Optimizing the Cost of Garbage Collection in F2FS Using Working Set Strategy

Xu Wang [1, †], Boxin Yang [2, †, *]

[1] Department of Software Engineering, Yunnan University, College of Software, Chenggong, KunMing, Yunnan, China

[2] Department of Computer Science, College of Information Technology, Beijing Language and Culture University, Haidian, Beijing, China

* Corresponding Author Email: 201911580814@stu.blcu.edu.cn

[†]These authors contributed equally to the work

Abstract. As an important part of the operating system, the file system has always received wide attention, and different types of file system architectures have been constantly proposed and practiced, this time, we mainly focus on the log structured file system (LFS) type flash friendly file system (F2FS) file system. The existing garbage collection algorithm has offered a interface for future improvement. Hence, we tried to improve existing garbage collection algorithms by implementing the working set strategy in victim select algorithm and analyze the existing deficiencies of the F2FS file system. Under the original strategy, when F2FS selects a segment, it will select the section with the fewest valid blocks. After we change it, we have completed the balance consideration of the change time, the number of valid blocks, and the proportion of changes. It is expected that the Flash device will have the longer life. By using the proposed technique, the cost of garbage collection has reduced by 10%. At the same time, based on the cold and hot data separation characteristics of the F2FS file system, we propose optimization suggestions for the cold and hot data separation strategy.

Keywords: Log Structured File System; F2FS; Garbage Collection; Hot and Cold Data Separation.

1. Introduction

Log structured file system (LFS), which is a log structured file system, has the characteristics of out-place update and sequential write. When updating a file, the updated data will not be written to the original location of the file, but the updated data block will be written to the next location, and the original data block will be marked as invalid. Therefore, there is also a garbage collection mechanism, which is responsible for collecting invalid blocks.

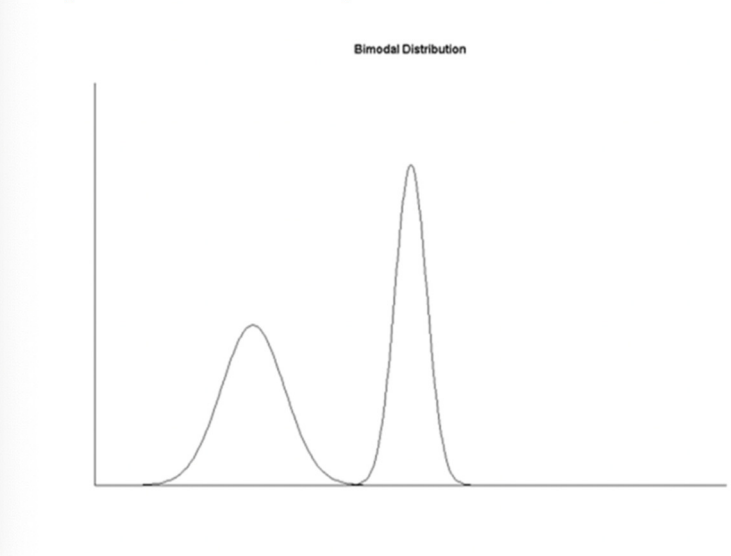

Figure 1. The bimodal distribution

Flash friendly file system(F2FS), is an LFS system. The biggest difference between it and the previous LFS system is the introduction of node address table (NAT) and hot/cold data separation strategy, which effectively solves the overhead caused by the previous file system updating inodes layer by layer when updating data (also known as the wandering tree, avalanche effect). The hot, warm, and cold data are written by F2FS into several segments. Since the data in a hot segment might be modified frequently, the related block is quickly rendered invalid. When the hot segment is chosen as a target for trash collection, it then contains a large number of invalid blocks. The directory entry and file meta-data are classified as hot or warm data according to the current hot/warm/cold separation policy of F2FS.[1], due to the cold and hot separation of the data, the distribution of the valid blocks is closer to the bimodal distribution (bimodal distribution refers to the concentration of more times near the two fractions in the distribution, so that the degree distribution curve has two humped peaks, so it is called bimodal distribution. Typically, it looks like FIGURE 1, the abscissa indicates how hot or cold the data is, and the ordinate indicates its proportion in valid blocks.).

That is, at any time, the number of valid blocks in the segment is concentrated around the lower and higher frequencies, so that frequent recycling of trivial data blocks can be avoided, thereby reducing the operation cost. However, the policy has much room for further improvement. F2FS is a Linux file system that is optimized for use with current flash storage devices. The file system is based on append-only logging, and flash storage characteristics were taken into consideration when making important design decisions. The core concepts, data structures, methods, and performance of F2FS are discussed in the work of Lee C, Sim D, Hwang J, et al..[2] The original layout of F2FS can be seen in FIGURE 2.

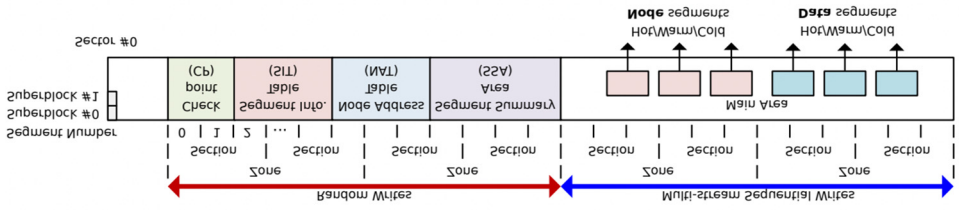

Figure 2. The original layout of F2FS [2]

Several file systems have been suggested and put into practice for embedded systems that employ raw NAND flash memory as storage.[3-7]These file systems take care of all chip-level problems including wear-leveling and faulty block management while having direct access to NAND flash storage. F2FS, in contrast to these systems, is designed for flash storage devices that have a specific controller and firmware (FTL) to handle low-level operations. These flash storage devices are becoming more prevalent.

F2FS is a log-structured-based file system, and garbage collection is a very important feature of the log-structured file system. Because of its unique data allocation method, this type of file system will always maintain a segment manager when running. In the writing process, each block allocated by the user is taken from a segment manager, and the data in the corresponding location of the segment manager is updated according to the address of the block, marking the block as used.

When the user needs to update the file data, the file system will update the data by means of off-site update, that is, the file system will write the user's updated data into a new block and invalidate the old block. Based on the characteristics of the log-structured file system, the main function of Garbage Collection(GC) is to recycle these invalid blocks for the continued use of the file system. The GC of F2FS is divided into foreground GC and background GC: the foreground GC generally runs when the system space is tight, and the purpose is to reclaim the space as soon as possible; while the background GC is performed when the system is idle, the purpose is to not affect the user experience. The situation reclaims a certain space. Foreground GC is usually triggered during checkpoint or write process, because F2FS can sense the space usage. If the space is not enough, it will often trigger the foreground GC to speed up the recovery of space, which means that when the

file system space is insufficient, the performance may be reduced. decline. Background GC is triggered by a thread at intervals.

2. Research Objectives and Method

2.1 Get Familiar with the Layout and Various Data Structures of F2FS File System

The F2FS file system has plenty of heuristic features such as the hot-cold data separation,the implement of NAT,the switch between normal logging and random logging,etc. It's necessary for us to get familiar with those features before we took any future moves.

2.2 Feasibility of Improving Garbage Collection Algorithms.

F2FS uses the variable cost to represent the cost of the segment of gc, and the value of cost is different according to different algorithms. The purpose of the get_victim_by_default function is to find a segment with the lowest cost for recycling, so you need to set a maximum cost before finding it, which is used to traverse and reduce the cost step by step.

The authors of F2FS has left many interfaces for future improvement. In the garbage collection part,the authors has provided a functional interface for victim select. Therefore,besides the current victim select strategy,developers can make their own victim select algorithm work easily.

2.3 Function Realization of Garbage Collection Algorithm.

2.3.1 Basic Unit of Garbage Collection Algorithm.

The smallest unit of F2FS is block, the upper one is segment, the upper one is section, and the highest is zone. The recycling unit of gc is section. By default, a section is equal to a segment, so 512 blocks are recycled for each section.

2.3.2 Start Function

The start function of GC is start_gc_thread, which is executed when F2FS is mounted, and its function is to create a gc thread.

2.3.3 Key Functions

The function of struct gc_inode_list gc_list in variable initialization is to add the inodes affected by the gc section to this list. Because not all blocks in a section are invalid,F2FS will only select sections with relatively more invalid blocks for gc. Therefore, when some valid blocks need to be migrated, their inodes will be affected, so these inodes need to be chained together for centralized processing.

It should be noted that the functions are the __get_victim function and the do_garbage_collect function, where __get_victim selects the most suitable segment for gc according to the number of invalid blocks and other factors, and then passes it to do_garbage_collect for gc.

2.4 Changed Content

Under the original strategy, when F2FS selects a segment, it will select the section with the fewest valid blocks. After we change it, we have completed the balance consideration of the change time, the number of valid blocks, and the proportion of changes. It is expected that the Flash device will have the longer life.

2.5 Detect the Disadvantages of F2FS.

The directory entry and file meta-data are classified as hot or warm data under the current hot/warm/cold separation policy of F2FS. The policy may yet be greatly improved, though. It doesn't take normal files' hotness and coldness into account, in particular.[1]

2.6 Research Method

Because Nand flash device read and write is in page units, and erase is in block units. For the write operation of Nand, it can only change from 1 to 0, but not from 0 to 1. Therefore, the erase operation must be performed on nand first, that is, 0 becomes 1, and then write (to make the corresponding 1 become 0). Through reading and understanding the source code, according to the characteristics of the NAND flash device, design a new garbage collect algorithm, Based on the Cost-Benefit algorithm, we add the consideration of the proportion of valid blocks. The Cost-Benefit algorithm is an algorithm that considers the last modification time and the number of invalid blocks at the same time. Because it is equivalent to frequently modified data, it is not worth performing GC. After GC, they are quickly modified. At the same time, due to the characteristics of remote update, invalid blocks continue to be generated. For data that has not been modified for a long time, it can be considered that invalid blocks continue to be generated relatively less frequently after migration. The core of the Cost-Benefit algorithm is:

$$cost = \frac{1-u}{2u * age} \tag{1}$$

u: Indicates the proportion of the valid block in the section, age represents the last modification time of the section

1-u: Indicates the income after gc for this section

2u: means overhead (read + write)

age: indicates the last modification time

At present, garbage collection algorithms are basically designed around the idea of how to erase each block more evenly. These algorithms are based on the "assume" that the block wear resistance is completely consistent, but in fact, each page and block The wear resistance state of the flash memory is different. When all blocks on a flash memory are erased on average, some blocks must become bad blocks first, while other blocks will have a longer wear resistance. This is determined by the wafer design process. Under the current process conditions, it is still impossible to ensure that each block is completely consistent, and there must be some errors in the wafer. What is uncontrollable is that these errors are randomly distributed.If the wear resistance of each block is originally unbalanced, artificially maintaining the same result of erasing each block through an algorithm will not prolong the overall life of the flash memory, but will damage the optimal life of the flash memory.Therefore, we tried a new strategy, intending to choose a block with a higher input-output ratio, a longer unmodified time, and a larger proportion of invalid blocks for GC.

After determine the target code segment, modify and add the data structure and functions involved, and recompile the changed code into the Linux kernel module to enable it to run without bugs.

3. Some Ideas on Cold and Hot Separation Strategy

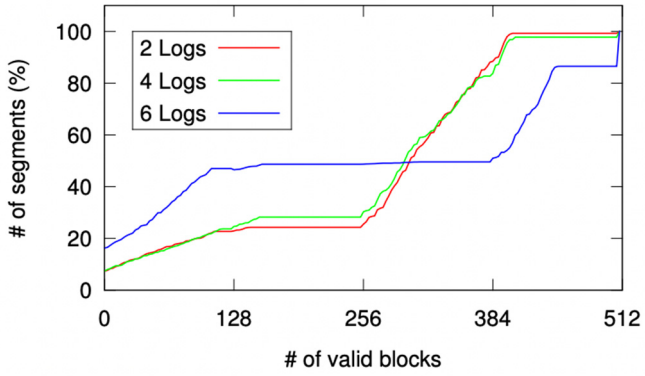

Figure 3. The effect of hot-cold data separation [2]

As mentioned earlier, the strategy of hot and cold data separation is adopted because it can make the distribution of valid blocks in any segment appear bimodal. In the papers published by the F2FS system, the above separation strategy can achieve the following effect: when all six logs are opened, that is, the number of valid blocks in any segment is concentrated around the two peaks (< 128) and (> 384) like we can see in FIGURE 3. And the number of segments with no valid blocks and all valid blocks is also significantly higher than when only two or four logs are opened.

In the original design, the maximum number of log areas can be expanded to 16, which is related to the fact mentioned above that the effect is better when the number of log areas is increased. Whether the finer division of data beyond the "hot and cold" degree can increase the gap between the peaks of the bimodal distribution of valid blocks should be our exploration direction based on the existing information.

4. Working Set Strategy in the Garbage Collection of F2FS.

Based on the characteristics of the log structured file system, the main role of GC is to reclaim these invalid blocks for the file system to continue to use. The GC of F2FS is divided into front-end GC and back-end GC: the front-end GC generally operates under the condition of tight system space, with the purpose of recovering space as soon as possible; The background GC is performed when the system is idle, so as to reclaim a certain space without affecting the user experience. In general, the foreground GC is triggered at the time of checkpoint or write process, because F2FS can sense the space utilization rate. If the space is insufficient, the foreground GC will often be triggered to speed up the recovery of space, which means that the performance may decrease when the file system space is insufficient. The background GC is triggered by a thread at an interval of time.

Using the Working Set, we changed the way F2FS chooses its victim segments. On a Linux system built on VMware, we assessed the expenses associated with trash collection for the original F2FS and the modified F2FS. The operating systems F2FS 1.15.0 and Ubuntu 20.04 are employed. A NM620 SSD is the intended storage device.

Three actual workloads were used: web browser, camera, and music player. The identical input items were created for both the original and modified implementations. The total number of block copies for all GCs during the Web Browser workload under the original F2FS is shown in Fig. 4, and it is also shown in Fig. 4 with the updated F2FS. Therefore, it is clear that the improved victim segment selection policy for the GC can decrease GC costs.

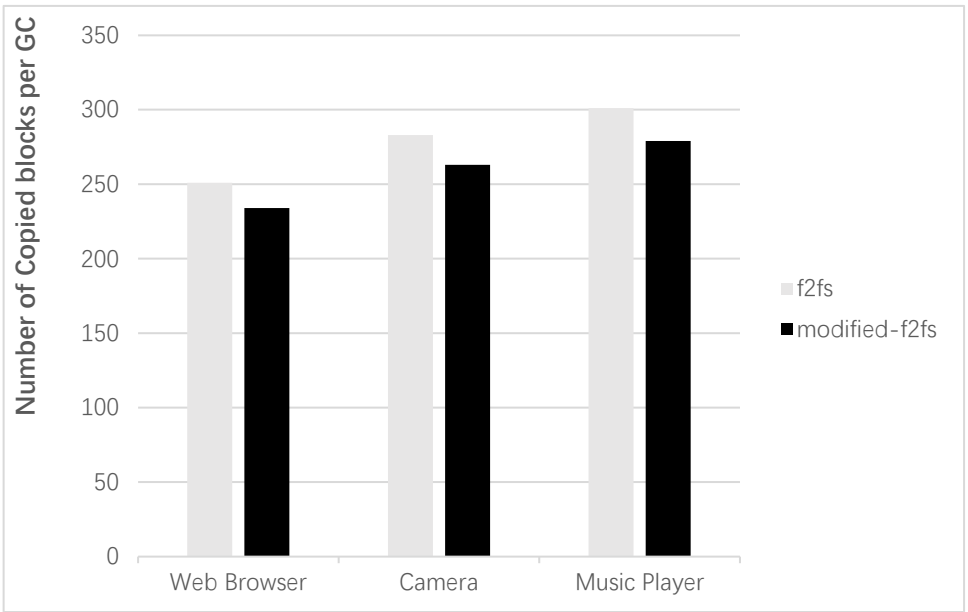

Figure 4. The comparison on GC costs under different workloads

5. Some Disadvantages that Cannot be Ignored

SSDs are made up of a number of flash chips, each of which is made up of several blocks. The atomic unit of read and write operations is a flash page, which makes up each block's predetermined number of pages. A flash translation layer (FTL) is created for the address mapping between logical addresses (LA) and physical addresses in order to determine the physical access location of I/O requests (PA). The SSD controller is outfitted with an internal RAM for caching these LA-to-PA address mappings, also known as a mapping cache, in order to translate from LA to PA efficiently.[9]

Fewer address mappings are cached since RAM is expensive and has limited space.[10] As a result, additional I/O operations are needed to preserve data integrity between flash memory and built-in RAM.[11]

The peculiarities of the NAND flash media become evident under specific flash storage device operation scenarios. For instance, Min et al. note that repeated random writes to an SSD will result in internal media fragmentation and impair the sustained SSD performance.

6. Conclusion

In the process of this research, we made changes to the GC algorithm of the F2FS file system, and completed the balance consideration of the change time, the number of valid blocks, and the proportion of changes, hoping to make the Flash device have a longer lifespan. At the same time, we also put forward the future prospect of the cold and hot separation strategy of the F2FS file system. The current cold and hot separation strategy has shortcomings. As mentioned in the previous article, the cold and hot data separation strategy is adopted because it can make the data separation in any segment. The valid block distribution is bimodal, and the author can expand the log area to 16 at most in the original design, which is related to the fact that the increase in the log area mentioned above achieves better results, and the data is beyond "hot and cold". Whether a finer division of the degree can increase the gap between the peaks of the bimodal distribution of valid blocks should be our exploration direction based on the existing information.

References

[1] Choi D, Shin D. Semantic-Aware Hot Data Selection Policy for Flash File System in Android-Based Smartphones[C]// 2013 International Conference on Parallel and Distributed Systems. IEEE, 2013: 444-445.

[2] Lee C, Sim D, Hwang J, et al. {F2FS}: A New File System for Flash Storage[C]//13th USENIX Conference on File and Storage Technologies (FAST 15). 2015: 273-286.

[3] Unsorted block image file system. http: //www.linux-mtd.infradead.org/ doc/ubifs.html.

[4] Yet another flash file system. http://www. yaffs.net/.

[5] Engel J, Mertens R. LogFS-finally a scalable flash file system[C]//12th International Linux System Technology Conference. 2005.

[6] Kim J, Shim H, Park S Y, et al. Flashlight: A lightweight flash file system for embedded systems[J]. ACM Transactions on Embedded Computing Systems (TECS), 2012, 11(1): 1-23.

[7] Woodhouse D. JFFS: The journalling flash file system[C]//Ottawa Linux symposium. 2001, 2001.

[8] Gao C, Di Y, Deng A, et al. F2FS aware mapping cache design on solid state drives[C]//2018 IEEE 7th Non-Volatile Memory Systems and Applications Symposium (NVMSA). IEEE, 2018: 31-36.

[9] Zhou Y, Wu F, Huang P, et al. An efficient page-level FTL to optimize address translation in flash memory [C]//Proceedings of the Tenth European Conference on Computer Systems. 2015: 1-16.

[10] Min C, Kim K, Cho H, et al. SFS: random write considered harmful in solid state drives[C]//FAST. 2012, 12: 1-16.

[11] Gupta A, Kim Y, Urgaonkar B. DFTL: a flash translation layer employing demand-based selective caching of page-level address mappings[J]. Acm Sigplan Notices, 2009, 44(3): 229-240.