공학석사 학위논문

Execution of a Soft Real-Time Multitask Program on NAND Flash Memory with Guaranteed Reliability

NAND 플래시 메모리 상에서 수행되는 연성 실시간 멀티태스킹 프로그램의 신뢰성 보장 기법

2018 February

서울대학교 대학원

전기 · 컴퓨터공학부

Gadimli Nushaba

Execution of a Soft Real-Time Multitask Program on NAND Flash Memory with Guaranteed Reliability

지도교수님 이창건

이 논문을 공학석사학위논문으로 제출함

2017 년 10 월

서울대학교 대학원

컴퓨터 공학부

Gadimli Nushaba (뉴샤베)

뉴샤베의 학위논문을 인준함

2017 년 12 월

위 원 장 민상렬 (인)

부 위 원 장 이창건 (인)

위 원 하순회 (인)

Abstract

Execution of a Soft Real-Time Multitask Program on NAND Flash Memory with Guaranteed Reliability

Gadimli Nushaba
Department of Computer Science and Engineering
The Graduate School
Seoul National University

As solid-state drives based on flash technology are becoming a staple for persistent data storage in data centres. Especially, solid-state drives based on the NAND flash technology in non-volatile storage for storing data. Yet it is still facing some difficulties in executing program codes on a NAND flash memory as a consequence of supporting only page-based reads. But with the constructed method of recent years so-called mRT-PLRU (Multitasking Real-Time constrained combination of Pinning and LRU), which forms a generic framework to use inexpensive non-volatile a NAND flash memory storage for storing and executing real-time programs in a multitasking environment, it allows executing program code on a NAND flash memory. However, since this technique does not consider reliability problems it exhibits some issues when performing read operation - especially read disturb errors which occur after repetitive read operations for each block, there is a limitation in terms of reliability in the execution of a real-time program code. Therefore, we propose a new approach to execute soft real-time tasks on NAND flash memory with

minimal RAM size while considering read disturb errors. For this, our approach has been performed in two steps in order to execute multiple real-time tasks stored in a NAND flash memory with the minimal requirement of a RAM size. In the first step so-called, per-task analysis, we analyse execution time of each task one by one by changing RAM size and check blocks in NAND flash memory which are accessed during execution of the task corresponding to optimal pinning and LRU combination for each individual task. In the second step so-called, stochastic-analysis-in-loop-optimization, we use those functions from the per-task analysis for all tasks as an input to allocate minimum possible RAM size to execute all tasks while meeting given deadline meet probabilities. In this step, we consider additional delay needed to prevent read disturb errors.

As a consequence of configured technique, compared with other approaches, this technique significantly minimised a RAM size requirement. Motivated by this, under this kind of real-time system circumstance, we propose a new approach to guaranteeing reliability, especially focusing on the read disturb problem on the NAND flash memory while executing multiple tasks. To this end, our approach rewrites the blocks considering overhead for a rewrite of NAND blocks. Those blocks with high read count, before exceeds the threshold limit for each block, should be rewritten earlier with the reserved time slots. At the end, checking of all given tasks schedulability with an exact stochastic analysis for probabilistically guaranteeing the reliability of read operation on NAND flash memory that the deadline is met for all given tasks.

Keywords: Soft real-time program, NAND flash memory, reliability, multitasking, mRT-PLRU

Student Number: 2015-22147

Contents

List of Figures

List of Tables

1 Introduction

As solid-state drives based on flash technology are becoming a staple for persistent data storage in data centres. Especially, solid-state drives based on the NAND flash technology in non-volatile storage for storing data. For many products, flash memory is the best option for its large capacity, low power consumption, shock resistance, fast random access performance. Yet it is still facing some difficulties in executing program codes on a NAND flash memory as a consequence of characteristic limitations such as it is supporting only page-based reads, only on a 2 KB page size for a read operation performance allowed, and no byte-level random accesses enabled. Regarding latterly mentioned reasons, it could have been almost making impossible for program code execution on NAND flash memory.

During the previous years, there was proposed and used a few approaches regarding the program execution on the NAND flash memory. On the architecture in Figure 1.1, the most commonly used approach in the industry is shadowing [1]. In shadowing approach, since all program codes in NAND flash memory are copied into RAM and codes in RAM are executed, a large amount of RAM is required. However, because the price of RAM is much higher than the price of NAND flash memory, shadowing has a problem in terms of cost efficiency. In order to reduce RAM usage, [10] approached a NAND based technique, *demand paging mechanism* for low-end embedded systems with NAND flash memory as a

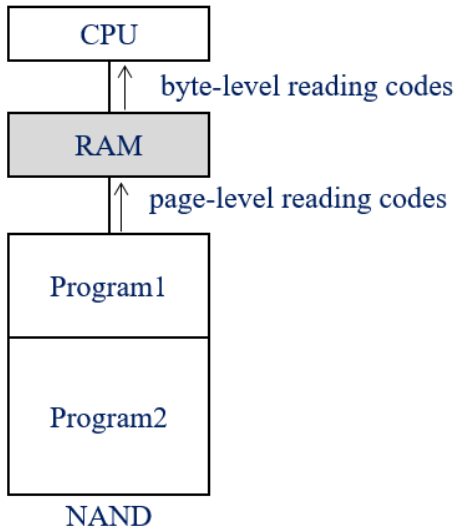


Figure 1.1. An example of architecture of NAND flash memory system

secondary storage that read pages into RAM when they are actually accessed while replacing pages as needed. However, this approach does not provide any real-time program execution requirements because of unpredictable page-faults.

To overcome this problem, [6, 7] propose needs a technique to execute programs on NAND flash memory minimizing the required RAM size, satisfying the program's real-time system requirements. More specifically, when programs on NAND flash memory are soft real-time tasks, [6, 7] propose minimal RAM size to execute all programs while meeting given deadline meet probabilities of the programs. However, since [6, 7] do not consider to guaranteeing the reliability of NAND flash, more specifically read disturb during a read operation from NAND to RAM which occurs after repetitive access of reading operation for each block,

as there is a limitation in term s of reliability in [6, 7]. In order to manage reliability issues on NAND flash memory, there have been proposed many approaches recent years. [2,4] proposes to deal with current issues with developed and improved *ECC and Bad Block Management*, [3] proposes to manage reliability issues with developed *EC solution by concatenating trellis coded modulation* with significantly improves performance along with *BCH codes* only, when [5] more focus to improve reliability via *Dynamic Threshold* and developed coding schemes working with dynamic verification thresholds that tolerating retention process which significantly reduces BER. However, none of these studies considers or provide real-time system requirements.

In this thesis, therefore, we propose an approach to executing soft real-time tasks on NAND flash memory with minimal RAM size while considering NAND flash memory's reliability, specifically read disturb errors of read operation as it is the current need of the mobile embedded system industry. For this, we perform two-step procedure. In the first step, we analyze execution times of each task by changing RAM size and check blocks in NAND flash memory which are accessed during execution of the task. In the second step, based on the analysis result, we find minimal RAM size to execute all tasks while meeting given deadline meet probabilities. In this step, we consider additional delay needed to prevent read disturb errors.

The rest of this thesis is organised as follows. Section 2 surveys the related works. Section 3 briefly introduces the background and defines our specific problem description. Then in Section 4 proposes our method for guaranteeing reliability for soft real-time multitask program executions on NAND flash memory. Section 5 evaluates our proposed method. Finally, Section 6 concludes the paper and brings up future works.

2 RELATED WORKS

Based on recent years works on NAND flash memories very little has been published on the internals of solid state drives. The majority of researches has been published either about the development of algorithm and data structure for such as address mapping, block cleaning, wear levelling or the development of hybrid memory systems. In a hybrid setting, flash memory is mostly utilized as a cache for hard disk drives to reduce overall power consumption of I/O systems. One of that methodology [12, 13] is employing an embedded system (or prototype system) with attached flash memory on it. A number of wear-levelling techniques have been proposed to balance the wear on flash memory blocks within an SSD to improve endurance and they are discussed in [14]. The proposed techniques include threshold based schemes using per-block erase-counters, techniques that use randomization to choose to erase blocks, and those that separate hot and cold blocks when making wear levelling decisions [14].

However, none of these studies addresses the issues of satisfying real-time requirements along with guaranteeing the reliability of NAND flash memory also wasn't there one of the main focuses. In [19, 20] works, they are addresses the issue of the page-based-read operations in terms of probabilistically guaranteeing real-time program executions, later these approaches enlarged to [6, 7]. However, the proposed techniques do not address read disturbs or any NAND reliability problems, simply assuming that page faults on flash memory are handling by page

fault handler, every time NAND page load to RAM invaliding previous page. Later these study had been practised under hard real-time system circumstances [8]. Most of the studies focused to avoid inter-task conflicts in the cache, cache-partitioning techniques [21, 22, 23] partition the cache so that each task uses only its assigned partition. The idea is similar to our RAM partitioning. However, those works don't have any consideration for possible improvement by combining pinning and LRU in each partition along with the guaranteed reliability of read operation.

3 BACKGROUND AND PROBLEM DESCRIPTION

Because of the physical nature of the NAND flash memory [15], we know that we cannot execute directly program code from NAND. Therefore, the program codes stored in the NAND flash memory need to perform read operation from NAND to RAM for execution of the real-time program. While performing a read operation, as a consequence it causes read disturb, which is happening when the program reads frequently accesses data from a flash page, which cause the threshold voltages of other unread pages in the same block to shift to the higher value of read count. While a single threshold voltage shift is small, such shifts can accumulate time, eventually becoming large enough to alter the state of some cells and hence generate *read disturb* [9, 11]. In our case each time when RAM requests code to execute from NAND, eventually frequently requested code's read count hits the threshold and read disturb occurs. Figure 3.1 illustrates the read disturbs while read operation on NAND.

In the original method in [7], without consideration of any possible read disturb, given task set of the program code execution, and considered read delay for a read operation from NAND to RAM, all tasks had successfully met their deadline, without any violation. Figure 3.2 illustrates how original task set scheduling changes when we have to consider read disturb as well. For this, our problem is, accumulate the block read count for each task and reschedule the tasks with consideration of predetermining read count of block rewrites, in the way that all

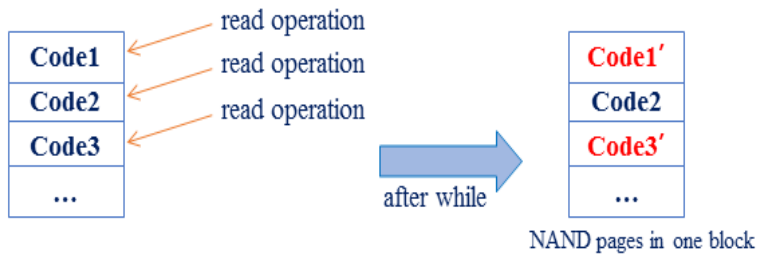


Figure 3.1. Read disturb error on NAND flash memory

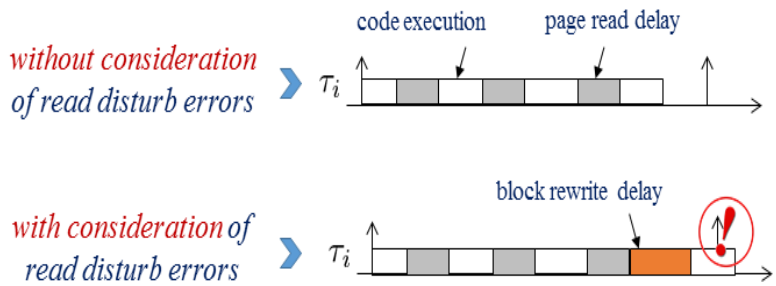


Figure 3.2. Consideration of read disturb errors

tasks meet their deadlines, and it will be probabilistically guaranteed.

3.1 Reliability on NAND Flash Memory

NAND flash memory is organized into blocks where each block consists of a fixed number of pages. Each page stores data and corresponding metadata and ECC information. The number of blocks and pages vary with the size of the flash memory chip. NAND flash memory assume a block device interface. Currently used interfaces are Fibre Channel (FC), parallel SCSI (Small Computer System

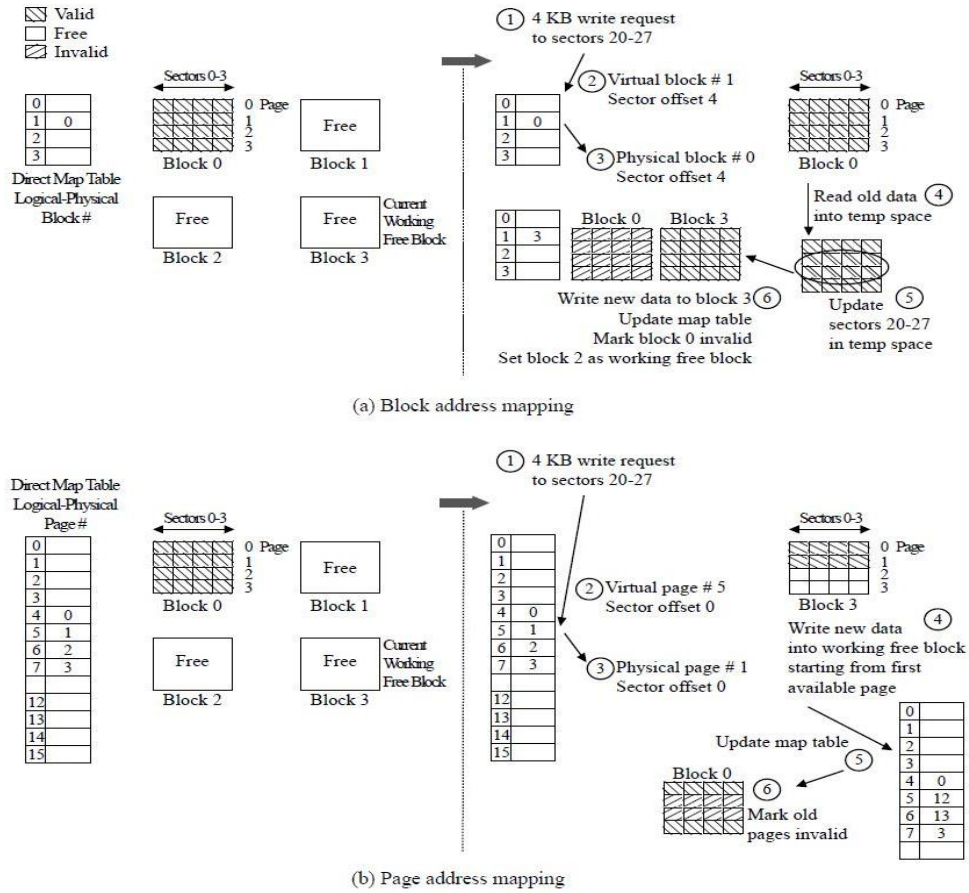


Figure 3.3. *An example of* Block and Page-based

Interface), parallel ATA (Advanced Technology Attachment), serial ATA (SATA), and serial attached SCSI (SAS). FTL layer converts requests' logical block address into physical page address in the flash memory and initiates read/write commands in the NAND interface layer. Address translation is one of the many activities of FTL later. Moreover, FTL also implements wear levelling algorithms. Wear levelling ensures that memory cells in an array are equally used - homogeneous

Characteristics	Large Block	Small Block
Block size	65536 bytes	16384 bytes
Page size	2048 bytes	512 bytes
OOB size	64 bytes	16 bytes
Read Page	25 µs	36 µs
Read OOB	25 µs	10 µs

Table 1. NAND flash specification based on Read and Write command accessing time

distribution of erase cycles. Mapping techniques are implemented at FTL layer and much more complicated than logical to physical address mapping in conventional hard disk drives. *Block mapping, page mapping, virtual-physical address mapping, LBA-PBA mapping* are all commonly used terms to address these sophisticated mapping algorithms and data structures. Figure 3.3 illustrates an example for block-based mapping and page-based mapping procedure. Most of the typical address mapping algorithms use two map tables. *A direct map table* will provide the physical location of data using its logical address. *An inverse map table* will store with each block of data its logical index and is used to generate the direct map table. In our case we are claiming to direct map addressing from NAND flash memory and only focusing on read operation.

Problem Description: We are given a set of *n* periodic tasks $\Gamma = \{\tau_1, \tau_2, \tau_3, \dots, \tau_n\}$. Each task τ_i is characterized by 4 tuples: $\tau_i = \left(C_i, T_i, D_i, Prob_i^{th}\right)$, where C_i

denotes the program that the task τ_i execute at every period. T_i denotes the period of τ_i, D_i denotes he relative deadline of τ_i. The j^{th} execution of τ_i is called the j^{th} job of $\tau_i$and denoted by J_{th}. Each job should be completed before its deadline with a probability higher than a threshold $Prob_i^{th}$. The given set of *n* periodic tasks are scheduled by a fixed priority scheduling algorithm, where τ_i has a higher priority than τ_i, if $i < j$.

The reason of why we are splitting the RAM size between Pinning and LRU is, if the given *n* tasks will share entire RAM, in that case conflict between tasks to access the RAM would be unavoidable. Therefore, partitioning approach will assign to each task to the dedicated partition of RAM size.

- S_i – the allocated entire RAM size of τ_i,
- S_i^{pinnig} – pinned RAM pages,
- $S_i^{LRU} = (S_i - S_i^{pinnig})$ – remaining RAM pages for demand-paging the non-pinned pages of C_i based on LRU, and
- $Read\ threashold\ \boldsymbol{TH}$ – the maximum number of each NAND page read operations that can guarantee the stored data is stable. If a block whole cumulated number of read operations count exceeds ***TH***, then block should be copied to other empty block.

Here, we are aiming to minimize the total RAM size, while satisfying the probabilistic deadline requirements $Prob_i^{th}$ for every task $\tau \in \{\tau_1, \tau_2, \tau_3, \dots, \tau_n\}$, with the rewrite delay,

Minimize $\sum_{i=1}^{n} Si,$

Subject to $dmp_i \geq Prob_i^{th}$ for all $\tau \in \Gamma$

where dmp_i is deadline meet probability of τ_i.

Under this task set and probabilistic temporal constraints, our problem is how we can minimize $\sum_{i=1}^{n} Si$ and guaranteeing the deadline meet probability of all n tasks, and at the same time eliminate *read disturbs* on NAND flash memory, which is happening where reading data from a flash page cause the threshold voltages of other unread pages in the same block to shift to higher value. While a single threshold voltage shift is small, such shifts can accumulate time, eventually becoming large enough to alter the state of some cells and hence generate *read disturb errors*, while we also have guaranteeing the deadline meet probability [18] of all *n* tasks.

4 OUR PROPOSED METHOD

In our proposal method, we have to use a heuristic algorithm to find a near optimal solution instead of searching enormous solution. This method has been configured in two steps in order to execute real-time tasks stored in NAND with minimal usage of a RAM and provide reliability while program execution.

4.1 PER-TASK ANALYSIS

In the first step called, per-task analysis it analysis each task one by one to find the optimal relation between allocated RAM size versus execution time distribution, and accumulate a block read count corresponding to the number of times any pages in the block have been read. For simplicity, from now on, we will explain each step with corresponding example. With the given executable file C_i for task τ_i, we preload all program codes into RAM from NAND and run them with sample input data stream until a large enough number of jobs L. From first run kernel-level profiler gives us execution time et_{ij} of each J_{ij} jobs. After running C_i one more time with same sample stream we will have our *page reference sequence* donated by $\phi_i = (P_i^1, P_i^2, \dots, P_i^{Mi})$, where P_{ij}^k is the k^{th} referenced page and M_{ij} is the total number of page references during execution of J_{ij} jobs. This two runs gives us execution times and page reference sequence for the L sample jobs of τ_i described as follows:

$$J_{ij} : et_{ij}, \phi_{ij} = \left(P_{ij}^1, P_{ij}^2, \dots, P_{ij}^{M_{ij}}\right), 1 \leq j \leq L \quad (1)$$

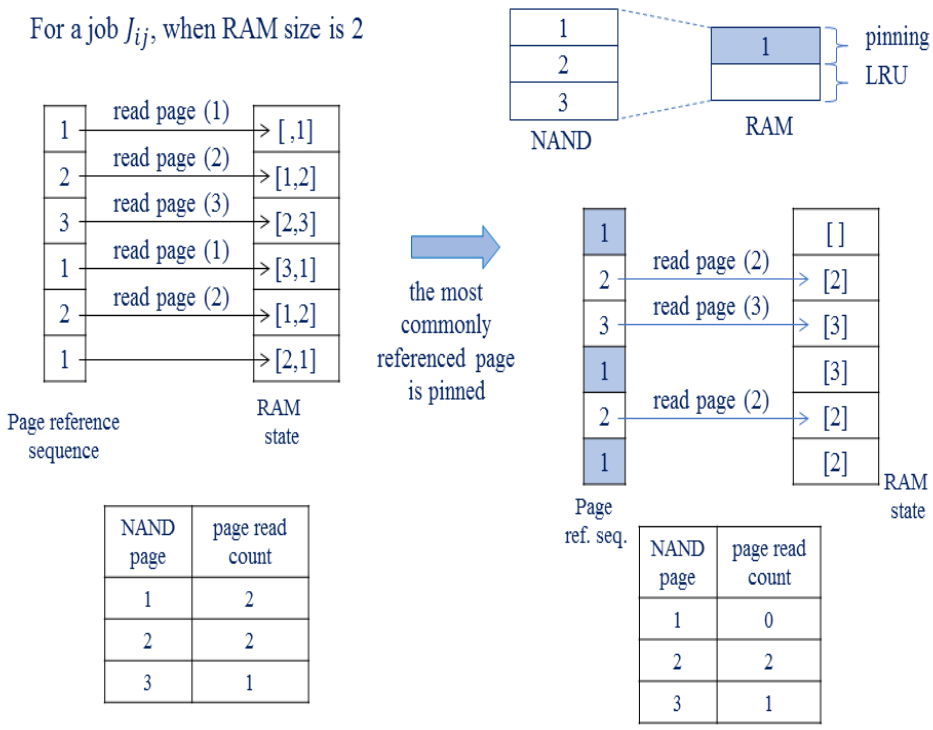
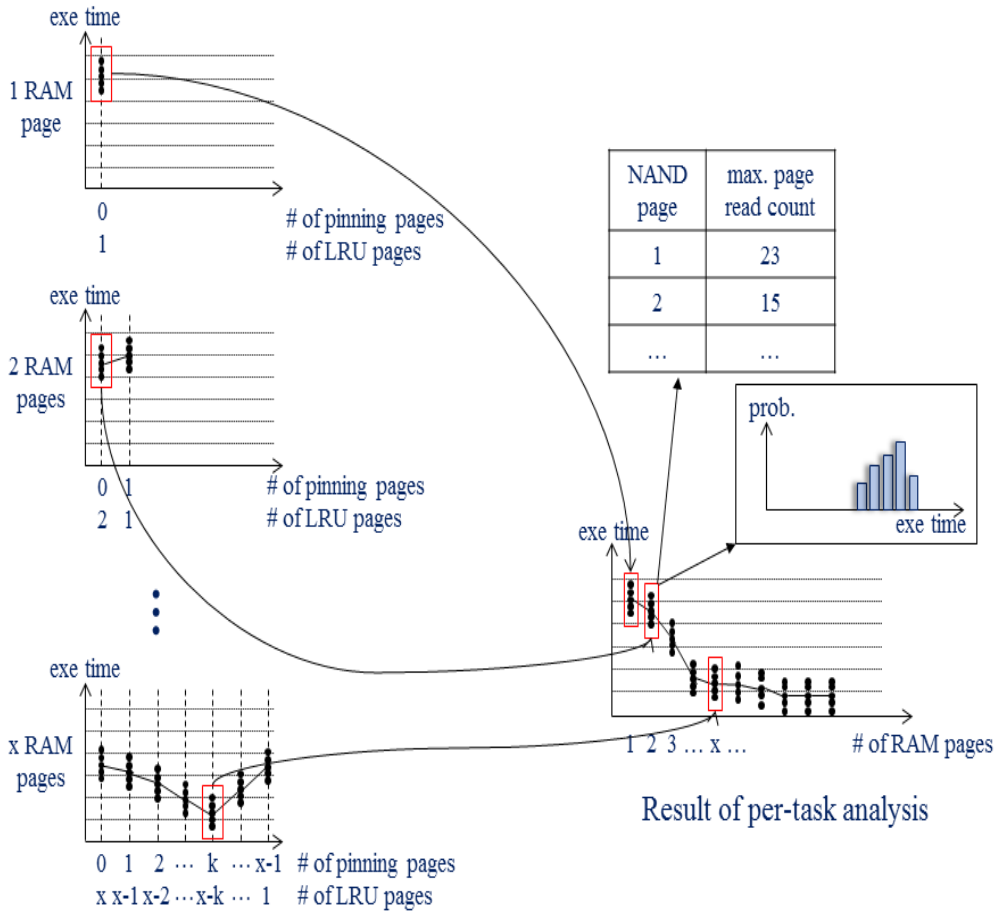


Figure 4.1.1. Example for finding optimal pinning/LRU combination with page read counts

With this trace information now we can investigate varying of given RAM size for accumulated each page's read count corresponding to the number of times any pages in the block have been read, as shown in Figure 4.1, and ergo the optimal partitioning between LRU and Pinning in terms of estimating the average execution time of all L jobs.

When the total RAM size $S_i = x$, there are k -pinning and $(x - k)$ -LRU frames. Therefore, we can estimate the execution times of all L sample jobs as follows:



$$exe\ time\ of\ J_{ij} = et_{ij} + F_{ij} \times T_{fault}, 1 \leq j \leq L \quad (2)$$

Figure 4.1.2. Per-task analysis

where T_{fault} is the time overhead for handling a single page fault as it has been depicted in Table 1.

Averaging those execution times of L sample jobs, we can have average execution time and max read counts of pages, and considering all the LRU/Pinning configuration, we can find the optimal combination for minimization of avg. execution time of τ_i and keep the execution time distribution also. In Figure 4.2.

illustrates the per task analysis, in the right side of graphic the shows the collection of optimal points (marked in dashed boxes for each combination) for total RAM size range values, that illustrates the function for RAM partitioning, optimal average execution time and max. page read count of L sample jobs. With computed functions donated by $avgE_i(S_i)$we can step to the following *stochastic-in-loop optimization.*

4.2 STOCHASTIC ANALYSIS FOR PROBABILISTIC SCHEDULABILITY

In the second step called, stochastic analysis in loop optimization, using $avgE_i(S_i)$ functions from per-task analysis as a input for all tasks, conducts a frequentative convex optimization process, which is fundamentally based on [31, 32]. The optimization process tries to satisfy the, $dmp_i \geq Prob_i^{th}$ for all $\tau \in \Gamma$ condition, to allocate minimum possible RAM to the tasks. The deadline meet probability dmp_i not only effected by execution time of τ_i but also execution time of high priority tasks that running at the same with the $avgE_i(S_i)$ functions, to meet all the $Prob_i^{th}$ requirements as quickly as possible, it increase the total RAM size (S_i) as small as possible. But instead of traditional resource budget we are using a stochastic analysis based on [9, 18] to probabilistically guarantee tasks scheduling deadlines with the delay for block rewrites, which is happing for the block read count once it reaches a predetermined number, Figure 6 illustrated an example how

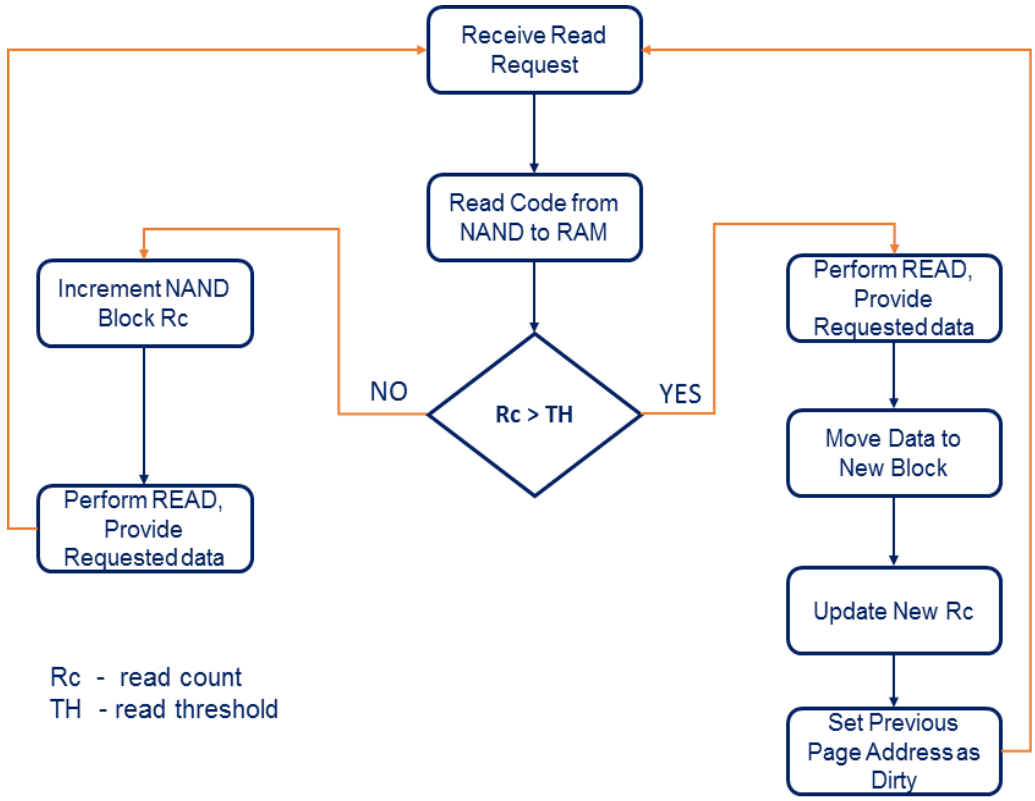


Figure 4.2.1. Workflow of moving data to the new block

and when rewrites required for to avoid read disturb.

For the stochastic analysis, we are using execution time distribution $\{\varepsilon_1, \varepsilon_2, \varepsilon_3, \dots, \varepsilon_n\}$, from per task analysis as an input for all tasks, considering all preemption scenarios with probabilistic convolution, and read count delay we are computing response time distribution $\{R_1, R_2, R_3, \dots, R_n\}$ as an output. From computed R_i we can calculate dmp_i of τ_i by summing the probabilities, where $R_i \leq D_i$. Regarding to determining the dmp_i, system's average utilization function

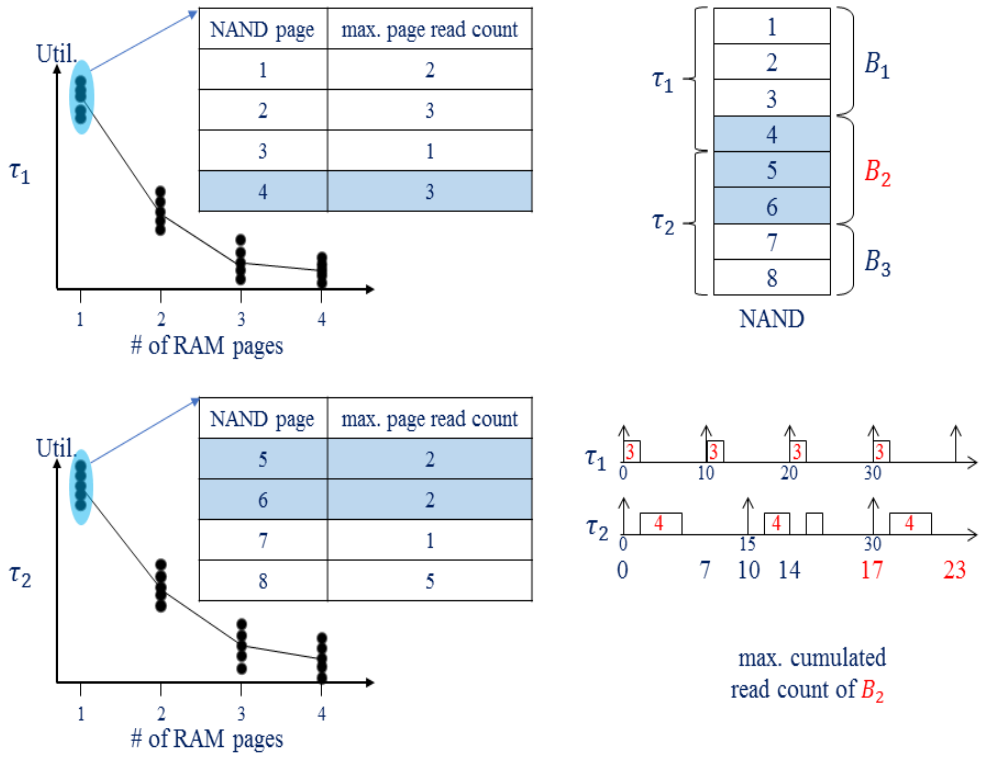


Figure 4.2.2. Example for finding a period of the block rewriting task for B_2

is playing an important role. Hence, if average utilization function $avgU_i(S_i)$ is getting smaller, then dmp_i's value is getting higher, means, it's getting easy to guarantee all the tasks deadline meet, of course we have to make sure that total RAM size is still small enough. For this, first transform $avgE_i(S_i)$ function of each task, to get the average utilization function $avgU_i(S_i)$ divide the $avgE(S_i)$ function by the period T_i:

$$avgU_i(S_i) = \frac{avgE_i(S_i)}{T_i} \quad for\ all\ \tau \in \Gamma \qquad (3)$$

From $avgU_i(S_i)$, we are constructs a convex approximated function: $\overline{avgU_\iota(S_\iota)}$ which is consists of only convex-hull frontiers:

$$\overline{avgU_\iota(S_\iota)} = \left\langle \begin{pmatrix} S_i^1 \\ avgU_i^1 \end{pmatrix}, \begin{pmatrix} S_i^2 \\ avgU_i^2 \end{pmatrix}, \cdots, \begin{pmatrix} S_i^{L_i} \\ avgU_i^{L_i} \end{pmatrix} \right\rangle \quad (4)$$

With this convex approximated function $\overline{avgU_\iota(S_\iota)}$ for every task for all $\tau \in \Gamma$ our stochastic analysis conducts the convex optimization. In this optimization, we perform the stochastic analysis [9] to guarantee that the every task can meet deadline meet probability threshold $Prob_i^{th}$. For this it requires to repeatedly increase S_i until all the tasks can satisfy with their $Prob_i^{th}$. For the complete and formal description on this optimization process, readers are referred to [7].

During the convex optimization, for each RAM assignment, to prevent read disturb errors, we add tasks for block rewriting. The execution times of these tasks are same as the time for writing a block. For each rewriting task, we have to find a period. For example, if there are two tasks τ_1 and τ_2 with given T_1=10, T_2=15, and TH= 20, the period of the block rewriting task for the block B_2 is going to be 30 as shown in Figure 4.2.2. Then, when performing the convex optimization, we also consider block rewriting tasks as highest priority tasks.

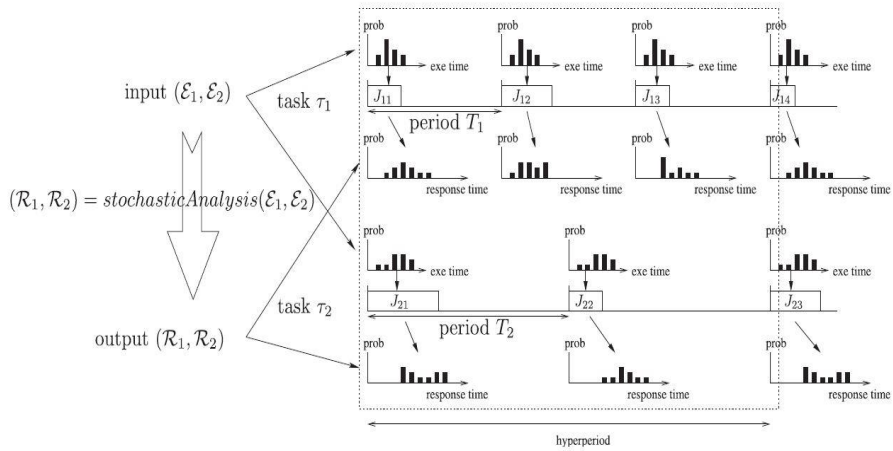


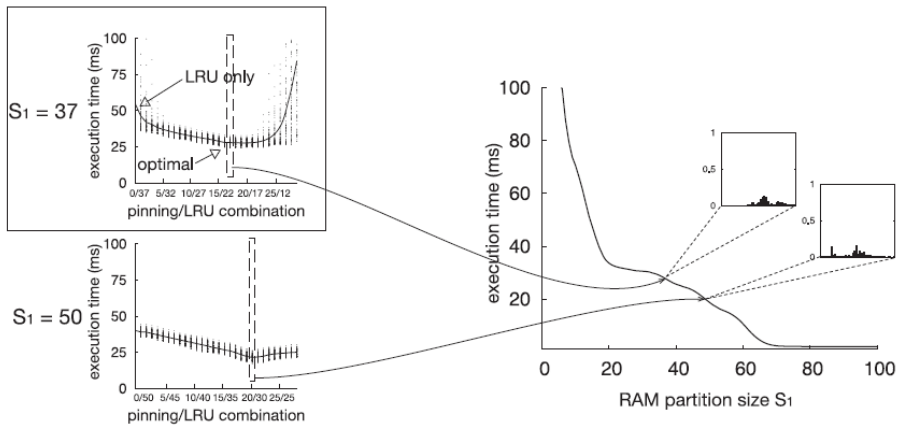
Figure 4.2.3. Stochastic analysis of response times distribution, Adapted from [7]

For more, and detailed optimization process description, for readers referred to check [7, 9, 18].

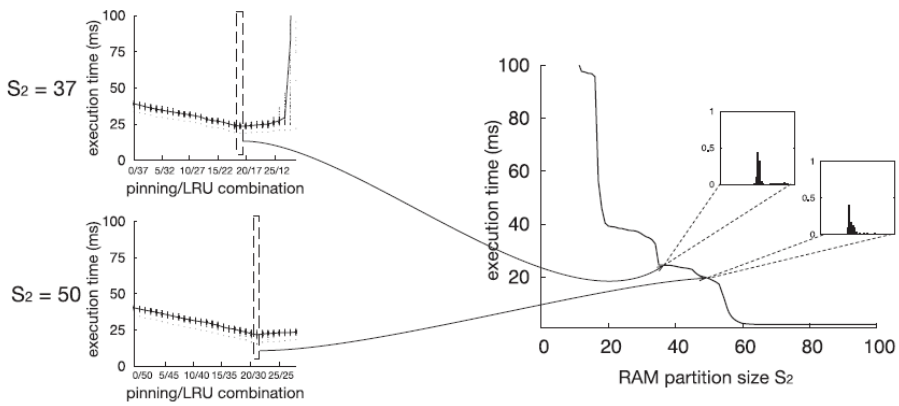
5 EXPERIMENTS

This section justifies the propped approach through both trace-driven simulation and actual implementation. As a prototype system we also used same device on previous studies experiments, which is for learning phase was Samsung Q1UMPC equipped with Intel A110 800Mhz processor, 64 GB NAND flash memory and 1 GB RAM. Top of this prototype system has been implemented with kernel-level profiler by modifying the Linux kernel 16.04. As a typical multi-tasking soft real-time application we are considering a mobile video decoder and encoder task should run concurrently. For the video decoder and encoder program we use h.264 decoder/encoder made by ITU-T Joint Video Team. When the two tasks concurrently running, they are scheduled by the fixed-priority scheduling with the decoder task having a higher priority. The deadlines are assumed to be the same as periods.

We use the page size from Linux's perspective which is 4 KB, from now on by page we mean a 4KB page. Which is two consecutive NAND page which is consist of max. page read count, by block we mean all tasks NAND pages with high read count collected in blocks. For the page-fault handling overhead T_{fault} we actually us measure time 60ms, which is adopted from [11], it also includes additional delay except physical delay for prevent read disturb errors on NAND flash memory while read operation. One rule of thumb regarding the deadline meet probability



(a) decoder case



(b) encoder case

Figure 5.1.1. Results for per-task analysis

threshold is 95% for both decoder and encoder tasks to ensure that human cannot perceive any quality distortion for the video playback. Thus, for $Prob_i^{th}$ 95% is accepted as a default.

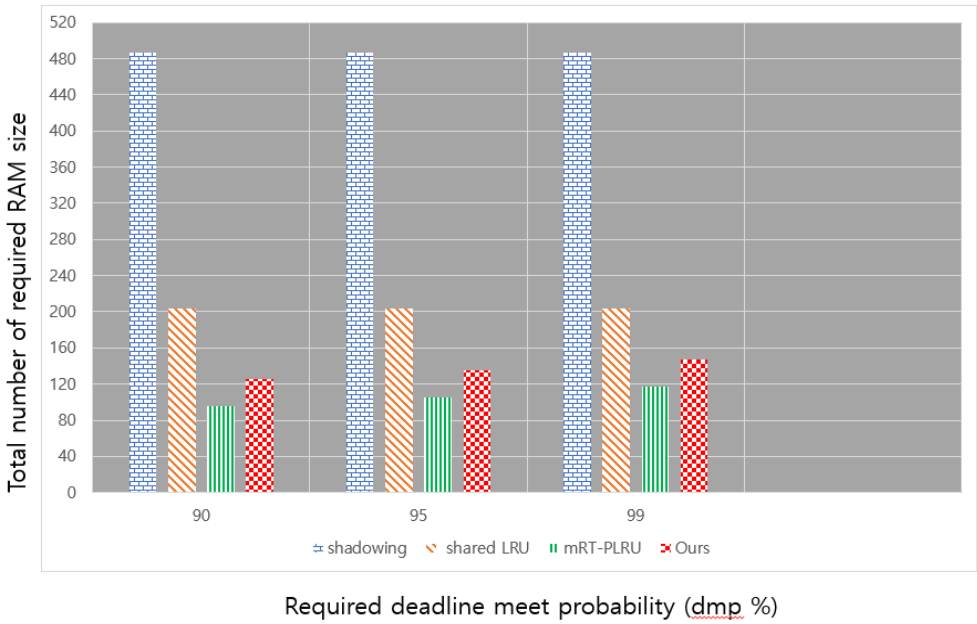


Figure 5.1.2. Comparison of required RAM sizes

In our learning phase, all our sample content consists of 1000 video frames. Figure 5.1.1. describes the result of our per-task analysis for two tasks, which is, decoder and encoder.

The results for total RAM size comparison with other approaches are illustrated in Figure 5.1.2. In the figure 5.1.2. required total RAM sizes has been compared as follow:

- **Shadowing:** It is traditional approach to preload all the program code pages into RAM and execute them from there, which is occupies too much RAM size.

- **Shared LRU:** It represent the default RAM usage provided by Linux OS, where all tasks share the entire RAM space using LRU-based demand paging. Due to unpredictable inter conflicts, it is hard to determine the required RAM size for probabilistic deadline guarantee. Thus there have been used only conversation's estimation.
- **mRT-PLRU:** It is proven that this method applying pinning and LRU combination, allocates RAM size minimum, however in terms of read disturb prevention, this method shows less reliableness.
- Ours: Despite is requiring RAM size more than original mRT-PLRU method, but we got to be able guaranteeing the reliability during read operation, and in terms of general comparison our approach shows best results, where it is requiring less that shadowing and shared LRU.

6 CONCLUSION

This thesis presents a new method for improvement of reliability problems on *mRT-PLRU* (Multitasking Real-Time constrained combination of Pinning and LRU), which is a generic framework to use NAND flash memory for real-time program executions with the minimal usage of RAM. This method consists of two steps. In the first step called, per-task analysis, each task one by one is analyzed, to find the function of the optimal relation between allocated RAM size versus execution time distribution, and accumulate a block read count. In the second step called, stochastic analysis, we use functions from per-task analysis results as an input for all tasks, conduct an iterative convex optimization with stochastic analysis to allocate minimum RAM to the tasks such that their deadlines are probabilistically guaranteed with the considered delay for block rewrites, to avoid read disturb errors.

Future work. For further work, we will examine the aforementioned method through experimentation due to consider varying read time, which depends on the number of bit flips as well.

요약 (국문초록)

NAND 플래시 메모리 상에서 수행되는 연성 실시간 멀티태스킹 프로그램의 신뢰성 보장 기법

플래시 저장장치 기술에 기반한 SSD 가 데이터센터를 위한 차기 스토리지 기술로서 각광받고 있다. NAND 플래시 기술에 기반한 SSD 는 데이터를 저장하는 비휘발성 기술이다. 그러나 NAND 플래시 메모리는 페이지 기반 read 만을 지원한다는 점에서 코드 실행에 걸림돌이 되는 문제들이 남아 있다. 최근에 개발된 mRT-PLRU(Multitasking Real-Time constrained combination of Pinning and LRU)기술은 저렴한 비휘발성 NAND 플래시 메모리를 멀티태스킹 환경에서 실시간 프로그램을 실행하고 저장하는 데 사용할 수 있도록 하는 프레임워크이다. 이 기술은 NAND 플래시 메모리에 저장된 여러 실시간 태스크들을 RAM 을 최소한도로 사용하면서 실행할 수 있도록 해 준다. mRT-PLRU 는 복수 개의 실시간 태스크 수행 시 데드라인의 만족 가능성이 확률적으로 보장되면서 최소한도의 RAM 만 사용하도록, 필요한 최적 RAM 크기를 결정한다. 그러나 이 프레임워크에서 해결되지 못한 문제로, 신뢰성(reliability)을 저하시키는 NAND 플래시 메모리의 read disturb 문제가 있다. 본 논문에서는 이 문제를 해결하여 NAND 플래시 메모리에서 신뢰성을 보장할 수 있는 새로운 방법을 제시한다. Read disturb 에러를 피하기 위해서는 NAND 페이지를 rewrite 하는 오버헤드를 고려하며 rewrite 을 해야 한다. 본 논문에서 제안하는 방법은 read 가 잦은 페이지의 경우 reserved time slots 를 이용하여 read 횟수가 일정 문턱값을 넘기 전에 rewrite 되도록 하는 것이다. 모든 태스크들이 스케줄 가능한지 체크하기 위한 방법으로, exact stochastic analysis 를 사용하여 데드라인의 만족 가능성이 확률적으로 보장되는지를 확인한다.

주요어: 연성 실시간 프로그램, NAND 플래시 메모리, 신뢰성, 멀티태스킹, mRT-PLRU

학번: 2015-22147

References

[1]. Toshiba America Electronic Components, Cost Savings with NAND Shadowing Reference Design with Motorola MPC8260 and Toshiba CompactFlash, 2002.

[2]. Jiang Xiao-bo, Tan Xue-qing, Huang Wei-pei, *"Novel Reliability Evaluation Method for NAND Flash Memory"* in IEEE TENCON 2015.

[3]. Shu Li, Tong Zang, *"Improving Multi-Level NAND Flash Memory Storage Reliability Using Concatenated BCH-TCM Coding"* in IEEE Transactions on very large scale integration(VLSI) systems, Vol.18, 2010.

[4]. Zhang Juntao, Yao Kun, *"Research on Reliability Improvement of NAND Flash Memory in Fat File System"* in 2010 International Conference on Biomedical Engineering and Computer Science, 2010.

[5]. Wang Kang, Youguang Zhang, Mingbang Wang, Guoyan Li, *"Improving Flash Memory Reliability with Dynamic Thresholds: signal Processing and Coding Schemes"* in 7th International ICST Conference on Communications and Networking, CHINACOM, 2012.

[6]. Jong-Chan Kim, Duhee Lee, Chang-Gun Lee, Kanghee Kim, *"RT-PLRU: A New Paging Scheme for Real-Time Execution of Program Codes on NAND Flash Memory for Portable Media Players"* in IEEE Transactions on Computers, Vol.60, 2011.

[7]. Duhee Lee, Jong-Chan Kim, Chang-Gun Lee, Kanghee Kim, *"mRT-PLRU: A General Framework for Real-Time Multitask Execution on NAND Flash Memory"* in IEEE Transactions on Computers, Vol.62, 2013.

[8]. Kyoung-Soo We, Chang-gun Lee, Kyongsu-yi, Kwei-Jay Lin, Yun Sang Lee, "*HRT-PLRU: A New Paging Scheme for Executing Hard Real-Time Programs on NAND Flash Memory*" in IEEE Transactions on Computers, Vol.63, 2014.

[9]. K. Kim, J.L. Diaz, L.L. Bello, J.M. Lopez, C.-G. Lee, and S.L. Min, "*An Exact Stochastic Analysis of Priority-Driven Periodic Real-Time Systems and Its Approximations*," IEEE Trans. Computers, vol. 54, no. 11, pp. 1460-1466, Nov. 2005.

[10]. C. Park, J. Lim, K. Kwon, J. Lee, and S.L. Min, "*Compiler-Assisted Demand Paging for Embedded Systems with Flash Memory*," Proc. Fourth ACM Int'l Conf. Embedded Software, pp. 114-124, Sept. 2004.

[11]. Y. Cai, Y. Luo, S. Ghose, E. F. Haratsch, K. Mai, O. Mutlu, "*Read Disturb Errors in MLC NAND Flash Memory: Characterization, Mitigation, and Recovery"* Dependable Systems and Networks (DSN), 2015 45th Annual IEEE/IFIP International Conference on, pp. 438-449, 2015.

[12]. S. Baek, J. Choi, D. Lee, and S. H. Noh. "*Performance Characteristics of Flash Memory: Model and Implications."* In *Proc. 3rd International Conference on Embedded Software and Systems*, Daegu, Korea, pp. 162-173, 2007.

[13]. S. Baek, S. Ahn, J. Choi, D. Lee, and S. H. Noh. "*Uniformity Improving Page Allocation for Flash Memory File Systems*." In *Proc. 7th ACM & IEEE International Conference On Embedded Software*, pp. 154-163, 2007.

[14]. E. Gal and S. Toledo. "*Algorithms and Data Structures for Flash Memories. ACM Computing Surveys"*, 37(2):138–163, June 2005.

[15]. B. Schroeder, R. Lagisetty, A. Merchant "*Flash Reliability in Production: The Expected and The Unexpected",* in 14th USENIX Conference on File and Storage Technologies (FAST '16), pp.67-80, 2016.

[16]. D. Yan Wu, S. Fan Chen, C. Lin, Y. King, "*Dummy Reads Scheme for Lifetime Improvement of MLC NAND Flash Memories",* in IEEE Transactions on Device and Materials Reliability, Vol, 16, No.4., Dec. 2016.

[17]. M. Breeuwsma, M. jongh, C. Klaver, R. Knijff, M. Roeloffs, "*Forensic Data Recovery from Flash Memory",* in Small Scale Digital Device Forensics Journal, Vol.1, June, 2007.

[18]. J.L. Dı´az, D.F. Garcı´a, K. Kim, C.-G. Lee, L. LoBello, J.M. Lo´pez, S.L. Min, and O. Mirabella, "Stochastic Analysis of Periodic Real-Time Systems," Proc. 23rd Real-Time Systems Symp., pp. 289-300, 2002.

[19]. J.-C. Kim, D. Lee, C.-G. Lee, K. Kim, and E.-Y. Ha, "*Real-Time Program Execution on NAND Flash Memory for Portable Media Players,"* Proc. IEEE 29th Real-Time Systems Symp., pp. 244-255, Nov. 2008.

[20]. D. Lee, C.-G. Lee, and K. Kim, "*A Generic Framework for Soft Real-Time Program Executions on NAND Flash Memory in Multi-Tasking Embedded Systems,*" Proc. 30th IEEE Real-Time Systems Symp, pp. 93-104, Dec. 2009.

[21]. D.B. Kirk, "*SMART (Strategic Memory Allocation for Real-Time) Cache Design,*" Proc. IEEE 10th Real-Time Systems Symp., pp. 229-237, Dec. 1989.

[22]. B.D. Bui, M. Caccamo, L. Sha, and J. Martinez, "*Impact of Cache Partitioning on Multi-Tasking Real-Time Embedded Systems*," Proc. IEEE 14th Int'l Conf. Embedded and Real-Time Computing Systems and Applications, pp. 101-110, 2008.

[23]. A. Wolfe, "*Software-Based Cache Partitioning for Real-time Applications*," Proc. Third Int'l Workshop Responsive Computer Systems, Sept. 1993.

[24]. Parthey, D.: *Analyzing real-time behavior of flash memories.* Diploma Thesis, Chemnitz University of Technology (April 2007).

[25]. Z.Qin, Y.Wang, D. Liu, Z.Shao "*Real-Time Flash Translation Layer for NAND Flash Memory Storage Systems,*" in IEEE 18th Real Time and Embedded Technology and Applications Symposium,pp.35-44,2012.

[26]. Y. Wang, Z. Qin, R. Chen, Z. Shao, Q. Wang, S. Li, and Laurence T. Yang "*A Real-Time Flash Translation Layer for NAND Flash Memory Storage*

Systems," in IEEE Transactions on Multi-Scale Computing Systems, vol. 2, no. 1, Jan.-Mar. pp.17-29, 2016.

[27]. S.Choudhuri and T. Givargis "Real-Time Access Guarantees for NAND Flash Using Partial Block Cleaning," in SEUS 2008, LNCS 5287, pp. 138–149, IFIP International Federation for Information Processing 2008.

[28]. Li-Pin Chang and Tei-Wei Kuo "*A Real-Time Garbage Collection Mechanism for Flash-Memory Storage Systems in Embedded Systems*," in ACM Transactions on Embedded Computing Systems (TECS), Vol. 3, pp. 837-863 Nov.2004.

[29]. J.E. Brewer and M. Gill, editors. Nonvolatile Memory Technologies with Emphasis on Flash. IEEE Press, 2008.

[30]. W. Hsu and A. J. Smith. "Characteristics of I/O Traffic in Personal Computer and Server Workloads." *IBM Systems Journal*, vol. 2, no. 2, pp. 347-372, April 2003.

[31]. C. Lee, J. Lehoczky, R. Rajkumar, and D. Siewiorek, "On Quality of Service Optimization with Discrete QoS Options," Proc. IEEE Real-Time Technology and Applications Symp., June 1998.

[32]. R. Rajkumar, C. Lee, J. Lehoczky, and D. Siewiorek, "Practical Solutions for QoS-Based Resource Allocation Problems," Proc. 19th Real-Time Systems Symp., Dec. 1998.