Advances in Computer Science Research, volume 62
2nd Joint International Information Technology, Mechanical and Electronic Engineering Conference (JIMEC 2017)

The Comparison Between T*_Tree And B+_Tree On Modern Hardware Revisited

Ahmad Abdullah*[1], Qingfeng Zhuge*†[2]

*College of Computer Science, Chongqing University, Chongqing 400044, China
†Key Laboratory of Dependable Service Computing in Cyber Physical Society, Ministry of Education, Chongqing 400044, China
[1] Ahmad_abdullah1@hotmail.com, [2] qfzhuge@gmail.com

Keywords: index structure, main memory databases, T*-Tree, B+Tree

Abstract

Database indexes have a significant impact to speed up access to particular item or items. While B+Tree is the most important index structure for disk-based database management systems (DBMS), T*-Tree is optimized for main memory access and widely used for in memory database system (IMDB). however, the prior researches made performance evaluation between T*-Tree and B+Tree on hardware with different memory hierarchy characteristics and single core CPU due to some of these studies published nearly two decades ago. In this paper, we implemented the state of the art of T*-Tree and B+Tree for main memory use. The two indexes structure were tested on modern hardware. Unlike the previous experimental results, our test performed shows that B+Tree is faster than T*-Tree in the most of the test cases.

1 Introduction

recently, IMDB was raised because Random access memory becomes more density and cheaper. Nowadays it becomes common for home PC to have 8 gigabytes or more of random access memory.

Because of the system contains big memory, it becomes feasible to store and manage database within main memory. It is quite likely that databases, at least for some applications where real time access and high data performance are required, will eventually fit entirely in main memory [6,7].

that is causing a growing number of IMDB researchers became interested in it architecture, data organization, and access methods.

This paper focuses on the use of the index structure to access data which reside in main memory.

In the end of 80's, Lehman and Carey evolved the T-tree from AVL-Tree and B-Tree as an index structure for main memory database [6]. in that time, experiments indicate that T-Tree outperform B+-Tree in main memory [3] proposed T*-Tree, which is an improvement from T-tree for better use of query operations, including range queries and which contains all other good features of T-tree.

Because of its good overall performance, the T-tree and its variant have been widely accepted as a promising index structure for main memory databases.

for example, Oracle TimesTen [4] uses T-tree indexes, which are optimized for main memory access [5].

A study from decades ago have used hardware with different memory hierarchy characteristics and CPUs (no multi-core back then, let alone inhomogeneous ones) that had a hard time to saturate the memory interface.

Today, T-Tree and its variants are bound to suffer horribly because of their poor locality, both in the sense of expected block/page transfer counts and in the sense of cache locality.

Compare this to the excellent access locality of B-Trees in general and B+Tree in particular (not to mention cache-oblivious and cache-conscious versions that were designed explicitly with memory performance characteristics in mind) [1,2].

according to our experiments, it seems that T-Tree and its variants have nothing to offer in the way of performance, given that the times of commodity hardware with a single-level memory 'hierarchy' have been gone for decades.

In this paper, we reimplemented T*-Tree the most advanced variant of T-Tree and B+-Tree as they described from them authors. then, extensive experiments are conducted on modern hardware. The experimental results show that the B-Tree index provides better performance than T*-Tree on modern hardware.

2 T*-Tree index structure

In this section, we first briefly describe the structure of the T*-Tree [3], complete T*-Tree shown in figure 2.

The T*-tree is a binary tree with many items in a node. By this design, T*-Tree keeps the intrinsic binary search nature which is inherited from the AVL-Tree, and has the good update and storage utilities of the B-Tree by having many items in a node.

T*-Tree enhanced the structure of the T-Tree by adding an additional pointer, called a successor pointer which points directly to the successor node for each node in T*-Tree. Figure 1, represent T*-Node, each node of a T*-Tree has a pointer to the successor node, which makes it easier to scan sequentially by using a simple linked list rather than a tree traversal for query processing, such as range query. This pointer can also be used to pass down directly to the leaf node due to insert/delete underflows.

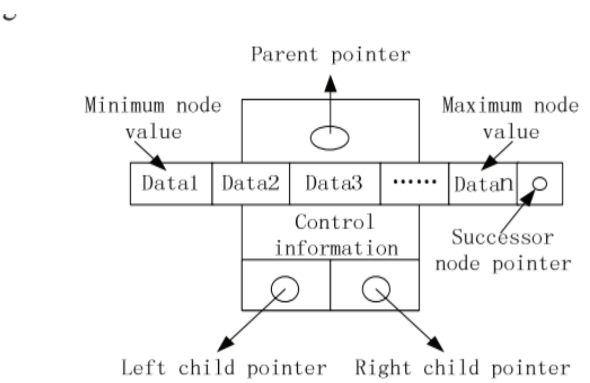

Figure 1:Structure of a T*-node

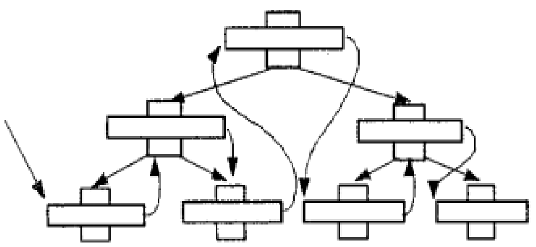

Figure 2: T*-Tree diagram

2 Experiment results and Discussions

To test the performance between T*-Tree and B+Tree, we developed a test suit. test cases imitated all possible operations which database management systems (DBMS) perform on index structure such as data insertion, deletion, search and range queries. The implementation of both algorithms was done in min memory style and implemented in C programing language. All experiments were run using a machine with Intel Core 5CPU @2.30GHz, 8 GB main memory, and running on Windows 10.

Datasets are generated by random number generator. Each tree structure with varying node sizes (from 20 to 100) was tested for the following test cases:

2.1 Offline index insertion

In offline mode the new index is built while the old index is empty. To evaluate the insert cost, one million items were inserted into each index structure. we inserted unique values to the indexes structure. so, insert operation do the additional search operation to ensure that the item was not already in the index.

the results of this test are shown in figure 3

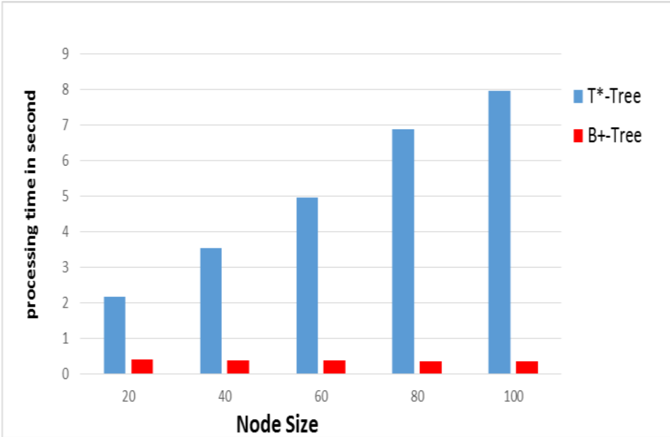

Figure 3: Offline index insertion

The figure clearly shows, B+-Tree is faster than T*-Tree at offline insertion.

2.2 Online index insertion

In online mode the new index is built while the old index has already one million values. To evaluate the insert cost, another one million items were inserted into each index structure. we inserted unique values to the indexes structure. so, insert operation do the additional search operation to ensure that the item was not already in the index.

the results of this test are shown in figure 4

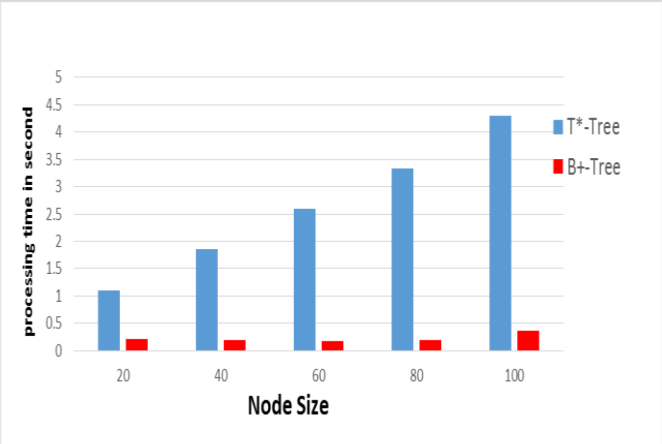

Figure 4: Online index insertion

The figure shows, B+-Tree is faster than T*-Tree at online insertion.

2.3 Delete items from index

To get a more realistic delete cost, we first insert one million data items to the index structure and then we only delete 400,00 data items from the index structure was deleted. deleting all the Tree elements would give the false analysis of the delete cost [6]
the results of this test are shown in figure 5

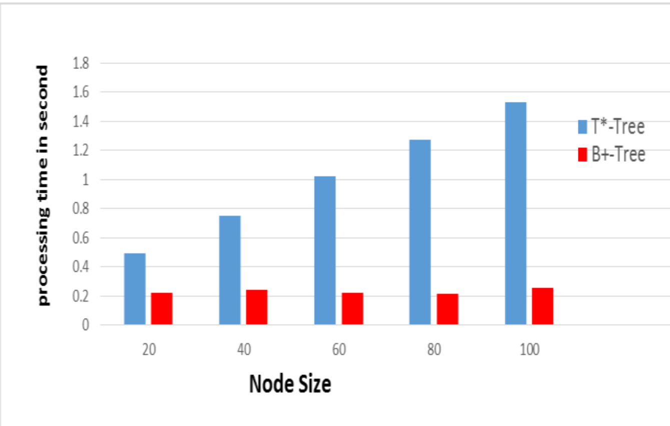

Figure 5: Delete items from index

As we see, B+-Tree is faster than T*-Tree at Deletion operation.

2.4 Searching

To measure the search speed of the indexes structure, each index was searched for 400,000 different random elements. each element requiring a new search.
the results of this test are shown in figure 6

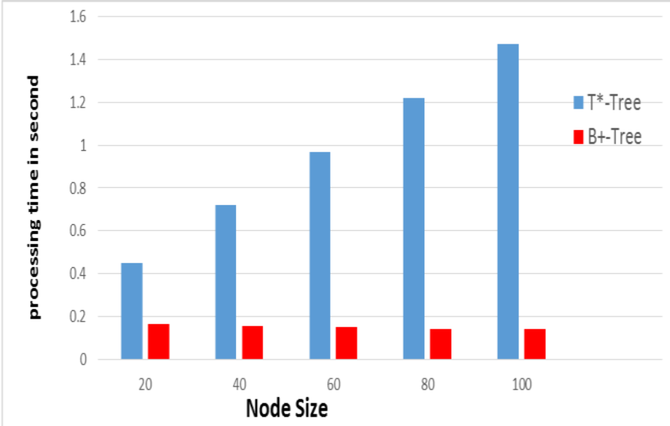

Figure 6: Searching

Again, B+-Tree is faster than T*-Tree at search operation.

2.5 Range Query

DBMS often need to query a list of values corresponding to upper and lower bounds.
To measure the cost of range query for each index structure, 100,000 random range queries were tested. each range query testing different amount of searching. the lower and upper bound of the range is chose randomly. each index structure first search for the lower bound item, then scan the elements until it finds or pass the upper bound item.
the results of this test are shown in figure 7

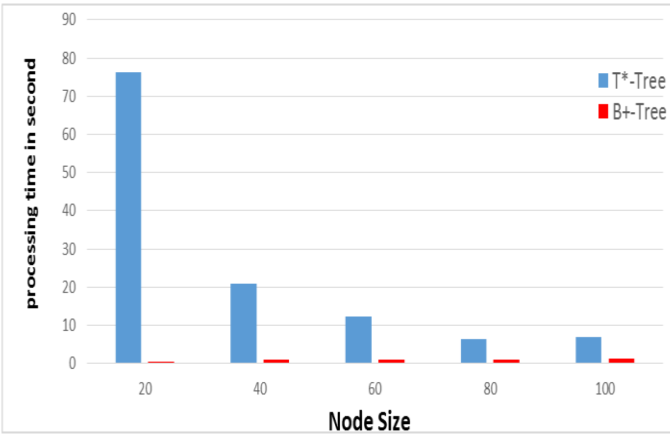

Figure 7: Range Query

The figure shows that in the case of range query, the B+-Tree
provided excellent performance against the T*-Tree.

3 conclusion

In this paper, we revisited the performance comparison between the T*-Tree and B+-Tree on modern hardware. Unlike previous results of related work, our experiment results indicate that when the data reside in memory the B+-Tree index provides better performance than T*-Tree index in the most of index operations. The reason is mainly due to the previous reported works have used old hardware with different memory hierarchy characteristics and single CPU.

Acknowledgements

This work is partially supported by National 863 Program 2015AA015304, NSFC 61472052, Chongqing Research Program cstc2014yykfB40007.

References

[1] J. Rao, K. A. Ross. "Making B+- trees cache conscious in main memory", In Proceedings of the 2000 ACM SIGMOD international conference on Management of data (SIGMOD '00). ACM, New York, NY, USA, 475-486,(2000) .

[2] K. Kyungwha, S. Junho, L. Ig-hoon. "Cache Conscious Trees: How Do They Perform on Contemporary Commodity Microprocessors?",Springer Berlin Heidelberg,pp.189--200,(2007).

[3] K. R. Choi ,K. C. Kim. "T*-tree: a main memory database index structure for real time applications," Proceedings of 3rd International Workshop on Real-Time Computing Systems and Applications, Seoul, 1996, pp. 81-88, (1996)

[4] " Oracle TimesTen", Oracle, [Online]. Available: https://www.oracle.com/database/timesten-in-memory-database/index.html, (2016)

[5] "Oracle TimesTen In-Memory Database Introduction”, [Online]. Available: http://download.oracle.com/otn_hosted_doc/timesten/701/TimesTen-Documentation/intro.pdf, (2007).

[6] T. J. Lehman and M. J. Carey. A Study of Index Structures for Main Memory Database Management Systems. In Proceedings of the 12th International Conference on Very Large Data Bases (VLDB '86), (1986).

[7] J. Hu, H. Yang, Q. Xiong, "Research of main memory database data organization," 2011 International Conference on Multimedia Technology, Hangzhou, pp. 3187-3191,(2011).