International Journal of Internet, Broadcasting and Communication Vol.14 No.4 148-153 (2022)
http://dx.doi.org/10.7236/IJIBC.2022.14.4.148

IJIBC 22-4-17

A Study on Metadata Management Based Modified Htree for Smart Device File Management

Youn-A Min

Professor, Applied Software Engineering, Hanyang Cyber University, Korea
yah0612@hycu.ac.kr

Abstract

In keeping with the fourth industrial revolution, IoT-based environments are being implemented everywhere, and as a result, there is an increasing number of cases in which various types of sensed data are utilized efficiently. In particular, the number of multimedia application cases using smart devices is increasing, and there is an increasing number of cases in which large amounts of data are used as fast and convenient as small amounts of data. Given the increasing size of files used in smart devices, the study designed a modified Htree based on journaling meta data management in Ext4-based smart devices and analyzed performance through experiments of various methods. Performance evaluation analysis using IOZone and Hdread shows that the data size of 100Mbyte compared to the data input / output of existing htree is 10% or more, and that of the super large capacity file of 500Mbyte or more is 5% or more.

Keywords: *Smart device; Meta data management; Htree*

1. Introduction

IoT is a natural by-product of the development of the ICT industry and is rapidly emerging as a trend.

The spread of IoT contributes to the development of national competitiveness by providing various convenience to individual life and developing various industries that can utilize data[1,2].

Most of the devices for IoT spread are smart devices. From the viewpoint of smart data management and search, as the IoT spreads, the number of devices that collect a large amount of usable data has been increased, and data management in consideration of a vast amount of data characteristics has been increasing [1-3]. Gartner, an American consulting firm, predicted that global IT spending would reach $4.5 trillion by 2022 and predicted that data center system spending would be the highest, suggesting that the importance of data management systems has increased [4].

As announced by Gartner, Android-based smart devices are increasingly being commonplace as major devices for network and data transmission with various sensors in IoT environments [4-5].

Android-based smart devices commonly use the Ext4 file system, and the Ext4 file system has features such as extents and journaling that are superior to existing file systems when storing large files. Figure 1 shows the data input / output performance for Ext4 compared to the existing file systems (Ext2 / Ext3)

Manuscript Received: September. 23, 2022 / Revised: September. 28, 2022 / Accepted: October. 2, 2022
Corresponding Author: yah0612@hycu.ac.kr
Tel: +82-2-2290-0872, Fax: +82-2-2290-0872
Professor, Applied Software Engineering, Hanyang Cyber University, Korea

[1][6]. Performance analysis of reading and writing data of relatively small size less than 1Mbyte was analyzed through IOZone (Jan,2016)[1,7,13]. However, as the ratio of important information such as personal information input / output through smart device increases, it is necessary to secure the stability of personal information related data through journaling and to improve the performance of reading / writing.

The study improved the stability of data management by journaling and modified the htree algorithm so that Ext4 can read / write data more quickly and stably.

2. Related work

2.1 Filesystem-Ext4

Ext4 file system are file system that supports 1Ebyte volume and 16Tbyte file. It is efficient for mass storage system, uses journaling method by default and is widely used in current Linux operating system [1, 2]. Ext4 distinguishes and manages disks in block units of 4 Kbytes, and manages 32,768 blocks as block groups. It also uses Flex, a large block grouping unit that groups 16 block groups. Extent base management, the extent is one descriptor indicating the range of the consecutively stored physical block [8] and holds detailed information on the storage location of the file composed of blocks. Extent allows retrieval of data through up to 2 concatenations, thus reducing metadata [9]

Ext4 in ode Extents allocates 60bytes larger than the existing file system and has the following structure.

Table 1. Ext4 inode structure [1]

```
header(12bytes) | extent0(12bytes) | extent1(12bytes) | extent2(12bytes) | extent3(12bytes)(a) Allocation of extents
struct ext4_extent {
_le32 ee_block ; / first logical extent covers */
_le16 ee_len ; /* number of blocks coveres by extent */
_le16 ee_start_hi; /* high 16 bits of physical block*/
_le32 ee_start_io; /* low 32 bits of physical block */}; /* from ext4_extents.h */(b) Structure
```

Each extent consists of a group of blocks. For a minimum of 4k, four extents can be used to store a 16k file, and a maximum of 128Mbyte can hold a file size of 512M if four extents are used [1]. If four extents are exceeded, an Htree operated by a hashing algorithm is used [7, 8]. Journal checksum is a way to calculate the CRC32 value for a journal transaction and store it in the journal checksum block. The Journal Checksum feature makes it easier to search for errors through a comparison of CRC32 values in the event of a transaction failure, and allows data to be written to disk regardless of whether header blocks and metadata blocks are written to disk [1]. Flexibility of in node and block allocation, Ext4 has 256 bytes of inode size that can store additional information compared to existing file system and can store metadata of special nature such as file access control list for additional space [1,10]. The extent of the in ode is only 60 bytes, and the leaf nodes use the whole block [1].

If eh-> depth is false the visit algorithm for the leaf node block is executed, and if not, the visit algorithm for the index node block is executed. Table 2 shows the leaf node block [11].

Table 2. Ext4 node selection algorithm [6, 8, 14]

```
struct ext4_extent_header *eh = (struct ext4_extent_header *)block;
if(eh ->depth)
do_index_node(eh, block)
else
do_leaf_node(eh, block)
```

3. Metadata Management Based Modified Htree for Smart Device File Management

3.1 Modeling & Research method

This is a management method for large files that are used when 4 or more extents are used for the Htree used by the Ext4 file system.

Today in various multimedia contents, such as listening to e-learning course through mobile, etc. the file size of 512Mbyte or more, not called a large-capacity file anymore.

The read performance of Ext4 file system is much higher than that of other file systems. However, it was observed that writing is not superior to existing file system due to journaling and journal checksum.

Table 3 shows the pseudo-code that manages the buffer block in this study.

Table 3. Pseudo code for buffer block management

```
Buffer block management ()
Input: Metadata for the file
Output: Buffer block management and metadata modification
{
While (Unmount)
{
If (size of file < 512 Mbyte )
        { Managing metadata in the buffer block management table}
If (size of file >= 512Mbyte || size of file <10Gbyte)
        { Buffer block management table metadata => Transfer to parent node management table }
If (file size >=10Gbyte)
      Start metadata management using B * tree
}
```

Root node information is inserted in the first area of the parent node management table, and metadata for the large amount of data to be inserted is managed from the following page

3.2 Performance evaluation analysis

IOZONE was used for the analysis[12][13]. We experimented with IOZONE. In this experiment, the file size was set from 100Mbyte to 2Gbyte, and evaluated the write performance of the existing Ext4, which was relatively lacking, compared to other file systems.

Also, -U and –o options were set in the experiment so that the file system's buffer cache is applied when reading and writing files.

By applying the buffer cache, the file system is remounted to flush the buffer cache at the time of read request, and write is performed synchronously at the time of write request.

In the experiment, the buffer cache application is set so that the experimental environment is set to be similar to the environment in which actual file I / O occurs.

Table 4. IOZONE setting value

Benchmark tool	factor	setting value
IOZONE (result : Kbytes/sec)	Seq.Read / Seq.Write	file size : 100M,512M,1G
		Record Length : 4K
	Rnd.Read / Rnd.Writ	file size : 100M,512M,1G
		Record Length : 4K

Table 5. Write and rewrite performance analysis using IOZone

1) Writing and rewriting performance evaluation for 100Mbyte files

Division	Ext4	Modified Ext4	Improvement rate
Writer Report	258511	286996	11%
Re-writer Report	248613	278700	12%

2) Writing and rewriting performance evaluation for 512Mbyte files

Division	Ext4	Modified Ext4	Improvement rate
Writer Report	112929	123501	9%
Re-writer Report	144192	146738	2%

3) Writing and rewriting performance evaluation for 1Gbyte files

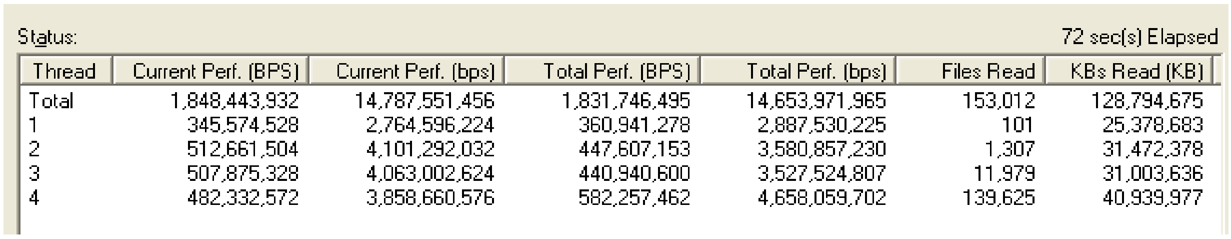
Division	Ext4	Modified Ext4	Improvement rate
Writer Report	79487	81903	3%
Re-writer Report	84079	84301	0%

Hread can measure the total number of reads and performance of files over a period of time [12].

Since benchmark tools for this test are only Windows-based performance tests, VMware was installed in Linux and experiments were conducted through mounting NMMFS and Ext4.

The files added to the experiment are shown as Threads in Figure 26. Thread 1 is a large capacity 700 Mbyte avi file and Thread 2 is a smaller large file which is a 500 Mbyte avi file. Thread 3 and 4 are respectively 100Mbyte avi files and 200Kbyte avi files.

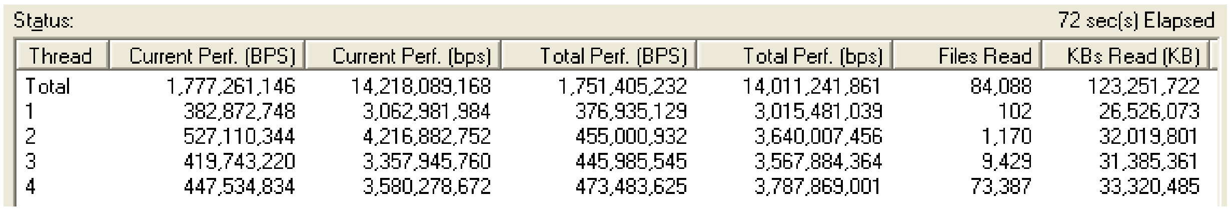
[Figure 6] Reading performance analysis using Hread



Status: 72 sec(s) Elapsed

Thread	Current Perf. (BPS)	Current Perf. (bps)	Total Perf. (BPS)	Total Perf. (bps)	Files Read	KBs Read (KB)
Total	1,848,443,932	14,787,551,456	1,831,746,495	14,653,971,965	153,012	128,794,675
1	345,574,528	2,764,596,224	360,941,278	2,887,530,225	101	25,378,683
2	512,661,504	4,101,292,032	447,607,153	3,580,857,230	1,307	31,472,378
3	507,875,328	4,063,002,624	440,940,600	3,527,524,807	11,979	31,003,636
4	482,332,572	3,858,660,576	582,257,462	4,658,059,702	139,625	40,939,977

The proposed Ext4 file system



Status: 72 sec(s) Elapsed

Thread	Current Perf. (BPS)	Current Perf. (bps)	Total Perf. (BPS)	Total Perf. (bps)	Files Read	KBs Read (KB)
Total	1,777,261,146	14,218,089,168	1,751,405,232	14,011,241,861	84,088	123,251,722
1	382,872,748	3,062,981,984	376,935,129	3,015,481,039	102	26,526,073
2	527,110,344	4,216,882,752	455,000,932	3,640,007,456	1,170	32,019,801
3	419,743,220	3,357,945,760	445,985,545	3,567,884,364	9,429	31,385,361
4	447,534,834	3,580,278,672	473,483,625	3,787,869,001	73,387	33,320,485

Figure 1. Ext4 file system Evaluation with Hread

If the modified Htree algorithm based on the modified journaling meta-data management as described above is used, it is possible to see an improvement of 5% or more even in the case of large files compared to the existing Ext4.

4. Discussion

The use of smart devices is increasing, IOT-based life is becoming more active, and data utilization using smart devices is increasing. Especially, as the use of large-capacity multimedia data increases, researches on efficient management of large-capacity files through smart devices are needed.

The study proposed a modified Htree based on journaling metadata for effective large file management among Ext4 file system data management methods used in smart devices based on the Android operating system. The study divided into 1) when 4 or fewer extents are needed, 2) when 4 or more extents are needed and Htree is used 3) when very large files are used, proposed a method to utilize buffer block management or use parent node table, and in case of very large files, proposed a method to manage metadata effectively using modified B*tree.

As suggestions of this study, for large files larger than 1 Gbyte, there was no significant performance change. This is due to the additional effort to manage in the parent node table and the computation time needed to convert to a modified B*tree. It is planned in the future to continue research in order to address these overhead of computation time.

References

[1] C. Swenson, R. Phillips et al., "File system journal forensics. In: Advances in digital forensics", IFIP international federation for information processing, Boston: Springer, vol. 242, pp. 231-44, 2007.

[2] K. Fairbanks, “An analysis of Ext4 for digital forensics”, Digital Investigation, vol. 9, pp. S118-S130, Aug. 2012.

[3] Chen, Ping-Xiang et al., "Facilitating the Efficiency of Secure File Data and Metadata Deletion on SMR-based Ext4 File System", 2021 26th Asia and South Pacific Design Automation Conference, pp.728-733 Jan, 2021

[4] Data forecasts gartner: Available at "https://www.gartner.com/en/newsroom/press-releases/2022-06-14-gartner-forecasts-worldwide-it-spending-to-grow-3-percent-in-2022"

[5] Kyoungho Koo, "LOCKED-Free Journaling: Improving the Coalescing Degree in EXT4 Journaling", NVMSA, August 2020

[6] K. Wu et al., "Towards an unwritten contract of intel optane ssd", Proc. of USENIX HotStorage, 2019.

[7] V. Tarasov, E. Zadok and S. Shepler, "Filebench: A flexible framework for file system benchmarking", The USENIX Magazine, vol. 41, no. 1, pp. 6-12, 2016.

[8] J. H. Myeong, I. C. Hwang, O. Y. Kwon, "Design Flexible Deduplication Filesystem on Personal Computer Environ ment," International Conference on Future Information & Communication Engineering, Vol. 10, No. 1, pp. 277-280, 2018.

[9] A. Brinkmann, S. Effert, M. Heidebuer and M. Vodisek, "Influence of adaptive data layouts on performance in dynamically changing storage environments", 14th Euromicro International Conference on, pp. 8, 2006.

[10] C. Frost, M. Mammarella, E. Kohler, A. De los Reyes, S. Hovsepian, A. Matsuoka, et al., "Generalized file system dependencies", ACM SIGOPS Operating Systems Review, vol. 41, no. 6, pp. 307-320, 2007.

[11] Y.-S. Chang and R.-S. Liu, "OPTR: Order-preserving translation and recovery design for ssds with a standard block device interface", Proc. of USENIX ATC, 2019.

[12] J. S. Bucy, J. Schindler, S. W. Schlosser, G. R. Ganger, The DiskSim Simulation Environment Version 4.0 Reference Manual, Technical Report CMU-PDL-08-101, 2008.

[13] IOzone Filesystem Benchmark, [online] Available: http://www.iozone.org/.

[14] http://kernel.org/doc/Documentation/filesystems/ext4.txt