B+ Tree and Implementation in Filesystem

Satria Ady Pradana (13510030)
Informatic Engineering
School of Electrical Engineering and Informatics
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
Satria.ady@students.itb.ac.id

***Abstract*—In computer science there are needs to organize data stored in storage media efficiently. Several approaches has been introduced, one approach is using B+ tree data structure. This paper will cover the B+ tree data structure and its implementation filesystem and data organization.**

***Index Terms*—B+ tree, tree, key, filesystem, data organization.**

1. INTRODUCTION

Although filesystem has been created in early stage of computer development, many improvement and change has been done. Therefore, the development of the filesystem is still go on even until this time.

People has implement many approach to create a better and efficient filesystem. One of technology behind nowaday is B+ tree. As the natural behavior of B+ tree, it has been chosen and proved as a suitable approach to implement filesystem. Some filesystem examples that implement B+ tree are NTFS, ReiserFS, NSS, XFS etc.

1.1. B+ TREE

1.1.1. B+ TREE HISTORY

B+ tree is a rooted tree which is a variant of B tree. B Tree itself was first described in the paper Organization and Maintenance of Large Ordered Indices. Acta Informatica 1: 173-189 (1972) by Rudolf Bayer and Edward M. McCreight. There is no single paper introducing the B+ tree concept.

1.1.2. B+ TREE DEFINITION

B+ tree is a balanced m-ary search tree. Furthermore, a B+ tree or B plus tree is a type of tree which represents sorted data in a way that allows for efficient insertion, retrieval and removal of records, each of which is identified by a key. It is a dynamic, multilevel index, with maximum and minimum bounds on the number of keys in each segment or vertex or node.

B+ tree guarantee that the tree structure will be balanced. All leaves are always height balanced and located at same level. There will not be leaves longer than others. This also guarantee that processing time taken will be equal.

B+ tree also a search tree. The tree order the key in a way that it allows a fast searching. The subtree which has less value key are located left. While the greatest value key is always placed at the most right place.

1.2. FILESYSTEM

A filesystem is means to organize data by providing procedure to store, retrieve and update data, as well as manage the available space on the device(s) which contain it.

Filesystem play a great role as it provides mechanism to control access to data. Ensuring realibility is a major responsibility of a filesystem.

File system are used on the data storage devices such as hard disk drives, floppy disk, optical disk, or flash memory storage devices to maintain the physical location of computer files.

2. B+ TREE STRUCTURE

2.1. NODES STRUCTURE

The B+ tree consist of two type of nodes: leaves and internal nodes (non-leaf nodes). However, the data, or sometimes referred as records, are always stored in the leaves. All leaves are then linked sequentially.

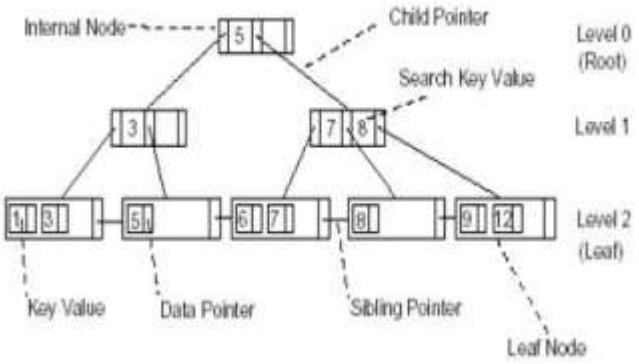

Figure 1: Node structure

Every node in B+ tree can have several key and children variables. The number of children nodes is not bounded to a fix number or fix size like in binary tree. Instead, B+ tree can dynamically expand its branch while sort the key at the same time.

Every key links to a child node. This node also has keys with value less than its parent's key. Both internal nodes and leaves have key and both of them are used as index to data. Although both seems similiar, but the keys in both node types are different in function. All the keys in internal nodes are served as dummies. They are used only as indexes and guide to the real data in the leaves (pointing to where the data are) while the key in leaves may duplicate as key in internal nodes. Those keys are the keys that point directly to data.

The searching for a value in the B+ tree always starts at the root node and moves downward until it reaches a leaf node.

The order, or branching factor b of a B+ tree, measures the capacity of nodes (i.e. number of children nodes) for internal nodes. The actual number of children for a node, denoted as m, is constrained such $\lceil b/2 \rceil < m < b$. The root is an exception internal node, it is allowed to have as few as two children. The leaves have no children, but are constrained so that the number of keys must be at least $\lfloor b/2 \rfloor$and at most b-1.

When B+ tree is in a condition nearly empty, it only contain one node, which is a leaf node. The single leaf is also referred as a root in this case. In this condition, this node is permitted to have as little as one key if necessary and at most b.

For instance, if the order of a B+ tree is 9, each internal node (except the root) may have children between 5 and 9 while the root may have between 2 and 9.

Another example can be seen in figure below:

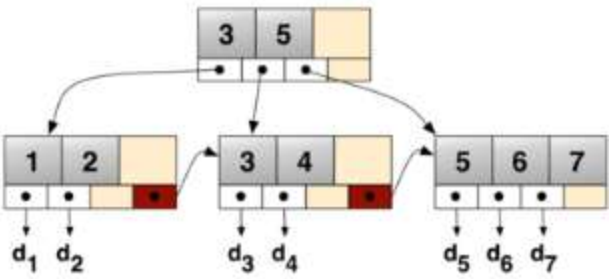

Figure 2: Simple B+ tree example.

The figure above explain another example of B+ tree where the 7 keys (denoted by number 1 to 7) link into 7 data values (denoted by d1 to d7). In this case, there is only root and three leaves. The root is bounded by order 3 or b = 3. This leads that the node can hold at most 3 children at the same time. As shown in the figure, root has 2 key which are 3 and 5. This key will guide to the actual data. If the key sought is less than 3 than it will be guided to first child. If that condition is not satisfied (key sought is greater than or equal 3) the root will traverse to next key which is 5. If the sought key is less than 5 (and greater than or equal 3) the searching will be guided to second node. Last if the key sought is 5 or beyond, the pointer will be redirect to third child. The linked list (red) is used for a rapid in-order traversal.

If we recall the constraint of branching factor and see the previous figure, we could get conclusion that each internal nodes must be filled half fully at minimum. So far, the insertion and deletion algorithms must guarantee that each node in the tree will be at least half full.

2.2. PHYSICAL REPRESENTATION

As described in section 2.1, the tree know two types of node: internal nodes and leaves. This section will furthermore cover the physical representation of each node type.

2.2.1. INTERNAL NODES

An internal node in B+ tree consists of a set of key values and pointers. The set of keys and values are ordered so that a pointer is followed by a key value. The last key value is followed by one pointer.

Each pointer points to nodes containing values that are less than the key. The last pointer in an internal node is called the infinity pointer. It points to a node containing key values that are greater than the last key value in the node.

When an internal nodes is searched for a key value, the search begins at the leftmost key value and moves rightwards along the keys.

2.2.2. LEAF NODES

A leaf node in B+ tree consists of a set of key values and data pointers. The set of key values and data pointers are ordered by the key values. The data pointer points to a record or block or the actual data identified by the key value.

Searching a leaf node for a key value begins at the leftmost value and moves rightwards until a matching key is found or else the search will return with no data.

2.3. B+ TREE CHARACTERISTICS

The depth or height of B+tree can be computed by following formula:

$$h = \log_b(nb - n + 1) - 1$$

where h is the height, b is the branching factor, and n is the total nodes in tree (including root and leaves).

The formula above can be proven as below:

1. As the node has branching factor b, the number of children will be at most b.
2. In the k-th level, the number of node would be b^k
3. The total number of node of h-height would be $n = \sum_{k=0}^{h} b^k = \frac{b^{h+1}-1}{b-1}$ (h is counted from 0)
4. By simple algebra we know $b^{h+1} = n(b-1) + 1$
5. Using logarithm we get $h = \log_b(n(b-1) + 1) - 1$ which is similiar to ordinary tree formula.

In spite the height formula is similiar with other m-ary tree, the B+ tree branches (factor b) can be larger. Thus,

the base of logarithm can be large and the height can be as minimum.as possible. This is one of B+ tree benefit, as the data grow large, the height can be kept as minimum as possible.

Other characteristics can be founded in B+ tree. For a *b*-order B+ tree with h-levels of index, the tree have following properties (these properties would not be cover furthermore):

- The maximum number of records stored is $n_{max} = b^h - b^{h-1}$
- The minimum number of records stored is $n_{min} = 2\left\lceil\frac{b}{2}\right\rceil^{h-1}$
- The minimum number of keys is $n_{kmin} = 2\left\lceil\frac{b}{2}\right\rceil^{h-1} - 1$
- Space required to store the tree is *O*(n)
- Inserting a record requires $O(\log_b n)$ operations.
- Finding a record requires $O(\log_b n)$ operations.
- Removing a record requires $O(\log_b n)$ operations.

3. FILESYSTEM AND B+ TREE IMPLEMENTATION

3.1. DATA REPRESENTATION ON THE DISK

A file and a directory is a collection of information that connected and stored in storage media. Generally, operation on the file can be one of creating, writing, reading, removing, searching, opening, and closing.

Storage media (often referred to disk based media) physically is a disk. To store a data on disk, file system will divided disk into regions.

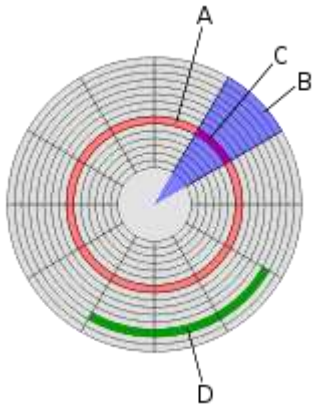

Figure 3: A disk structure.
(A) Track
(B) Geometrical sector
(C) Track sector
(D) Cluster

A track (as marked by A and red color) is a circular path on the surface of the disk which information is magnetically recorded.

A sector is a subdivision of a track. Each sector stored a fixed amount of user data. In the term of mathematic, the word sector means a portion of a disk between a center, two radii and a corresponding arc (see figure 3, item B) which is shaped like a slice of pie. Thus the disk sector (Figure 3, item C) refers to the intersection of a track and mathematical sector.

A cluster or allocation unit is the unit of disk space allocation for files and directories. To reduce the overhead of managing on-disk data structures, the file system does not allocate individual disk sector, but contiguous group of sector, called cluster. There on a disk that used 512-byte sectors, a 512-byte cluster contains one sector and a 4-kibibyte (KiB) cluster contains eight sectors.

A cluster is the smallest logical amount of the disk space that can be allocated to hold a file. Storing a small files on files on a filesystem with large clusters will waste disk space and must be avoided.

A cluster size may vary in several file system but typical cluster size are ranged from 1 sector (512 B) to 128 sectors (64 KiB) in common.

A cluster need not to be physically contiguous on the disk; it may span more than one track or if sector interleaving is used, may even be discontiguous within a track. This condition is still legal as the sectors are still logically contiguous.

A whole data in computer can be represented as block in the disk. A block is a combination of some cluster needed to store data (it can be actual data or a pointer, later in section 3.3). The data can be placed in one block or accross many blocks.

3.2. FILESYSTEM ORGANIZATION

A filesystem is a system that managing the operation in the disk or storage media. It is responsible for organizing files and directories, and keeping track of which area of the media belong to which file and which are not being used. It will provide access to certain block and retrieve information in those block. The flow of access occured in a process can be as shown in below:

1. Application programs
2. Logical file system
3. File – organization module
4. Basic file system
5. I/O control
6. devicess

Thus are the cycle that a filesystem must maintain. The application program is not allowed to directly access the disk, but using filesystem.

3.3. INDEXING PROBLEM AND B+ TREE IMPLEMENTATION

The development of storage media has growth very fast in term of speed and capacity. But still the issue is still the same. The data located on the disk are distributed on the many side of the disk. Data allocated on the disk is placed

in some blocks.

The computer can only transfer one block a time between main memory and the disc. This means that to retrieve one record a whole block of data also be retrieved. Therefore, it is important to know in which block a record is stored and to retrieve the minimum number of block when looking for a record.

The problem filesystem must handle is deciding in which block each record should be placed and what information should be stored in addition to the record to allow the record to be retrieved easily.

Therefore, to allocate data and addressing a correct block containing data, an access mechanism is needed. The mechanism is known as indexes and the scheme is known as Indexed Allocation.

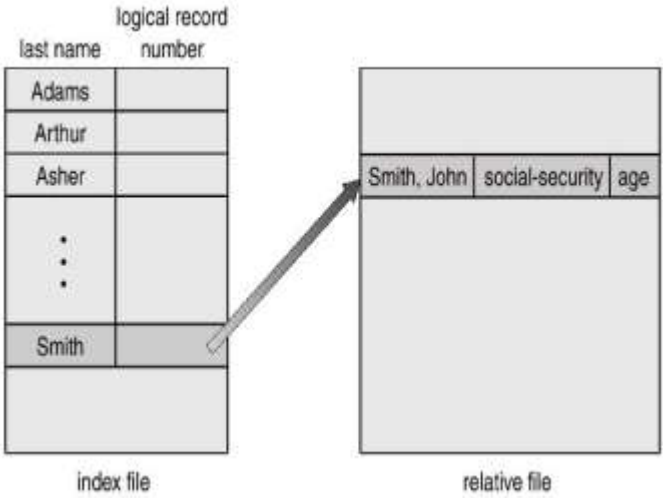

Figure 4: Simple indexing

The mechanism of indexing is quite simple. The actual data location are pointed with some certain pointers. Those pointers then will be combined in a block name index block. Every file have own index block. A directory contain address of index block.

Thus, the scheme above are quite similiar with the properties of B+ tree. The data blocks can be considered as actual data or record. It is then located in the leaves strcture and then pointed by approriated pointer in upper nodes. In figure 5 example, the data block are the relative file while the pointer that point the data is denoted by logical record number. The key to identify the data is the last name.

The problem of indexing is in the size of blocks. If index block is small, the block would not enough to hold a big size file. While if the block is too large, there will be waste as described in section 3.1.

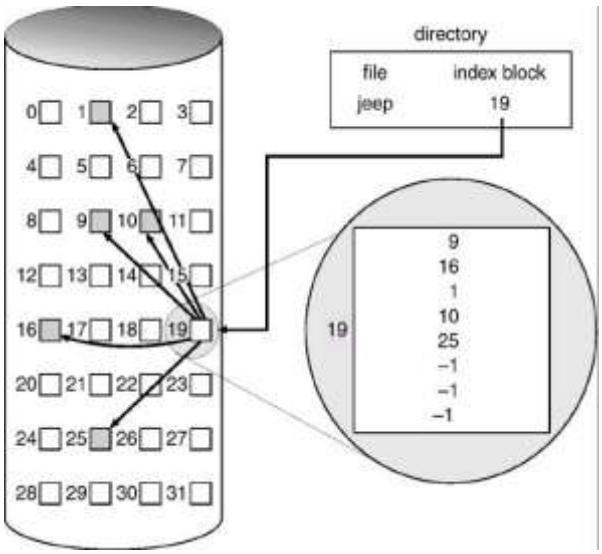

Figure 5: Indexed allocation scheme.

In above figure, the block containing whole data are 1,9,10,16, and 25. But the actual data are arranged in order 9,16,1,10, and 25. Those each data block are sliced or piece of whole data. Each data then pointed by a pointer and placed in a single index block (block 19).

Mechanism used in indexed allocation cover: linked scheme, multilevel index, combined scheme. This section will also cover implementation of B+ tree in this scheme.

Linked scheme mechanism can link several index block. A large file containing large data. A single block would not be enough to accomodate that file, so do the index block. As the data distributed across many blocks, pointer are needed to make sure the integrity of data. If pointer is not enough to be hold in whole index block, the last pointer of this block will point to another index block. That index block will hold a next pointer. Thus the integrity of data will be a file will be kept. But, if pointer only need one index block, the last pointer will be null.

Thus, the scheme can be covered by using B+ tree. The pieces of data pointer can be hold in the a single leave node. If the data is large enough, the pointer to next leave node is provided to give immediate access to next piece data. Therefore, a performance of accessing data can be high and process will take less operation.

In multilevel index, index block in first level will point to second level index blocks that will point to the address of data. The mechanism can be repeated until third level or fourth level depend on the needed amount of data.

Thus the multilevel index is a direct implementation of B+ tree as the index can be placed in various level of tree.

Combined scheme will combine direct block and indirect block. A direct block defined as a block containing a pointer that point the data directly while the indirect block will point to another block. An indirect block has three pointers: first pointer point to a single indirect block. Second pointer points to double indirect block. Third pointer points to triple indirect block.

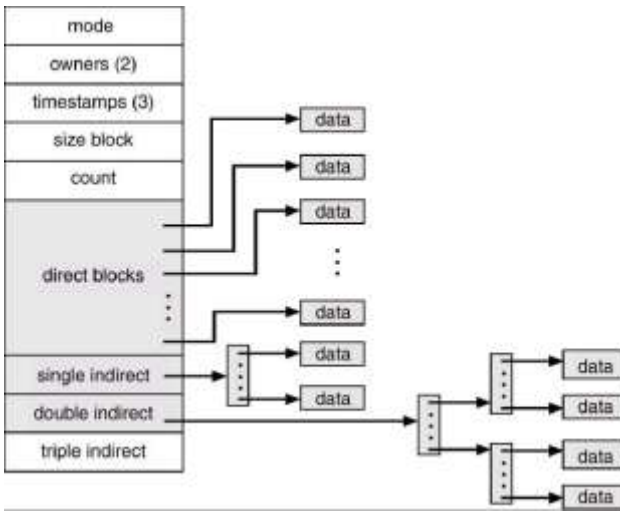

Figure 6: Direct and Indirect Indexes

The figure 6 shows the basic of multilevel indexing and each indirect can be implemented as a regular B+ tree. Each index block branching and point to another index block. Thus the cycle must be kept as minimal.

The performance of indexing will be depended on index structure, file size, and position of block.

4. CONCLUSION

B+ tree can be implemented to address problem in filesystem. Using approriate approach, the data can be managed and organized well.

5. REFERENCE

Munir, Rinaldi. "Matematika Diskrit". 2010. Penerbit Informatika.

Siang, Jong Jek. "Matematika Diskrit dan Aplikasinya pada Ilmu Komputer". 2009. Penerbit ANDI.

Rosen, Kenneth H. "Discrete Mathematics and Its Applications Fifth Edition". 2003. McGraw-Hill.

WikipediaFoundation, Inc. http://en.wikipedia.org/. (access time: November 29th, 2011).

Chandra, Ian. "PC DOS versi 3.10". 1988. Gramedia.

6. DECLARATION

With this I, Satria Ady Pradana, declare that my paper is originally written by me, not a transliteration, translation, nor plagiarism of another paper.

Bandung, 12 Desember 2011

signed

Satria Ady Pradana (13510030)