Advanced Science and Technology Letters
Vol.108 (Mechanical Engineering 2015), pp.36-40
http://dx.doi.org/10.14257/astl.2015.108.09

A Hybrid B+tree Hash Index for Efficiency Improvement in a NAND Flash Memory

Huijeong Ju[1], Sung-Je Cho[2]

[1]Dept. of Education, Dongbang Culture Graduate University
60, Seongbuk-ro 28-gil, Seongbuk-gu, Seoul, 136-823, Korea
Corresponding Author : dbaguru@daum.net
[2]Dept. of Education, Dongbang Culture Graduate University
60, Seongbuk-ro 28-gil, Seongbuk-gu, Seoul, 136-823, Korea
chosj715@daum.net

Abstract. Recently due to the development of wireless communication techniques, transaction processing within a mobile computing environment has been transferring to a memory based database from a disk based database. A memory based database, unlike a disk based database, uses up only a small capacity of memory and operates on a low-power environment; thus uses a NAND flash memory for fast transaction processing most of the time. This NAND flash memory has its strengths in its light weight, high-speed approaching velocity, and its durability. However due to the NAND flash memory's structure, it takes up lots of time in self-placement modifications and thus creates problems in performance deterioration due to its slow processing speed of writing, deleting, and modifying operations. In order to mend this performance deterioration problem, a model that reduced the maximum amount of problems by using split function of the overflow bucket extended hash index has been introduced. The problem of this model is that problems still exist regarding performance deterioration since the split function of the overflow bucket takes form of a sequential file processing. Thus in this said study, it seeks to introduce a hybrid B+tree hash model that has modified the preexisting problems of the existing model by designing a B+tree for the overflow bucket's sequential file processing. Also it has proven that the suggested model excels the performance of the preexisting model through a performance review.

Keywords: B+tree, Hybrid Hash Index, NAND Flash Memory

1 Introduction

Recently, due to the popularization of portable electronic devices such as smart phones or PDAs, NAND flash memories with faster approaching velocity and lower power-consumption is being used more often over disk based storing mediums. The reason why the NAND flash memory is being used is because it is able to store more amounts of data compared to its small sized, and consumes less power compared to other memories. Especially, it is a non-volatile memory that maintains data regardless of a power supply[1,2]. Also its other strengths are that the data of the NAND flash memory can be input/output directly from the NAND flash memory without having to load the it on the main memory, and that it is very durable to external shocks[2,3].

However, although the NAND flash memory's reading speed is fast; its downside is that its writing speed in comparison is slow. This is because a NAND flash memory can only save data in an initialized block; and because without a pre-initialized block, it takes time to conduct an operation in order to initialize a different block[4]. Also since it is impossible to in-place-update a NAND flash memory, a problem occurs where the speed slows down in the process of erasing a said block and then conducting a writing operation if and when a modification operation occurs[5].

Thus a Flash Translation Layer was studied that will allow the file system used in a disk based saving medium to be applied to a NAND flash memory using the characteristics of the NAND flash memory. The FLT solved the problems the NAND flash memory had so that when a modification operation was run with a characteristic that doesn't start up their reading, writing, and modifying operation identically, both the address re-mapping model and the NAND flash memory field could be used for the linearizability of data and the assignment of a new block[5]. However the frequent use of a modification operation causes in-place-updates; which causes overall performance degeneration. Thus in this said paper it seeks to suggest a hybrid B+tree has index after improving the previously studied hybrid hash index. The previously studied hybrid hash index kept the overflow bucket on the delete/modification record ratio without running the split function using the hybrid has index right away in order to lower the number of operations on the NAND flash memory.

The structure of this paper is as follows. On the 2nd page, related studies will be explained; while the 3rd page explains the hybrid B+tree hash index introduced in this said paper. The 4th page compares and contrasts the introduced model and the preexisting model through a simulation, while the 5th page describes the conclusion.

2 Related Researches

2.1 Circular Hashtag Model [7]

A Circular Hashtag model is a Hash Index Model modified to be fit for a NAND flash memory, and has a reduced reading, writing, and elimination operation by postponing the bucket division as much as possible. Also it has reduced the use of over-flow chains, and has upgraded its performance by using the inside space of the bucket list as much as possible. When the bucket divides, not too many overflow chains are used for the structure restrains the use of overflow chains as much as possible. However if the number of keys saved after setting a threshold value reaches a certain amount, the bucket is divided like any other hashing models. Also in the process of division, all overflow chains are eliminated and re-entered into the bucket list. The division process divides the whole, and does not modify already-existing buckets[7]. However in the process of bucket division, a problem where the number of elimination operations rises from frequent page cancellations due to the frequent use of overflow chains is present.

2.2 A Flash Memory Based B+Tree for Efficient Range Search[8]

Although a node corrections in a disk based B+tree is conducted easily with an in-place-update, excessive in-place-update costs occur when a B+tree is saved and used on a flash memory. If a reef node that has an identical parent node is saved on the same block as the said parent node, and the modified information is also saved on the same block, the consistency of the B+tree can be maintained without using efficient memory space or the modification of a parent node; solving the problem. Also it was made possible so that the reef nodes and parent nodes on the flash memory block can be saved in a p-node block form, and removed internal fragmentation by setting reef nodes at a byte unit[8]. However if a parent node isn't cashed, a full table scan is conducted; shortage of renewal nodes or frequent p-node block initializations from the overflow of reef nodes can bring about performance deterioration problems.

2.3 NAND Flash Memory Based Hybrid Hash Index[6]

A hybrid hash index has been introduced to reduce writing and elimination operations about large amounts of in-place-updates from frequent insertion, deletion, and modification of a record. A hybrid has index conducts either a merge operation or a division operation after a merge to reduce additional operations of the division operation when an overflow occurs. Also when deleting a record by conducting a deletion operation, it does not delete the said record directly off the index; but inserts the key that is to be deleted into the bucket, and carries on the deletion process through a merge operation or a merge-then-divide operation. In other words, the system performance was improved by avoiding and postponing direct deletion operations. A search engine allows for a fast search by starting the search from the bucket with the most recently inserted record; for it searches bucket addresses with the hash values gained from hash functions[6]. However a NAND flash memory based hybrid hash index shows its weakness by spending too much time conducting record searches of buckets allocated in the overflow field in order.

2.4 B+tree

A B+tree is a balanced tree that has levelled height from the root node to the reef node; and is part of the index structure that directly searches for records with data using the key value. Also, the B+tree's each node has n/2 ~ n child nodes; which is composed two different sets - a sequence set that is only made up of reef nodes, and an index set only made up of all other nodes excluding reefs. A reef node has [the address of the key value data record], while the inside nodes of the index set has [key value] saved[9,11].

Many models using B+tree was researched for the performance improvement on the NAND flash memory. A flash memory based B+tree supports range searches using terminal node links, and uses space efficiently[9]. This paper seeks to research the performance improvement of the NAND flash memory using a B+tree.

3 A B+tree Applied Hybrid B+tree Hash Index

This paper conducts an operation on a record by receiving a page unit when carrying out a transaction from the NAND. The weakness is that it takes up a lot of time on the memory elimination operation for an in-page-update on the page.

Previous studies have presented a hybrid hash index model that has applied an overflow model in order to solve this problem. This model distributes overflow buckets to each bucket if and when an overflow occurs; and by not carrying out the division operation immediately, it has reduced additional cost on the operation[6]. This model saves the operation results after conducting the transaction in a sequential file form by allocating an overflow bucket in each bucket. however although additional elimination operations were reduced, time was still over-spent on searching for storage location when conducting additional operations for writing and modifying. Thus the basic idea of this study is to find a way to shorten operation time by applying a B+tree applied hybrid hash index on carrying out a transaction in a sequential file form of an overflow. The Figure 1 below is the outline of the Hybrid B+tree Hash Index.

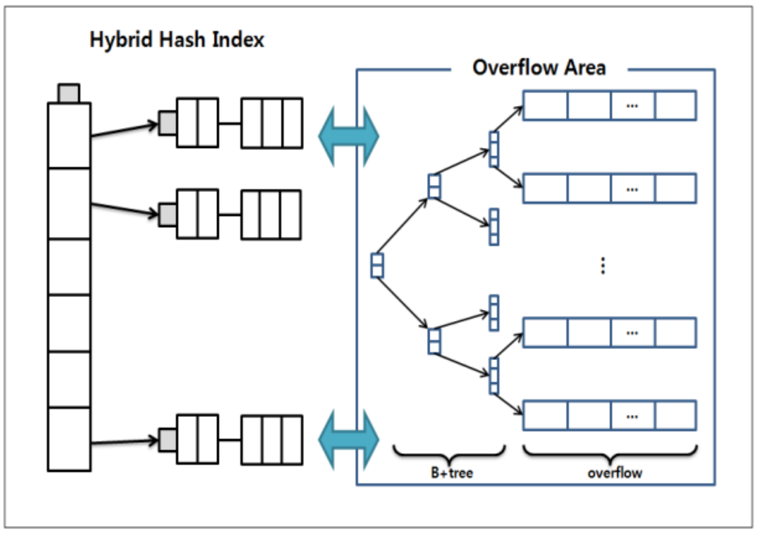

Fig. 1. Outline of the Hybrid B+ Hash Index

The structure of the Hybrid B+tree Hash Index is made up of a hybrid hash index and an overflow. The hybrid hash index operates the division and deletion of extendible hashing indexes, while the overflow has applied the B+tree index model.

The specific algorithm is as follows. First, when the hybrid hash index's bucket is full the overflow field's bucket is allocated. Second, when the overflow's bucket is placed, a specific bucket that is to be allocated shall be searched by finding the reef node using the B+tree within the overflow field.

The advantages of this model are as follows. First, since only the key value is saved within the B+tree's inside node due to structural reasons and saves the key value and the address of the relevant bucket on the reef node, storage space can be used efficiently. Second, since the reef node is connected sequentially, it is possible to do direct processing and range processing.

4 Conclusion

This paper has suggested a new hybrid B+tree hash index for the storage system of the NAND flash memory. The hybrid B+tree hash index manages the overflow buckets allocated to each bucket in case of an overflow. Thus it shortens writing and eliminating operation time due to modification and deletion of frequently occurring records. Henceforth, more research on how to materialize the suggested index structure in this paper on a flash memory is to be done.

References

1. Moon, M.J., Roh, H.C., Park, S,H.: “Database-based Flash Memory File System for Mobile Devices.” Korea Computer Congress, Vol.36, No.1, pp.49-52 (2009)
2. Hwang, E.D., Cha, J.H.: “A Recovery Mechanism applying the Shadow-Paging technique to Flash Memory based LFS.” Korea Computer Congress, Vol.31, No.2, pp.199-201 (2004)
3. Atsuo, K., Shingo, N., Hiroshi, M.: “A Flash-Memory Based File System.” Prceedings of the 1995 USENIX Technical Conference.
4. Brian, D., Markus, L.: “Designing with Flash Memory,Annabooks” (1993)
5. Bae, Y.H.: “Design of A High Performance Flash Memory-based Solid State Disk.” Journal of Computing Science and Engineering, Vol.25, No.6, pp.18-28 (2007)
6. Yoo, M.H., Kim, B.K., Lee, D.H.: “Hybrid Hash Index for NAND Flash Memory-based Storage Systems” Database, Vol.2, No.39, pp.120-128 (2012)
7. Han, D.Y., Kim, K.S.: “A Circular Hashing Index for Flash Memory Storage” Korea Computer Congress, Vol.39, No.1, pp.180-182 (2012)
8. Lim, S.C., Park, C.S.: “A Flash Memory B+-Tree for Efficient Range Searches” JKCA, Vol.13, No.9, pp.28-38 (2013)
9. Bayer, R., MCCreight, C.: “Organization and Maintenance of Large Ordered Indexes” Acta Information, Vol.1, pp.173-189 (1972)
10. Comer, D., “The Ubiquitous B-Trees” ACMComputing Surveys, Vol.11, No.2, pp.121-137 (1979)
11. Knuth, D.: “The Art of Computer Programming” Addison-Wesley Publishing Co.Inc. (1973)
12. Wu, C.H., Kuo, T.W., Chang, L.P.: “An efficient B-tree Layer Implementation for Flash Memory Storage Systems” ACM Transations on Embedded Computing Sytems, Vol.6, No.19, pp.1-20 (2007)
13. Nam, J.H., Park, D.J.: “The Efficient Design and Implementation of the B-Tree on Flash Memory.” Korea Information Science Society, Vol.34, No.2, pp.109-118 (2007)