Check for updates

Partial Expansions for File Organizations with an Index

DAVID B. LOMET
Wang Institute of Graduate Studies

A new way to increase file space in dynamically growing files is introduced in which substantial improvement in file utilization can be achieved. It makes use of partial expansions in which, instead of doubling the space associated with some part of the file, the space grows at a slower rate. Unlike previous versions of partial expansion in which the number of buckets involved in file growth is increased by less than a factor of two, the new method expands file space by increasing bucket size via "elastic buckets." This permits partial expansions to be used with a wide range of indexed files, including B-trees. The results of using partial expansions are analyzed, and the analysis confirmed by a simulation study. The analysis and simulation demonstrate that the file utilization gains are substantial and that fears of excessive insertion cost resulting from more frequent file growth are unfounded.

Categories and Subject Descriptors: D.4.3 [**Operating Systems**]: File Systems Management—*access methods*; *file organization*; H.2.2 [**Database Management**]: Physical Design—*access methods*; H.3.2 [**Information Storage and Retrieval**]: Information Storage—*file organization*

General Terms: Algorithms, Design, Performance

Additional Key Words and Phrases: B-trees, dynamic files, indexed files, partial expansions, storage management, storage utilization

1. INTRODUCTION

Most existing file organizations that cope well with growing files do so by increasing the space used to contain some portion of the file by a factor of two. Thus B-trees [1] typically grow by node splitting, in which the entries of a full node of the tree are divided equally between the existing node and new node. Such growth by local doubling is also a property of many of the dynamic (extensible) file organizations [2, 6, 8, 9, 10]. The result is that utilization within the nodes (or sections) of these files varies from 50 to 100 percent with an average utilization of approximately log (2) = .693 [8, 13]. Doubling of the entire file, as done in virtual hashing [5], leads to a similar result.

If growth could occur at a rate that is less than a factor of two, then utilization within these growing files would be improved. One early proposal to do this [3]

This work is a revised version of the identically named IBM Research Report, RC 11240, June 1985.
Author's address: Wang Institute of Graduate Studies, School of Information and Technology, Tyng Road, Tyngboro, MA 01879.
Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.
© 1987 ACM 0362-5915/87/0300-0065 $00.75

is embodied in the notion of what Knuth calls a B*-tree. The idea is to examine neighbors of a full node, and to shift entries from a full node to neighbors that still have available space. When a node and its neighbor are both full, then a new node is added to the B*-tree and the entries from two nodes are spread over three nodes, the new one and the original two. The result is that utilization does not fall below about 67 percent, and average utilization is similarly improved.

In a difficult but noteworthy paper, Martin [12] proposed a file organization in which local growth of an exponentially hashed file could occur at a rate less than a factor of two. His bucket allocation scheme was complex, however, and his method was designed to cope particularly with exponentially distributed hashed keys. Larson [4] invented the term "partial expansion" to characterize a growth regime in which more than one growth event is necessary in order for a file to double, doubling being called a "full expansion." He applied the technique to Litwin's linear hashing [6] by incrementally increasing the number of buckets in a small group of buckets from two to three, and then from three to four, thus doubling the number of buckets in two steps. His analysis showed that utilization improved substantially. He computed results for one, two, and three partial expansions per full expansion, and his results showed how utilization improves for linear hashing as the number of partial expansions per full expansion increases, as one would expect.

The potential penalty for applying the partial expansion technique to raise file utilization has been that insertion cost is increased. The file growth process occurs more frequently, causing data to be moved between buckets (or nodes).

With B-trees, not only does this increased data motion occur, but with three-for-two splitting in B*-trees, each entry inserted into a full node requires an extra disk access of the neighboring node for as long as the node is full and its neighbor is not. The result is that not only is the insertion process more complex when growth occurs, but it can be both more complex and more costly for many insertions prior to growth occurring.

One would like to realize the increased utilization produced by partial expansions, while avoiding excessive insertion cost. For B-trees, three-for-two splitting is rarely used because of the extra insertion cost. However, in this paper we propose a new form of partial expansion which minimizes this extra insertion cost. This new technique is also applicable to a number of dynamic hashed file organizations [2, 9], to tree index methods with multibucket nodes [8, 10], and to the new mixed tree/hashed file organization proposed in [7]. In fact it can be applied to almost any file in which there are multiple nodes addressed by means of an index of almost any kind.

This paper is organized as follows: The new partial expansion technique is described in Section 2. Section 3 analyzes the impact of partial expansions with a single growth rate on storage utilization. Because of page size granularity, partial expansions with unequal growth rates will be required in order to yield a full expansion. This is analyzed in Section 4. A simulation study is described in Section 5 which confirms the results of the analysis. The impact of partial expansions on insertion cost is treated in Section 6. Related to insertion cost is the size of the index, which is considered in Section 7. The paper ends with a short discussion in Section 8.

2. ELASTIC BUCKETS

A careful reading of the introduction reveals that file expansions have been realized by increasing the number of buckets that are used to store the contents of some growing subset of the key space (or hashed key space) of the file. This shows up clearly in the discussion of three-for-two node splitting for B-trees. In order to do partial expansions via growth in the number of buckets, the growth process must be able to "fill" a number of buckets prior to adding a bucket. The smallest number of buckets that can be added is one, which, for B-trees with their single bucket per node, doubles the space locally. For some hashed files, with multiple bucket nodes, for instance, BEH hashing [9], growth can be more finely controlled. However, partial expansions for these files require redistributing the contents of one bucket over two different buckets of a new node. This redistribution is also required for linear hashing [4, 6], which does not have nodes in any conventional sense.

We wish to consider the mechanics of how a file might grow so as to avoid moving large amounts of data and incurring large numbers of disk accesses. To explain the technique being proposed, it is necessary to distinguish three different units of disk storage, pages, nodes, and buckets, which are defined below.

(1) *Page*: The smallest physical unit of storage allocation on the disk. It is the smallest unit of data that can be read from or written to the disk. All other units are an integral number (one or more) of pages. Each page has a unique disk address. Page size, in bytes, is usually defined by the file system.
(2) *Node*: The logical unit of allocation of storage on the disk. A node is a logically contiguous set of pages that can be referenced by a single disk address. It is the largest unit for which the file system guarantees logical contiguity.
(3) *Bucket*: A logically contiguous set of pages within a node. The set of buckets forming a node is a partition of the node. A bucket is the storage unit that the access method, for its own purposes, is required to access in its entirety, that is, an access method always reads an integral number of buckets.

Note that page size is determined by the file system and node size (in pages) is made known to the file system, acting as the allocator of disk space. (In analogy with main memory storage allocation, a page is analogous to a byte and a node is analogous to a storage area, i.e., data structure.) Thus both of these quantities are known to the file system. The file system attempts to map the logically contiguous pages of a node onto physically contiguous disk pages so as to maximize disk performance when reading some subset (perhaps all) of the node. This will usually be accomplished by mapping a node to some part of a physical extent or to some small number of extents, should extents be smaller than node size. We assume that the file system is prepared to honor I/O access requests, whose operands are expressed in terms of the triple:

⟨node address; offset of first page of part of node desired; size, in pages, of part⟩.

The bucket is a storage unit defined by, and known to, the access method. It represents some fractional part of a node. (In analogy with programming

languages, a bucket corresponds to a component of an allocated data structure.) While each bucket may have a different size within a node, we expect bucket size to typically be constant for a given node so as to permit array type addressing of the buckets. The file system will not usually be aware of the bucket structure of a node. The access method uses buckets analogously to the way programming languages use scalars. That is, buckets like scalars, to be effectively used, must be accessed in their entirety.

File growth, in the context of the file system described above, is accomplished in two distinct ways. First (and, conventionally, as performed with B-trees), nodes may split so that one node is replaced by two nodes and the data contained in the original node divided between the two nodes. Second, a node may grow in size, that is, be replaced by a single new node which is comprised of a larger number of pages than the original node, thus providing additional room for growth.

When a node grows, the access method is presented with the choice of either (1) leaving the bucket size unchanged and increasing the number of buckets or (2) leaving the number of buckets unchanged and increasing the size of the buckets. This second alternative, which we call "elastic" buckets, is the method advocated here.

In the remainder of this paper, the notion of elastic buckets is pursued in the context of B-tree files. B-trees do not have multibucket nodes. Rather, each node must be read in its entirety and hence is a single bucket node. (Elastic bucket partial expansions can be used with multibucket nodes and will be explored in a forthcoming paper [11].) Because of this, we will frequently use the terms *bucket* and *node* interchangeably, since the unit of storage being referenced is the same. If an important distinction is being made, this will be emphasized and terms will be used with precision.

As an example, consider a B-tree with single bucket nodes in which the smallest bucket size is two pages. When a node (bucket) overflows, instead of splitting it into two nodes, we replace the two-page node with a three-page node. The utilization drops to 67 percent instead of the 50 percent that splitting would result in. When a three-page node overflows, we split it into two nodes, but these nodes are only two-page nodes, not three-page nodes. Hence, the initial utilization of the two new nodes is 75 percent. These two partial expansions have resulted in a doubling of the storage used to contain records in the local key subspace. This is thus a full expansion in two steps.

Larger numbers of partial expansions per full expansion are also possible. We can begin with a three-page node, overflow resulting in growth to a four-page node, which, when it overflows, grows to a five-page node. Finally, when a five-page node overflows, it splits into two three-page nodes, completing the doubling or full expansion in three steps.

What we wish to know is the effect of partial expansions on file performance characteristics. Because of the page size granularity of buckets, a full expansion will typically be accomplished by two or more partial expansions of different rates. In the first example above, a two-page node was replaced with a three-page node, a growth factor of 1.5. When the three-page node split into two two-page nodes, the growth factor was 1.33. Ideally, choosing two partial expansions per

full expansion, we would like the growth rate to be 1.414, that is, the square root of two. We analyze partial expansions for both a single growth rate and for differing growth rates. We shall see that different growth rate partial expansions come quite close to equaling the results of ideal, single growth rate partial expansions.

3. SINGLE GROWTH RATE PARTIAL EXPANSION

In order to analyze the performance of partial expansions, we make three assumptions.

(1) The numbers of entries in buckets of the file are growing at a uniform rate, that is, the percentage growth of entries in each bucket during any time interval is the same for all buckets.
(2) The distribution of the number of buckets at any given utilization remains constant, that is, the file is at steady state with respect to this attribute. Note that this implies that our results apply to files that have grown to contain a large number of buckets.
(3) Each bucket contains a large number of records. During the analysis, limits are taken. Thus, our analysis will not be accurate for small numbers of records per bucket. The larger the number of records, the more accurate will our results be. This will be discussed again later in the paper.

The question that we wish to answer is, what is the average utilization of a file in which, when a bucket reaches maximum utilization, it is replaced by a bucket whose size is larger by some fixed percentage? The result we seek is average utilization expressed as a fraction of the maximum utilization of any bucket. It is apparent that in hashed files the maximum utilization may well be less than 100 percent, since buckets of a multibucket node fill unevenly, even when the hashed keys are distributed uniformly. Even with B-trees, because buckets are not necessarily a multiple of record size, maximum utilization will frequently not be 100 percent. A "relative" result will remind us of this.

Let $u_{\max}$ be the maximum utilization of buckets of the file. Further, let R be the growth factor of the file. That is, when a bucket's utilization reaches $u_{\max}$, the bucket is replaced with one whose size is R times the size of the original bucket. Equivalently, a large number of buckets with utilization of $u_{\max}$ can be replaced with new buckets such that the space of the new buckets is a factor of R larger than the space of the original buckets. Since the number of records per bucket is large, the minimum utilization of a bucket will be

$$u_{\min} = \frac{u_{\max}}{R}.$$

Various buckets will have differing utilizations. Our steady state assumption means that the fraction of buckets at any utilization will be constant. Let these utilizations be $u_{\min}$, $u_{\min}R^{i/m}$, ..., $u_{\min}R^{(m-1)/m}$. These utilizations are chosen so that file growth by a factor of $R^{1/m}$ causes buckets with utilization $u_{\min}R^{i/m}$ to become buckets with utilization $u_{\min}R^{(i+1)/m}$. Then, letting m go to infinity in the limit gives us results for u as a continuous distribution.

We define N_i to be the number of pages with a utilization of $u_{\min}R^{i/m}$. As the file grows, the number of buckets at any given utilization also grows. In particular, after an increase in number of entries by a factor of $R^{1/m}$, we have

$$N_0^{\text{new}} = RN_{m-1}^{\text{old}},$$

and, for $0 < i < m$,

$$N_i^{\text{new}} = N_{i-1}^{\text{old}}.$$

Normalizing relative to the number of buckets with utilization equal to $u_{\min}$, we define c_i as the ratio of the number of buckets with utilization $u_{\min}R^{i/m}$ to the number of buckets with utilization $u_{\min}$.

Then

$$c_0 = \frac{N_0^{\text{new}}}{N_0^{\text{new}}} = 1 = \frac{RN_{m-1}^{\text{old}}}{N_0^{\text{new}}} = Rc_{m-1}\frac{N_0^{\text{old}}}{N_0^{\text{new}}}.$$

Hence, we have that

$$\frac{N_0^{\text{old}}}{N_0^{\text{new}}} = \frac{1}{Rc_{m-1}}.$$

Further,

$$c_i = \frac{N_i^{\text{new}}}{N_0^{\text{new}}} = \frac{N_{i-1}^{\text{old}}}{N_0^{\text{new}}} = c_{i-1}\frac{N_0^{\text{old}}}{N_0^{\text{new}}} = c_{i-1}\frac{1}{Rc_{m-1}}.$$

This leads to

$$c_1 = \frac{1}{Rc_{m-1}} \quad \text{and} \quad c_i = (c_1)^i = \left(\frac{1}{Rc_{m-1}}\right)^i,$$

with

$$c_{m-1} = \left(\frac{1}{Rc_{m-1}}\right)^{m-1}.$$

Solving for c_{m-1} yields

$$c_{m-1} = (1/R)^{(m-1)/m}.$$

Thus

$$c_i = (1/R)^{i/m}.$$

Knowing the normalized number of pages with any given utilization, we can determine the total number of normalized pages NP for all utilizations. Further, we can determine the total normalized space NS used, in number of pages, by the data. Dividing the second quantity by the first yields average utilization. We begin by summing the number of normalized pages, which is

$$\text{NP} = \sum_{i=0}^{m-1}(c_i) = \sum_{i=0}^{m-1}\left(\frac{1}{R}\right)^{i/m} = \frac{1-(1/R)}{1-(1/R)^{1/m}}.$$

The normalized space used is

$$\mathrm{NS} = \sum_{i=0}^{m-1} c_i u_i = \sum_{i=0}^{m-1} \left(\frac{1}{R}\right)^{i/m} \frac{u_{\max}}{R}(R)^{i/m} = \left(\frac{u_{\max}}{R}\right)m.$$

Dividing NS by NP yields the average utilization of the file. It is here that the analysis most explicitly assumes a large number of entries per page, so as to avoid dealing with record size granularity and a large number of pages. This also yields a closed-form solution for the analysis and is similar to what was done in [8]. Thus,

$$u_{\text{file}} = \lim_{m\to\infty} \frac{\mathrm{NS}}{\mathrm{NP}} = \lim_{m\to\infty} \frac{u_{\max}}{R(1-(1/R))} m\left(1 - \left(\frac{1}{R}\right)^{1/m}\right).$$

But, we have that

$$\lim_{m\to\infty} m(1 - (1/R)^{1/m}) = \log(R).$$

Thus,

$$u_{\text{file}} = \frac{u_{\max}}{R-1} \log(R).$$

Using this result, we can compute average file utilization for various growth factors. We are interested in the range of $1 < R \leq 2$, which is the range of growth factor of interest for partial expansions.

Particular values of R are of interest for our study of partial expansions. To determine the effect of E partial expansions on file utilization, where all partial expansions have the same growth rate, and, when completed, the file has doubled, requires that $R = 2^{1/E}$. Average file utilization is plotted against number of partial expansions in Figure 1. Notice that the greatest gain in utilization, going from .693 to .837, results when we use two partial expansions to double a file, instead of a full expansion. Each doubling of the number of expansions needed to double the file space halves the difference between average utilization and maximum utilization. Thus, gains for additional partial expansions quickly become marginal. Indeed, after four partial expansions, when file utilization is .916 percent of maximum utilization, each additional expansion increases utilization less than 2 percent.

4. UNEQUAL GROWTH RATE PARTIAL EXPANSIONS

While we might ideally like to have uniform growth rates for each partial expansion, so as to maximize file utilization, in a real system with finite (as opposed to infinitesimal) pages, we must deal with differing growth rates. Recall that two-page buckets expanding to three pages, then splitting into two two-page buckets, produce two growth rates, $\frac{3}{2}$ and $\frac{4}{3}$. In this section we wish to determine what effect these differing growth rates have on file utilization. The same kind of analysis is done here as in Section 3.

We begin by considering the two partial expansion case, and then will generalize to an arbitrary number of partial expansions. Let R_1 be the growth in space

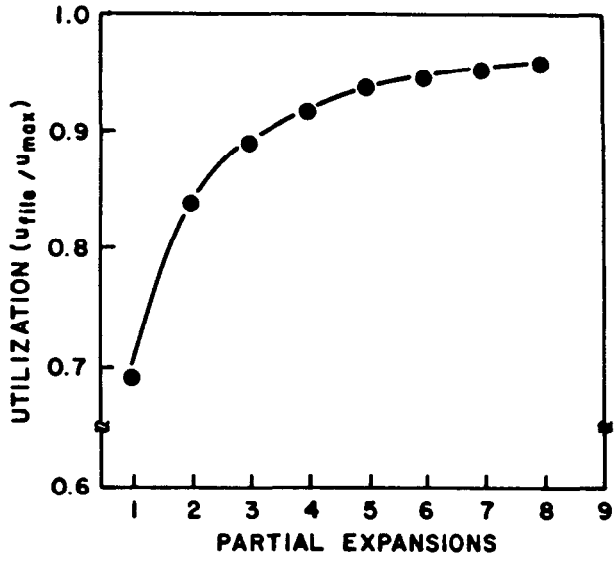

Fig. 1. Utilization is plotted against the number of partial expansions required to achieve that utilization. The growth factor for a growth regime employing E partial expansions per full expansion is $R = 2^{1/E}$.

produced by the first partial expansion, with R_2 the growth produced by the second. Further, let u_{max1} be the maximum utilization achieved after the first partial expansion, and u_{max2} be the maximum utilization achieved after the second. There are corresponding values for the minimum utilizations. Then

$$R_1 = \frac{u_{max2}}{u_{min1}} \quad \text{and} \quad R_2 = \frac{u_{max1}}{u_{min2}}.$$

Our steady state assumption requires that the fraction of buckets at any utilization be constant. Since we have two growth rates for space, our utilizations will be influenced by both the rates. Thus the utilizations that will be tracked are $u_{min1}, \ldots, u_{min1}(r_1)^{i/m}, \ldots, u_{min1}(r_1)^{(m-1)/m}, u_{min2}, \ldots, u_{min2}(r_2)^{j/n}, \ldots, u_{min2}(r_2)^{(n-1)/n}$. Analogously to the single growth rate case, we define N_i to be the number of pages with a utilization of $u_{min1}(r_1)^{i/m}$, for $0 \leq i < m$. Further, we let N_{m+j} be the number of pages with a utilization of $u_{min2}(r_2)^{j/n}$ for $0 \leq j < n$. Note here that

$$r_1 = \frac{u_{max1}}{u_{min1}} \quad \text{and} \quad r_2 = \frac{u_{max2}}{u_{min2}}.$$

In addition, n and m are related. Specifically,

$$(r_1)^{1/m} = (r_2)^{1/n} \quad \text{and} \quad R_1 R_2 = r_1 r_2.$$

The analysis here follows that for a single growth rate, but reflects the two partial expansions. Thus, after an increase by a factor of $(r_1)^{1/m}$ in number of entries, we have the following recurrence relations for pages following the first

partial expansion:

$$N_0^{\text{new}} = R_1 N_{m+n-1}^{\text{old}},$$

and, for $1 \leq i < m$,

$$N_i^{\text{new}} = N_{i-1}^{\text{old}}.$$

At $u_{\max 1}$, pages undergo the second partial expansion. Thus the second partial expansion affects these pages, and we have

$$N_m^{\text{new}} = R_2 N_{m-1}^{\text{old}},$$

and, for $1 \leq j < n$,

$$N_{m+j}^{\text{new}} = N_{m+j-1}^{\text{old}}.$$

We again normalize these numbers relative to the number N_0 of $u_{\min 1}$ pages by dividing by N_0. Doing this, then solving for c_{m+n-1}, and finally substituting the result into the equations yields

$$c_i = \left(\frac{1}{R_1 R_2}\right)^{i/(m+n)} = \left(\frac{1}{r_1}\right)^{i/m}$$

for $0 \leq i < m$; and

$$c_{m+j} = R_2\left(\frac{1}{R_1 R_2}\right)^{(m+j)/(m+n)} = \left(\frac{R_2}{r_1}\right)\left(\frac{1}{r_2}\right)^{j/n}$$

for $0 \leq j < n$.

Computing the average utilization of a file undergoing two partial expansions requires once again that we compute the normalized number of pages (capacity) and the normalized space required by the data. Both these quantities now require summations over both partial expansions. Thus, let NP_{Total} denote the total normalized number of pages and NS_{Total} the total normalized space required by the data, with NP_i the normalized number of pages and NS_i the normalized space required for data, both for the ith partial expansion. Then

$$\text{NP}_{\text{Total}} = \text{NP}_1 + \text{NP}_2 = \sum_{i=0}^{m-1} c_i + \sum_{j=0}^{n-1} c_{m+j}.$$

After substituting for the c_i and c_{m+j} terms, doing some algebraic manipulation, and performing the summations, we have

$$\text{NP}_{\text{Total}} = \frac{1}{1 - (1/R_1 R_2)^{1/(m+n)}}\left[\left(1 - \frac{1}{r_1}\right) + \left(\frac{R_2}{r_1}\right)\left(1 - \frac{1}{r_2}\right)\right].$$

Similarly,

$$\text{NS}_{\text{Total}} = \text{NS}_1 + \text{NS}_2 = \sum_{i=0}^{m-1} c_i u_i + \sum_{j=0}^{n-1} c_{m+j} u_{m+j}$$

$$= \sum_{i=0}^{m-1} \left(\frac{1}{r_1}\right)^{i/m} u_{\min 1}(r_1)^{i/m} + \sum_{j=0}^{n-1} \left(\frac{R_2}{r_1}\right)\left(\frac{1}{r_2}\right)^{j/n} u_{\min_2}(r_2)^{j/n}.$$

After algebraic manipulation and summation, this results in

$$\mathrm{NS}_{\mathrm{Total}} = u_{\mathrm{min1}}(m) + \left(\frac{R_2}{r_1}\right)u_{\mathrm{min2}}(n) = u_{\mathrm{min1}}(m + n).$$

Finally,

$$u_{\mathrm{file}} = \frac{\mathrm{NS}_{\mathrm{Total}}}{\mathrm{NP}_{\mathrm{Total}}} = \frac{u_{\mathrm{min1}}(m + n)(1 - (1/R_1R_2)^{1/m+n})}{[(1 - (1/r_1)) + (R_2/r_1)(1 - (1/r_2))]}.$$

If we take the limit as m goes to infinity, then n will also go to infinity. Making use of the previously used logarithm limit yields

$$u_{\mathrm{file}} = \frac{u_{\mathrm{min1}}\log(R_1R_2)}{[(1 - 1/r_1) + (R_2/r_1)(1 - 1/r_2)]}.$$

Expressed in terms of the utilizations of each of the partial expansions, we have

$$u_{\mathrm{file}} = f_1u_1 + f_2u_2$$

where

$$f_i = \frac{(u_{\mathrm{max2}}/u_{\mathrm{maxi}})(r_i - 1)}{[(u_{\mathrm{max2}}/u_{\mathrm{max1}})(r_i - 1) + (r_2 - 1)]} = \frac{(1/u_{\mathrm{maxi}})(r_i - 1)}{[(1/u_{\mathrm{max1}})(r_1 - 1) + (1/u_{\mathrm{max2}})(r_2 - 1)]}$$

and

$$u_i = \frac{u_{\mathrm{maxi}}\log(r_i)}{r_i - 1}.$$

The quantity u_{file} can also be expressed as

$$u_{\mathrm{file}} = \frac{\log(r_1r_2)}{[(1/u_{\mathrm{max1}})(r_1 - 1) + (1/u_{\mathrm{max2}})(r_2 - 1)]}.$$

It is possible to generalize this analysis to arbitrary numbers of partial expansions. The result for k partial expansions, in terms of the utilizations for each partial expansion, is

$$u_{\mathrm{file}} = \frac{\sum_{i=1}^{k} \mathrm{NS}_i}{\sum_{i=1}^{k} \mathrm{NP}_i} = \sum_{i=1}^{k} f_iu_i,$$

where

$$f_i = \frac{\mathrm{NP}_i}{\mathrm{NP}_{\mathrm{Total}}} = \frac{(1/u_{\mathrm{maxi}})(r_i - 1)}{\sum_{i=1}^{k} (1/u_{\mathrm{maxi}})(r_i - 1)}$$

and

$$u_i = \frac{u_{\mathrm{maxi}}\log(r_i)}{r_i - 1}.$$

Table I[a]

E	Utilization (ideal)	Utilization (realizable)
1	.693	.693
2	.837	.832
3	.889	.885
4	.916	.913
5	.932	.930
6	.943	.941
7	.951	.949
8	.957	.956

[a] The "ideal" utilization is computed assuming a constant growth rate for each partial expansion of $R = 2^{1/E}$ where E is the number of partial expansions. The realizable utilization is computed assuming growth rates of $R_i = (E + i)/(E + i - 1)$ from $i = 1$ to $i = E$.

Alternatively, we can express the result in terms of the full expansion as

$$u_{\text{file}} = \frac{\log(\prod_{i=1}^{k} r_i)}{\sum_{i=1}^{k} (1/u_{\text{max}i})(r_i - 1)}.$$

In order to compare the different growth rate partial expansions with single growth rate partial expansions, the maximum utilizations reached for the different partial expansions are set to the same value, that is, $u_{\text{max}i} = u_{\text{max}j}$ for all values of i and j. We then choose to do the partial expansions using the smallest possible bucket sizes, in terms of number of pages. This choice makes the growth rates as unequal as possible, and hence maximizes the distance from the single growth rate case. The results for the various realizable partial expansions are tabulated in Table I and compared with the ideal single growth rate partial expansion involving the same number of steps. As can readily be seen, using realizable (unequal) growth rate partial expansions has a negative impact on utilization, but the impact is very small indeed.

5. SIMULATION STUDY FOR "SMALL" BUCKETS

A simulation study involving "small" bucket sizes was done so as to attempt to confirm the results of the analysis. Our particular concern was that the analysis assumed that records were infinitesimal in order to compute the recurrence and that a limit was taken in order to produce the closed-form solution. The simulation dealt with specific finite-size buckets. To approximate the uniform growth process that the analysis assumes, keys from a uniform distribution were inserted into a simulated B-tree file, and the nodes were expanded or split as required by the growth process.

Ten thousand records were inserted into the B-tree during each simulation run. The smallest bucket size was varied from 12 to 48 records capacity. Twelve-record buckets, or multiples of 12, were chosen so as to produce integral-size buckets for the various partial expansions. (A smallest bucket size of 6 was also

Table II[a]

Page size	E	Sim($u_{max=1}$)	Standard error	Sim($u_{max} = M/M + 1$)
6	1	71.89	0.10	61.62
	2	88.75	0.14	76.07
	3	93.56	0.05	80.19
12	1	70.83	0.23	65.38
	2	85.78	0.21	79.18
	3	91.28	0.10	84.26
	4	94.22	0.11	86.97
24	1	69.92	0.43	67.12
	2	84.18	0.16	80.81
	3	90.14	0.15	86.53
	4	92.71	0.12	89.00
36	1	68.80	0.11	66.94
	2	83.92	0.30	81.65
	3	89.32	0.23	86.91
	4	92.38	0.24	89.88
48	1	70.43	0.51	68.90
	2	83.93	0.38	82.22
	3	89.42	0.32	87.60
	4	92.34	0.21	90.46

[a] The relative utilization for varying numbers of partial expansions, as produced by simulation. The "Sim($u_{max} = \cdots$)" column gives the utilization assuming first that $u_{max} = 1$ and then that $u_{max} = M/(M + 1)$. M is the smallest bucket's capacity in records.

simulated, but not for four partial expansions.) The number of partial expansions involved varied from one to four, that is, from doing conventional B-tree node splitting to using four different bucket sizes. Four runs were taken at each setting of the expansion and bucket size parameters. The average of the utilizations for these four runs is presented in Table II and can be compared with the analysis results of Table I.

It is apparent from Table II that the simulation results very closely match the analysis results, even for a bucket size as small as 12 records. Further, and not unexpectedly, the agreement with analysis improves as the bucket size increases. It is interesting to note that the simulation results when maximum utilization is 1.0 were consistently better than the analysis results by a small margin, with only one exception. The actual maximum utilization will vary between 1.0 and $M/(M + 1)$, where M is the capacity of a bucket in records. Thus, these results, both of analysis and simulation, appear to be quite robust. Given a "uniformly growing" file, there is a high probability that the utilizations achieved will be very close to the results reported here. Note that these bounds bracket the analysis results very tightly, and the standard error is very small.

6. EFFECT OF PARTIAL EXPANSIONS ON INSERTION COST

Frequently, improvements in one performance attribute must be purchased with a negative performance impact on some other attribute. This does not seem to be the case with partial expansions. Each doubling of some part of a file will involve a larger number of distinct file growth steps, this number being precisely

the number of partial expansions per full expansion. However, partial expansions involving bucket growth are less costly than those involving node splitting. This is analyzed below in terms of the disk accesses required for file growth.

We make the following assumptions regarding the costs of file growth:

(1) There is no disk access cost for allocating a bucket or for freeing a bucket. Thus, our analysis assumes the bookkeeping for disk storage is done largely in main memory. There may be the need for an occasional write of this information onto the disk, but the number of such writes is assumed to be very small compared to the number of allocates and frees. While this assumption is not true of all current systems, it can be achieved without undue difficulty. It should be noted that, if allocation and freeing of buckets requires disk access, the results reported here will not accurately reflect either the absolute or relative costs.

(2) File growth occurs during an insertion to the same bucket that will be expanded. Thus, extra accesses to read and write some other expanding bucket will not be needed. This assumption will be true of B-trees and extendible hashing [2], but will not necessarily be true of other hashing methods or of indexing methods using multibucket nodes. As discussed in [9], the disk access cost for file growth that must be added to normal insertion cost will be very low for multibucket nodes, and so file growth costs per record will be small for all forms of expansion.

(3) Any cost for absorbing overflow records into the expanding "primary" bucket is not included. This analysis is essentially for B-trees. It ignores overflow records, which are virtually inevitable for hashed files. Handling overflow records adds substantially to file growth costs, as Larson has shown [4].

(4) The cost of updating levels of an index higher than the bottom level (the level that references the data nodes) is insignificant. Typically, index buckets have a fanout of 100 or more. Thus, the fraction of the time that higher index levels need to be updated is typically less than 2 percent of the update frequency for the bottom index level. Thus we ignore this cost. Note that this reduces the cost for node splitting, and hence has a slightly negative impact on the relative performance of partial expansions versus full expansions. However, the impact is not significant.

(5) Index entries must include a field that indicates the size of the buckets within a node. For single bucket nodes, this corresponds to having a node-size field. The number of different bucket sizes is E, the number of partial expansions per full expansion. Thus $\lceil \log(E) \rceil$ bits are required for the size field. Given the number of partial expansions in the useful range (2–4), only one or two bits are required for this field. Its impact on the size of index terms will be small, and will only affect growth frequency above the bottom level of the index, which is already being ignored.

Given the above assumptions, which are admittedly favorable to partial expansions, but nonetheless readily achievable in a B-tree organized file system, we incur the following disk access costs for file expansion:

(1) The cost of forming the new node required of node splitting is two extra disk accesses per bucket over the cost of a simple insertion that does not expand

the file. One disk access writes the new node (the write required of an insertion in the absence of file expansion takes care of writing the old node). The second disk access writes a new index page, which contains the newly inserted index term for the new bucket.

(2) The cost of a partial expansion in which the size of the bucket is increased is one disk access. This access writes a new index page, which contains the updated index term for the expanded bucket.

The result of this is that, for multiple partial expansions per full expansion, each of the smallest buckets has a file growth cost of two disk accesses. The cost for each of the next larger size bucket is one disk access more than this. Hence its total cost is three disk accesses, two to create it as a small bucket, and one more to expand it. Each increase in bucket size adds another disk access to the cost of that size bucket. If we let b_i be the number of buckets in the ith partial expansion and E be the number of partial expansions, then the total cost of having grown the file is

$$\text{Cost}_E = \sum_{i=1}^{E} (2 + i - 1)b_i.$$

To determine the expansion cost per record, we divide our expansion cost by the number of records in the file. This is

$$\text{Records} = u_{\text{file}}\text{NP}_{\text{Total}}\text{Pagesize}.$$

Thus

$$\frac{\text{Cost}_E}{\text{Records}} = \frac{1}{u_{\text{file}}\text{Pagesize}} \sum_{i=1}^{E} \frac{(1 + i)b_i}{\text{NP}_{\text{Total}}}.$$

For our realizable partial expansions, when $i > 1$, we expand bucket size by one page each time a bucket fills up. Then

$$R = \frac{E + i - 1}{E + i - 2}.$$

When $i = 1$, this is the case where we have split our largest bucket into two of our smallest buckets, yielding

$$R = \frac{2E}{2E - 1}.$$

Then the number of buckets in each partial expansion is

$$b_i = \frac{\text{NP}_i}{\text{bucketsize}_i} = \frac{\text{NP}_i}{E + i - 1},$$

where bucketsize_i is the number of pages per bucket of the ith partial expansion. Thus

$$\frac{\text{Cost}_E}{\text{Record}} = \frac{1}{u_{\text{file}}\text{Pagesize}} \sum_{i=1}^{E} \frac{1 + i}{E + i - 1} f_i,$$

where, as before, $f_i = \text{NP}_i/\text{NP}_{\text{Total}}$.

Finally, we wish to normalize the cost, so that the cost is a function of the number of partial expansions and is not confused by the absolute size of the buckets. Thus, instead of the cost of expansion per record, we wish to compute the cost of expansion per smallest bucket in the partial expansions. The cost per record must thus be multiplied by the capacity of the smallest bucket, which, since the number of pages in the smallest bucket equals the number of partial expansions, is Pagesize $\times E$. We then have

$$\frac{\text{Cost}_E}{\text{Smallbucket}} = \frac{E}{u_{\text{file}}} \sum_{i=1}^{E} \frac{1+i}{E+i-1} f_i.$$

This result is an attribute of the file expansion process and does not depend on bucket size. Cost per record is found by dividing by the capacity, in records of the smallest bucket. Since our normalized cost per full small bucket is constant, by the analysis, we can see that increasing smallest bucket size will result directly in a reduction in the expansion cost per insertion.

We can also derive the expansion costs discussed above from our simulation. As before, our results are the average of four runs. It should be noted that, while utilization changes little from run to run, the fraction for the buckets of various sizes changes more substantially. Nonetheless, the average is in generally good agreement with the analysis. This leads to expansion cost per small bucket being in good agreement with the analysis. The results for both analysis and simulation are reported in Table III.

The interesting thing to note concerning the results in Table III is that the cost of inserting a full small bucket of records into a file is least with two partial expansions. The cost of the original B-tree bucket splitting is 20 percent higher. In fact, while the costs for higher numbers of partial expansions increase, these costs do not exceed the B-tree bucket splitting cost until there are five partial expansions per full expansion. These results should not be too surprising. Each set of partial expansions has a smallest bucket size that equals the full expansion bucket size. Thus all other buckets in the higher order partial expansions are larger than this. Hence there are fewer of them. Thus, even though their costs are higher, their higher capacities and higher utilizations result in an overall savings.

So long as the number of records per smallest bucket is sufficiently large (e.g., 10 records), the cost that file expansion adds to insertion cost will be modest. At 10 records per bucket, the insertion cost, without file growth, is typically about three disk accesses, two to access the bucket via the index and one to write it back with the new record inserted. (We assume that all but the bottom level of the index and the data records themselves are held in main memory.) At 10 records per bucket, file growth adds less than 10 percent to the cost of insertion until the number of partial expansions exceeds five. The file growth cost is less than 20 percent for up to eight partial expansions.

7. INDEX SIZE IMPLICATIONS

The reduced insertion cost is a consequence of the reduced number of buckets required for the file. This reduced number of buckets is also of interest because it is directly proportional to the number of index entries for the file, and hence

Table III[a]

E	$E/E = 1$	$(E + 1)/E$	$(E + 2)/E$	$(E + 3)/E$	Cost
1	1.00 (1.00)				2.89 (2.87)
2	0.40 (0.43)	0.60 (0.57)			2.41 (2.38)
3	0.26 (0.26)	0.43 (0.38)	0.32 (0.36)		2.52 (2.52)
4	0.19 (0.18)	0.33 (0.29)	0.26 (0.27)	0.22 (0.27)	2.78 (2.69)

[a] The file growth cost, in number of disk accesses required for each full smallest bucket (Cost), is shown. Also shown is the relative abundance of different size buckets. The column headed by $(E + i)/E$ gives the number of buckets whose size, relative to the smallest bucket, is given by the expression. The results shown are from the analysis and, in parentheses, from the simulation with page size of 48 records.

the index size. This reduction in index size can result in either increased performance from a fixed-size buffer for containing index pages, or decreased index page buffer size for constant performance. The reduction in number of buckets is a result of two factors, increased bucket utilization and increased average bucket size. We determine the relative number of buckets produced by a partial expansion regime as a fraction of the buckets produced by a full expansion.

Let Records be the number of records inserted into the file, u_E be the utilization achieved for a file undergoing E partial expansions per full expansion. Then the total number of pages is

$$\mathrm{NP}_{\mathrm{Total}E} = \frac{\mathrm{Records}}{u_E \mathrm{Pagesize}}.$$

Let $\mathrm{Size}_{E,i}$ be the relative size of a bucket in the ith partial expansion in a regime with E partial expansions, where $\mathrm{Size}_{E,1} = 1$. Then the number of buckets of this size is

$$\mathrm{NB}_{E,i} = \frac{f_{E,i}\mathrm{NP}_{\mathrm{Total}E}}{\mathrm{Size}_{E,i}}.$$

The total number of buckets is

$$\mathrm{NB}_E = \sum_{i=1}^{E} \mathrm{NB}_{E,i} = \frac{\mathrm{Records}}{u_E \mathrm{Pagesize}} \sum_{i=1}^{E} \frac{f_{E,i}}{\mathrm{Size}_{E,i}},$$

where $f_{E,j}$ is the fraction of pages in a partial expansion regime of E partial expansions in the ith partial expansion. Dividing this by NB_1, which is the number of buckets from a full expansion, and which is

$$\mathrm{NB}_1 = \frac{\mathrm{Records}}{u_1 \mathrm{Pagesize}},$$

yields

$$\frac{\mathrm{NB}_E}{\mathrm{NB}_1} = \left(\frac{u_1}{u_E}\right)\left(\sum_{i=1}^{E} \frac{f_{E,i}}{\mathrm{Size}_{E,i}}\right).$$

The first term is the factor in the bucket number reduction due to improved storage utilization, while the second term is the factor due to increased average bucket size. The expansion ratio for the ith partial expansion is

$$r_{E,i} = \frac{E + i - 1}{E + i - 2}$$

for $i > 1$ and

$$r_{E,i} = \frac{2E}{2E - 1},$$

and the relative size of its buckets is

$$\text{Size}_{E,i} = \frac{E + i - 1}{E}.$$

Finally, from our previous analysis, and assuming all buckets have the same maximum utilization, we have

$$u_E = \frac{u_{\max} \log(2)}{\sum_{i=1}^{E} (r_{E,i} - 1)}$$

and

$$f_{E,i} = \frac{r_{E,i} - 1}{\sum_{i=1}^{E} (r_{E,i} - 1)}.$$

Then, substituting and performing algebraic manipulation yield

$$\frac{\text{NB}_E}{\text{NB}_1} = \frac{E}{E(2E - 1)} + \sum_{i=2}^{E} \frac{E}{(E + i - 1)(E + i - 2)}.$$

Mathematical induction can be used to show that this is

$$\frac{\text{NB}_E}{\text{NB}_1} = \frac{E}{2E - 1}.$$

The results for various partial expansion regimes are presented in Table IV. The ratio of buckets for a partial expansion regime versus a full expansion, as determined by the analysis, is presented. In parentheses are the ranges of bucket ratios encountered in the simulation study. As before, the agreement between analysis and simulation is quite good. Also given are the factors in this reduction that can be attributed to improved utilization and to increased average bucket size.

Note, in Table IV, that the limit of possible reduction in number of buckets is also presented. The important point in presenting the limit is that it demonstrates that most of the possible reduction is achieved when $E = 2$, that is, two-thirds of the total possible gain. Further, very little is to be gained from regimes employing more than four partial expansions.

Table IV[a]

E	Ratio	Utilization	Bucket size
1	1.000	1.000	1.000
2	0.667 (0.658–0.677)	0.833	0.800
3	0.600 (0.594–0.610)	0.783	0.766
4	0.571 (0.563–0.574)	0.760	0.752
Inf	0.500	0.693	0.721

[a] The ratio of number of buckets resulting from partial expansions to the number of buckets produced by node splitting (i.e., a full expansion) is given in the "Ratio" column, first the results of analysis, then the range observed during the simulation runs. The factors in this ratio attributable to the differences in utilization and to the differences in bucket size are also given. The relative number of buckets is equal to the relative number of index terms for the various partial expansions.

8. DISCUSSION

The notion of elastic buckets, that is, buckets that come in a number of sizes and that permit file space growth to occur at a rate that is less than a factor of two, permits the use of partial expansions with most indexed files in a completely straightforward way. Disk space allocation is somewhat more complicated than with single-size buckets, but is readily provided. However, if disk allocation is not done carefully, external storage fragmentation can result in substantially lower effective disk utilization. The analysis and simulation that have been done have treated files without overflow records, of which B-trees are the prime example.

For B-trees, the case for performing partial expansions is compelling. Improved utilization and insertion performance and reduced index size are all positive results of using two or three partial expansions per full expansion. These two regimes achieve most of the benefits that partial expansions can yield, while minimizing the extra potential cost of disk management. Two partial expansions per full expansion yield almost half of the utilization gain that is possible, 14 percentage points (to 83 percent from 69 percent) of a total of 31 percentage points. Going to three partial expansions results in a total gain of 19 percentage points to 88 percent, or almost two-thirds of the total potential gain.

The smaller number of buckets required when partial expansions are employed has two results. First, in terms of insertion cost, the two-partial expansion regime is best, with a file growth cost that is about 15 percent less than the full expansion cost. The insertion cost impact is not adverse until we reach five partial expansions. Second, the amount of space needed by the index is decreased. The two partial expansion regime requires only two-thirds of the index terms of full expansions, which is two-thirds of the gain achievable using the partial expansion technique. Going to three partial expansions results in six-tenths of the index terms of full expansions, four-fifths of the potential gain possible.

Larson [4] has demonstrated the gains to be achieved from partial expansions for one hashing technique. However, each hashing technique will necessitate a separate analysis. The existence of overflow records adds a substantial extra cost to file growth and hence to insertion cost. However, as Larson demonstrates, partial expansions appear to provide a favorable trade-off between utilization and search performance gains versus increased insertion cost, particularly as

bucket size increases. Again, the trend is toward smaller gains and larger insertion cost penalties as the number of partial expansions is increased past three. Elastic buckets reduce the shuffling of records between buckets. This reduces the need to either recompute the hashed key value or to store the hashed key value with its associated record. This reduction in algorithmic complexity is difficult to quantify but is no less real.

One final caveat is that the analysis and simulation results for partial expansions are derived assuming a "uniformly growing" file. The results for the full expansion case are derived in the same way. While this seems to be a reasonable assumption, in that usually the initial records added are a fair sample of the records that will be subsequently inserted, not all files grow in this fashion. However, no growth technique is likely to be optimum for all possible file growth scenarios. Further, partial expansions place a lower bound on utilization that is better than the lower bound produced by full expansions. Instead of 50 percent utilization, a two partial expansion regime will have a minimum utilization of 67 percent, while three partial expansions will have a minimum utilization of 75 percent. (Both these minimums are for the largest growth factor expansion needed to realize a particular number of partial expansions.)

Note. After this report was in essentially finished form, we learned from Georges Gardarin, Patrick Valduriez, and Yann Viemont at the 1985 SIGMOD Conference that INRIA's SABRE system uses a form of elastic bucket partial expansion. The problem they were solving arose because node splitting in SABRE is done digitally, based on a binary "signature." This leads to an unequal division of the contents of a node. Their multipage nodes permit the node splitting to likewise be an unequal division, approximately proportional to the division of the data. The new smaller nodes are then permitted to increase in size so that this process can be repeated.

REFERENCES

1. BAYER, R., AND MCCREIGHT, E. M. Organization and maintenance of large ordered indices. *Acta Inf. 1*, 3 (1972), 173–189.
2. FAGIN, R., NIEVERGELT, J., PIPPENGER, N., AND STRONG, H. R. Extendible hashing—A fast access method for dynamic files. *ACM Trans. Database Syst. 4*, 3 (Sept. 1979), 315–344.
3. KNUTH, D. *The Art of Computer Programming. Vol. 3, Sorting and Searching.* Addison-Wesley, Reading, Mass., 1973.
4. LARSON, P. Linear hashing with partial expansions. In *Proceedings of the 6th Conference on Very Large Data Bases* (Montreal). 1980, pp. 224–232.
5. LITWIN, W. Virtual hashing: A dynamically changing hashing. In *Proceedings of the 4th Conference on Very Large Data Bases* (Berlin). 1978, pp. 517–523.
6. LITWIN, W. Linear hashing: A new tool for file and table addressing. In *Proceedings of the 6th Conference on Very Large Data Bases* (Montreal). 1980, pp. 212–223.
7. LITWIN, W., AND LOMET, D. The bounded disorder access method. In *Proceedings of the 2nd Conference on Data Engineering* (Los Angeles). 1986, pp. 38–48.
8. LOMET, D. B. Digital B-trees. In *Proceedings of the 7th International Conference on Very Large Data Bases* (Cannes). 1981, pp. 333–344.
9. LOMET, D. B. Bounded index exponential hashing. *ACM Trans. Database Syst. 8*, 1 (Mar. 1983), 136–165.
10. LOMET, D. B. A high performance, universal, key associative access method. In *Proceedings of the ACM SIGMOD Conference on Management of Data* (San Jose, Calif.). 1983, ACM, New York, pp. 120–133.

11. LOMET, D. B. A simple bounded disorder file organization with good performance. Tech. Rep. TR-86-13 (submitted for publication), Wang Institute (Sept. 1986).
12. MARTIN, G. Spiral storage: Incrementally augmentable hash addressed storage. Theory of Comput. Rep. 27, Univ. of Warwick, Coventry.
13. YAO, A. C. -C. Random 3-2 trees. *Acta Inf.* 9 (1978), 159–170.

Received September 1985; revised June 1986; accepted June 1986