B+ tree Preference over B Tree

Amruta Kudale(A20330095)
Computer Science
Illinois Institute of Technology
Chicago, USA
akudale@hawk.iit.edu

***Abstract*— the purpose of this paper is to discuss the preference of B+ tree over B tree or any other indexing technique. Most of the modern commercial RDBMS are using B+ trees as their default indexing structure reason being the efficiency of B+ tree and the performance attribute of it. The cache hit is better for B+ trees. Various indexing techniques are discussed with their pros and cons.**

Keywords—indexing, B tree, B+ tree, has table, sequential indexing, search key, data value, root node, SQL Server, MySQL, Microsoft, Oracle.

I. Introduction (*Heading 1*)

This template, modified in MS Word 2007 and saved as a " Database management is a collection of programs that allows a user to specify the structure of the database. The user defines the schema of the database where the data is being stored, insertions, deletions and modifications can be done to the data also control access permissions can be set. Control access is used to give permission to specific set of users group who have permissions to edit the database depending on their role.

Relational Database Management system is most popular used database management system used today. SQL is used to implement RDBMS. RDBMS has collection of tables, meaning the data is stored in rows and columns of the table format. Importantly it forms bases for all the commercial database management systems like MS SQL Server, IBM DB2, Oracle, MySQL, and Microsoft Access [1]. The main feature of RDBMS is that the database is spread across several tables because of which is preferred over traditional DBMS as the changes made to one table does not affect the other tables.

Reason for the commercial RDBMS to use RDBMS is [2]:

- Data sharing is simpler and thus development of applications is faster.
- Redundancy of data is less.
- Data integrity is maintained as data is stored in one place.
- Physical data independence is provided.
- Same data can be viewed differently by different users.
- Database is expandable easily.
- Strong data protection is provided.
- Good backup and recovery system.
- Provides multiple interfaces because of which many users can access the database.

II. Indexing

Data in database is stored in an ordered fashion based on the primary key or any uniquely identified value. Indexing is a data structure technique which can efficiently retrieve records based on indexing done on these uniquely identified values [3]. Indexing is done based on the indexing attribute or on ordered field which is the file is arranged orderly. Types of indexing

a) Primary index : Indexing based on the primary key field
b) Secondary index : Indexing based on non-ordering field
c) Clustering index : Indexing based on ordered non-key field

There are two type of ordered indexing:

1. Dense index: There is index record for every search key. It takes more space in memory but advantage is that we get the access to record immediately.
2. Sparse index: There is index record contains search key value and pointer to a block of records. All the ordered records are formed in blocks and accessed through one single search key value of the block. Where the record can be searched through sequential search in the block after getting the access to the correct block. This take less space in memory but requires search time in the block.

III. Indexing Technique

A. Sequential indexing

In sequential indexing the records are arranged in sequential order. The indexed sequential file consists of the data file that contains the records in sequential manner. Secondly it consists of the index file which consists of the primary key and its address in data file. Advantage of sequential indexing is records are written and read in sequential manner. If the primary key is known to search a record then we can simply search for it in the index file and directly get the address of the record from the data file. Index file is sorted one. Alternate index can be created to fetch the records from the data file.

New records can be appended at the end of the file. Insertion is simple it is done at the end of the file. Deletion requires searching for the records sequentially and then delete which is difficult. Updating of a record can be done by simply overwriting the record if they are of same size. [4]

B. B Tree

B trees have a predefined range and may contain variable number of children. Since the range is fixed so if any node is inserted or deleted from B tree the internal nodes of the tree may be joined or split to balance the range of the tree. Every internal node contains number of keys between d and 2d, where d is the order of the B-tree. Any node can have children between d+1 to 2d + 1. B-tree is of sorted order. As mentioned early the index is adjusted and balanced on any insertion or deletions in the tree. As per Knuth's definition [20], B-tree of order n (maximum number of children for each node) is satisfied following property: 1. every node has at most n children. 2. Every node has at least n/2 children. 3. The root has at least two children if it is not a child node. 4. All leaf node at the same level. 5. A non-Leaf node have n children contains n-1 keys. It best case height of B-tree is logmn and worst case height is logm/2n. Searching in B-tree is similar to the binary search tree. Root is starting then search recursively from top to bottom. Within node binary search is typically used. Apple's file system HFS+, Microsoft's NTFS [21] and some Linux file systems, such as btrfs and Ext4, use B-trees. B-tree is efficient for the point query but not for range query and multi-dimensional data [19]. Spatial data cover space in multidimensional not presented properly by point. One dimensional index structure B-tree do not work well with spatial data because search space is multidimensional. [5]

By default, the Oracle creates a B tree index. In a b-tree, you traverse the tree until the required node is found. The classic B-tree structure has branches from the root that lead to leaf nodes that contain the data. The Oracle database implements the b-tree index in a little different manner. An Oracle b-tree starts with only two nodes, one root and one leaf. The root contains a pointer to the leaf block and the values stored in the leaf block. As the index grows leaf bocks are added to the index. To find a specific row, we look at the root to find the range of values in each leaf and then go directly to the leaf node that contains the value we are looking for. Since the root contains only pointers to leaf blocks, a single root node can support a very large number (hundreds) of leaf nodes. B-tree indexes are very powerful. Oracle has used b-tree indexes for many years. [6]

In SQL Server, indexes are organized as B-trees. Each page in an index B-tree is called an index node. The top node is called the root node in the tree. The bottom level of nodes is called the leaf nodes in the index. Any index levels between the root and the leaf nodes are called as intermediate levels. The underlying table data pages are contained in leaf nodes of clustered index. The root and intermediate level nodes contain index pages that contain index rows. Each index row contains a key value and a pointer to either an intermediate level page in the B-tree, or a data row in the leaf level of the index. The pages in each level of the index are linked in a doubly-linked list. [7]

C. B+ Tree

B tree is multi-level index format, which is dynamic and height balanced binary search trees also known as balanced sorted tree that . It is an efficient disk based data structure that stores key-value pairs. Single level index records becomes large as the database size grows, which also degrades performance. But it supports efficient look up of key.

All leaf nodes of B+ tree denote actual data pointers. Each non-leaf node with m index entries has m+1 children pointers. The leaf nodes contain data entries. Leaf pages are chained in a doubly linked list. B+ tree ensures that all leaf nodes remain at the same height, thus balanced. Additionally, all leaf nodes are linked using link list, which makes B+ tree to support random access as well as sequential access. [3]

B+ tree offers: 1. Fast record search 2. Fast record traversal and 3. Maintaining sorted tree structure without overflow pages. [8]

B+ tree contains key-value pair. The keys are an ordered list and there are pointers that point to values at lower level nodes in the tree. The pointers give a range of probable key values that can be found on next lower level. Insertion in B+ tree begins at the root node, finding the next adjacent keys between or close to the value being found and following the corresponding pointer to the next node in the tree. Recursive implementation of this algorithm results in finding the value in the tree or results in non-existence of it. [9]

A clever balancing technique is used in B+ tree to make sure that all the leaves are on the same level of the tree, each node must always be half full of keys, and the height of the tree is always at most ceiling (log (n)/log (k/2)) where n is the number of values in the tree and k is the maximum number of keys in each block. That means if more keys are present in the node less number of pointer traversal is required to search the desired value. This is crucial in a database because the B+ tree is on disk. Reading a single block takes just as much time as reading a partial block, and a block can hold a large number of keys/pointers.

In the leaves in B+ tree contain next sibling pointer which facilitate for faster iteration through contiguous block of values. This is extremely useful when dealing with range queries where the search for value can be done by traversing the sibling values until the desired value is found. It's especially efficient if the implementation guarantees that leaf blocks are contiguous on disk. Insertions in B+ trees need to deal with overflow and underflows by splitting the nodes if required. Deletions in B+ trees are efficient and mirror reversal of insertion algorithm. To avoid underflow after deletion it merges the nodes. Additional feature of B+ tree is if data is stored in sequential file, sorted by a key, then one can very efficiently bulk load the sequential file into a B+ tree. For disk based sorting B+ tree can be used as a sorting algorithm. [8]

IV. Advantages of B+ Trees over Other Indexing Structures

A. *B trees*

1. In B+ tree data is ordered in sequential linked list and the data is stored in leaf nodes but in B tree the leaf nodes cannot be stored in linked list due to lack of the sibling pointers.
2. In B+ tree the data is stored in leaf node so searching of any data requires scanning only of leaf node alone, whereas in B tree data is stored in non-leaf nodes as well and to search any data value the complete tree needs to be traversed.
3. Insertion in B+ tree is simple compared to B tree.
4. Deletion is tedious in B tree as compared to B+ tree, as data is present in internal nodes as well so.
5. As the data is not there in non-leaf node so the fan out is higher in B+ tree, tree depth is kept shorter and thus there are less reads done from secondary storage. [11]

B. *Sequential Indexing*

B+ tree is preferred over sequential indexing as it saves lot of time and memory. Finding a data requires to traverse the index file of sequential index. It will take more time for large files.

C. *Hash Table*

B+ tree are memory efficient. Hash table too are memory efficient but in some case it all depends on the hash function that is used to store the data. But in overall B+ tree perform fairly uniform. The basic operations of insertion, deletion and lookup takes O (log n) time. B+ trees do not exhibit any worst case behavior. On an average hash table takes O (1) to access, insertion, deletion and look up. [11]

V. Commercial RDBMS use B+ tree

To support fast random access or range searches on column values database systems allow creating secondary indexes (B+-trees [14]) on tables. A secondary index row comprises of indexed column values and row identifier for the corresponding row in the base table. For a conventional table, where data is typically organized as a heap, the location of a table row does not change during a table operation such as insert, delete, or update. For such cases, secondary indexes with physical row identifiers are ideally suited, where the row identifier holds physical location of the table row. Most commercial database systems, including IBM DB2 [15], Sybase Adaptive Server [18], Microsoft SQL Server [16], and Oracle [17] support such physical secondary indexes for their conventional tables. [13]
SQL Server uses B+ tree format to store its indexes. There are a few exceptions – which are not indexed at all are temporary hash indexes, created during a hash join operation, or column store indexes. However, all key-based clustered and non-clustered persisted SQL Server indexes are organized and stored as B+ Trees. SQL Server terms each node in tree to be a page. Two page types are found in each index. The first type is the data page which is type 1 page. Every leaf level node in SQL Server B+ tree index has a one data page. The type 2 pages are the intermediate index pages thus, every non-leaf node is type 2 page which contain data pages. But there is pointer to each row that identifies next child page, it can be either page type 1 or 2 depending where in the B+ tree it is located. [12]

VI. Conclusion

The conclusion would be that B+ trees are better for the secondary indexing purpose as it is memory efficient and save lots of money as there are less trips done to access data from disk reason being the hit rate is good. There is no redundancy of data in the B+ tree. Most of the commercial RDBMS like SQL, MySQL, Oracle8, and Microsoft require high performance, high reliability, portability and scalability that can be achieved through B+ tree.

References

[1] http://www.tutorialspoint.com/sql/sql-rdbms-concepts.htm

[2] http://www.slideshare.net/rdbms/rdbms-1405766?qid=46d9d90e-8413-4334-a0c0-90cc9174c848&v=qf1&b=&from_search=10

[3] http://www.tutorialspoint.com/dbms/dbms_indexing.htm

[4] http://www.tutorialspoint.com/cobol/cobol_file_organization.htm

[5] Parth Patel, Deepak Garg "Comparison of Advance Tree Data Structures" International Journal of Computer Applications (0975 – 8887) Volume 41– No.2, March 2012.

[6] http://www.dba-oracle.com/t_garmany_easysql_btree_index.htm

[7] https://msdn.microsoft.com/en-us/library/ms177443.aspx

[8] http://dblab.cs.toronto.edu/courses/443/2014/05.btree-index.html

[9] http://www.quora.com/What-is-a-B+-Tree

[10] http://stackoverflow.com/questions/870218/b-trees-b-trees-difference

[11] http://cs.stackexchange.com/questions/270/hash-tables-versus-binary-trees

[12] http://sqlity.net/en/2445/b-plus-tree/

[13] Eugene Inseok Chong, *Souripriya Das, Chuck Freiwald, Jagannathan Srinivasan, Aravind Yalamanchi*, *Mahesh Jagan*nath, Anh-Tuan Tran, Ramkumar Krishnan "B+-Tree Indexes with Hybrid Row Identifiers in Oracle8i" 2001 IEEE

[14] Comer, D. The Ubiquitous B-Tree. Computing Surveys, 11(2), pp 121-137, June 1979.

[15] The SQL Reference. DB2 Universal Database Version 5.2 Publication.

[16] Microsoft SQL Server, SQL Server 7.0 Storage Engine, White Paper, October 19980.

[17] Oracle8i Concepts, Release 8.1.5: Oracle Corporation, Part Number A67781-01, February 1999.

[18] Sybase SQL Server, Transact-SQL. (Jser's Guide, Document ID:32300-01-1100-02, December 1995.

[19] Hung-Yi Lin, "A Compact Index Structure with High Data Retrieval Efficiency", International Conference on Service Systems and Service Management, pp. 1–5, 2008.

[20] Knuth, Donald, "Sorting and Searching, The Art of Computer Programming", Addison-Wesley, ISBN 0-201-89685-0 , Vol. 3 (Second ed.), Section 6.2.4, Multiway Trees, pp. 481–491, 1998.

[21] Mark Russinovich, "Inside Win2K NTFS, Part 1", Microsoft Developer Network, Retrieved 2008-04-18.