Performance Management Information System Development

Dhruv Seth
Senior Analyst
Wells Fargo India

Vidisha Raj
Assistant Systems Engineer
Tata Consultancy Systems

ABSTRACT
An organization namely a government, corporate house or even a school nowadays relay on information collected for a huge amount of data. Storing the data, handling it to get the relevant record and presenting it gives the organizing body an aid to make a decision. Similarly, a management information system (MIS)[2] is used in corporate houses and other similar institutes to provide control of all the businesses involving people, documents, technologies, and other procedures used by management accountants to solve problems such as costing a product, service or a business-wide strategy. Our project was to develop a similar system which could store the data in a flat file and fetch it in a lesser time and display it on a UI for the user. We developed the static library in C language in Linux environment and used GTK+2.0 library[1] to build the User Interface to present the records.

Keywords
MIS[4], Management Information System, Indexing, Development, C-ISAM[5].

1. INTRODUCTION
An 'MIS' is a planned system of the collecting, processing, storing and disseminating data in the form of information needed to carry out the functions of management. According to Philip Kotler "A marketing information system consists of people, equipment, and procedures to gather, sort, analyze, evaluate, and distribute needed, timely, and accurate information to marketing decision makers." The terms MIS and information system are often confused. Information systems include systems that are not intended for decision making. The area of study called MIS is sometimes referred to, in a restrictive sense, as information technology management. That area of study should not be confused with computer science. IT service management is a practitioner-focused discipline. MIS has also some differences with Enterprise Resource Planning (ERP) as ERP incorporates elements that are not necessarily focused on decision support.

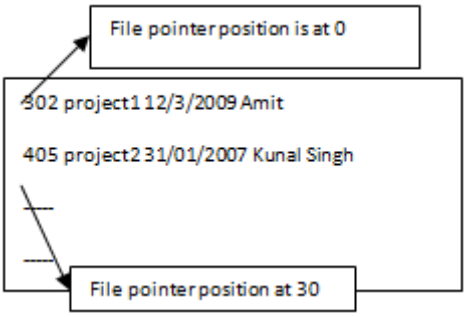
2. PROPOSED METHOD
The project implements the modified version of the b-tree for maintaining a database management system, which includes all the facilities of a basic database system. The concept used to decide the line of action was C-ISAM library. C-ISAM stands for Indexed sequential access method, it is a library used to maintain a database. However, the library was not available as it was a propriety library of IBM Informix. So, we worked out the solution, which used the concept of C-ISAM, using a modified Bayer's Tree. The project includes a backend implementation of the modified BTREE version, which handles the database storage strategies. The frontend of the projects handles the basic GUIs and the services which the application provides. The project focuses on development of MIS. This maintains the database of few tables regarding the Sales and Delivery in a Company. The fact is that a file is stored as a sequential data of records. For example, a table of Project has three fields for example, Project number, name, date, project engineer, would be like:

Figure 1. Showing a sample record

Serial No.	Name	Date	Employee Name	…
302	Project 1	12/3/2009	Amit	…
405	Project 2	31/01/2007	Kunal Singh	…

Storing the record in a form of "struct" data type, for each table, would be a solution for the problem of the mode of storage. But, the records stored in the file, would not return a reliable file pointer. Therefore, we had to store the database in the simple style of storing in a sequential format as shown in the Figure 2.

Figure 2. A way to store data in a file is shown.



The records can be identified uniquely by the primary key of the table. Hence, using the feature of the key, one can search any record in a database. Now, the problem was of storing the primary key with the record's position is the only chance of getting any desired record. Hence, we stored the file position against the primary key of each record. This is also called indexing of each record.

2.1 Implementation of Indexing
As the data is being stored in the file, the starting location of each record, in the sequential file, with the primary key is stored in a file named "master index". Now, the indexes of the records are stored in a file. The question arises that in which way one should implement this search of primary key, to access the record's location in the database file. Other than that, searching of an index in the Master Index file would result into a larger time complexity, if the records are larger in number. The contents for a master index file are shown in Figure 3.

Figure 3. The master index file is shown.

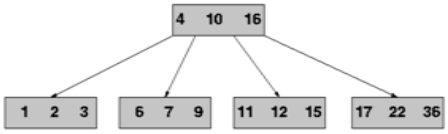
FILE POINTER	PRIMARY KEY
0	304
87	945
120	1023
1345	99

2.2 Proposed Solution Strategy

To solve the above stated problem, one can store the indexes in a form of data structure. This implementation is being done using the concept of BTREE (Bayer's Tree), which stores the primary keys in a form of tree, which have a root and leaves. The original BTREE is being modified in the project, so as to make it customized to our requirement. The concept of BTREE is to store the data values in a sorted order and search the key as per the value of the current node with the search key.

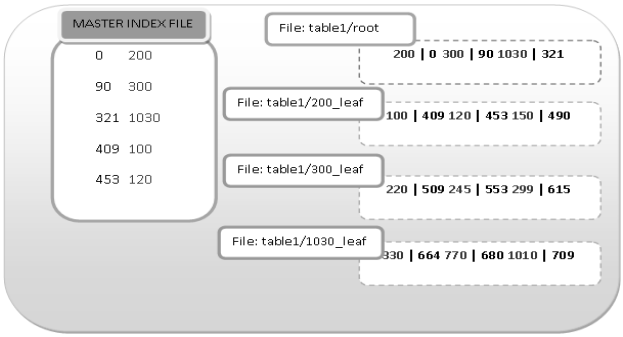
For example, represented in the Figure 4.

Figure 4. Showing a B-TREE.



The regular BTREE has the factors of minimizing the search for a leaf node by a factor know as t, which states that the maximum nodes a leaf can have is 2t-1. In the modified BTREE, implemented, the maximum keys stored in a leaf is set constant as 3, as the number of records being stored is assumed to be less. Further, the searching for the required key in the BTREE is done using simple binary search, as the keys are stored in ascending (or already sorted) order. This method is adopted because when the amount of records exceeds a certain level, say 10MB, then the usage of RAM would increase and the system might consume more and more memory on increase of records. Instead the storage of each leaf is been on the disk and whenever the leaf is required the corresponding leaf file is accessed for that purpose. This way the usage of RAM decreases, though it decreases a factor for I/O performance. This method is devised keeping in mind the number of records in the database to be above a million. Below figure (Figure no. 5) shows an implementation of B-tree.

Figure 5. Implementation of the B-TREE



2.3 Implementation Details

The implementation of the problem was done in a simple fashion. The steps involved are:

For storing the nodes, in a format of BTREE

1. START

2. The user enters the record, the primary key is stored in the MASTER INDEX FILE, along with the file location.

3. As the record is saved, the MASTER INDEX FILE is read and the modified BTREE is formed as:

a. The first record is stored in the root file, as seen in the figure below.

b. As the records are saved, the root file is filled, along with their file pointer positions, in an ascending order.

c. After the root is completely filled, the records are filled following the order of leaves i.e in ascending order.

d. The naming of the leaves are named as table/first node_leaf, which states that it contains the nodes having value less than primary key of the first node of the root.

e. Similar is with every leaf.

f. STOP

For searching of the nodes, in the BTREE:

1. START

2. The BTREE is stored in an ascending order, the value to be searched is first picked, for matching, in the root node.

3. The root leaf is searched for the match, in a binary search mode.

4. If the value is matched, by a node in the leaf, the result is returned with the corresponding file pointer.

5. If the value is not found in the current leaf, the related position of the next leaf to be searched is returned to the recursive function.

6. The function returns if the value is not found, when the function reaches the last leaf without a match.

7. STOP

The root of the BTREE is been stored as a file namely "table1/root", the keys having lesser value than the key 200 in the root is been stored as "table1/200_leaf", and so on.

The deletion process:

1. START

2. The user gives the primary key of the record to be deleted.

3. The root leaf is being searched for the match.

4. If the value is not matched, then user is given the notification.

5. If the value is found in the BTREE, the value of the file position and the primary key is set to -1 in the MASTER INDEX FILE.

6. After that, the current BTREE of the record is deleted. A new BTREE is formed with the modified MASTER INDEX FILE.

7. Also, the database file is modified with an invalid value for the set of record, to be deleted.

8. STOP

The modification process:

1. START

2. The user gives the primary key of the record to be modified.

3. The primary key value is matched in the current BTREE.

4. If the value is not found, the user is given notification of the result.

5. If the value is found, the user is given a dialog box, which contains the current values of the record. Then the user, can alter or change the record details and save the new record.

6. As the primary key has a default file pointer stored against it in the MASTER INDEX FILE. The value is used to alter the record in the database file.

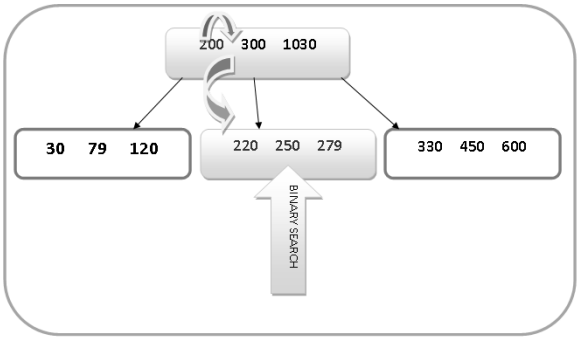
7. STOP

SEARCHING IN A BTREE (example as shown in Figure 6)

Command: Search 250, in the BTREE.

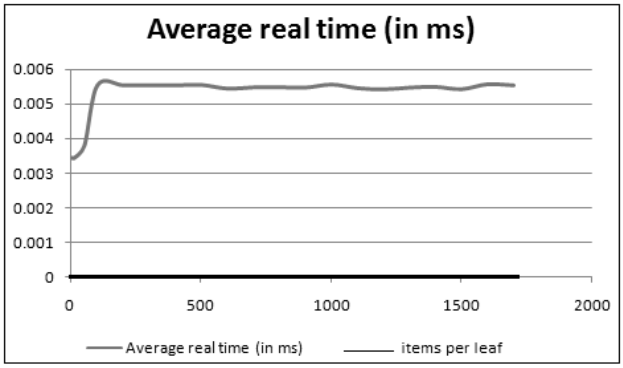
The above mechanism will render the complexity of searching a key from a large database system to mere O(log n). This method is adopted because when the amount of records exceeds a certain level, say 10MB, then the usage of RAM would increase and the system might consume more and more memory on increase of records. Instead the storage of each leaf is been on the disk and whenever the leaf is required the corresponding leaf file is accessed for that purpose.

Figure 6. Searching mechanism



This way the usage of RAM decreases, though it decreases a factor for I/O performance. This method is devised keeping in mind the number of records in the database to be above a million. The above problem has the time complexity of log(n). As the BTREE has an implementation in the hard disk rather than RAM. This states that, if the numbers of records are larger the size of memory used by the application will be less. But the I/O operations will render the system to slow; this problem can be dealt with the help of the BTREE itself, by increasing the number of nodes to be stored in each leaf, which is set by DEFAULT as 3 per leaf. This gives flexibility to the problem according to the number of records stored in a table. Below is the graph which shows the average real time (in ms) vs number of items per leaf taken after implementing the algorithm to search an item on a Pentium 3 (1.65 Ghz) system having Ubuntu 8.10 (linux kernel 2.26) with varied items per leaf set from a range [3-1700] items per leaf.

Figure 7. Graph generated after implementation of the algorithm to search an item.



3. ACKNOWLEDGMENTS

Our special thanks to Mr. K. K Majumdar, General Manager, HCL Info services Ltd. and Ratika Pradhan, Faculty member from Sikkim Manipal Institute of Technology for reviewing the project.

4. REFERENCES

[1] Beginning Linux Programming by Neil Mathew and Richard Stones. Publisher Wrox Publishing inc. 4th Edition.

[2] "Development strategy of MIS for small and medium-sized enterprise"- Guo-Shun Lin Dong Xiang Qing Chang Shipping Manage. Coll., Dalian Maritime University.

[3] "Development of a comprehensive management information system for a large R&D laboratory"- Panella, R.F.; Jendrian, F.H.; ASTARS Program Branch, Wright Lab., Wright-Patterson AFB, OH.

[4] "Evolutionary database design and development in very large scale MIS"- M. C. Filteau, S. K. Kassicieh and R. S. Tripp.

[5] IBM Informix C-ISAM Version 7.x for UNIX and Linux, official e-book by IBM Informix. (ISBN - GC27-1491-00)

[6] "Systems Development in Information Systems Research" – Jay F. Nunamaker, JR., Minder Chen, and Titus D.M. Purdin.