International Journal of Multimedia and Ubiquitous Engineering
Vol.10, No.10 (2015), pp.283-290
http://dx.doi.org/10.14257/ijmue.2015.10.10.28

Flash Memory Based Failure Recovery Model by Using the F-Tree Index

Sung-Soo Han[1*] and Chang-Ho Seok[2]

[1] *Department of Statistics and Information Science, Bucheon University, Korea*
[2] *Department of Statistics and Information Science, Seoill University, Korea*

Abstract

Recent advancement in mobile miniature information equipment has led to a rapid growth in the usage of storage device based on flash memory. Also, Flash memory is being used as the next generation's storing device and shows rapid growth in massive storage device such as the SSD due to the distinguishable features such as low power consumption, strong durability, approach velocity, non-vibration, non-noise etc. These Flash drives uses B+-Tree index universally. However, B+-tree index has a flaw which downgrades the performance due to the repetitive writing request caused by the insertion of node. This dissertation suggested the F-ISLD (F-Tree Index Segment Log Directory) method which uses the F-Tree index. To prove the method's superiority the suggested method and BISLD (B+-Tree Index Segment Log Directory) was compared. According to the assessment, the overall performance was increased by 29%.

Keywords: *Flash Memory, Index, B+-Tree, F-Tree, Database*

1. Introduction

Recently, flash memory is highly praised as a storing device for portable computer that is equipped with SSD which has cheap price, low energy consumption, portability, permanent storage, non-volatile, and bulk storage. Flash memory with bulk storage, has an outstanding compatibility with other interfaces and for this reason it is a substitute storage medium for HDD [1-2].

Existing flash memory generally uses index affiliated with B-Tree. However, B-Tree has a problem when it comes to, writing by page, and deleting arithmetic by blocks. Due to the Insertion and deletion of Tree node write instruction frequently occurs. This write instruction degrades the storing performance and has a life-shortening effect on flash memories. Therefore, to improve these defects this dissertation is suggesting a method to use F-tree to reduce the burden of write instruction and delete instruction; thus, establishing an outstanding performing index and will be able to recover fast from malfunction. By using an F-Tree index, this study constitutes a system that has the ability to quickly recover in the occurrence of disorder by using the log directory-building F-ISLD technique (F-Tree Index Segment Log Directory technique: hereinafter referred to as the F-ISLD technique).

The following research constructed a system that can quickly recover from malfunction by using F-ISLD method which is a method that uses F-Tree index to construct a Log Directory. The method that this research is suggesting is that the Log to administrate altered information for failure recoveries. Also, when there is an alteration in root node all the data will be committed and check pointed. And when there is a malfunction recovery, work will be performed by the Log information. Therefore, Through the F-ISLD method it is possible to use the buffer in the flash memory and construct a high performing F-Tree index.

ISSN: 1975-0080 IJMUE

The paper is composed as follows; Chapter 2 examines the related research, Chapter 3 explains the suggested method, and Chapter 4 evaluates the performance. Chapter 5 is the conclusion and direction of the study.

2. Related Research

2.1. NAND Flash Memory's B+-Tree Index Buffer Recovery Method

This method is a method that builds a safe B-Tree through a slight overhead that occurs when establishing B-Tree in a Fast speed low power consumption flash memory based SSD. Also, it is a method that reduces recovery time after malfunction. B-Tree flash memory index uses a failure recovery administration method that makes all the data to go through checkpoints and to be committed to SSD so that it can recover its memory back to the committed stage when there is a malfunction. Today B-Tree is mostly used for efficient search in Bulk storage device. Therefore, the problems that can occur in the process of establishing B-Tree in the SSD can be found in the storage system that uses the Bulk SSD. To solve these problems, methods such as BFTL and IBSF was suggested. These methods doesn't save the index data that occurs when B-Tree is established, but temporarily stores it in the buffer and when the buffer is filled, then, reflects it to the B-Tree. Through these processes it reduces the number of write instruction and enhances the B-Tree performance. Especially, IBSF does not commit all the indices but only the ones that are expected to be exported. Also, among the buffers by deleting overlapping data it solved the problem of overhead which was a problem in BFTL. However, in the flash memory by using the buffer when establishing B-Tree it has a possibility of losing all the data when the power goes off due to malfunction. Especially, when the power goes off suddenly all the non-reflected index data will be lost, therefore, all the internally linked data could be lost Thus, a sound B-Tree cannot be established. Generally, Log is used for failure recovery, however, when there is a lot of loss due to malfunction and when there is no way of distinguishing the right Log information it can add up to high cost is recovery. Also, when using B+-Tree in searching in flash memories multiple write instructions occur due to various entries that B-tree node presents. Therefore, a concentrated write instruction will decrease the performance skill of a flash memory. Thus, the critical flaw when establishing B-Tree is that a lot of write instructions occurs in a particular area [3-7].

2.2. IBSF Techniques

In this technique, IBSF utilizes a software module that can insert either a flash layer or file switching. Also the order of the flash memory has been restricted for a more efficient build-up of the B-Tree. ISBF is made through the unit of the nodal information change that occurs due to the build-up of the B Tree, and keeps the buffer going temporarily. After the buffer is completed, the node is renewed by collecting all the units and the committed to one page. Also, ISBF reduced the amount of units by finding all units that were duplicated and either erase it or hold it as bit stream mode. Through this process, IBSF delayed the moments when buffers were packed, and reduced the number of writing operations. Therefore, ISBF can reduce costs because it only requires one operation for completion [5].

2.3. MR-Tree Technique

MR-Tree is a disk based index in space index form that does not require usage of the main memory but can still approach space data. Also, MR-Tree has a space index structure that has a modified form of the space data approachable R-Tree. In the main memory system, when the difference between the CPU's speed and the memory's speed is rising, it is important to reduce the amount of cash miss, and to take into account these

characteristics, MR- Tree has the slight advantage over R-Tree when it comes to being able to increase the usage rate of the middle node entry while lowering the tree's height. Also, through the use of the indexing technique in the flash memory, the efficient approach and management of big amounts of data is possible. However the Tree Index has regular changes in the insert, delete, and split actions. Also, even through the node entry change, writing operations and erasure operations happen frequently, causing the problem of lowering the performance. Therefore, in order to fix this problem the materialization of an efficient R-Tree and lowering of the buffering layer's writing operation frequency of the BFTL and the flash memory system was used to fix the low performance problem, but there is the problem of having additional search operations, and while the efficient approach and management of big data is possible, there is pressure of decoding and re-recording of the same sectors in the same location, when the insert/erase operations and rebalancing operations such as division/merge are carried out [8].

2.4. Crashed Flash File Restoration Technique

This technique is used for efficient resetting during irregular file system crash situations by changing the settings of working area to manipulate the writing tool to be used only in particular areas of the whole flash memory space. Also, working area data can be saved in promised space, so when initialization is carried out, the user can grasp the crafting of the most recently set operation. Through this structure, only the most recent work area is examined and initialized in crash situations. Through this, the amount of time it takes for a crash to be restored proportionately increases with the size of the working area. Through this technique, the problem of increasing expandability is resolved, for no matter how much the flash memory's capacity increases, the crash restoration time is proportionate to it. However, this technique has the weakness of the capacity of the flash memory not being proportionate to the restoration time [9].

3. Technique in Restoring Damaged Flash Memory Base through F-Tree

This research offers a flash memory base restoration technique by improving the B-Tree flash memory, which is the foundation of the base memory database and basic disk database, and using the F-Tree saving technique, which gets rid of pressure from the writing/erasing operations and is able to improve performance. The offered technique is using the node compression and complex operation compression characters in the B-Tree and merging it with the log directory of the F-Tree's improved structure.

3.1. F-Tree Index System Structure

The F-ISLD technique recovery structure is composed of an F-Tree index and log directory, and the diagram is shown in [Figure 1]

With a flash memory, it is impossible to convert each bit from 0 to 1 when carrying out write operations after elimination calculations. However it is possible to carry out writing operations converting 1 to 0, so when changing from 0 to 1 the elimination calculation to reset every bit to 1 must be conducted first. However, the write operation from 1 to 0 has a distinct characteristic in that it is possible without the elimination calculation and it is possible to have higher improvement of performance using this characteristic. In other words, on one page writing 1 to 0 can be repeated without the elimination calculation. Also, the pressure of elimination calculation is removed, extending the life of the flash memory and reducing the saving time.

The F-Tree algorithm is applied in this technique. When applying this algorithm certain malfunctions within the flash memory could occur, it is possible to recover the service

quickly through the RedoLog quick search engine. This is because the F-Tree is superior to others when browsing [10-11].

This technique creates an index range whenever the F-Tree is modified and maintains the buffer until it is full, when the commit is carried out. Also, it records and saves modified information on the log, and when the route node is modified, it commits every index range and executes Check point.

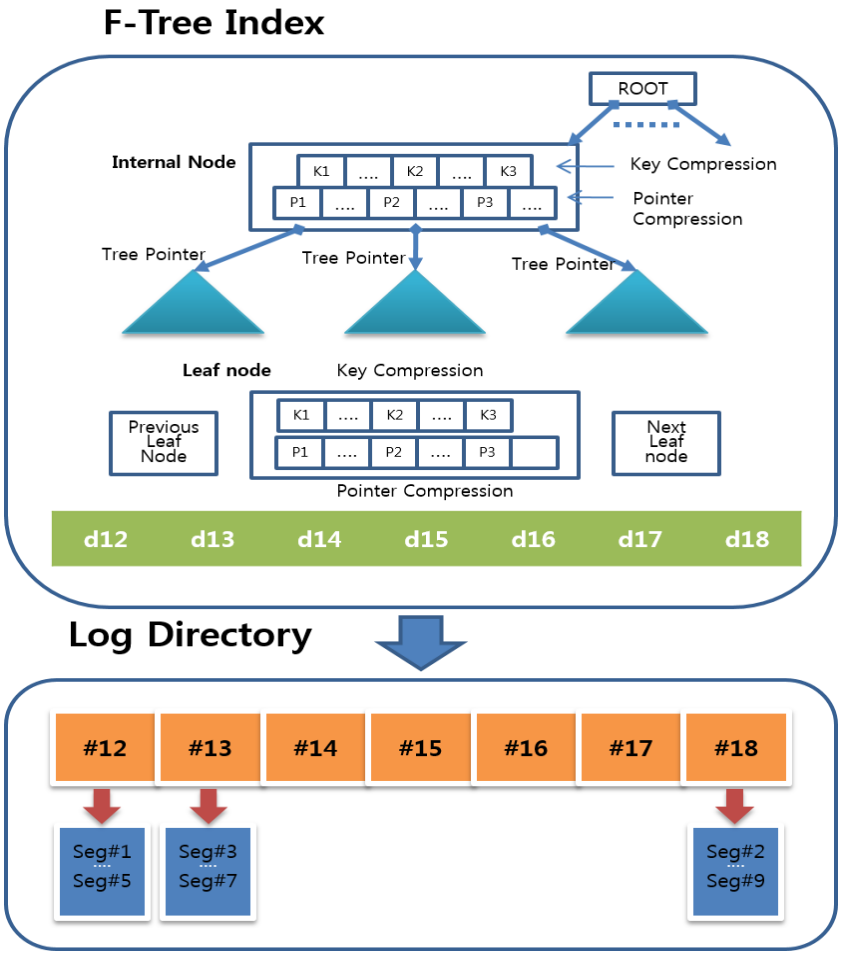

Figure 1. F-ISLD Technique Recovery Structure

3.2. F-Tree Index Algorithm

F-Tree is the structure of the original B+ Tree added with compressing and subsequent writing. The compression of nodes is executed by compressing the key part and the pointer part separately to increase compression efficiency. The F-Tree Index is based off the B+Tree and includes the basic structure of the B+Tree. In the index calculation, when there is insertion or deletion in the tree node, the search engine calculation execution occurs, the compression and recover calculations are added to the original calculations, and in the NAND Flash memory the subsequent writing calculation function is added to the node which it is to be stored. The F-Tree algorithm is applied to algorithm [10].

```
Struct Block_Header {// elimination segment block
BYTE        B_Status;// B_FREE : Not used after initialization
                     // B_WRITING : writing operation in process
                     // B_FULL : Full of pages using the blocks
```



```
                    // B_PREINVALID : Copying to recycle blocks
                    // B_INVALID : Invalid state
Int     No_Erase    //number of erase operations, erase operation counter
etc…// miscellaneous, Error-correcting code, check for faulty blocks, reserve blocks, various tag
information
}

Struct Page_Header {// The page the node will be saved
BYTE        P_Status; //P_FREE : Able to record due to initialization
                    //P_PREWRITE : Writing operation in process, not complete
                    //P_COMPLETE : The uncompressed page where the writing operation is
complete
                    //P_COMPRESSED : Page where 1st compression is complete
                   //P_COMPRESSED2 : Page where 2nd compression is complete
                   //P_PREINVALID : Recycle operation copying to another location
                   //P_INVALID : Invalid state, erased state
short Comp1_end;    // 1st compressed data last location offset
short Comp2_end;    //2nd compressed data last location offset
etc…// miscellaneous, page number, error-correction code, various tag information is stored
}
```

```
Struct Internal_Node {
        Short isLeaf: // 0: intermediate node
        Short num_of_children; //fan-out
        Node *pointer_to_children[MAX1];
        // array of pointer to subnode
        Key key/[MAX1-1];
};
```

```
Struct Leaf_Node {
        Short isLeaf; // 1 : leaf node
        Short num_of_object ; //fan-out
        Node *pointer_to_object[MAX2];
        // array of pointer to sub-object
        Key key[MAX2];
        Struct Leaf-node *next;
        // pointer to next leaf node
};
```

By applying the algorithm, compression is possible, thus reducing save space and the 1st writing is done in the front and afterwards during tree modification it is possible to continue writing in the back secondarily, reducing the number of elimination calculation by half and increasing the tree process performance as a whole, compositing a superb tree index. The way to simulate is by creating a transaction of fixed reading and writing of numbers per second and as a flash memory data base environment main performance evaluation measurement, measures the transaction handling.

4. Performance Evaluation

In this research a simulation was conducted to prove the performance of the F-ISLD technique using the F-Tree. In this simulation the comparison target search technique to the F-ISLD method is the BISLD method using the B+Tree. The tools used are CSIM (Schwetman, 1992) simulation language and C++. The hardware the experiment was conducted on was Intel i7-4770 (3.6GHz) CPU and main memory of 4GB, Hard disk 500GB, the operating system used is the Windows 2000 server. Transaction process means how many transactions were processed per second.

4.1. Experiment Result and Analysis

The results comparing the F-ISLD method introduced in this thesis and the original BISLD method is as follows

Table 1. Recovery Performance Comparison Graph (Measurement :ms)

Transaction	BISLD	F-ISLD	Improvement Rates
400	1.04	0.74	29%
700	0.95	0.68	29%
1,000	0.90	0.65	29%
1,300	0.87	0.62	29%
1,600	0.85	0.60	29%
1,900	0.83	0.59	29%
2,200	0.81	0.58	29%
2,500	0.80	0.57	29%
2,800	0.79	0.56	29%
3,100	0.78	0.55	29%

The transactions created per second are increased from 400 to 3,100 with a unit of 300. The recovery process time shows the distribution from 0 to 1, and displays the lowest F-ISLD method 0.55 to the highest BISLD 1.04. Also, the two methods' performances are compared and the improvement ratio is marked.

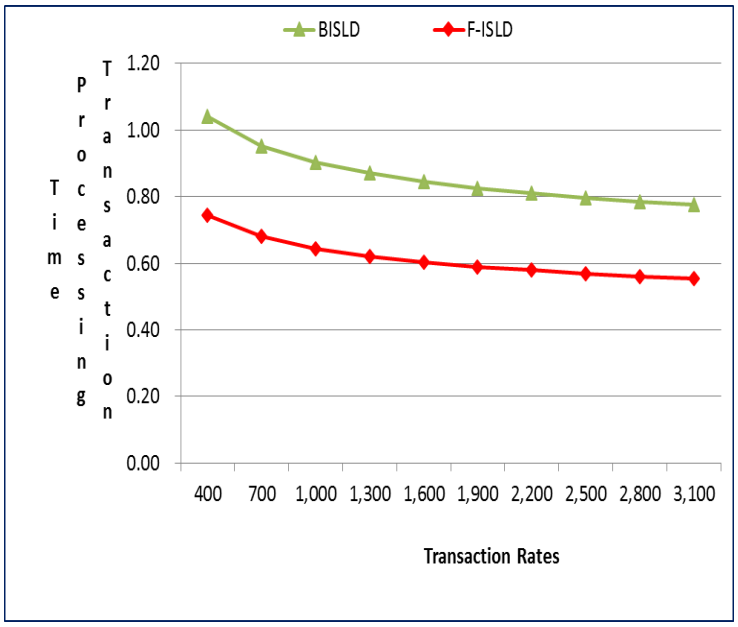

Figure 2. Recovery Time Performance Comparison

Figure 2 displays the performance evaluation results of the original method and the proposed method using a graph. In the graph, it is confirmed that the proposed method improved 29% more than the original method.

5. Conclusion

In this research, in relation to the search and save method of the recently popular portable saving device, the flash drive, the structure of the B+Tree Index which is widely used as the original tree based search and save technology was analyzed. The B+Tree has

problems in it due to the frequent writing technology during the insertion and deletion of nodes, the memory performance decreases and lowers the durability so when there are defects the recovery is slow. Therefore, in this thesis, to fix the problems occurring during recovery, a method was introduced that includes the index using the F-Tree and also allows it to recover simultaneously. The F-Tree search method takes into consideration the disadvantages of input and output of the flash drive, whose writing calculation and elimination calculation is comparatively extremely slow, and used the new compressed continuous writing method to reduce the number of low speed elimination calculation. Therefore, through the use of the F-Tree, the response and process performances of the tree calculation improved. Also, in this thesis the F-ISLD was built using the F-Tree on flash drives, and through the speedy recovery of faults the recovery performance was improved. In the comparative evaluation results, it is confirmed that there was a recovery time decrease and performance improvement of an average of 29% compared to the BISLD method. In the future, there needs to be continuous research on improving the recovery from defects on portable storage devices that use the NAND flash memory.

References

[1] S. E. Corporation, "PM810", Data Sheet, **(2011)**.
[2] T. S. Chung, "A survey of Flash Translation Layer", Journal of Systems Architecture, vol. 55, **(2009)**, pp. 332-343.
[3] C. H. Wu, L. P. Chang and T. W. Kuo, "An Efficient B-Tree Layer for Flash-Memory Storage Systems", Proc. RTCSA, Tainan, Taiwan, **(2003)**, pp. 409-430.
[4] C. H. Wu, L. P. Chang and T. W. Kuo, "An Efficient R-Tree Implementation over Flash-Memory Storage Systems", Proc. Lf ACM CIS'03, New Orleans, Louisiana, USA, November 7-8, **(2003)**, pp. 17-24.
[5] H. S. Lee, B. K. Kim, Y. D. Joo and D. H. Lee, "An Efficient Recovery Management Scheme for an Index Buffer of B+tree based on NAND Flash memory", Journal of Database Research Society, vol, 27, no.03, **(2011)**, pp.20-40.
[6] H. S. Lee and D. H. Lee, "An Efficient Index Buffer management Scheme for Implementing a B-Tree on NAND Flash Memory", Data & Knowledge Engineering, vol. 69,no.9, **(2010)**, pp. 901-916.
[7] T. S. Chung, D. J. Park, S. Park, D. H. Lee, S. W. Lee and H. J. Song, "A survey of Flash Translation Layer", Journal of Systems Architecture: the EUROMICRO Journal, vol. 55, **(2009)**, pp. 332-343.
[8] K. C. Kim and S. W. Yun, "MR-Tree: A cache-conscious main memory spatial index structure for mobile GIS, Web and wireless geo-graphic information systems", The 4th international workshop (W2GIS 2004), **(2004)**, pp. 167-180.
[9] H. C. Park and C. Yoo, "A Design of Efficient Crash Recovery Technique for NAND Flash File System", Journal of Korea Information Science, vol. 35, no. 2, **(2008)**.
[10] S. W. Byun, "F-Tree: Flash Memory based Indexing Scheme for Portable Information Devices", Journal of Information Technology Applications & Management, vol. 13, no. 4, **(2009)**, pp.257-271.
[11] S. S. Han and C. H. Suk, "Enhancement study on NAND flash memory-based failure recovery using F-tree", Advanced Science and Technology Letters, (Ubiquitous Science and Engineering 2015), vol.107, **(2015)**, pp.66-69.

