CMPUT 291 - Project 2

Berkeley DB & File Systems

Authors: Victor Frunza & Ian Oltuszyk
Instructor: Li-Yan Yuan
TA: Kriti Khare
University of Alberta
Due Date: April 7th, 2016

Victor Frunza & Ian Oltuszyk
CMPUT 291 - Project 2
University of Alberta

IMPLEMENTATION

Input was provided by storing four random keys into a char array prior to database population. Using this array in conjunction with a variable that increments and resets to 0 at a threshold of 4, we were able to reference them (in sequence) repeatedly between different searches.

Key Search

The key search was done by performing a single query for both binary tree and hash type databases.
Hash is expected to return with it's data in the shortest amount of time.

Data Search

The data search is done by setting the cursor to the first position and then iterating it traverses the whole database while finding matches. For the indexfile type, an associated hash database with reversed keys and data entries implements the key search mechanism to retrieve values quicker.
Neither database type is expected to do well.

Range Search

The range search is done differs for hash and binary tree types; btree uses the DB_SET_RANGE parameter, while hash iterates over the entire database. Indexfile uses the same mechanism as btree to search by range from the primary database.

IndexFile

Our indexfile was a hash database that was associated with a btree database. The indexfile featured reversed key/data pairs which allowed for a large increase in the execution time for data searches. This file also allowed the use of the primary database when searching for keys and searching over ranges, as it got the benefits of the btree file system where similar entries reside beside each other in memory.

Results

Key Search

	Trial 1	Trial 2	Trial 3	Trial 4	Avg
Btree	60	52	56	29	49
Hash	24	55	53	21	38
Indexfile	56	19	59	62	49

Note: All data is in microseconds.

Victor Frunza & Ian Oltuszyk
CMPUT 291 - Project 2
University of Alberta

Data Search

	Trial 1	Trial 2	Trial 3	Trial 4	Avg
BTree	64742	62814	64216	62163	63484
Hash	77550	78489	75224	76407	76918
Indexfile	100	67	62	112	85

Range Search

	Trial 1	Trial 2	Trial 3	Trial 4	Avg
BTree	7172	7340	8086	8556	7789
Hash	124560	150472	136110	133846	136247
Indexfile	7103	7513	6089	14090	8699

Analysis

Key Search

The indexfile method uses the primary database for key searches so it's average result, 49 µs is justifiably identical to btree. Hash is faster than both btree and indexfile, averaging 38 µs. This is due to it's much faster retrieval of keys as positions inside of an array, rather than having to constantly determine the key's position relative to a node in a binary tree.

Data Search

Indexfile is considerably faster with it's inverted key/value entries in a hash table, and it computes this task much more efficiently averaging 85 µs. Indexfile is the superior mechanism because it does not have to iterate over the whole database, unlike hash and btree; it implements the key search on secondary database's keys (representing data).

Range Search

Indexfile uses the same mechanism btree uses to search, thus giving it similar results. Btree is most successful because it's iteration is limited, and not over the entire database. The lower limit is searched for, then the database iterates until hitting the upper limit, preventing the program from checking all 100,000 key/value pairs. The Hash algorithm has to iterate over the whole database file in order to find the similar keys which are distributed randomly throughout the whole file.

Note: All data is in microseconds.