E-trees paper is a node partitioning scheme for B-trees to enhance concurrency. The main contribution of this method is to reduce the tree height and improve performance. Parallel processing of large node B-trees may not improve the response time and may even degrade the performance. The proposed method achieves better performance while avoiding this anomaly.

Index Terms—B-trees, parallel processing, queueing models, cost-effective when the size of a file is large.

I. INTRODUCTION

Many proposals have been made for parallel processing of database systems [4], [11]. Several of these proposals utilize parallel scanning of flat files [8], [12]. However, these scanning methods may not be cost effective when the size of a file is large. It is because the search time grows linearly with file size in these methods.

For queries requiring multiple hits, a file can be distributed among parallel devices for efficient parallel processing. Several such techniques have been proposed for processing queries in parallel [3], [6], [10].

In this paper, we will investigate data distribution methods for B-tree type indexes to enhance concurrency in processing queries. B-trees are the most commonly used tree type index structure for external databases [1], [2]. We propose a node partitioning scheme for large node B-trees for parallel processing. In the proposed scheme, each node is partitioned into multiple subnodes to be distributed for parallel processing. We also propose a new approach to process these subnodes. We will call the proposed B-tree structure PNB-trees (partitioned node B-trees), while the standard B-trees will be called conventional B-trees.

The main results presented in this paper are that the parallel processing of PNB-trees reduces access time, increases throughput, and minimizes the frequency of tree restructuring. We do not claim that the basic structure of PNB-trees is different from B-trees. However, we use a different approach to process and search PNB-trees. The PNB-tree approach exploits parallel scanning by developing a new physical structure for a B-tree.

The remainder of this paper is organized as follows. In Section II, we present the basic structure of PNB-trees, and the search and update algorithms. The hardware environment for the PNB-tree construction is also discussed in this section. The important parameters of the PNB-tree are described in Section III. Section IV presents a performance comparison between PNB-trees and conventional B-trees. Finally, PNB-tree construction for various disk technology is discussed. Section VI contains the conclusion.

II. THE PNB-TREE

PNB-trees are constructed on synchronized disks where read/write heads of all disks are located at the same position [15]. The PNB-tree is a B-tree with each node partitioned into s subnodes, and all the subnodes of a node are stored on s different disks. The format of the node is 

\[(K_1, P_1), (K_2, P_2), \ldots, (K_m, P_m)\]

where \(K_i\) is the key value and \(P_i\) is the address of the node subnode in each disk. This index node is partitioned into s subnodes, and all the subnodes of a node are accessed by the same address pointer. All the subnodes have the same format except the first one, where the key value in the first record is omitted. The leaf nodes which contain the data are also partitioned into s subnodes. However, PNB-trees differ from conventional B-trees in that the key values within a node in PNB-trees are not required to be in sorted order. This needs extra computation to determine the child address, but the amount of data movement between disks is reduced considerably. This is because a subnode does not overflow. The rule of locating a data record is defined recursively as follows:

Let \(R = \{(\phi, P_1), (K_1, P_1), (K_2, P_2), \ldots, (K_m, P_m)\}\) be a current index node. Then the data record with a key value \(K\) is in the descendant node pointed by \(P_1\), if \((K, P_1) \in R - \{(\phi, P_0)\}\) and \((K - K_i)\) is the minimum nonnegative value among all \((K - K_i), i = 1, \ldots, m\). If there is no nonnegative \((K - K_i)\), the desired record is in the descendant node pointed by \(P_0\).

An example of a PNB-tree is given in Fig. 1. Here, each node is partitioned into two subnodes being stored on disks D1 and D2. When the value 56 needs to be searched, we first compute the difference 56 - \(K_s\) for all the key values in the root node. These values are \(-16, 37, -31, 24\). By following the pointer \(K_s\) such that 56 - \(K_s\) = 24, the node (32, 56, 41, 63) is obtained.

The reason for the use of unsorted nodes is to minimize the interdevice data movement. Suppose that a node is in sorted order, and is partitioned and stored as above. Whenever a subnode overflows, data movement between disks is required. However, this does not happen in unsorted node construction because inserted records can be placed in any subnode of a node.

A. The PNB-Tree Operations

1) Lookup: The lookup of a record starts from the root node and continues until the leaf. All the subnodes of a node are searched in parallel. Let \(P\) be the address of the node currently being accessed. Initially, \(P\) is the address of the root. The lookup procedure consists of two phases. At first, \((K, P)\) is broadcast to all the disks, where \(K\) is the desired key value. If \(P\) is the address of a leaf node, find the record and stop. If not, each disk finds an index record \((K_i, P_i)\) in its subnode such that \((K - K_i)\) is the minimum nonnegative value. In the second phase, find the minimum of these minimum nonnegative values at each disk. If no nonnegative value is found at any disk, repeat the first and the second phase by replacing \(P\) with \(P_0\). Otherwise, repeat the same by replacing \(P\) with \(P_j\), where \((K_j, P_j)\) is an index record and \(K_j\) is the overall minimum nonnegative value.

2) Insertion: To insert a record, apply the lookup procedure to find the desired leaf node. We then insert the record into any place within the node, if it has empty space. If the node is already full, it is sorted and then split into two nodes. The effect of node splitting on its parent node is handled recursively in the same way as in standard
B-trees. Fig. 2 shows the PNB-tree configuration after the key value 70 has been inserted into the tree of Fig. 1.

3) Deletion: The node underflow in PNB-trees is handled the same way as in B-trees. However, PNB-trees have an advantage over B-trees because the holes created by deletion can be easily filled by any records inserted. This is because the values within a node need not be kept in sorted order. In conventional B-trees, the holes can also be used but only at the cost of sorting the node each time.

B. Disk Hardware Architecture

For each disk we assume a simple microprocessor with a few thousand bytes of memory. All the subnodes of a node are processed locally by these processors. Only the matched record is sent to the host processor. For the second phase of a lookup procedure, we use a minimum finding hardware module to which all disks are directly connected. This module is a simple extension of a multiplet input comparator. As soon as each disk finds an index record \((K_i, P_i)\) such that \((\text{desired key value} - K_i)\) is minimum nonnegative value, it sends that index record to this minimum finding hardware. If a disk does not find a nonnegative value, it sends \((-1, \text{null})\). The minimum finding hardware determines the overall minimum nonnegative value and then broadcasts the corresponding address of the child node to all the disks. When all the disks reply with \((-1, \text{null})\), the address of a child node is the first pointer in the first subnode of a node. Since the communication involved is very simple and communication distance is short, we expect that the time for these operations is negligible compared to disk access time.

III. MOTIVATION OF PNB-TREES

The height of an index tree is the most important parameter in data retrieval time when the file is stored in the secondary storage. One approach to reduce the height of a B-tree is to partition a file horizontally, and construct a B-tree for each partition. These B-trees are searched in parallel. When the number of records in a file is \(n\) and is partitioned into \(s\) subsets, the height of each B-tree is approximately \(\log_s(n/s)\), where \(f\) is the fan-out of an index node. On the other hand, PNB-trees reduce the tree height by using a large node. When the size of index nodes is increased \(s\) times, the tree height becomes approximately \(\log_s(n)\) which is smaller than \(\log_s(n/s)\) for most practical cases. This approach also reduces the frequency of tree restructuring.

A. Compressed Height

Let \(s\) be the number of disks, \(n\) be the number of records, \(r\) be the size of a data record, and \(k\) be the key size. Let the block size be \(b\) bytes and the pointer size be \(p\) bytes, where a block is a unit of data transfer. This block corresponds to a subnode in PNB-trees and a node in conventional B-trees. Let \(h\) be the height of a PNB-tree, where \(h\) includes the level of data nodes. To compute the worst case height we assume every block to be half-full except the root node. Then, the maximum number of leaf nodes is approximately

\[
\frac{s^h b}{k + p} \left(\frac{b}{2(k + p)}\right)^{h - 2}.
\]

Since we need about \(2nr/b\) data blocks (leaf nodes), the relation

\[
\frac{s^h b}{k + p} \left(\frac{b}{2(k + p)}\right)^{h - 2} \geq \frac{2nr}{b}
\]

should be satisfied [7]. For all the examples in this paper, we will assume \(r = 200\) bytes, \(k = 30\) bytes, and \(p = 4\) bytes. Then, \(b \geq 2500\) if \(h = 3\), \(s = 4\), and \(n = 1\). Therefore, a 3 kilobyte block size is enough for making two-level indexes. Even for large files with 60 M records the block size of 5 kilobytes guarantees two-level indexes when eight disks are used. On the other hand, four level indexes are needed for this file when the conventional B-tree of 5 kilobyte block size is used. To guarantee two level indexes we need to use a block size of about 40 kilobytes. In this case, the block transfer time becomes quite significant.

The node size of PNB-trees is large, but I/O and computation time improves significantly because the large node is split into smaller subnodes and these subnodes are processed in parallel. Since the height of a B-tree decreases with increasing node size, the number of disk accesses in PNB-trees is minimized. Furthermore, the block transfer is reduced because only the subnode contributes to the block transfer time. Note that the height can also be reduced in conventional B-trees by making the node size large. But this results in a long block transfer time and more main memory processing time.

B. Reduced Frequency of Tree Restructuring

The B-tree restructuring due to a node overflow is expensive. One way to reduce the frequency of tree restructuring is to use a large
node. When the node size increases $k$ times, the frequency of tree restructuring due to a node overflow reduces by a factor of $1/c_k$, where $c$ is slightly larger than one. This is because the number of nodes in a B-tree decreases almost linearly with node sizes. The detailed analysis for computing the average number of nodes in a B-tree is given in [14].

IV. PERFORMANCE COMPARISON

In this section, we compare the response time of PNB-trees to conventional B-trees which are stored on parallel disks.

<table>
<thead>
<tr>
<th>Disk #</th>
<th>Performance Model</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.458</td>
<td>D1 D2 D3 D4 D1 D2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

A. Performance Model

For conventional B-trees, each data request is directed to the disk which contains the corresponding B-tree. Thus, a request queue is formed in front of each disk. In PNB-trees, all data requests form one queue.

Let $X$ be a random variable for the disk service time of a single disk. Then, $X = S$ (seek time) + $L$ (rotational latency) + $T$ (block transfer time). For conventional B-trees on parallel disks $RPS$ (rotational positioning sensing) miss delay should be added for disk access time [5]. $RPS$ miss delay happens due to the channel contention by concurrent disk I/O's. Let $X_i$ be the disk service time for disk $i$ in conventional parallel disks. Then, $X_i = S + L + T + RPS_i$. We define the disk response time to be the disk service time plus the queue waiting time for the disk service. The queue waiting time is computed as follows. The arrival process for disk access requests is assumed to be Poisson process with mean $R$. $L$ is uniformly distributed with mean one half of one disk rotation time and $S$ is assumed to be exponentially distributed. The number of $RPS_i$ misses is assumed to be geometrically distributed. Since the distributions of $S$, $L$, $T$, and $RPS_i$ are known, the mean and the variance of $X_i$ can be computed. We choose the $M/G/1$ queueing model to compute average queue waiting time. In the $M/G/1$ queueing model, the queue waiting time for disk $i$, denoted $W_i$, is given by

$$R_i(Var(X_i) + E[X_i^2])$$

$$2(1 - R_i E[X_i])$

where $R_i$ is the request rate for disk $i$ [16]. Thus, the average disk response time for disk $i$ is $E[X_i] + W_i$. Let $Z_j$ be the disk response time for a $j$th level node of a B-tree. We define the data response time $Z_j$ to be $Z_j = Z_1 + Z_2 + \ldots + Z_k$, where $k$ is the height of the B-tree. Thus, the average data response time $E[Z_j]$ is $R E[Z_1]$. Here, for all $j, j = 1, \ldots, k$

$$E[Z_j] = \frac{1}{2} \sum_{i=1}^{s} (E[X_i] + W_i)$$

where $s$ is the number of disks. For the PNB-tree model,

$$E[Z_j] = E[X] + R(Var(X) + E[X^2])$$

$$2(1 - R E[X])$$

where $R$ is equal to $\sum_{i=1}^{s} R_i$. Note that there is only one queue and there is no RPS miss delay in the PNB-tree model.

As described in Section II, PNB-trees use two-phase processing to locate a record. In the first phase, the computation time is overlapped with the data transfer time because processing can be done concurrently with the data transfer between a disk and a local buffer. For the second phase, we assume that it takes less than one millisecond to find the overall minimum value.

B. Average Data Retrieval Time

In this section, we assume IBM 3380 disk device, where $E[S] = 7.2$ ms, $E[L] = 8.3$ ms, and block transfer rate $= 3000$ kilobytes/second. When the height of a B-tree is 4, one data request requires four disk accesses. We want to compare the performance of conventional B-trees on parallel disks to PNB-trees on synchronized disks. As a typical example, four disks and eight disks are considered. In conventional parallel disks, data requests are usually skewed for some disks. The following skewed data request patterns are used in our performance analyses.

where $D_i$ represents disk $i$ and the number below $D_i$ represents the probability that some request is a disk $i$ request. Note that there is no skewed effect in the PNB-tree model because there is only one queue. Let the number of records be $M$ and all other parameters be the same as in Section III-A. When four disks are used, the data response time for conventional B-trees is 70.07 ms while the data response time for PNB-tree is 54.09 ms. We have used a 4 kilobyte block size with a data request rate of 2 per second. When eight disks are used, the block size is 2 kilobytes and the data request rate is 2 per second, the data response time of conventional B-trees is 77.85 ms and that for the PNB-trees is 51.85 ms. The root node is assumed to reside in the main memory for all response time computation. The height of the B-tree is computed with the assumption that every node is half-full.

Many parameters need to be considered in comparing the response time of conventional B-trees on parallel disks and PNB-trees on synchronized disks. Here, we show response time comparisons for some typical parameter values. Figs. 3, 4, and 5 show the response time comparison for various disk speeds, when the data request rate is 20/s. Disk speed $i$ in these figures denotes $1/i \times$ (average disk access time of IBM 3380 disk).

We consider three different models for performance comparison. The first is conventional B-trees with height 4. The second is conventional B-trees with height 3, and the third is PNB-trees with height 3. The second model uses an increased block size to reduce the height of the conventional B-tree. In the figures, 1, 2, and 3 denote the response time of the first, second, and third model, respectively.

Fig. 3 shows the results for four parallel disks with 1 M records. 4 kilobyte and 13 kilobyte block sizes have been used for the first and second model, respectively. The block size for PNB-trees is 4 kilobytes. The figure shows that the response time of conventional B-trees with 13 kilobyte block size is better than that of conventional B-trees with 4 kilobyte block size. However, large block sizes are not always better than small block sizes. For example, Fig. 4 shows the case for eight disks with 60 M records. Here, we use 10 kilobyte and 48 kilobyte block sizes for the conventional B-trees, and 6 kilobyte block size for the PNB-trees.

We see that when the file size is large, increasing block size to reduce the height increases response time in conventional B-trees. This is because the block transfer time is significant for large block size. Furthermore, response time suffers more $RPS$ miss delay due to the long block transfer time. On the other hand, PNB-trees perform better than conventional B-trees for both small and large block sizes. When database sizes are small, the performance of PNB-trees and large block size B-trees is almost the same. The saturation request rate is defined to be a data request rate which makes the queue length infinite. The PNB-tree achieves a larger saturation request rate than the conventional B-tree in the worst case, where every request goes to the same disk. Note that in PNB-trees the worst case is the same as the average case. When

$1)$ is data collected from a relatively well tuned system [5]. $2)$ is data collected from our environment.
The number of records is 1 M and eight disks are used with 4 kilo-
byte block size, the PNB-tree is saturated at 21 data requests/second
but the conventional B-tree is saturated at 14 data requests/second.
Therefore, PNB-trees can handle larger data request rate in the worst
case.

One problem of PNB-tree organization is an increased queue
length. This is because all disk requests go to the same queue. For
normal data request rate, the queue length is usually very small (less
than 0.1). But for high data request rate (e.g., more than 15 data
requests or more than 45 disk access requests, if the height of a
B-tree is 3), the queue waiting time is no longer negligible and the
PNB-tree suffers from long queue waiting time.

We also give the maximum data request rate (threshold point) un-
til the response time of the PNB-tree is smaller than the conventional
B-tree. Fig. 5 shows this result for various disk speeds. 1 M records
with four disks are used. All other parameters are the same as in
Section III-A. We can observe that threshold limit increases almost
linearly with disk speed.

V. OTHER STRATEGIES FOR THE PNB-TREE ORGANIZATION

We have described the PNB-tree to be a B-tree with unsorted
nodes. All the performance analyses presented so far are based on
the synchronized disks. However, several variations to this basic ap-
proach are possible. For example, PNB-trees can be constructed on
a set of asynchronous disks. These variations give different perfor-
mance tradeoffs.

A. Synchronized Disks with Sorted Nodes

When the key values of a node are kept in sorted order, the over-
all minimum finding hardware is not required. Here, the average
response time is about the same as that of the basic approach. How-
ever, the frequency of tree restructuring increases in this approach.
This is because a subnode within a node can overflow. Note that
there is no subnode overflow for the unsorted case.

B. Asynchronous Disks with Unsorted Nodes

In PNB-trees, constructed on synchronized disks, only one address
pointer is used to locate all the subnodes of a node. On asynchronous
disks, on the other hand, multiple pointers are needed to locate them.
Note that asynchronous disks do not use synchronizing disk-arms and
are the same as conventional parallel disks. This approach gives near
100% storage utilization because subnodes are dynamically allocated.
One problem of this approach is the increase disk access time. This
is because the disk access time for asynchronous disks is determined
by the worst case time among all the disks. When the response time
of each disk is assumed to be uniformly distributed between a and b,
E[Z] = (sb + a)/(s + 1), where Z is the disk response time of PNB-
trees constructed on s asynchronous disks (see the Appendix). When
s becomes large, E[Z]/E[X] is approximately 2b/(a + b). In other
words, when b > a, the response time of PNB-trees on asynchronous
disks is almost twice that of PNB-trees on synchronized disks.

C. Asynchronous Disks with Sorted Nodes

The disk access time for a successful search does not increase
in this case. This is because a match can be found without waiting
for responses from all disks. Therefore, the disk access time for a
successful search is almost the same as that of synchronized disks.
However, for an unsuccessful search, it takes the same amount of
time as in Section V-B.

The storage utilization is the same as in Section V-B but the fre-
quency of node restructuring increases as in synchronized disks with
sorted order.

VI. CONCLUSION

In this paper, we proposed parallel processing of B-tree nodes to
improve the response time of file accesses. This approach is based
on the partitioned node B-tree called the PNB-tree. We compared the
performance of PNB-trees to conventional B-trees on parallel disks.
Even though PNB-trees have only one disk access request queue
compared to many parallel request queues as in conventional B-trees
on parallel disks, the queue length for PNB-trees has been found
to be very small. Since the effect of queue waiting time is small,
the response time is mostly dependent on the number of levels of
the B-trees. Therefore, the height reduction of the B-tree along with
parallel processing within a node gives a better performance than
parallel processing of conventional B-trees on parallel disks. We also
show that the frequency of tree structuring due to node overflow
is considerably reduced in PNB-trees. We have proposed several
variations of the PNB-tree approach giving different performance
tradeoffs.
APPENDIX

AVERAGE DISK ACCESS TIME FOR ASYNCHRONOUS DISKS

$X_i$'s are independent and identically distributed random variables whose probability density function and distribution function are $g(x)$ and $G(x)$. Then the distribution function of $Z = \max(X_1, X_2, \ldots, X_s)$ is

$$ F(z) = P[Z \leq z] $$

$$ = P[X_1 \leq z]P[X_2 \leq z] \cdots P[X_s \leq z] $$

$$ = (G(z))^s $$

when all $X_i$'s are $U(a, b)$, $g(x) = 1/(b - a)$ and $G(x) = (x - a)/(b - a)$. Then

$$ F(z) = \left( \frac{z - a}{b - a} \right)^s. $$

The expected value of $Z$ is

$$ E[Z] = \int_a^b z f(z) \, dz, $$

where $f(z) = F'(z) = \frac{s}{(b - a)^s} (z - a)^{s-1}$

$$ = \int_a^b \frac{s}{(b - a)^s} z(z - a)^{s-1} \, dz. $$

Let $z = a + t$. Then $dz = dt$

$$ E[Z] = \int_0^{b-a} \frac{s}{(b - a)^s} (a + t)^{s-1} \, dt $$

$$ = \frac{sb + a}{s+1} $$

$$ E[Z] = \frac{2(sb + a)}{(s+1)(a + b)} $$

$$ \lim_{s \to \infty} \frac{E[Z]}{E[X]} = \frac{2b}{a + b}. $$

ACKNOWLEDGMENT

We would like to thank the referees for their helpful comments.

REFERENCES