DataBase Management System

Lecture Notes

Performance Optimization with Indexing

Date – 11/10/2011

1. Introduction:

The following class compares the performance of various indexing schemes with respect to operations like scan, equality search, range search, insertions and deletion.

2. Measures of Performance:

1. Time Complexity of operations
2. Space complexity of the indexing scheme
3. Ease of scalability of the database

*we will mainly discuss the time complexity of different operations

3. Comparison of Indexing structures:

3.1 Indexing structures to be compared:

1. Simple sorted file
2. Static tree index over a sorted file
3. Dynamic tree index over a sorted file

3.2 Review of terminology

P – Total number of pages in the file
D – Disk access time, i.e. time taken to load a page from disk to main memory
R – Number of records on a page

3.3 Sorted file

Consider a file which contains records that are sorted over the search key attribute

R_1	R_2	R_3	R_4	…	…	…	…	…	R_{M-1}	R_M

Performance of this file over some operations is as follows:

Scan – The cost is **PD** because all pages must be examined. Note that this case is no better or worse than the case of unordered files.

Equality Search – We assume that the equality selection is specified on the field by which the file is sorted. We can locate the first page containing the desired record or records, with a binary search in $\mathbf{log_2 P}$ steps. Each step requires a disk I/O, hence the total cost for equality search is $\mathbf{D log_2 P}$.

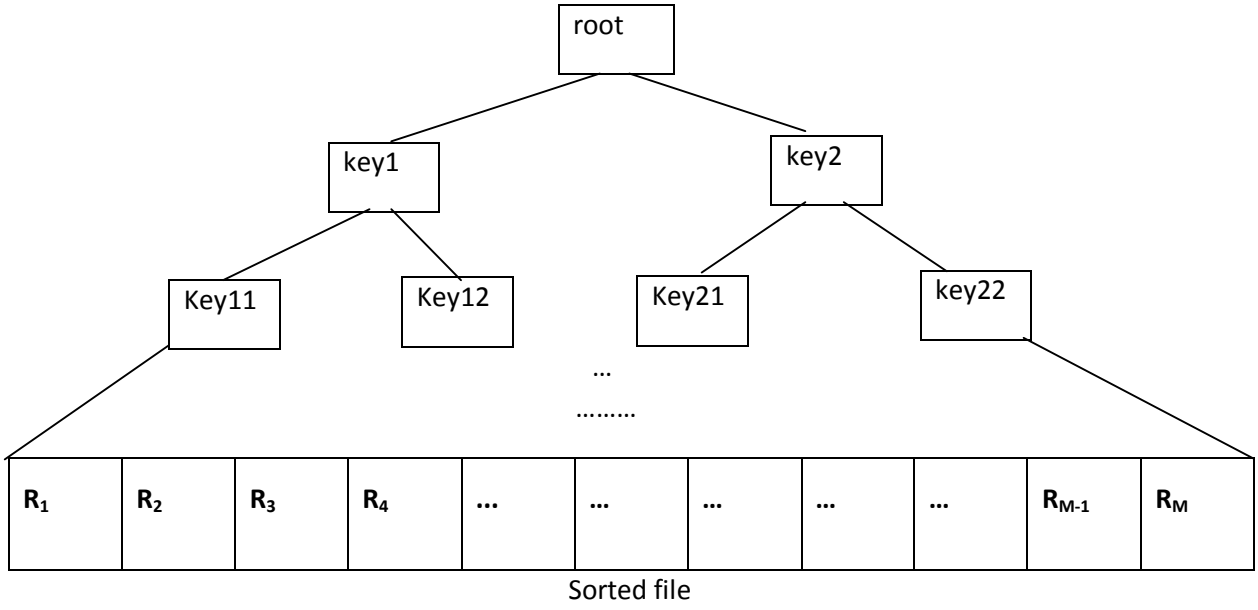
Range Search – The first record that satisfies the selection is located as it is for search with equality. Subsequently, data pages are sequentially retrieved until a record is found that does not satisfy the range selection; this is similar to an equality search with many qualifying records. So total cost comes out to be $\mathbf{D log_2 P}$ + **(# Matching Pages)**

Insertion – To insert a record while preserving the sort order, we must first find the correct position in the file, add the record, and then fetch and rewrite all subsequent pages. So cost for insertion will be (**PD + search**).

Deletion – We must search for the record, remove the record from the page, and write the modified page back. So cost for deletion is same as of insertion i.e. (**PD + search**).

But it is clear that the sorted file gives bad performance with the operations equality search (automatically affects range search), insertion and deletion. In further sections we will see how tree based indexing schemes help improve this performance.

3.4 **Static tree Index**



Static here means that the keys in the tree always remain the same (fixed in the beginning), whatever be the range of the data stored on the file.

Disadvantage: since the keys are fixed there is no self balancing in the tree and over time, some part of the tree may become much more heavily used than other parts and the reverse will be true for some parts.

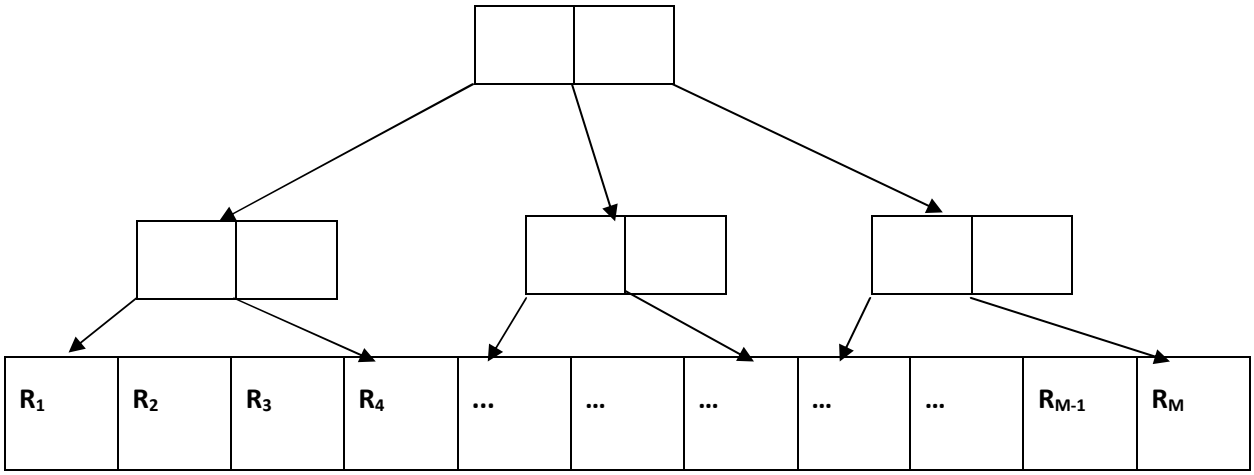
Improvements in performance:

1. Scan – for scanning the entire file, we can ignore traversing the index. So the time remains same as in a sorted file i.e. PD
2. Equality search - improved because a large portion of the tree is rejected each time we descend a level in the index tree.
3. Range search – this also gets improved as equality search improves.
4. Insertion – much more improved; same as equality search
5. Deletion – this also gets improved drastically on the same lines as insertion.

3.4 Dynamic Tree index – B+ Trees

Some important points with respect to a B+ Tree are the following:

1. All records are stored at the leaf nodes
2. The capacity of each leaf node is fixed, say R
3. Apart from root, every other node (both leaf and internal) has to be at least 50% filled. This is variable though and can e considered as a constant a (0<a<1)



P.S.: it is easy to see that as a (the percentage of node that should be always filled) increases, the cost associated with the restructuring of index during insertion and deletion operations will increase.

Improvements in performance:

1. Scan – same as that of sorted file and static tree index
2. Equality search – as good as static indexing; since there is no overcrowding in any part of the tree now, so the number of nodes to be traversed to reach the correct node is nearly same for all paths in the tree.
3. Range search – improves as a result of equality search as it is a clustered index.

4. Insertion – though now it would take less time to search for the correct location to insert the new record, but if the page where the record needs to be inserted is full, then the overhead of restructuring the index due to creation of a new page fades it out.
5. Deletion – deletion has the same overheads of restructuring as insertion.

Advantages of dynamic tree index:

1. Search is much more frequent operation and dynamic index beats the static index where overcrowding is expected in the tree.
2. Insert/delete operations are costly in rare cases only when a node becomes full or falls below the required threshold.
3. Most databases don't show frequent insertion and deletions of records; rather only modification.

Disadvantages of static tree index:

1. We can't carry out multiple insertion/deletion operations simultaneously.

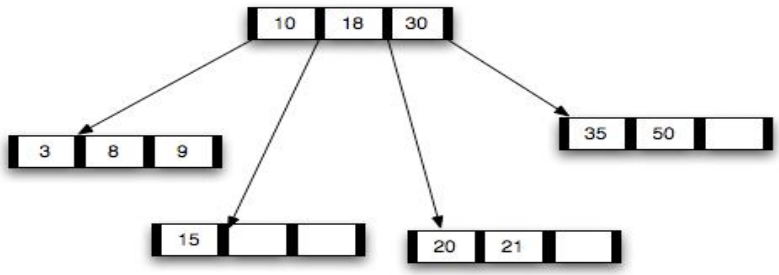
Suggested solutions in dynamic indexing:

1. Schedule the restructuring of the index at a fixed interval of time rather than every now and then

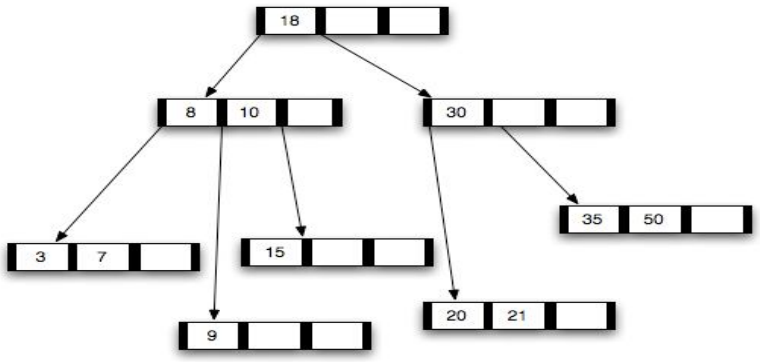
So it is concluded that though dynamic tree index gives better performance in most cases, the choice of the indexing scheme purely depends on the domain of use.

Restructuring in a dynamic tree index:

1. **Insertion**
   Suppose we insert a new record in the file and it is required to create a new page to insert the record. In this case we can't just create the new page (node) and insert the new record, but we also have to redistribute records in the nodes so that the new page is full beyond its threshold size.
   a. **Splitting of nodes**
      Suppose the page at whose end the record should have been inserted is full and has records R records already. Then we split the page into two pages having a common parent and insert the new record in the appropriate node.



B+ Tree, before inserting 7 in leaf node



B+ Tree, after inserting 7 in leaf node

b. Redistribution of records with the neighboring pages.
Suppose some of the neighboring pages have records much above the threshold occupancy, then some records from those neighbors can be shifted to the new page and value of the keys in the parent node can be appropriately changed.

2. Deletion
Suppose we delete a record and it causes the corresponding page to fall below its threshold occupancy, then we need to again redistribute records or merge the records of two different pages.

a. Merging
Just the inverse of splitting in insertion, here we merge to neighboring into one by combining their records and replace it by their parent.

b. Redistribution of records
Here again some records can be borrowed from the neighboring pages and the values of the keys in the parent node appropriately changed.

In today's world, most of the databases are expected to grow. Keeping this in mind we generally prefer splitting of pages in case of insertions, while preferring redistribution of records in case of deletion.

Submitted By:-

Ayush Parolia (09010165)
Pranav Gupta (09010161)