Suspend-aware Segment Cleaning in Log-structured File System

Dongil Park, Seungyong Cheon, and Youjip Won

Hanyang University, Seoul, Korea
$\{idoitlpg|chunccc|yjwon\}$@hanyang.ac.kr

Abstract

The *suspend* feature of the modern smart device practically suppresses the background segment cleaning of the log-structured file system. In this work, we develop *Suspend-aware Segment Cleaning* for the log-structured file system. We seamlessly integrate the segment cleaning into the suspend module of the smartphone OS so that the log-structured file system can reclaim the free segments without interfering with the foreground user activity. The Suspend-aware Segment Cleaning consists of two key ingredients: (i) *Virtual Segment Cleaning* and (ii) Utilization-based Segment Cleaning. We implement Suspend-aware Segment Cleaning in commodity smartphone which uses the log-structured file system, F2FS, as its stock file system (Moto G). F2FS with Suspend-aware Segment Cleaning consolidates 6× more segments than stock smartphone does. With Suspend-aware Segment Cleaning, the F2FS consolidates 2× more segments even with suspend mode on than the case where the phone is always on.

1 Introduction

After over two decades since its inception [1, 2, 3], the log-structured file system is being massively deployed on the commercial device[1] [4]. This is primarily because the append-only nature of the log-structured file system very well addresses the physical characteristics of the NAND flash device: asymmetry between read and write latency and inability of overwrite. Further, the recently announced log-structured file system, F2FS [4] has been reported to effectively resolve the excessive IO behavior in Android IO stack [5].

One of the key concerns in smartphone is the battery lifetime of the device. Various sophisticated techniques [6, 7], are being employed to minimize the energy consumption of the device. These works lower the CPU clock speed and/or turn off the subset (or all) of the hardware components, e.g. screen, and DRAM to reduce the energy consumption. The modern smartphone cannot dispense with suspend mode. The suspend feature leaves less opportunity to OS to execute various background managerial activities. The log-structured file system occasionally consolidates the invalid file system blocks to make a room for incoming write operations. The log-structured file system periodically performs this segment cleaning in background manner since the foreground segment cleaning directly interferes with the ongoing IO requests.

The background segment cleaning of the log-structured file system and the suspend feature of the modern smartphone directly conflicts with each other. The suspend feature deactivates most of the hardware components prohibiting the background segment cleaning routine from functioning properly. As a result, when the file system utilization is high, the application can be exposed to extreme IO latency due to the foreground segment cleaning activity of the log-structured file system.

In this work, we develop *Suspend-aware Segment Cleaning* for log-structured file system. We seamlessly integrate the segment cleaning activity into the suspend module of smartphone OS. Our scheme consists of two key technical ingredients: Virtual Segment Cleaning and Utilization-based Segment Cleaning. We implement Suspend-aware Segment Cleaning at F2FS of commercially available smartphone, Moto G. With Suspend-aware Segment Cleaning, the F2FS consolidates 6× as many segments as the stock F2FS does.

2 Background

2.1 Log-structured File System and Segment Cleaning

The log-structured file system clusters the data and the metadata together, aggregates multiples of these as a single large unit, *segment*, and synchronizes them to the disk in append-only manner. The log-structured file system needs to occasionally consolidate the valid file system blocks to create the free segment. This operation is called *segment cleaning*. The log-structured file system have been believed to behave poorly as file system utilization increases [1, 3, 8, 9]; the prime suspect for performance hit being the overhead of *segment cleaning*. Blackwell et al. [3] pointed out that the segment cleaning overhead can bring 40% performance drop in transaction processing workload. Matthews et al. [8] showed that file system performance starts to drop significantly as file system utilization increases beyond 70%. For flash-memory based storage, Kawaguchi et al. [9] showed that the random write performance at 90% file system utilization is

[1] Moto G, Moto X, Nexus 7

1/3 and 1/5 of the file system performance with 60% and 30% file system utilization, respectively. Their results convey the same message as what Rosenblum et al. [1] illustrated in their original work — with LFS-Greedy victim selection, write cost quintuples when disk utilization increases from 40% to 90%.

Most log-structured file systems adopt two types of segment cleaning: foreground and background segment cleaning. Foreground segment cleaning is activated when there are not enough free segments to write the log sequentially, e.g., under the very high disk utilization (over 95%). Foreground segment cleaning reclaims free segments till sufficient segments are cleaned. It is also called on-demand segment cleaning. Background segment cleaning is periodically activated, e.g. 30 sec. and reclaims free segments [1, 4, 10] to prepare for further write requests. Background segment cleaning should not be activated too frequently because it can block other IO requests and hinder user interactions. Both foreground and background segment cleaning are reclaim one segment per execution, which is 2MB in F2FS by default.

2.2 Suspend Mode of Smartphone

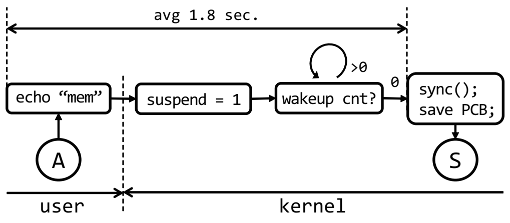


Figure 1: Entering the suspend mode: A: active state, S: suspend state

With few exceptions, the smartphone adopts *suspend* feature to save the energy. The smartphone is put into suspend mode [7] when not in use, typically after 5 - 15 sec. of silence. It freezes all the processes and tasks and then calls suspend callback function in all device drivers. Then, it deactivates all the components except the essential ones, e.g. modem. Fig. 1 illustrates the detailed steps for suspending a device.

The state transition between the active and the suspend state (or mode) consists of a series of steps. The state transition from active to suspend mode starts when the screen is turned off. When the Android platform detects the screen-off signal, the user library (`libsuspend`) writes a character string "`mem`" to `/sys/power/autosleep` file via `sysfs` and sets the suspend flag. It also asks the kernel to check the suspend flag. Kernel updates the internal suspend state if the suspend flag is set. When the suspend state is set, the kernel waits until all outstanding wakeup events finish. After all wakeup events finish, the kernel synchronizes the dirty page cache entries, saves the states of the processes to NAND flash memory and finally deactivates the hardware.

3 Problem Assessment

3.1 The conflict of interests

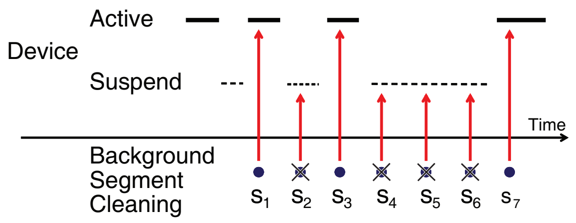


Figure 2: Background Segment Cleaning and Suspend Mode

The background segment cleaning is one of the essential features of the log-structured file system. The log-structured file system allocates separate thread for background segment cleaning. The cleaner thread is periodically activated and consolidates the valid blocks if the number of free segments is smaller than a certain threshold. When entering the suspend mode, the OS kernel freezes all processes. It freezes not only the user processes but also the kernel tasks including segment cleaning thread of the file system. The suspend feature of the modern mobile device practically deprives the log-structured file system of background segment cleaning. Fig. 2 illustrates this situation. Among seven scheduled background segment cleanings (s_1, ..., s_7), only three coincides with the active period of the device. In reality, it can be much worse.

3.2 Suspend Mode in Reality

We examine the behavior of suspend feature in modern smartphone. We capture the block IO accesses and major function calls involved in suspending a device[2] from the real settings (May 1st, 2015 ~ May 19th, 2015, Nexus 5, two users). We analyze the length of active intervals, suspend intervals and the suspension latency. The suspension latency denotes the time to suspend the phone, i.e., the interval from the LCD is turned off till the time when the smartphone OS calls `sync()` to synchronize the dirty page cache entries.

According to our experiment, in 75% of the time, the smartphone stays suspended. For 95% of all active intervals, the length of active interval is less than 4 sec. in user A and 10 sec. in user B (Fig. 3), respectively. When the smartphone is woken up, it takes 5.6 sec. till it is suspended again on the average. The background segment

[2]`enter_state()`, `sys_sync()`, `suspend_prepare()`, `suspend_devices_and_enter()`

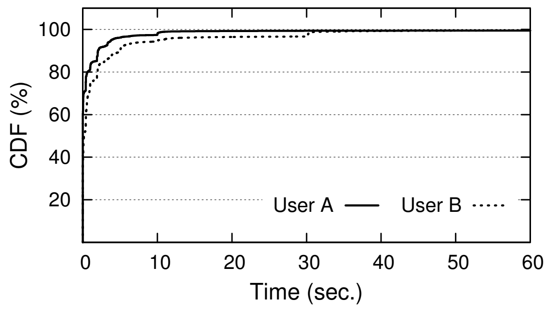


Figure 3: CDF of the length of active period

cleaning can consolidate only 1/4 of the segments compared to the case when the device is always active. Despite its promising property of the log-structured file system, the suspend feature of the smartphone can expose the IO request to extreme delay [1] due to the foreground segment cleaning.

Log-structured file system may use hardware timer to execute a background segment cleaning in periodic manner. Hardware timer literally wakes up the smartphone and all OS and application threads resume executing. This not only entails unnecessary power consumption but also prohibits the background segment cleaning from executing because the awaken thread may access the storage device.

There is important observation. The suspension latency is quite long: the average is 1.8 sec. with $P(X \geq 1.7) = 0.99$ (Fig. 1). The kernel spends most of this period on waiting for all wakeup events to be cleared. We exploit this characteristics and develop *Suspend-aware segment cleaning*. We seamlessly integrate the segment cleaning into suspend module, which does not interfere with the foreground user activity.

4 Suspend-Aware Segment Cleaning

4.1 Design

We find that there exists sufficient slack for performing a segment cleaning when a device is going into suspended state. In Suspend-aware Segment Cleaning, we exploit the suspend feature of the smartphone in performing the segment cleaning not to interfere with the foreground IO. We propose to activate the segment cleaning routine when the LCD is turned off.

The mechanism is simple. When the LCD is turned off, the segment cleaning module examines the file system utilization and the ratio of the invalid blocks. When the condition for segment cleaning is met, the segment cleaning module starts and iteratively claims the free segments until the suspend module calls `sync()`. When the kernel calls `sync()` to synchronize the dirty page cache entries, the segment cleaning module automatically stops. Our segment cleaning module is designed to stop when it detects outstanding IO requests in the queue.

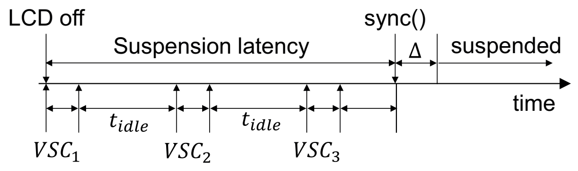
```
SegCleaning ← On ;        /* when LCD is turned off */
function SuspendAwareSegmentCleaning()
    $t_{idle}$ ← 500 ;   /* default sleep time 500 msec */
    while true do
        if SegCleaning = On then
            sleep($t_{idle}$) ;
        end
        else
            wait for SegCleaning = On ;
        end
        calculate $t_{idle}$ with disk utilization ;
        if IO subsystem is not idle or $p_{inv} < p^*_{inv}(u)$ then
            SegCleaning ← Off ;
            continue ;
        end
        do Virtual Segment Cleaning ;
        if no victim segment then
            SegCleaning ← Off ;
        end
    end
end
```

Figure 4: Pseudo-code of Suspend-aware Segment Cleaning

Suspend-aware Segment Cleaning exploits the large slack (1.8 sec. on the average) between the time when LCD is turned off and the time when The OS starts to freeze the processes. It does not interfere with the foreground user processes nor does it increase the time to suspend a device.

Fig. 4 illustrates the pseudo code for Suspend-aware Segment Cleaning. The Suspend-aware Segment Cleaning consists of two key technical ingredients: Virtual Segment Cleaning and the Utilization-based Segment Cleaning.

4.2 Virtual Segment Cleaning



VSC_i : i_{th} Virtual Segment Cleaning

Δ: Time for synchronizing dirty page cache entries

Figure 5: Suspend-aware Segment Cleaning with three Virtual Segment Cleanings

The Suspend-aware Segment Cleaning routine does not entail any filesystem write operation; instead, it selects the victim segment from the file system, fetches the valid blocks in the victim segment into page cache, and marks these page cache entries as dirty. This process repeats until the power management module freezes the thread. The Suspend-aware Segment Cleaning module delegates the synchronization of newly reclaimed segments to the suspend module. Suspend module calls `sync()` to synchronize the dirty page cache entries and the newly reclaimed segments can be synchronized to file system via suspend module before putting the device into the suspend mode. We call this as *Virtual* Segment Cleaning.

Fig. 5 illustrates the Suspend-aware Segment Cleaning. When LCD is turned off, virtual segment cleaning starts. After the first virtual segment cleaning, VSC, the thread sleeps for a certain interval, t_{idle}, and then starts the second virtual segment cleaning. This process continues until the suspend module calls `sync()`. The reason we introduce an idle period between the successive virtual segment cleaning is to throttle the amount of reclaimed segments.

4.3 Utilization-based Segment Cleaning

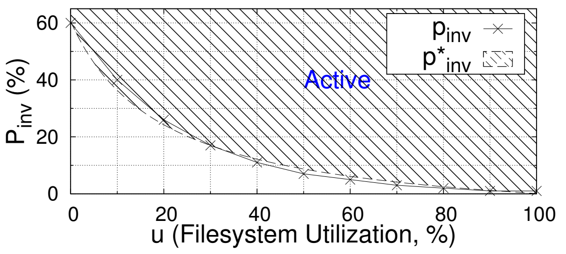


Figure 6: Threshold for activating the segment cleaning

The Suspend-aware Segment Cleaning is activated every time when the LCD is turned off. It may cause too frequent segment cleaning, which leads to the reduction of the NAND flash lifetime due to excessive write operations. Based upon the file system utilization, we establish a condition for triggering the segment cleaning and the interval between the successive virtual segment cleanings, t_{idle}.

There are three types of blocks in the log-structured file system: *valid*, *invalid*, and *free*. We introduce the notion of invalid block ratio, p_{inv} as $\frac{n_{invalid}}{n_{valid}+n_{invalid}}$, where n_{valid} and $n_{invalid}$ denotes the number of valid blocks and invalid blocks in the file system partition, respectively. File system utilization, u, corresponds to $\frac{n_{valid}}{n_{valid}+n_{free}+n_{invalid}}$. For a given file system utilization, u, we establish a minimum invalid ratio, $p_{inv}(u)$. When LCD is turned off, we check if the fraction of invalid block is greater than $p_{inv}(u)$ and trigger segment cleaning only when this condition is satisfied. The underlying philosophy is to trigger segment cleaning only when there are sufficient amount of invalid blocks, and to reclaim more segment as the file system utilization increases.

We establish the threshold, $p_{inv}(u)$, as follows. For empty file system, i.e. $u = 0$, the segment cleaning starts only when the fraction of invalid blocks is larger than 60%, i.e. $p_{inv}(u)$ is set to 0.6. The threshold of triggering the segment cleaning decreases by the factor of 2/3 as the file system utilization increases by 10%. The threshold value can be represented by Eq. 1.

$$p_{inv}(u) = 0.6 * (2/3)^{u/10}, 0 \leq u \leq 100\% \qquad (1)$$

Exponential function is computationally very expensive and cannot be used in the Kernel code. To reduce the computational complexity, we approximate this formula as in Eq. 2.

$$p^*_{inv}(u) = (1450/(u+20) - 12)/100 \qquad (2)$$

In Fig. 6, the shaded region denotes the region where the Suspend-aware Segment Cleaning becomes active.

We regulate the amount of reclaimed segments via adjusting the idle period, t_{idle}. We like to reclaim at least two segments and at most six segments before a device goes into suspend mode. Given the average suspend latency of 1.8 sec., we set the minimum and maximum value of t_{idle} to 300 msec and 900 msec, respectively, The length of the idle period is determined by the file system utilization, u and the invalid block ratio, p_{inv} as in Eq. 3.

$$t_{idle}(p_{inv}, u) = 300 + 600 \left(\frac{1 - p_{inv}}{1 - p^*_{inv}(u)} \right) \text{ms} \qquad (3)$$

5 Experiment

We implement Suspend-aware Segment Cleaning on F2FS. We use Motorola Moto G (Android 4.4.2, Linux Kernel 3.4) in our experiment. We perform two experiments: (i) the number of reclaimed segments under Suspend-aware Segment Cleaning and (ii) the effectiveness of the Utilization-based Segment Cleaning.

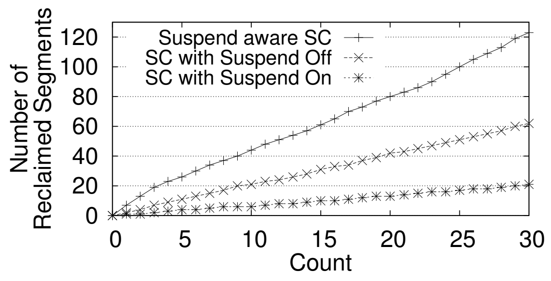


Figure 7: The Number of Reclaimed Segments

We examine the number of segments reclaimed in three situations: (i) Suspend-aware Segment Cleaning with suspend-mode on, (ii) Stock segment cleaning with suspend mode on and (iii) Stock segment cleaning with suspend mode off. Segment cleaning in F2FS is activated in 30 - 60 sec. interval, and it reclaims one segment on each segment cleaning. Fig. 7 illustrates cumulative number of segments reclaimed in each case. By turning on the suspend mode, the F2FS reclaims only 1/3 of the segments of what it used to collect via background segment cleaning. With Suspend-aware Segment Cleaning, it reclaims 6× number of segments than the stock F2FS does. Even with occasional suspension, Suspend-aware Segment Cleaning collects 2× number of segments as the stock F2FS does with the device being always active.

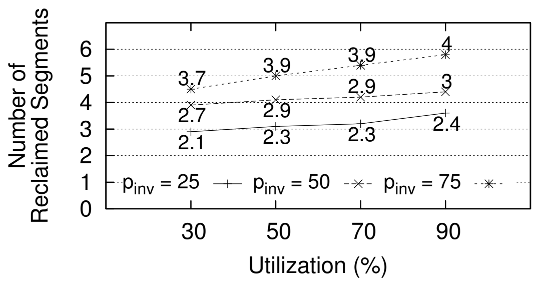


Figure 8: Number of Reclaimed Segments vs. the File System Utilization (u = 30, 50, 70, 90%) and Fraction of Invalid Blocks ($p_{inv} = 25, 50, 75\%$)

We examine the effectiveness of Utilization-based Segment Cleaning. We vary the file system utilization (u = 30, 50, 70, 90%) and fraction of invalid blocks in the file system partition ($p_{inv} = 25, 50, 70\%$). We put the device to suspend mode 50 times and count the total number of reclaimed segments. We develop an Android application for accurate replay of the test cases. Fig. 8 illustrates the result of experiment. It shows the number of reclaimed segments obtained from the physical experiment and from the analytical model (Eq. 2 and Eq. 3). The points denote the result of the experiment and each point is annotated with the theoretical value. As the file system utilization increases the utilization-based segment cleaning effectively increases the number of reclaimed segments. Likewise, the number of reclaimed segment increases with the ratio of the invalid blocks. The Utilization-based Segment Cleaning algorithm effectively regulates the number of reclaimed segments and the number of reclaimed segments precisely coincides with the theoretical value obtained from the model.

6 Conclusion

Background segment cleaning module of the log-structured file system and the suspend feature of the modern smartphone device directly conflict with each other. In this work, we develop a segment cleaning scheme for modern log-structured file system so that the background segment cleaning can be seamlessly integrated with suspend module of the smartphone OS. We develop Virtual Segment Cleaning and Utilization-based Segment Cleaning to exploit the `sync()` operation of existing suspend module in synchronizing the consolidated file system blocks and to regulate the amount of consolidated segments to avoid excessive cleaning, respectively. The result is very promising. Suspend-aware Segment Cleaning claims 6× as many segments as the stock background segment cleaning does.

Acknowledgment

This work is supported by Brain Korea 21 Program for Leading Universities & Students, and by MSIP(Ministry of Science, ICT and Future Planning), IT R&D program MKE/KEIT (No. 10041608, Embedded System Software for New-memory based Smart Device).

References

[1] M. Rosenblum and J. K. Ousterhout, "The design and implementation of a log-structured file system," *ACM Transactions on Computer Systems*, vol. 10, no. 1, pp. 26–52, 1992.

[2] M. Seltzer, K. Bostic, M. K. McKusick, and C. Staelin, "An implementation of a log-structured file system for UNIX," in *Proc. of the USENIX Winter Conference*, 1993, pp. 307–326.

[3] J. H. Blackwell, Trevor and M. I. Seltzer, "Heuristic cleaning algorithms in log-structured file systems," in *Proc. of the USENIX ATC*, 1995, pp. 277–288.

[4] C. Lee, D. Sim, J. Hwang, and S. Cho, "F2FS: A new file system for flash storage," in *Proc. of the USENIX FAST*, 2015, pp. 273–286.

[5] S. Jeong, K. Lee, S. Lee, S. Son, and Y. Won, "I/O stack optimization for smartphones," in *Proc. of the USENIX ATC*, 2013, pp. 309–320.

[6] P. Pillai and K. G. Shin, "Real-time dynamic voltage scaling for low-power embedded operating systems," *ACM SIGOPS Oper. Sys. Rev.*, vol. 35, no. 5, pp. 89–102, 2001.

[7] A. L. Brown and R. J. Wysocki, "Suspend-to-RAM in Linux," in *Proc. of the Linux Symposium*, 2008, p. 39.

[8] J. N. Matthews, D. Roselli, A. M. Costello, R. Y. Wang, and T. E. Anderson, "Improving the performance of log-structured file systems with adaptive methods," in *Proc. of the ACM SOSP*, 1997, pp. 238–251.

[9] A. Kawaguchi, S. Nishioka, and H. Motoda, "A flash-memory based file system," in *Proc. of the USENIX ATC*, 1995, pp. 155–164.

[10] R. Konishi, Y. Amagai, K. Sato, H. Hifumi, S. Kihara, and S. Moriai, "The Linux implementation of a log-structured file system," *ACM SIGOPS Oper. Sys. Rev.*, vol. 40, no. 3, pp. 102–107, 2006.