INS($\mathcal{F}$)	inserting a record
DEL($\mathcal{F}$)	deleting a record given its identifier
UPD($\mathcal{F}$)	updating a previously retrieved record

Expressions for these functions are given in Appendices I and II. Note that other operations, such as batched retrieval and updating, could also be included in this list.

Files evolve due to record insertions and deletions. Performance deterioration, which often accompanies file evolution, occurs when operations

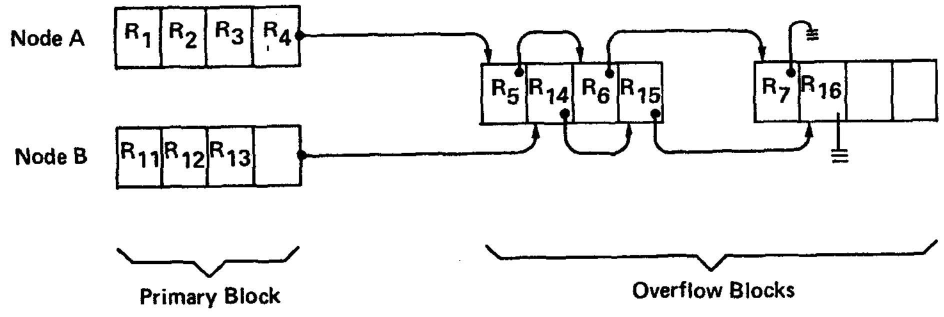
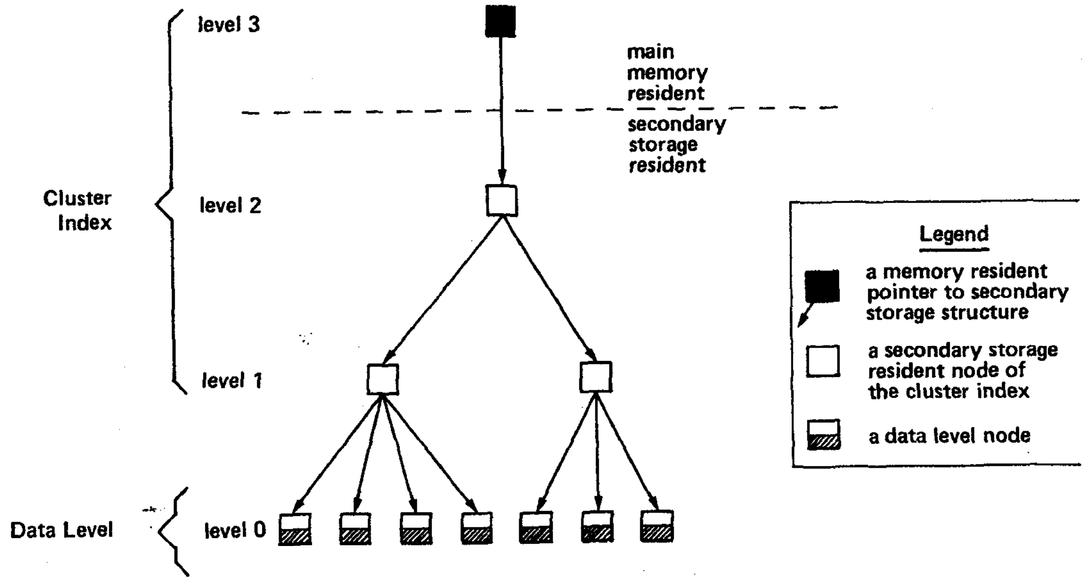


Figure 1. Structure of a Node with Overflow Records

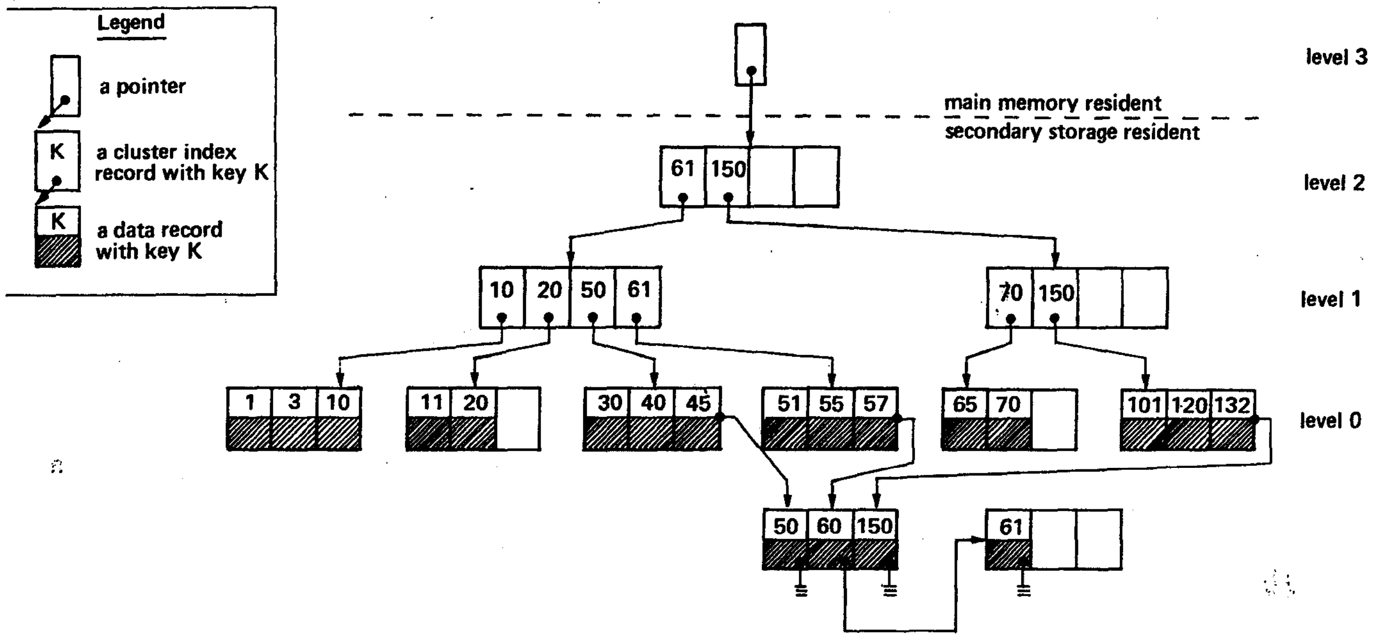


level 3
main memory resident
secondary storage resident
Cluster Index
level 2
level 1
Data Level
level 0
Legend
a memory resident pointer to secondary storage structure
a secondary storage resident node of the cluster index
a data level node

Figure 2. A Directed Tree Model of a File Structure

	design parameters
Fs	file structure: indexed sequential or B+ tree
R	record capacity of a data level block
Ri	record capacity of a cluster index block
M	minimum record occupancy of a data level node
Mi	minimum record occupancy of a cluster index node
	file parameters
N	number of records in file
L	height of file structure
Z	expected number of data level nodes
H	expected number of records in a primary block on the data level
G	expected number of records on an overflow chain on the data level
Ω	expected number of overflow records accessed when locating a single record within a data level node
Pfull	probability of inserting a record into a data level node whose primary block has no vacant record slots
Pund	probability that a record deletion will cause a data level node to underflow
Pmer	probability that when a data level node underflows, a node merging will occur
	cost parameters
S	storage cost of a block per unit time
A	transfer cost of a block

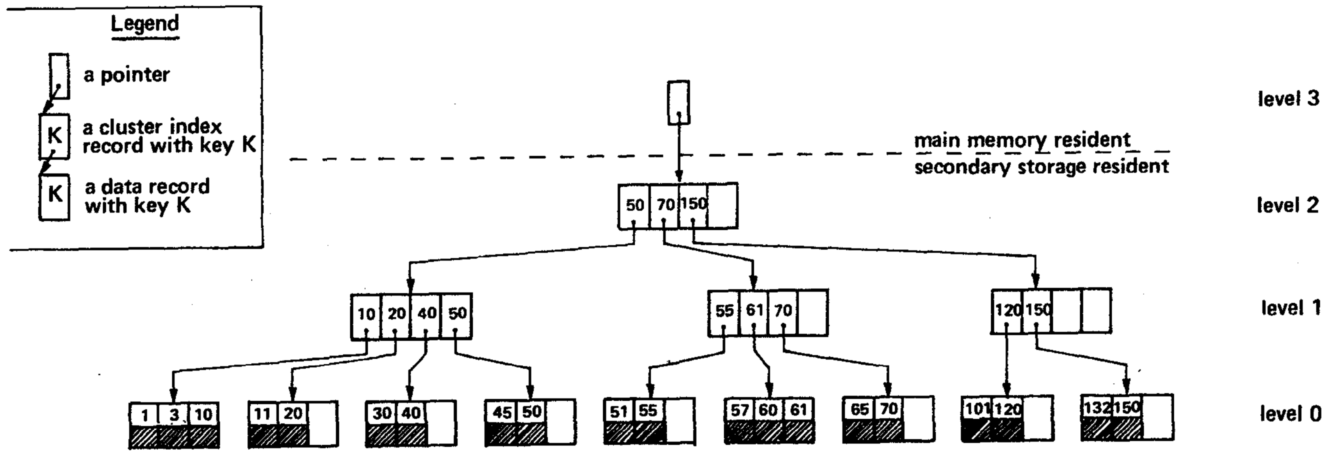
Table 1. Parameters of a File Descriptor



Fs	R	M	Ri	Mi	A	S
indexed sequential	3	0	4	1	A	S

N	L	Z	H	G	Ω	Pfull	Pund	Pmer
20	3	6	16/6	4/6	5/20	16/20	0	0

Figure 3. An Indexed Sequential File



Fs	R	M	Ri	Mi	A	S
B+ tree	3	2	4	2	A	S

N	L	Z	H	G	Ω	Pfull	Pund	Pmer
20	3	9	20/9	0	0	6/20	14/20	7/9

Figure 4. A B+ Tree

become more expensive to execute. For example, expected record retrieval costs increase as overflow chains become longer.

As a file evolves, values assigned to selected parameters of a file descriptor will change. Specifically, these are the file parameters of Table 1. Once it is known how a descriptor evolves, cost functions can be used to trace the evolution or deterioration of a file's performance.

The values assigned to file parameters are statistics of a distribution called a node occupancy distribution. Node occupancy distributions take the form T(r) = number of data level nodes that contain r records.[1] For example, the parameters Z and N have the following definitions:

$$Z = \text{number of data level nodes} = \sum_r T(r)$$

$$N = \text{number of records in file} = \sum_r r \times T(r)$$

Definitions for other file parameters are given in Appendix III. Owing to these relationships, the problem of modeling file evolution is identified with the problem of estimating node occupancy distributions.

General techniques are presented in [Bat80b] for estimating node occupancy distributions for hash based, indexed sequential, and B+ tree files, among others. Although it is beyond the scope of this paper to develop the techniques for deriving distribution equations, it is possible to summarize some relevant results. Let N_o be the initial file size and let s be the initial number of records stored in each data level node of a B+ tree or indexed sequential structure. (s is said to be the initial loading factor.) For an insertion only environment (i.e., no deletions), the node occupancy distribution for an indexed sequential file is:

P(I,r) = number of data level nodes containing r records after I records have been inserted

$$= \frac{N_o \binom{N_o - 1}{s} \binom{I}{r-s}}{r \binom{N_o + I - 1}{r}}$$

Note: $\binom{a}{b}$ is a binomial coefficient.

and for B+ trees (see [NaMi78]):

Q(I,r) = number of data level nodes containing r records after I records have been inserted

At B+ tree creation:

$$Q(0,r) = \begin{cases} N_o/s & \text{if } r=s \\ 0 & \text{otherwise} \end{cases}$$

Because overflow chains are not used, the value of s is constrained to be between the minimum and maximum number of records per node: $M \le s \le R$.[2] For other Q(I,r):

$$Q(I+1,M) = Q(I,M)\left(1 - \frac{M}{N_o+I}\right) + Q(I,R)\left(\frac{R}{N_o+I}\right)$$

$$Q(I+1,M+1) = Q(I,M+1)\left(1 - \frac{M+1}{N_o+I}\right) + Q(I,R)\left(\frac{R}{N_o+I}\right) + Q(I,M)\left(\frac{M}{N_o+I}\right)$$

$$Q(I+1,r) = Q(I,r)\left(1 - \frac{r}{N_o+I}\right) + Q(I,r-1)\left(\frac{r-1}{N_o+I}\right)$$

index ranges: $M+1 < r \le R$, $I \ge 0$

The above equations are valid only when the record capacity R of a primary block is even. For odd R, a node splits into two nodes with identical record populations. The corresponding equations are:

$$Q(I+1,M) = Q(I,M)\left(1 - \frac{M}{N_o+I}\right) + 2Q(I,R)\left(\frac{R}{N_o+I}\right)$$

$$Q(I+1,r) = Q(I,r)\left(1 - \frac{r}{N_o+I}\right) + Q(I,r-1)\left(\frac{r-1}{N_o+I}\right)$$

index ranges: $M < r \le R$, $I \ge 0$

No closed form solutions to these equations are known.

The equations defining P(I,r) and Q(I,r) were developed from the same assumptions (specifically (A3)) as the cost functions of Appendices I and II. Moreover, P(I,r) and Q(I,r) were validated by simulation studies. Equations that model the impact of insertions and deletions on node occupancy distributions are, in general, very complex. It is beyond the scope of this paper to present these equations, and for this reason we will confine our experiments to files that do not experience deletions.

P(I,r) and Q(I,r) estimate node occupancy distributions after I records have been inserted. Applying the definitions of Appendix III to an estimated distribution yields values for the file parameters. In this way, the evolution of a file descriptor is modeled.

Because values of a file descriptor change with time, let $\mathcal{F}_t$ denote the descriptor of a file at (the end of) time period t. $\mathcal{F}_0$ describes the file at creation time (i.e., before insertions and deletions occur). Also, let $N^{(\mathcal{F})}$ denote the number of records in a file whose descriptor is $\mathcal{F}$. To characterize the usage of a file, we introduce the following statistics:

fret	= number of times each record is retrieved via its identifier per time period
fsc	= number of file scans per time period
fins	= number of insertions per time period
fdel	= number of deletions per time period
fupd	= number of times each record is retrieved and updated per time period

Let STOR($\mathcal{F}$) be the storage cost of a file with descriptor $\mathcal{F}$ (see Appendices I and II). If a

[1]A node occupancy distribution may be viewed as an unnormalized probability distribution whose normalization constant is Z.

[2]For B+ trees, $M = \left\lfloor \frac{R+1}{2} \right\rfloor$

descriptor remains constant over a time period, the cost of using the file during period t is:

$$\begin{aligned}\text{Static}(F_t) = \ & (\text{fret}+\text{fupd})\times N^{(F_t)}\times \text{RET}(F_t,\text{ONE}) \\ & + \text{fsc}\times\text{RET}(F_t,\text{SCAN}) + \text{fins}\times\text{INS}(F_t) \\ & + \text{fdel}\times\text{DEL}(F_t) + \text{fupd}\times N^{(F_t)}\times\text{UPD}(F_t) \\ & + \text{STOR}(F_t)\end{aligned}$$

However, since file descriptors do evolve, the usage cost of a file during period t is more closely approximated by:

$$\text{Usage\_Cost}(\vec{F}_t) = \frac{\text{Static}(F_{t-1}) + \text{Static}(F_t)}{2}$$

where $\vec{F}_t$ denotes the descriptor pair $\{F_{t-1}, F_t\}$.

The primary objective of our study is to compare B+ tree and indexed sequential file performance. This will be accomplished, in part, by evaluating and comparing Usage_Costs of the two structures. Before this can be done, however, we need to know when an indexed sequential file should be reorganized. Once this is known, a fair comparison can be made.

3. A SOLUTION TO THE FILE REORGANIZATION PROBLEM

File reorganization is the problem of determining when a file should be reorganized. A closely related problem is loading factor selection, i.e., choosing an initial loading factor for a file. In the following paragraphs, a method for determining optimal reorganization points and loading factors for files with fixed lifetimes will be developed. We begin by presenting a solution to the file reorganization problem.[3]

Let the lifetime of a file be T time periods. At the end of each period, a decision is made to reorganize the file or not. The initial loading factor of a reorganized file is the constant s.

Let $(i,j) be the sum of the costs of: 1) constructing the file at the end of period i, 2) using the file during periods i+1 through j, and 3) dumping the file at the end of period j. Let $F_{i,t}$ be the descriptor of a file at the end of period t, where the file was constructed at the end of period i, and let CON(F) and DUMP(F) be the costs of constructing and dumping a file with descriptor F (see Appendix I). $(i,j) is given by:

$$\$(i,j) = \text{CON}(F_{i,i}) + \sum_{t=i+1}^{j} \text{Usage\_Cost}(\vec{F}_{i,t}) + \text{DUMP}(F_{i,j})$$

where $\vec{F}_{i,t}$ denotes the descriptor pair $\{F_{i,t-1}, F_{i,t}\}$.

Let Cost(t) be the minimal usage and reorganization cost for a file with a lifetime of t time periods. Clearly,

$$\text{Cost}(0) = 0$$

$$\text{Cost}(1) = \$(0,1)$$

To estimate Cost(2), note that the last reorganization was either at the end of period 0 or period 1. Choosing the situation with minimal cost yields:

$$\text{Cost}(2) = \min[\ \text{Cost}(0) + \$(0,2),\ \text{Cost}(1) + \$(1,2)\ ]$$

and in general,

$$\text{Cost}(t) = \min_{0\le i<t} [\ \text{Cost}(i)+\$(i,t)\ ] \qquad (1)$$

By incrementing the index t, and progressively building upon previous results, Cost(T) is obtained. Recording the value of i used in each Cost(t) calculation, for which Cost(i)+$(i,t) is minimal, determines the times at which the file should be reorganized. These are the optimal reorganization points for the file.

The loading factor selection problem is integrated into this framework by allowing s, the loading factor of a reorganized file, to become an optimization variable. $(i,j) becomes $(s,i,j), and (1) becomes:

$$\text{Cost}(t) = \min_{0\le i<t} [\ \text{Cost}(i) + \min_{s}\{\ \$(s,i,j)\ \}] \quad (2)$$

Recording the values of i and s used in each Cost(t) calculation, for which Cost(i)+$(s,i,t) is minimal, determines the optimal initial loading factors and reorganization points for the file.

Equation (2) can be evaluated efficiently using dynamic programming techniques (see [Ram80],[AHU74]).

4. EXPERIMENTAL RESULTS ON A HYPOTHETICAL FILE

In the following paragraphs, we present results of a number of different computation experiments. These experiments are not intended to be comprehensive; their purpose is to indicate the types of analyses and results possible using the techniques described in this paper.

Consider the indexed sequential file that is described by the values of Table 2. The file has an initial size of 20000 records, an insertion rate of 400 records per week, and a lifetime of 25 weeks. The block storage and access cost figures reflect current charging rates at the University of Florida's main computing center.[4] Solving equation (2) for the optimal loading factors and reorganization

[3]The proposed solution for file reorganization presented here is similar to that in recently circulated works by Ramirez ([Ram80]) and Hatzopoulos and Kollias ([HaKo80]). The approaches in these works are respectively based on data structures and index selection, quite different from our approach, but the method for determining reorganization points is similar.

[4]It is worth noting that the computing center charges significantly higher storage rates for files that are stored "online" than for files that are stored "offline". ("Online" files can be accessed by interactive programs; "offline" files can only be accessed accessed by batch programs.) We are considering an "online" file.

Descriptor Values: N_0=20000, R=20, M=0,
Ri=255, Mi=1, A=.00131, S=.05683
Usage Statistics: fret=fupd=.5/week,
fins=400/week, fdel=0/week
fsc=7/week
File Lifetime: T=25 weeks

Table 2. An Indexed Sequential File

points, the optimal strategy is to construct and reorganize the file using the loading factor of 19 records/node. Thus, the primary block of each data level node will contain one vacant record slot. Reorganizations occur at the end of weeks 2,4,6,8, 10,13,16,19, and 22. With this information, and using the equations developed in Section 2 and Appendix I, the performance evolution of the indexed sequential file can be predicted.

The height (L) of a B+ tree is bounded by:

$$\lceil \log_{(Ri)} Z \rceil + 1 \le L \le \left\lfloor \log_{(Mi)} \frac{Z}{2} \right\rfloor + 2$$

Occasionally, the height is well defined in that the upper bound equals the lower bound. Uncertainty arises when record insertions may cause the height of the tree to increase by one level; in such cases the upper bound and lower bound disagree. Our analysis of B+ trees is directed to those situations where insertions do <u>not</u> cause an increase in height. Predicting the expected height of B+ trees appears to be a formidable task, and it is left as an open problem. But more on this later.

Consider the B+ tree that is described by the values of Table 3. This file is identical to the file of Table 2; only the implementations differ. When the B+ tree is created, we assume that the initial loading factor (s) is 20, so that the storage utilization is 100%.[5] Using the equations developed in Section 2 and Appendix II, the performance evolution of the B+ tree can be predicted.

Our comparison begins with Figure 5 where the expected storage volume of both structures are displayed. Because records are densely packed in an indexed sequential file, there is little free space. Consequently, the storage curve reflects the linear increase in file growth. It is interesting to note the after a reorganization, the storage volume <u>increases</u> due to the vacant slot allocated per primary block. As for the B+ tree, the storage volume quickly balloons from a storage utilization of 100% at week 0, to 59% at week 7, and finally to 73% at week 25. This follows since the first few insertions cause nodes to split (originally all nodes were full), which effectively halves the initial storage utilization. Because a significant amount of free space now exists, further insertions cause few splits, so the storage volume remains essentially constant.

[5]The initial loading factor of 100% was chosen arbitrarily. Although initial loading factors do influence the initial performance of a B+ tree, the impact decreases with time (see [NaMi78],[Yao78]). A problem which is not addressed in this paper, but could be using the techniques that are presented, is determining good initial loading factors for B+ trees.

Descriptor Values: N_0=20000, R=20, M=10,
Ri=255, Mi=128, A=.00131, S=.05683
Usage Statistics: fret=fupd=.5/week,
fins=400/week, fdel=0/week
fsc=7/week
File Lifetime: T=25 weeks

Table 3. A B+ Tree File

In Appendix II, equations defining upper and lower bounds on expected B+ tree storage volume are presented. In our example, these bounds were so close that, for practical purposes, either estimate could be taken as the true storage volume.

Figure 6 displays the expected number of block accesses needed to retrieve all records of both structures. Such an operation is called a <u>scan</u>. The sawtoothed curve clearly illustrates the effects of reorganization; after each reorganization, overflow records are removed and the cost of a scan declines sharply. It is worth noting that at the end of week 13, the average length of an overflow chain reaches its maximum value, .35. Clearly, overflow chains do not have to be long to significantly impact the cost of a scan. Also note that .35 is specific to our problem. It is <u>not</u> a characteristic of all indexed sequential files that are about to be reorganized. There are realistic situations (not considered here) where the average overflow chain length can be much longer. As for the B+ tree, the scan curve naturally reflects the storage curve of Figure 5.

Figure 7 shows the expected number of block transfers needed to insert a record. For the indexed sequential structure, the first record inserted into a node will be stored in the vacant record slot of the primary block. This requires four transfers (three reads + one write). Subsequent insertions are stored in overflow and require additional transfers (to read and update overflow pointers). Thus, insertion into overflow is costly. For B+ trees, observe that the bounds on insertion performance are not good initially, but they quickly converge as the file grows. Furthermore, the number of transfers needed is just slightly above the minimum of four. This indicates that a majority of the time, splitting does not occur, and when it does, it is confined to the data level of a B+ tree.[6]

Figure 8 displays the number of transfers needed to delete a record, although in the file that we are considering, no deletions occur. For indexed sequential files, deletions rarely involve overflow records since the bulk of records reside in primary blocks. Consequently, deletions are efficient. For B+ trees, deletions are somewhat more expensive. The rise in the bounding curves around week 5 coincides with the increase of storage volume when many nodes have minimal occupancy; deletions from these nodes trigger node underflow activities. Notice again, there is a convergence of cost bounds.[7]

[6]This result has theoretical importance. It suggests that only the bottom level of a file structure needs to be modeled in detail.

[7]The bounds remain close, even out to week 100 where

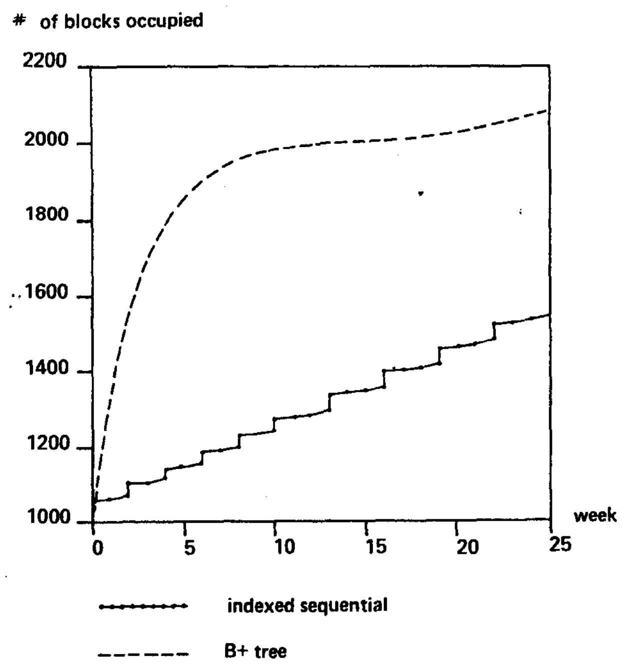


Figure 5. Expected Storage Volume

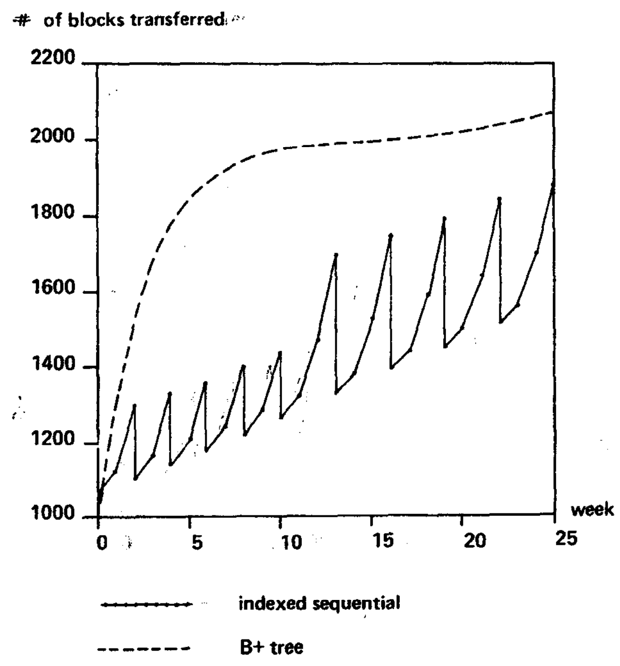


Figure 6. Expected Scan Efficiency

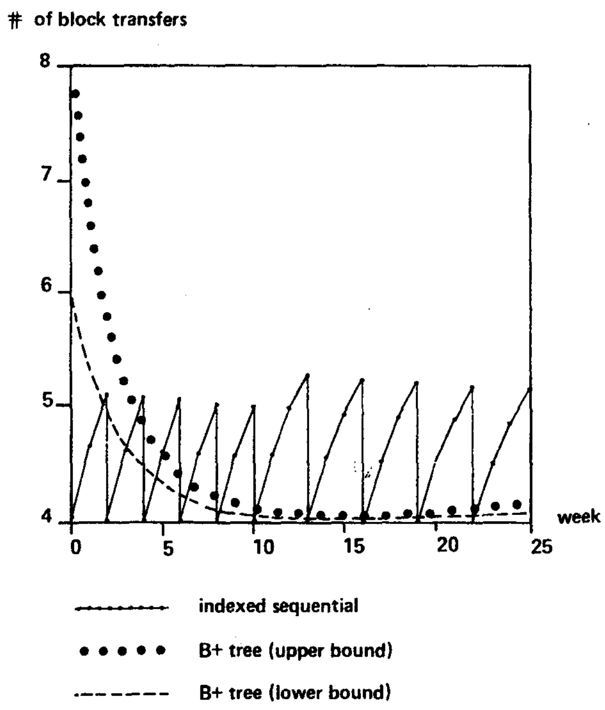


Figure 7. Expected Insertion Efficiency

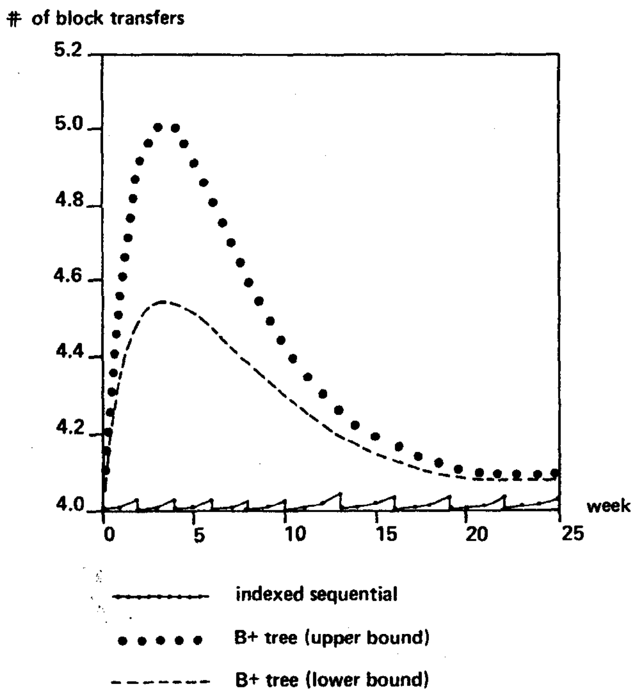


Figure 8. Expected deletion Efficiency

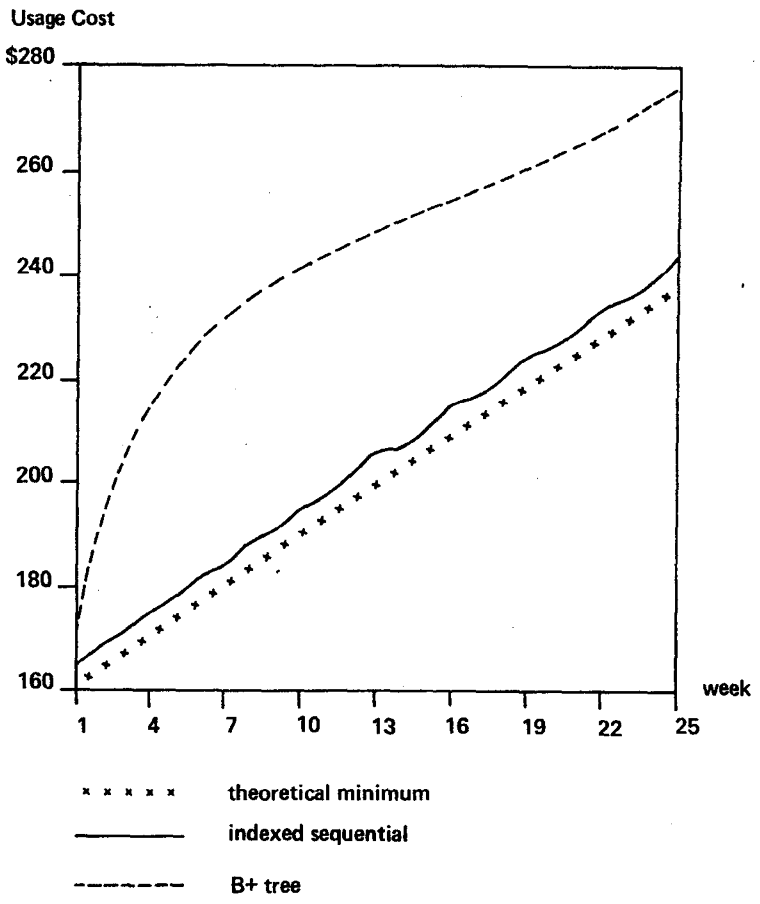


Figure 9. Expected Usage Cost Graphs

For the remaining operations of record retrieval and update, the differences observed between B+ tree and indexed sequential performance were negligible (i.e., less than 1%). Again, this result is specific to our problem; it need not be characteristic of all situations.

At last, we compare the Usage_Costs of the B+ tree and indexed sequential files (Fig. 9). For our problem we find that an indexed sequential file is slightly more efficient than a B+ tree. The predicted difference at week 25 is about 13%.[8]

In order to complete our preformance comparison, it is necessary to consider the cost of reorganizations. The sum of all indexed sequential constuction and dumping costs was slightly over $40. Such costs are easily amortized during the first few weeks of indexed sequential usage.

The efficiency of indexed sequential files is due to frequent reorganizations. At no time are many records in overflow. Consequently, indexed sequential performance closely approximates the theoretically minimal usage cost which is achieved if the file is continuously reorganized (see Fig. 9 and Appendix IV). B+ tree performance is less efficient due to its lower storage utilization.

Earlier we noted that the expected height of a B+ tree may not be known with certainty. Notice, however, that a B+ tree's height is usually greater than or equal to that of an indexed sequential file. Should it be greater, it is clear from the evidence just presented that indexed sequential files will still be preferred to B+ trees.

6. CONCLUSIONS

An analytic method has been presented for comparing B+ tree and indexed sequential file performance. The relative performance of B+ trees and indexed sequential files is application dependent; which structure is preferred varies in different situations. File usage, file size, record lengths, and block storage and access costs all have an impact on the final selection. In the situation that we examined, we noted that indexed sequential files

$DEL^-(F)=4.12\times A$ and $DEL^+(F)=4.21\times A$. The same is true for insertions where $INS^-(F)=4.13\times A$ and $INS^+(F) = 4.26\times A$.

[8] Although the Usage_Cost for B+ trees was bounded, the differences between bounds were negligible.

were slightly more efficient than B+ trees. Further experiments are necessary to assess the generality of these conclusions.

Acknowledgements. I thank Dr. M. Edelberg for suggesting the topic of this paper. I also thank Dr. K.C. Sevcik for his helpful insights and suggestions.

REFERENCES

[AHU74] Aho, A.V., Hopcroft, J.E., and Ullman, J.D., The Design and Analysis of Computer Algorithms, Addison-Wesley, Reading, Massachusetts, 1974.

[Bat80a] Batory, D.S. "Optimal File Designs and Reorganization Points", Computer Systems Research Group Tech. Rep. 110, University of Toronto, 1980.

[Bat80b] Batory, D.S. "An Analytic Model of Physical Databases", Ph.D. Th., University of Toronto, 1980.

[Com79] Comer, D., "The Ubiquitous B-Tree", ACM Computing Surveys 11,2 (June 79), pp. 121-138.

[HaKo80] Hatzopoulos, M. and Kollias, J., "A Dynamic Model for the Optimal Selection of Secondary Indices", Dept. of Applied Mathematics, Univ. of Athens, Athens, Greece, 1980.

[Knu73] Knuth, D.E., The Art of Computer Programming, Vol 3: Sorting and Searching, Addison-Wesley, Reading, Massachusetts, 1973.

[NaMi78] Nakamura, T. and Mizoguchi, T., "An Analysis of Storage Utilization Factor in Block Split Data Structuring Scheme", Proc. Very Large Data Bases Conf., Berlin 1978, pp. 489-495.

[Rei76] Reiter, A., "Some Experiments in Directory Organization - A Simulation Study", Proc. Int. Symp. Computer Performance, Modeling, Measurement, and Evaluation, (March 76), Harvard, pp. 1-6.

[HeSt78] Held, G. and Stonebraker, M., "B-Trees Re-examined", Comm. ACM 21,2 (Feb. 78), pp. 139-142.

[Ram80] Ramirez, R.J., "Efficient Algorithms for Selecting Efficient Data Storage Structures", Ph.D. Th., University of Waterloo, Waterloo, Ontario, 1980.

[Ston80] Stonebraker, M., "Retrospection on a Database System", ACM Trans. Database Syst. 5,2 (June 80), pp. 225-240.

[Yao77] Yao, S.B., "An Attribute Based Model for Database Access Cost Analysis", ACM Trans. Database Syst. 2,1 (March 77), pp. 45-67.

[Yao78] Yao, A.C., "On Random 2-3 Trees", Acta Informatica 9,2 (1978), pp. 159-168.

APPENDIX I. COST FUNCTIONS FOR INDEXED SEQUENTIAL FILES

The cost functions in this Appendix and those in Appendix II are based on the assumptions:

(A1) All records have an equal probability of being requested.

(A2) All records have an equal probability of being deleted.

(A3) Records that are inserted have identifiers that are randomly chosen from a static, perhaps lexicographically nonuniform, distribution of identifiers.[9]

The storage cost of a file is the file's storage volume in blocks times the storage cost per block:

$$\mathrm{STOR}(F) = (Z + \left\lceil \frac{G\times Z}{R} \right\rceil + \sum_{j=1}^{L-1} \left\lceil \frac{Z}{Ri^j} \right\rceil)\times S$$

Provided that the records of a file are already sorted on ascending identifiers, the cost of constructing an indexed sequential file is the cost of writing out all blocks of the file:

$$\mathrm{CON}(F) = (Z + \left\lceil \frac{G\times Z}{R} \right\rceil + \sum_{j=1}^{L-1} \left\lceil \frac{Z}{Ri^j} \right\rceil)\times A$$

Updating a previously accessed record requires a block write:

$$\mathrm{UPD}(F) = A$$

When each overflow record requires a separate access, we have:

$$\mathrm{RET}(F,\mathrm{ONE}) = (L + \Omega)\times A$$

$$\mathrm{RET}(F,\mathrm{SCAN}) = Z\times(1+G)\times A$$

A file dump retrieves all records of a file in ascending identifier order:[10]

$$\mathrm{DUMP}(F) = \mathrm{RET}(F,\mathrm{SCAN})$$

If no records have been deleted,

$$\mathrm{INS}(F) = \mathrm{RET}(F,\mathrm{ONE}) + (1 + 2\times \mathrm{Pfull})\times A$$

If deleted records are marked deleted in a primary block and are physically removed from an overflow chain,

$$\mathrm{DEL}(F) = \mathrm{RET}(F,\mathrm{ONE}) + (1 + \frac{G}{H+G})\times A$$

APPENDIX II. COST FUNCTIONS FOR B+ TREES

Denoting $\mathrm{STOR}^+(F)$ and $\mathrm{STOR}^-(F)$ as the upper and

[9] An example of a lexicographically nonuniform distribution are the names in a phone directory. There is a greater probability of selecting a name beginning with 's' than a name beginning with 'z'.

[10] In practice, file dumping is often accompanied by file backup, i.e., copying the file on tape. Although file backup is not included in our model, it could be with minor additions to the CON and DUMP cost functions. The additional terms would account for the cost of reading and writing a tape file.

lower bound to the expected file storage cost, we have:

$$\mathrm{STOR}^-(F) = (Z + \sum_{j=1}^{L-1} \left\lceil \frac{Z}{Ri^j} \right\rceil)\times S$$

$$\mathrm{STOR}^+(F) = (Z + \sum_{j=1}^{L-1} \left\lceil \frac{Z}{Mi^j} \right\rceil)\times S$$

When record insertions and deletions do not alter the height of a B+ tree, bounds on the expected cost of record insertion and deletion are:

$$\mathrm{INS}^-(F) = \mathrm{RET}(F,\mathrm{ONE}) + (1 + 2\times Pfull)\times A$$

$$\mathrm{INS}^+(F) = \mathrm{INS}^-(F) + 2\times(L-2)\times Pfull\times A$$

$$\mathrm{DEL}^-(F) = \mathrm{RET}(F,\mathrm{ONE}) + (1 + Pund\times(3 - Pmer))\times A$$

$$\mathrm{DEL}^+(F) = \mathrm{DEL}^-(F) + \begin{cases} 0 & \text{if } L \le 2 \\ Pund\times Pmer\times(3+2\times(L-3))\times A & \text{otherwise} \end{cases}$$

Other cost functions are:

$$\mathrm{RET}(F,\mathrm{ONE}) = L\times A$$

$$\mathrm{RET}(F,\mathrm{SCAN}) = Z\times A$$

$$\mathrm{UPD}(F) = A$$

APPENDIX III. FILE PARAMETER DEFINININITIONS

The following definitions are based on assumptions (A1)-(A3) and are applicable when no record deletions have occurred:

$$N = \sum_r r\times T(r) \qquad H = \sum_r \min(r,R)\times\frac{T(r)}{Z}$$

$$Z = \sum_r T(r) \qquad Pfull = \sum_{r\ge R} \frac{r\times T(r)}{N}$$

For indexed sequential files:

$$L = \left\lceil \log_{(Ri)} Z \right\rceil + 1$$

$$\Omega = \sum_{r\ge R} \frac{(r-R)\times(r-R+1)\times T(r)}{2\times N} \qquad Pmer = 0$$

$$G = \sum_{r\ge R} (r-R)\times\frac{T(r)}{Z} \qquad Pund = 0$$

For B+ trees:

$$\left\lceil \log_{(Ri)} Z \right\rceil + 1 \le L \le \left\lfloor \log_{(Mi)} \frac{Z}{2} \right\rfloor + 2$$

$$\Omega = 0 \qquad Pmer = \sum_{r=M}^{R+1-M} \frac{T(r)}{Z}$$

$$G = 0 \qquad Pund = M\times\frac{T(M)}{N}$$

APPENDIX IV. COST FUNCTIONS FOR A CONTINUOUSLY REORGANIZED FILE

Let F^* be the descriptor of an (imaginary) tree structured file with the properties:

1) it maintains 100% storage utilization,
2) new records are accommodated without node splitting or storing records in overflow,
3) deleted records do not leave vacant record slots,
4) the following identities hold: $Z = N/R$ and

$$L = \left\lceil \log_{(Ri)} Z \right\rceil + 1.$$

Tree structured files with such properties are said to be <u>continuously reorganized</u>. Cost functions for F^* are:

$$\mathrm{RET}(F^*,\mathrm{ONE}) = L\times A$$

$$\mathrm{RET}(F^*,\mathrm{SCAN}) = Z\times A$$

$$\mathrm{INS}(F^*) = \mathrm{RET}(F^*) + A$$

$$\mathrm{DEL}(F^*) = \mathrm{RET}(F^*) + A$$

$$\mathrm{UPD}(F^*) = A$$

$$\mathrm{STOR}(F^*) = (Z + \sum_{j=1}^{L-1} \left\lceil \frac{Z}{Ri^j} \right\rceil)\times S$$