An Adaptive Idle-Time Exploiting Method for Low Latency NAND Flash-Based Storage Devices

Sang-Hoon Park, Dong-gun Kim, Kwanhu Bang, Hyuk-Jun Lee,
Sungjoo Yoo, *Member*, *IEEE*, and Eui-Young Chung, *Member*, *IEEE*

Abstract—The market share of NAND flash-based storage devices (NFSDs) has rapidly grown in recent years since many characteristics, such as non-volatility, low latency, and high reliability, meet the requirements for various types of storage devices. However, the unique characteristic of NAND flash memories (NFMs), erase-before-write, causes problems for NFSDs from a performance perspective. Specifically, performance degradation is incurred by extra operations that serve to hide the bad characteristics of NFMs. In order to resolve this problem, many attractive methods have been proposed. Various algorithms for flash translation layers (FTLs) are representative methods that provide space redundancy to NFSDs for better performance. However, the amount of space redundancy is limited by the capacity of NFMs and thus, space redundancy is still insufficient for improving the performance of NFSDs. Consequently, a new type of redundancy, termed temporal redundancy, has recently been introduced for NFSDs. More precisely, the idleness of NFSDs is exploited so as to precede extra operations for NFSDs while minimizing the overhead of extra operations. In this paper, we propose an adaptive time-out method based on the Hidden-Markov Model (HMM) to efficiently utilize idle periods. In addition, we also suggest a simple scheduling scheme for extra operations that can be customized for general FTLs. The experimental results demonstrate that the proposed method yields performance improvements in terms of average write latency and peak latency, 74% and 76% better than the existing method, respectively, and approaching within average 9% and 5% of the optimal case, respectively.

Index Terms—Solid-state disk, NAND flash memory, idle-time

1 Introduction

As the annual amount of created, accessed, and copied information has been dramatically increasing [1], the need for dense and reliable memory has become a priority. Consequently, NAND flash memories (NFMs), one of the densest non-volatile memories, are widely used in a variety of applications and occupy a large portion of the entire memory market.

In many computing systems, NFMs are employed as secondary storage devices due to their several advantages over traditional magnetic disks. Since NFMs can be fully accessed electronically, elimination of all mechanical parts and seek the location of data is possible. Therefore, NAND flash-based storage devices (NFSDs) can have a smaller form factor, shorter response time, and higher shock resistance.

Two features of NFSDs are often considered to be the biggest obstacles that hinder the complete replacement of HDDs with NFSDs. The first obstacle is the higher cost-per-bit of NFSDs, which is expected to be solved in the near future due to the advanced process technology. The second obstacle is the erase-before-write feature, which is a unique characteristic of NFMs. NFMs do not intrinsically support in-place update. Every program command to an NFM requires a preceding erase command to the target location. Furthermore, the unit of an erase command is a block, which is a group of pages. As such, an update to a single page may cause every page containing valid data in the block to be copied to another block.

To hide this problem from the host system cooperating with NFSDs, many researchers have proposed an intermediate software layer, called a flash translation layer (FTL) and an FTL performs diverse extra operations, i.e. garbage collection, wear-leveling and etc. Numerous studies on FTLs have been devoted to reducing the cost of extra operations with various mapping granularities and mapping algorithms [2]–[8].

Most FTLs use over-provisioning blocks, also called log blocks or update blocks, to delay or reduce the cost of the extra operations of NFMs. FTLs forward updated data to the over-provisioning blocks in order to reduce extra operation overhead. In other words, existing FTLs utilize space redundancy to improve the performance of NFSDs.

In this work, we use temporal redundancy by exploiting the proper idle periods based on the adaptive time-out method for improving the latency of NFSDs. In fact, traditional HDDs have already exploited idle time for power reduction [9], [10]. Unfortunately, previous research on HDDs cannot be directly applied to NFSDs for two reasons.

First, in performance perspective, the power reduction attained for HDDs is a supplementary objective. Therefore,

• S.-H. Park, D. Kim, K. Bang, and E.-Y Chung are with the Department of Electrical and Electronic Engineering, Yonsei University, Seoul 120-749, Korea.
E-mail: {soskhong, digikim, khbang}@dtl.yonsei.ac.kr, eychung@yonsei.ac.kr.
• H.-J. Lee is with the Department of Computer Science and Engineering, Sogang University, Seoul 121-742, Korea. E-mail: hyukjunl@sogang.ac.kr.
• S. Yoo is with the Department of Electronics and Electrical Engineering, Pohang University of Science and Technology, Phohang 790-784, Korea.
E-mail: sungjoo.yoo@postech.ac.kr.

*Manuscript received 13 Jan. 2012; revised 29 Oct. 2012; accepted 06 Nov. 2012; published online 28 Nov. 2012; date of current version 29 Apr. 2014.
Recommended for acceptance by C.-W. Wu.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TC.2012.281*

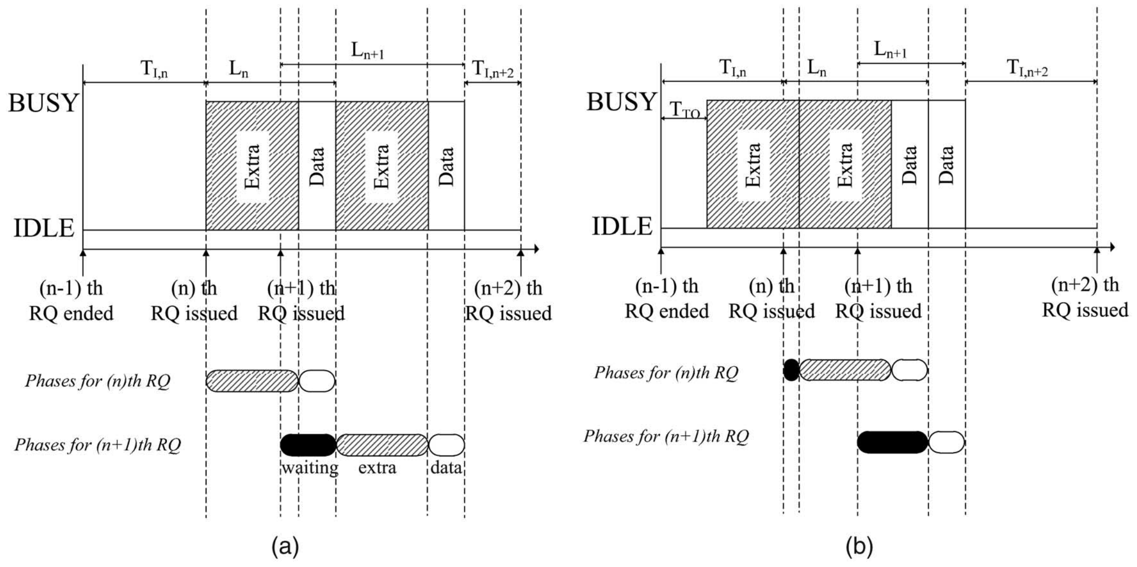


Fig. 1. Timing diagrams of an NFSD experiencing idle-time (a) if it does not exploits idle-time, and (b) if it exploits idle-time.

if a required performance level is defined, a degree of aggressiveness in exploiting idle-time can be found for HDDs. In contrast, NFSDs utilize idle periods to enhance their performance. Thus, operations performed during idle-time are not optional, but turn into essential jobs from the perspective of performance.

The second reason why HDD research cannot be applied to NFSDs pertains to the amount of performance penalties following the utilization of an idle period. The only penalty in HDDs is the wake-up penalty which usually has a fixed length. However, the extra operations of the FTL, which are performed by an NFSD during its idle-time, have a large variation in length. Therefore, more sophisticated methods are required to exploit the idle-time of NFSDs.

In this paper, we propose an idle-time exploiting algorithm for NFSDs. We categorize issues related to the utilization of idle-time in NFSDs into two categories: “when” and “which”. For “when,” the proposed algorithm determines useful idle periods. The proposed method effectively filters out useless or harmful idle-time[1] based on an adaptive method that uses the Hidden-Markov Model (HMM).

“Which” is a problem defined by selecting and scheduling extra operations to be executed within the chosen idle-time. “Which” strongly depends on the design objective to be accomplished by introducing temporal redundancy to NFSDs. Since our objective is an improvement in performance, we customized our method appropriately so as to achieve this goal.

The contribution of this work is two-fold. First, we propose an adaptive idle-time exploiting algorithm that can effectively select proper idle periods from various access patterns. Second, the algorithm contains a simple scheduling method that is proven to be the best method among our candidates. We also demonstrate the effectiveness of our algorithm in a quantitative manner by integrating our algorithm into a representative hybrid-mapping FTL.

1. Useless or harmful idle time is too short to be utilized for extra operations.

2 Backgrounds

2.1 Usefulness of Exploiting Idle-Time

We will first define the notations used in our work. Two timing diagrams representing the behavior of NFSDs with idle-time are shown in Fig. 1; $T_{I,n}$ represents the length of an idle period from the moment the service of the $(n-1)$th request is completed to the moment the next request is issued. This idle period can be utilized by NFSDs as their temporal redundancy for performance improvements.

The L_n term in Fig. 1 denotes the latency required to notify the host system that the (n)th request has been completed. The latency consists of a waiting phase, an extra phase, and a data phase, which are depicted in Fig. 1 as rounded black rectangles, rounded rectangles with diagonal lines, and rounded-white rectangles, respectively. The length of a waiting phase is measured as the time period over which a request is residing in the request queue of the NFSD. In other words, a request waits for its service in the request queue during its waiting phase until the NFSD finishes all NFM commands, data transfer, and other jobs for the previous request. This phase is skipped if the NFSD is idle when the request is issued as in the case of (n)th request in Fig. 1(a).

Extra and data phases consist of NFM commands. A data phase of a request is the time period required to access and transfer requested data from/to an NFM. Therefore, if a request is given, the length of its data phase can be simply computed according to the specification of the NFM and the length of the data given by the host system. However, an extra phase of the (n)th request, $T_e(n)$, has a variable length since the status of an NFM determines the amount of NFM commands. The extra phase may include erase commands to reclaim used blocks or copy-back commands to move valid data from one page to another page. The extra phase can be omitted when there are enough free pages to receive incoming data or the request is a read request. Otherwise, the extra phase must precede the data phase, to reclaim pages and blocks following the algorithm of the FTL.

The effect of exploiting idle periods is shown in Fig. 1(b). Unlike the case of Fig. 1(a), $T_{I,n}$ is utilized for extra commands for the $(n+1)$th request. For the case of the $(n+1)$th request in Fig. 1(b), no extra phase is required since free space is reclaimed during the previous idle-time.

The T_{TO} term indicates the time-out threshold value, which means that, since an idle period has begun, the execution of extra commands is initiated if there is no request until T_{TO}. Unfortunately, due to the longer execution time of extra commands when compared to $T_{I,n} - T_{TO}$, the (n)th request experiences a waiting phase that results in longer latency than that of the (n)th request in Fig. 1(a). However, the latency of the $(n+1)$th request depicted in Fig. 1(b) is reduced when compared to that of Fig. 1(a) due to the previously executed extra commands. Finally, even though the latency of the (n)th request increases slightly, since the latency of the $(n+1)$th request is significantly shortened, the average latency of the two requests is reduced.

It should be noted that, throughout this paper, we assume that the host system can be notified upon the completion of service for one request only when all the required phases are completed. This assumption is essential in order to guarantee data consistency between an NFSD equipped with its volatile cache and its host system in the event of sudden power outage.

2.2 Flash Translation Layer

FTLs can be classified by their mapping granularity, since this characteristic is one of the key factors that decide both the overall performance of FTLs and the size of the mapping table. Most FTLs are based on a block-level mapping table and utilize a small number of over-provisioning blocks that are managed in page-level. Such FTLs are called hybrid-mapped FTLs [4]–[7].

BAST [4], which is our target FTL, is a representative hybrid-mapped FTL. Like other hybrid-mapped FTLs, it is based on a block-level mapping table for the data blocks and utilizes a page-level mapping table to manage over-provisioning blocks (log blocks). In BAST, one over-provisioning block is related to only one data block.

Due to the restriction on the mapping table size and the capacity of NFSDs, the ratio of over-provisioning blocks to total blocks is usually limited. In addition, if a log block is full and needs to be written, it must be merged with its corresponding data block and reclaimed as a new log block. To complete this task, BAST provides two different types of merge operations, which are full merge operations and switch merge operations. Even though they require different conditions to be induced and incur different sequences of NFM commands, both of them include one or more erase commands which affect performance severely due to long execution time. Furthermore, in case of full merge operations, they usually include a large number of page copy commands which also degrade performance.

Even if page-mapped FTLs [2], [3], [11], which are expected to outperform hybrid-mapped FTLs, are employed in NFSDs, they cannot avoid extra operations caused by the characteristics of NFMs, either. Therefore, idle-time exploitation, which can hide the latency of extra operations from end-users, can further improve the performance of NFSDs with any types of FTLs.

3 Related Work

While the notion of NFSDs utilizing idle-time is not new, there have only been a few studies conducted on the subject. In [12], Seong et al. proposed a hardware architecture that NFSDs manage every write request from host in the background during idle-time. In other words, data from the host is written to faster volatile memories such as DRAM, while the moving of data to NFMs is performed in background. To maintain read performance, the hardware architecture is equipped with a foreground operation manager that pauses background operations to service read requests. However, with the problems like data consistency caused by the volatility, this architecture is not optimized to exploit idle periods. In terms of “when,” instead of determining efficiency of an idle period, it just executes commands of NFMs whenever possible. Therefore, it is a greedy time-out method with zero time-out threshold value. Also, they did not consider the scheduling of extra operations in the perspective of “;which”;. Furthermore, future requests hardly benefit by the exploitation of idle-time, since it executes operations only required by the current request.

On the other hand, FlexFS, in [13], includes a file system for MLC NFMs. One unique characteristic of MLC NFMs is that some of their portions can be used as SLC-like pages which are much faster than normal MLC pages. FlexFS uses this characteristic to employ a part of NFMs as fast as SLC NFMs. However, data in faster pages should be eventually moved to slower pages in a process called migration, since the number of SLC-like pages is limited. Therefore, the target task of “which” in FlexFS is the migration. In addition, to schedule migrations, FlexFS adaptively adjusts their execution time by adjusting the amount of migrated data. It is done by predicting the actual length of the future idle-time via a weighted moving average. The purpose of the adjustment is to avoid performance penalties incurred by migration during idle-time.

As an answer for “when,” FlexFS filters out useless idle-time with a simple time-out policy with a fixed T_{TO}. One unique aspect of FlexFS’s “when” is the triggering interval, which serves to reduce performance penalties incurred by the execution time of extra commands during an idle period, like in the case of the waiting phase of the (n)th request in Fig. 1(b). FlexFS triggers the execution of commands during idle-time according to the triggering interval.

While FlexFS improves the performance of NFSDs with MLC NFMs, it cannot be generally applicable to NFSDs due to two major problems. First, the future idle-time cannot be accurately predicted by a weighted moving average due to the non-stationarity of storage access patterns. Furthermore, the work amount for most of extra operations cannot be controlled as in migration. In migration, its work amount is highly controllable since a migration consists of page copying whose execution time is constant. In contrast, some other operations, whose work amount cannot be quantized, should be completed during idle periods. For instance, a merge operation cannot be quantized, since the whole valid pages in a target block should be copied to a new block. The second major problem in FlexFS is that it is conservative in utilizing the idle period. In detail, for a given idle period, it uses a long time-out threshold and schedules the migration periodically.

Therefore, the utilization of idle-time may be reduced if the scheduling interval is not appropriately set.

Real-time systems with NFSDs also exploit idle periods for garbage collections [14], [15]. In these systems, the deadline of each transaction is known. Therefore, the length of an idle period can be given precisely and the possibility of the latency penalty caused by executing tasks during idle-time is very small. Our target system, however, does not have the deterministic deadline. Therefore, an accurate prediction scheme to estimate the length of idle periods should be adopted to reduce the probability, which makes our problem more complicated.

4 Preliminaries

4.1 Characteristics of Idle Periods

It is important to understand the characteristics of idle periods given to NFSDs in this work. Besides the fact that the patterns of idle periods are very hard to be predicted if the system does not have deterministic deadlines, "bursty" access pattern of storage devices [16] is one of our motivations. By "bursty," we mean that the access patterns for NFSDs may include two clearly classfied types of idle periods.

The first type of idle periods (short idle periods) are very short and found between bursty requests while the second type (long idle periods) are very long and found between groups of burst accesses. We assumed the first type of idle periods should be filtered out and only the second type should be exploited since the length of the first type is too short to execute extra operations which consist of NFM commands spending very long time despite their appearance frequency.

4.2 Service Penalty and Aggressiveness of Idle-Time Exploitaition

Idle-time exploiting methods eventually incur performance degradation, since the idle-time prediction cannot be perfect. The performance degradation is a subset of a waiting phase introduced in Section 2.1 since the next request should wait in a queue until the hardware resources utilized by extra operations executed during idle-time are released. Specifically, we define this kind of waiting phase as a service penalty of the request in this work, and $T_p(n)$ refers to the service penalty of the (n)th request.

The service penalty is very similar to the wake-up penalty which is a critical metric in the HDD shutdown problem. However, it might be much more complex than the HDD shutdown problem which schedules only when to shutdown and wake-up the HDD. Unlike the shutdown of HDD which does not perform any particular operations, NFSDs actually execute operations and the operations occupy hardware resources. It means that, to minimize service penalties, a more tightly coupled scheduling policy is required and the policy should be responsible for controlling the aggressiveness of idle-time exploiting method.

4.3 Adaptive Threshold Value

Fig. 2 shows the latency of an NFSD with different input access patterns when various fixed threshold values are employed. In this case, the NFSD executed some useful tasks when the idle period was longer than the fixed threshold value.

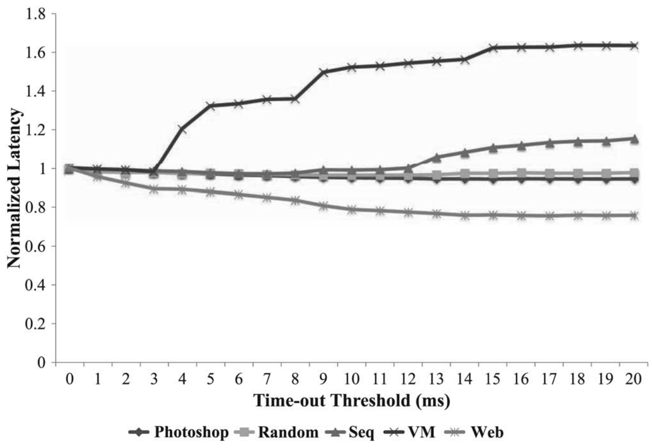


Fig. 2. Latency of an NFSD with various input access patterns and various time-out threshold values normalized to the latency of each access pattern with $T_{TO} = 0$ ms.

According to the input access patterns in Fig. 2, the optimal fixed threshold value, which maximizes performance, varies. Moreover, according to our simulation, a common trend is hardly found for the patterns.

Through this simulation, it can be inferred that in real computing systems with time-varying access patterns, a fixed threshold value does not guarantee the best long-term performance. Thus, to find the "when" of the proposed method, we need an adaptive method to dynamically track the optimal threshold value during runtime.

5 Proposed Method

5.1 Motivating Example

In addition to the selection of proper idle periods, scheduling of extra operation during idle-time is also a critical factor for determining performance improvement. However, achieving optimality of their scheduling is nearly impossible without an intensive offline analysis. In other words, the optimal scheduling policy requires perfect information about future input trace and the status of an NFSD.

Instead of nearly impossible optimal scheduling policy, our method is equipped with simple heuristic scheduling method inspired by a simple example shown in Fig. 3. First of all, we assume that an NFSD has four different instances of extra operations in its extra operation pool. Each instance of extra operations is depicted by E(n) and the width of each box indicates the execution time. Additionally, we also assume that each instance E(n) must be preceded by D(n), which is the data phase of the (n)th request R(n) to prevent the NFSD from malfunctioning as mentioned in Section 4.2. The arrival time of request is represented by dashed lines.

The timing diagram of the NFSD that does not exploit idle-time is depicted by the timing diagram labeled as *Without idle exploitation*. By our assumptions, the latency of all requests is degraded by extra operations.

The second timing diagram labeled as *Shortest-first* shows the NFSD with the idle-time exploitation method utilizing shortest-first (SF) scheduling policy. The policy executes extra operations in the ascending order of their execution time. Even though its total execution time is reduced thanks to the

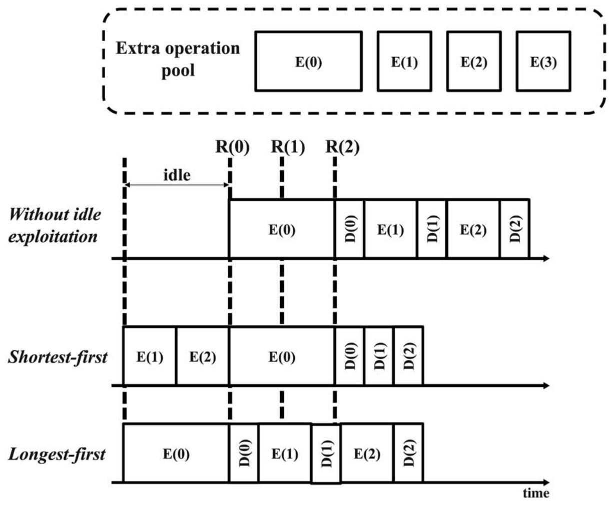


Fig. 3. Comparison of two different idle-time exploiting method.

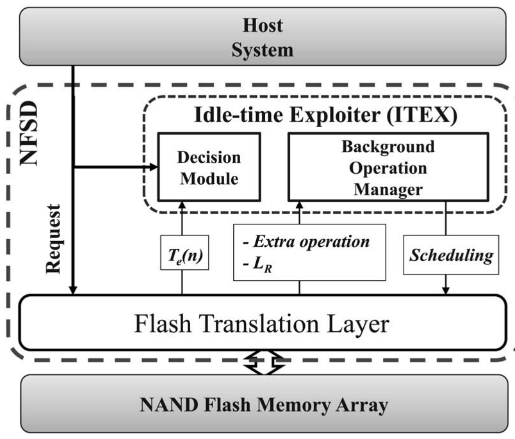


Fig. 4. Overview of proposed system.

utilization of idle period, the latency for request R(0) is not improved since E(0) should be executed before D(0).

The last timing diagram shows the NFSD with longest-first (LF) scheduling policy, which schedules extra operations in the opposite way to SF policy. Due to the execution of E(0) during idle-time, R(1) and R(2) suffer from the overhead incurred by extra operations. However, R(1) is handled quickly, since a portion of E(1) is handled during idle-time between R(0) and R(1).

Even though the latency of each request is different from each other, SF policy and LF policy shows the same total execution time. It is because two NFSDs do the same amount of operations, i.e., three extra operations and three data phases. However, in terms of both average and peak latency, LF policy outperforms SF policy for the above example. This motivates us to use LF policy as our scheduling policy.

The effect of LF policy is simple. At any moment, it can guarantee that average time required to execute extra operation is minimum. If every instance of extra operation has the same probability to be executed during an extra phase and must be executed eventually, then the expected length of an extra phase can also be minimized, thus both average and peak latency can be minimized.

5.2 Overview of Proposed Method

The overall structure of an NFSD employing the proposed method is shown in Fig. 4. Our method called idle-time exploiter (ITEX) is implemented in software and cooperates with an FTL.

ITEX consists of two sub-modules: a decision module and a background operation manager. The decision module is responsible for the ";when"; of the method. It detects both request issues from the host system and idle periods to identify useful idle periods. To determine the usefulness of an idle period, the decision module adopts an adaptive time-out-based scheme with the information such as $T_e(n)$ and $T_p(n)$ from other modules of the NFSD. The length of an extra phase, $T_e(n)$ is given by the FTL. The FTL records the time to execute extra operations while it translates address from logical to physical and passes the recorded time to the decision module. $T_p(n)$ is directly measured by the decision module from the moment an request is issued if the background operation manager is operating.

The background operation manager schedules the extra operations for the idle periods and it corresponds to the "which" of our problem formulation. It recieves the list of extra operations from the FTL and schedules them during idle periods selected by the decision module. Moreover, it controls the aggressiveness by a parameter called *restriction level*, L_R. It defines the portion of the used blocks over the total over-provisioning blocks. Therefore, as L_R decreases, the background operation manager becomes more aggressive in executing extra operations, since it will executes until only L_R% of blocks is occupied over the total over-provisioning blocks. To figure out how many blocks are currently used, the background operation manager receives the portion of used over-provisioning blocks from the FTL.

The operational steps of ITEX are described in Fig. 5. First, at the end of each request's handling, the decision module receives the measured $T_e(n)$ from the FTL and detects whether the current idle periods is longer than T_{TO} or not. If the idle periods is shorter than T_{TO}, the decision module initiates the feedback routine. Since our method is based on the HMM, the feedback routine observes the current symbol which is the measured information. Of course, since it does not execute any extra operation, $T_p(n)$ is set as zero. After the feedback routine, the new T_{TO} is updated. The details of the algorithm will be explained in the next section.

If the idle period is longer than T_{TO}, the background operation manager takes the role of control. It checks the list

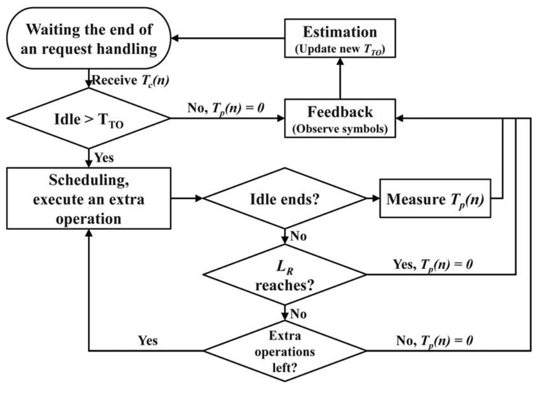


Fig. 5. Flow chart of ITEX.

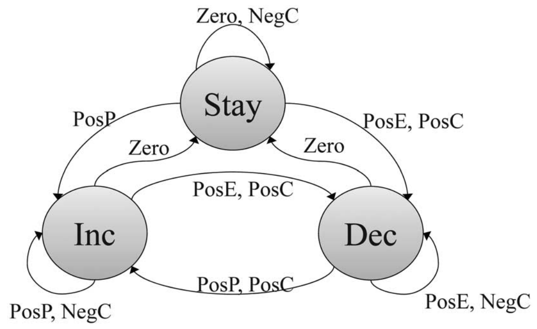


Fig. 6. State transition diagram of our modeling by the HMM.

of extra operations and schedules them for the FTL. At the end of execution of each scheduled extra operation, it executes another extra operation only if three conditions–the current idle period does not end, L_R is not reached and some extra operations are left–are satisfied. If a request is issued before currently executed extra operation finishes, the decision module measures $T_p(n)$ and initiates the feedback routine.

5.3 Decision Module

Our decision module calculates the time-out threshold value, T_{TO} based on the HMM. We describe the details of HMM modeling in the following sections.

5.3.1 System Modeling Based on the HMM

The Hidden Markov Model (HMM) is different from a regular Markov model in the perspective of visibility of its states. Since the states are not visible while the outputs (symbols) of the HMM can be observed, the sequence of state, Q_t, can only be guessed from the sequence of symbols, X_t. We also have to notice that two distributions are required, which are the state transition probability distribution A and the observation symbol probability distribution B, for futher explaination. A indicates the probability of transition among states, while B shows the probability that each symbol is observed. If initial conditions are added, the combination of A and B can represent the characteristics of the whole system.

To describe our system, which is an NFSD, we focused on the first problem among three basic problems using HMMs introduced in [17]. The problem focuses on computation of the most probable output sequence with negligible overheads by using an algorithm called forward algorithm and forward probabilities.

Our method uses the HMM to model an NFSD in the perspective of idle period it uses. However, we do not estimate the actual length of idle perids, since discrete representation of continuous values may cause problems, such as state explosion or extremely high computational complexity. Instead, we attempted to estimate the status of the NFSD which can be easily represented in a discrete way with information that can be gathered from the NFSD itself. To be specific, the discrete states are represented by the hidden states, while information from NFSD is used as symbols of the HMM.

We first define three states to model the changes in T_{TO}. The three states–Inc, Dec, and Stay states shown in Fig. 6 represent increasing, decreasing, and unchanging T_{TO}, respectively. For instance, if our system is in the Inc state, increasing T_{TO} is advantageous in exploiting idleness. Since states change only T_{TO} instead of defining the actual length of an idle-time, our method is expected to fully utilize sudden long idle-time. Also, the amount of T_{TO} changes at each state, ΔT_{TO}, is an important factor to trade-off the adaptation speed and the control resolution. We experimentally found the best value which will be discussed in Section 6.4.

TABLE 1
Definition of Symbols

Name	$\{P_p, P_e, P_s\}$
PosP	10x
PosE	01x
Zero	00x
PosC	111
NegC	110

Along with the state, the state transition condition of our model is defined as symbols given in Table 1. A symbol is a 3-tuple vector whose elements are P_p, P_e, and P_s, which are boolean variables computed by Eqs. 1, 2, and 3. $T_p(n)$ and $T_e(n)$ in these equations represent the time duration of the service penalty and extra phase of the (n)th request, respectively.

$$P_p = (T_p(n) > 0), \tag{1}$$

$$P_e = (T_e(n) > 0), \tag{2}$$

$$P_s = (\{T_p(n) + T_e(n)\} - \{T_p(n-1) + T_e(n-1)\} > 0). \tag{3}$$

Since we use two timing variables, $T_p(n)$ and $T_e(n)$, which can be either zero or bigger than zero, all combinations of two variables can be represented by two boolean variables, P_p and P_e. However, for the situation that both boolean variables are true, we need to specify the status of the NFSD, thus P_s is added to indicate the sum of $T_p(n)$ and $T_e(n)$ is whether bigger or smaller than that of the previous time stamp, $n-1$. More detailed description of each variable is given below.

- P_p: This value is true when the NFSD suffers from service penalties ($T_p(n) > 0$). As a result, it means that the current exploitation of idleness is too aggressive.
- P_e: This value indicates the situation that extra phases degrade performance ($T_e(n) > 0$). Therefore, to reduce the effect, the more aggressive ITEX would be helpful.
- P_s: This value is taken into consideration only above two variables are all true. More specifically, it is true when both $T_p(n)$ and $T_e(n)$ are bigger than zero and the sum of them is bigger than that of $T_p(n-1)$ and $T_e(n-1)$. As a result, if it is true, it means that performance degradation at the current time stamp is larger than that of the previous time stamp.

By combining the above three boolean variables, the decision module composes the following symbols shown in Table 1.

- PosP is the symbol observed only when P_p is true and P_e is false as shown in Table 1. As explained above, it means that ITEX is too aggressive. Therefore, to be conservertive, a state transition is made to the Inc state so as to generate a longer T_{TO}.
- PosE is the symbol representing the opposite situation to PosP. Therefore, a state transition to the Dec state is made.

- PosC is observed when all of the three boolean variables are true. It means that the sum of $T_p(n)$ and $T_e(n)$ is increasing. It implies that the current direction of changing T_{TO} is wrong, if the current state is Inc or Dec. Therefore, the decision module changes its state to the opposite side, like from Inc to Dec or from Dec to Inc. Note that, if a PosC symbol is observed when the current state is Stay state, our method makes a transition to the Dec state since we want to become more aggressive in utilizing idle periods.
- NegC has an opposite meaning to that of PosC. Therefore, it does not incur a state transition to further reduce the latency.
- Lastly, a Zero symbol incurs state transition to the Stay state since it is the most desirable status of an NFSD. Hence, we do not need to change the threshold value.

5.3.2 Estimation and Feedback

During the estimation step, which is depicted in Fig. 5, we adopt the forward probability of the HMM, the state transition probability distribution A and the symbol probability distribution B. By computing forward probability, the decision module computes the most probable next symbol and the next state transition according to that symbol.

As explained also in Fig. 5, the feedback step is executed regardless of whether or not the idle period is exploited. With the actual symbol composed by $T_p(n)$ and $T_e(n)$, an actual state transition is figured out and A and B are updated. Since this feedback step is performed for every request, the threshold value and probability distributions A and B, can be rapidily changed so as to adapt to the input trace.

5.4 Background Operation Manager

The role of the background operation manager is to schedule extra operations and execute them during idle periods which are selected by the decision module.

As shown in Fig. 4, the background operation manager interacts with the FTL for performing extra operations. More precisely, the FTL provides the list of extra operations to be performed, then the background operation manager schedules them. Finally, the FTL performs the scheduled extra operations until one of the three conditions shown in Fig. 5 is satisified. With this interactive cooperation of the background operation manager and the FTL, the FTL manages the address mapping table exclusively for the data consistency.

Scheduling of proper extra operations is performed based on their execution time as mentioned in Section 5.1, which is the LF policy. However, it is designed to adopt other policies such as SF policy for future extension and the comparison purpose.

One additional thing that should be considered by the background operation manager is L_R. In fact, securing enough over-provisioning blocks during idle periods provides better performance by reducing the number of extra phases. However, at the same time, it may incur not only service penalties but also shortened life-time of NFMs since it should include NFM erase commands and is executed in the speculative way.

Consequently, L_R is a simple and effective value that provides the controllability for performance improvement including the amount of service penalties and the life-time of NFMs by restricting the aggressiveness of the background operation manager.

TABLE 2
Timing Information of NFMs

Name	Value	Name	Value
T_{prog}	200us	T_{BER}	2ms
T_R	25us	T_{wc}	25ns
T_{rc}	25ns		

5.5 Configuring an ITEX

To configure an ITEX, three parameters are required–ΔT_{TO}, L_R, and scheduling policy. For convenience, a combination of the parameters will be presented as {ΔT_{TO}, L_R, scheduling policy}. For example, {2, 20, LF} denotes that the ITEX starts to execute extra operations when an idle period is longer than 2 ms with longest-first policy. Additionally, the background operation manager will not operate when the portion of occupied over-provisioning blocks is less than 20%.

6 Experiments

6.1 Experimental Setup

In order to evaluate the performance of the proposed method, we implemented a trace-driven simulator that includes the FTL with ITEX, timing information, and the configuration of the NFMs. The target FTL is BAST [4], which is one of the representative hybrid-mapped FTLs. The simulator reports various metrics introduced in the next section including the latency, the number of executed erase commands, timing information of each access such as $T_p(n)$ and $T_e(n)$, and etc.

The timing information of the NFMs conforms to the specifications outlined in [18]. Important timing information is given in Table 2. The simulator is equipped with the multichannel/multi-way (specifically, 8-channel/4-way) architecture like the one introduced in [19] and its FTL utilizes the hardware resources in the way proposed in [20].

To drive the simulator, we prepared various input traces, which are shown in Table 3. All of these traces are collected by DiskMon [21] while applications are running. Especially, VM trace contains the smallest amount of idle-time among all traces. Random, VM and Web traces can be classified into random access traces since their average number of written sectors are relatively small.

Before starting simulation, the simulator fills NFMs by writing data to incur many extra operations to obviously show the effect of the proposed method. The data is not only randomly located one but also located at the logical addresses accessed by the input trace. Additionally, for a comparison, experiments are conducted using other methods as follows.

- **Normal**: It is simply a normal FTL without exploiting idle periods.
- **FlexFS**: It is the fixed T_{TO} scheme proposed in [13]. Its decision module has a fixed $T_{TO} = 1000$ ms and the triggering interval is 20 ms. Originally, in [13], 15 ms is proposed for the interval. We need to modify this since one unit of extra operation is different from original FlexFS. Furthermore, we adopt only its decision module

TABLE 3
Used Trace Information

Name	Avg. $T_{I,n}$ (ms)	Total Length	Write Ratio (%)	Avg. Written Sectors	Optimal T_{TO} (ms)	Duration (sec)	Description
Photoshop	19.26	23498	29	121.294	15	453	Modifying and saving images using a graphical software.
Random	8.79	33740	99	8.276	9	296	Benchmark for random accesses.
Seq	13.11	10000	94	1821.82	7	131	Benchmark for sequential accesses.
VM	3.40	10990	98	22.01	3	37	Storing the status of virtual machine.
Web	24.56	26079	68	23.85	17	641	Surfing internet.

and use our background operation manager to cooperate with an FTL, since FlexFS is a file system designed for a certain architecture of NFSDs.

- **Fixed_Optimal**: It is the fixed T_{TO} scheme obtained by time-consuming offline simulation by sweeping T_{TO} from 1 to 20 ms. The optimal T_{TO} values are selected in the perspective of the shortest average write latency and they are depicted in the sixth column of Table 3 when $L_R = 10\%$ is used. This method is called static ideal one since it gives the best performance when a static T_{TO} is applied.

6.2 Evaluation Metrics

To prove the efficiency of the proposed method, we define several metrics as follows.

- Average write latency (L) is a good metric to measure the macroscopic capability of NFSDs. Especially, the write latency is used since write requests are usually severely affected by expensive extra phases.
- Peak latency per sector (L_{peak}) is defined as the maximum latency per one sector writing. This metric is important, because some SSDs suffer from the phenomenon called *freezing* which means that users experience intermittent suspensions of systems due to their NFSDs and is caused by long peak latencies.
- Erase count (N_E) is one of essential metrics, since NFMs have limited life-time. The total number of erase operations, which can be controlled by L_R of the background operation manager, should be compared to estimate their life-time.
- Utilizable length of idle-time (L_{UI}) is defined as the total length of utilizable idle-time which are selected by the decision module. Therefore, this metric is used to evaluate the efficiency of the decision module. If the decision module uses the time-out policy, it can be computed as

$$L_{UI} = \sum T_{I,n} - T_{TO,n} \quad \forall T_{I,n} > T_{TO,n}, \tag{4}$$

- where $T_{I,n}$ is the length of an idle period before the (n)th request is issued and $T_{TO,n}$ is the time-out threshold value at that time.
- Erase-aware latency improvement (I_{EA}) is a newly defined metric to show the ratio of the improvement in latency to the cost for the improvement. It is defined as

$$I_{EA} = \frac{L_{Normal} - L}{((N_E - N_{E,Normal})/N_R) + 1}, \tag{5}$$

- where N_R is the number of total requests, L_{Normal} and $N_{E,Normal}$ are the latency and the number of erase commands of an FTL that does not exploit idle-time, respectively. The last added one in the denominator prevents the equation from divergence. According to Eq. 5, the physical meaning of I_{EA} is improvement in latency over additional erase commands per request. Erase commands are accumulated as more requests are induced, thus the number of handled requests should be considered. Higher I_{EA} is preferred since higher performance improvement with the smaller cost means higher efficiency.

TABLE 4
Real Values of L (ms)

Scheme	Photoshop	Random	Seq	VM	Web
Normal	8.8	7.7	13.3	8.3	4.9
FlexFS	3.6	4.5	13.2	8.3	1.0
Adaptive	2.1	0.3	10.3	5.3	0.4
Fixed_Optimal	2.0	0.3	10.3	3.8	0.4

We conducted parameter sensitivity analysis and optimization of ITEX to find the optimal configuration which is {1, 10, LF}. The performance comparison with other method based on the defined metrics will be shown in the next section. After the performance comparison, the effects of varied parameters will also be presented.

6.3 Performance Comparison

6.3.1 Average Write Latency

Table 4 shows how much the proposed ITEX, which is referred as Adaptive, improves the average write latency. On average, L of ITEX is 87% shorter than that of Normal. Compared with the static ideal case of Fixed_Optimal, ITEX gives only 8% longer write latency. If VM trace is excluded, the difference between ITEX and Fixed_Optimal is only 1.6% at most. The case of VM trace includes the smallest amount of

TABLE 5
Real Values of L_{peak} (ms)

Scheme	Photoshop	Random	Seq	VM	Web
Normal	18.7	10.4	18.7	10.8	18.7
FlexFS	18.7	9.5	18.7	10.8	18.7
Adaptive	2.3	0.7	4.4	2.5	4.5
Fixed_Optimal	2.3	0.7	4.4	1.7	0.8

idle-time among traces as shown in Table 3. In such a case, ITEX should incur service penalties due to its speculative execution of extra operations during short idle periods of VM trace. In spite of the inefficiency of adaptation for VM trace, ITEX accomplishes 37% improvement in L compared to Normal, whereas FlexFS fails to achieve any improvement due to its fixed long T_{TO}.

With the considration of average written sectors in Table 3 and timing of NFMs in Table 2, it can be said that ITEX and Fixed_Optimal eliminate most of extra operations affecting L especially for Random and Web traces. For other traces, ITEX and Fixed_Optimal reduce extra operations compared to Normal and FlexFS but do not completely remove them.

6.3.2 Peak Latency

Table 5 shows L_{peak} values of four schemes. ITEX gives about 83% shorter L_{peak} than Normal on average. Furthermore, for the former three traces, i.e. Photoshop, Random and Seq, ITEX and Fixed_Optimal show the same L_{peak}. It means that ITEX executes extra operations affecting L_{peak} as much as Fixed_Optimal does during idle-time.

Futhermore, Table 5 shows that some traces suffer from the maximum L_{peak} of our NFSD which is 18.7 ms and includes one page program, 64 page copy and 2 erase oprations without idle-time exploitation. It means that the NFSD may incur frequent freezing phenomena when requests with large data arrive. ITEX and Fixed_Optimal shows much lower L_{peak}, thus reduced number of freezing phenonmenons can be expected.

However, for VM and Web traces, ITEX fails to achieve the same L_{peak} as that of Fixed_Optimal. In the case of VM trace, like L in the previous section, ITEX is not able to secure the same amount of idle-time as Fixed_Optimal does because of insufficient idle-time of the trace. In other words, ITEX executes fewer extra operations than Fixed_Optimal due to smaller L_{UI}, thus some extra operations affect L_{peak} by incurring T_e larger than zero.

On the other hand, ITEX for Web trace shows even larger gap with Fixed_Optimal in spite of the longest idle-time of the trace. Based on our analysis, it is because ITEX exploits idle-time more aggressively than Fixed_Optimal does. Therefore, T_p degrades L_{peak} instead of T_e. It is also proven by the fact that the average T_{TO} of ITEX for Web trace, which is 13.58 ms, is shorter than that of Fixed_Optimal, which is 17 ms.

6.3.3 Utilizable Idle-Time

To compare decision module's efficiency of ITEX with other schemes, Table 6 shows the values of L_{UI}. Results for Normal scheme are omitted since it does not utilize any idle-time. ITEX shows only average 4% shorter L_{UI} than Fixed_Optimal. It is due to the adaptability of changing its T_{TO} dynamically according to varying access patterns. Furthermore, for Random trace, ITEX gives longer L_{UI} than Fixed_Optimal, which means it gives more chances to exploit idle-time to the background operation manager.

TABLE 6
Real Values of L_{UI} (ms)

Scheme	Photoshop	Random	Seq	VM	Web
FlexFS	2.4E5	7.8E4	1.3E3	0	3.3E5
Adaptive	2.9E5	1.8E5	3.3E4	1.1E4	3.2E5
Fixed_Optimal	2.9E5	1.6E5	3.4E4	1.4E4	3.3E5

Among results, there are two extreme cases in terms of L_{UI} of FlexFS. First, in the case of Web trace, FlexFS shows the longest L_{UI}. This can be understood by the average $T_{I,n}$ shown in Table 3. Web trace has the longest average $T_{I,n}$ among the traces and it actually indicates the length of the second type of idle periods introduced in Section 4. Therefore, FlexFS gives comparable L_{UI} only for traces such as Web and Photoshop since the most of second type idle periods in those traces are longer than T_{TO} of FlexFS.

On the other hand, VM trace shows the opposite extreme case. FlexFS shows zero L_{UI} for the trace and it explains why FlexFS fails to accomplish any performance improvement as shown in Section 6.3.1. This is because all second type idle periods of VM trace are shorter than T_{TO} of FlexFS and filtered out unlike Web and Photoshop traces.

6.3.4 Life-Time of NFMs

The life-time of NFMs is inversely proportional to the number of erase operations NFMs experience. Table 7 shows the values of N_E. The proposed method performs almost the same amount (0.2% larger) of erase operations as Fixed_Optimal and has 2.9% larger N_E than Normal on average.

The differences in N_E between Normal and the other methods are relatively large especially for Web trace. It is due to long idle periods in the trace. During the long idle periods, the background operation manager has more chances to execute extra operations. Thus, idle-time exploiting methods show relatively larger N_E than Normal. Furthermore, since ITEX and Fixed_Optimal give much longer L_{UI} than FlexFS, they execute even more extra operations than FlexFS. Thus, their N_Es tend to be larger than those of Normal and FlexFS as shown in Table 7.

In addition to N_E, since an block erasure is usually accompanied by live-page copying, we also measure the number of live pages copied and show the results in Table 8. Similar to N_E, three idle-time exploiting methods copy more live pages than Normal. Quantitatively, the difference between the proposed method and Normal is 5.51% on average while Web trace incurs the maximum difference, which is 14.65%. Even

TABLE 7
Real Values of N_E

Scheme	Photoshop	Random	Seq	VM	Web
Normal	2.17E6	6.33E5	7.17E6	5.14E5	5.87E5
FlexFS	2.18E6	6.42E5	7.17E6	5.14E5	6.09E5
Adaptive	2.20E6	6.54E5	7.22E6	5.22E5	6.25E5
Fixed_Optimal	2.20E6	6.54E5	7.22E6	5.29E5	6.25E5

TABLE 8
Number of Live-Page Copying

Scheme	Photoshop	Random	Seq	VM	Web
Normal	4.95E7	3.50E6	2.16E8	3.70E6	5.53E7
FlexFS	4.99E7	3.63E6	2.16E8	3.70E6	6.01E6
Adaptive	5.03E7	3.74E6	2.18E8	3.83E6	6.34E6
Fixed_Optimal	5.03E7	3.74E6	2.18E8	3.90E6	6.33E6

though the ratio is not small, the overhead is marginal compared to the improvement in average write latency achieved by the proposed method, which is 87% as shown in Section 6.3.1.

Additionally, since the life-time of NFMs is affected by standard deviation of each NFM block's erase count, the values are shown in Table 9. If the proposed method would incur large increase in standard deviation compared to Normal, the life-time of NFMs could be degraded even though the variation of N_E caused by idle-time exploiting methods is not huge. However, as shown in Table 9, the difference in standard deviation for four methods are small. Furthermore, the proposed method shows smaller standard deviation than Normal except for Seq trace. Qunatitatively, FlexFS, Adaptive and Fixed_Optimal show 2.8%, 4.1% and 4.1% smaller standard deviation than Normal on average. Even for Seq trace, the increase is negligible since it is less than 1%.

Even though the idle-time exploiting methods including the proposed method shows the smaller standard deviation than Normal for current simulation results, standard deviation largely depends on the wear-leveling policy the FTL employs and the proposed method is orthogonal to the policy. Therefore, the distribution of block erasures will converge to the same point eventually regardless of the exploitation of idle periods by the proposed method.

6.3.5 Erase-Aware Latency Improvement

Putting together L and N_E, Fig. 7 depicts the efficiency I_{EA} of three different methods. Normal is omitted since its efficiency is zero. On average, the proposed method achieves 89% of efficiency compared to Fixed_Optimal while FlexFS does only 60%. Moreover, in contrast to I_{EA} of FlexFS which fluctuates with varying traces, ITEX maintains the similar level of I_{EA} regardless of traces except VM trace due to its adaptability.

For Photoshop and Web traces, I_{EA} of FlexFS is 7% and 16% larger than Fixed_Optimal, respectively. This is because, as explained in Section 6.3.3, FlexFS is suitable for traces containing long idle periods. It can not only exploit proper (long enough) idle periods but also reduce speculative execution of extra operations which reduces redundant erase counts. Therefore, its efficiency represented by I_{EA} can be ameliorated despite of its lower performance improvement. As a future work, ITEX will be improved to reduce redundant execution of extra operations even with long idle periods.

TABLE 9
Standard Deviation of Each NFM Block's Erase Count

Scheme	Photoshop	Random	Seq	VM	Web
Normal	123.06	29.00	124.50	41.71	32.52
FlexFS	122.84	25.67	124.51	41.71	31.79
Adaptive	122.87	25.38	124.68	39.12	32.03
Fixed_Optimal	122.92	25.43	124.64	39.12	31.98

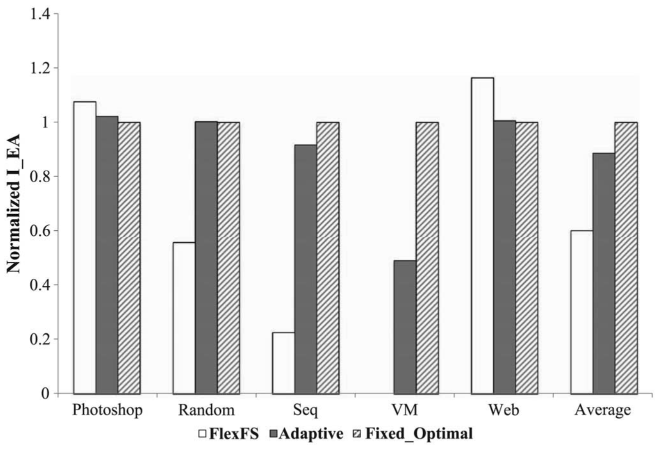


Fig. 7. I_{EA} normalized to that of Fixed_Optimal.

6.4 Parameter Sensitivity Analysis and Optimization of ITEX

In the following sections, we show how the optimal configuration of ITEX, which is {1, 10, LF}, is obtained. The sections may not only give the details of the optimal configuration, but also seeds of optimization for designers who consider different metrics as their major metrics.

6.4.1 Impact of ΔT_{TO}

The overall trends shown in Fig. 8 show that smaller ΔT_{TO} can give longer idle-time. The configuration of ITEX except ΔT_{TO} is fixed, i.e., LF policy and $L_R = 10\%$.

According to our analysis, there are two major drawbacks of coarse step size, i.e., large ΔT_{TO}. First, since the background operation manager starts to work only after T_{TO}, the utilizable length of each idle period is reduced by T_{TO}. Therefore, if ΔT_{TO} is large, the reduction also becomes large, thus L_{UI} is decreased.

Second, the decision module ignores idle periods smaller than ΔT_{TO}. In other words, the decision module loses ability to tune T_{TO} for mid-range idle periods, i.e., idle periods greater than zero but shorter than ΔT_{TO}. When the mid-range idle periods are exploited, it may incur service penalties but

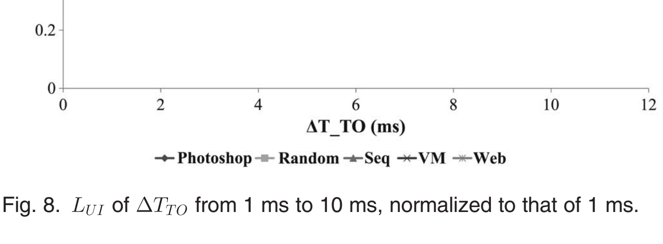


Fig. 8. L_{UI} of ΔT_{TO} from 1 ms to 10 ms, normalized to that of 1 ms.

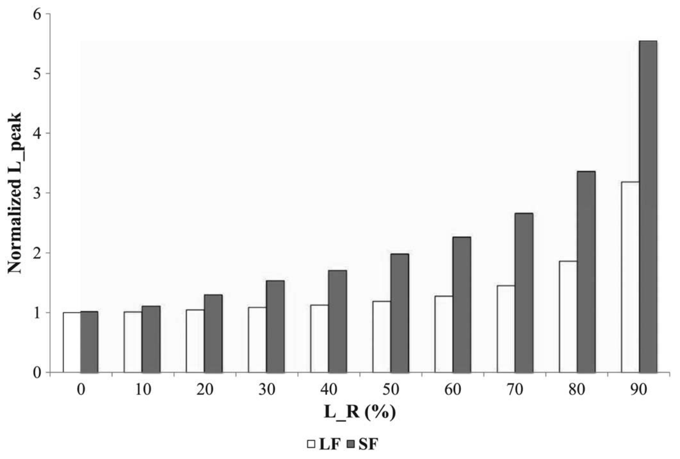


Fig. 9. L_{peak} of Web trace with varying L_R. All results are normalized to that of 1, 0, LF.

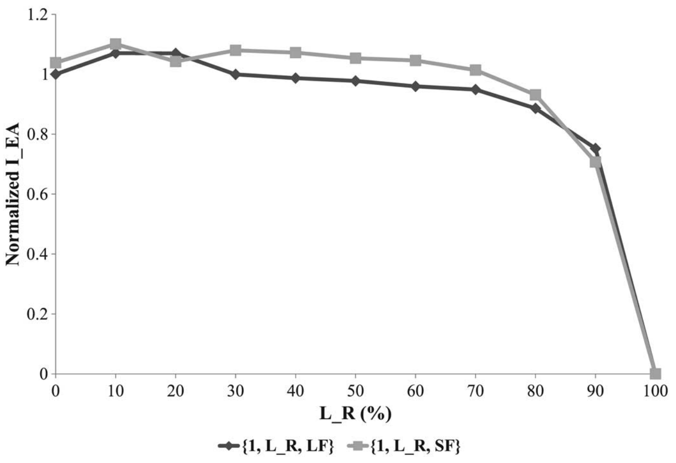


Fig. 10. Average I_{EA} of five traces with varying L_R. All results are normalized to that of 1, 0, LF.

also reduce $T_e(n)$ simultaneously. As a result, the fine-grained ΔT_{TO} enables ITEX to minimize the sum of $T_p(n)$ and $T_e(n)$, not only $T_p(n)$ and the process will be done automatically by the HMM modeling of the decision module.

We do not consider ΔT_{TO} smaller than a millisecond since NFM commands contained by extra operations usually require milliseconds of execution time. Hence, too small ΔT_{TO} cannot incur sufficient changes of T_{TO}. Moreover, adaptation speed would be too slow to track the dynamically changing behavior of input patterns. Consequently, 1 ms is chosen as the best ΔT_{TO}.

6.4.2 *Impact of Scheduling Policy*

Fig. 9 shows the effects of two different scheduling policies, SF and LF, on L_{peak} under various values of L_R for the Web trace. Even though only the result for the Web trace is depicted, all other traces have similar trends. Since LF policy provides lower L_{peak} regardless of the values of L_R, the background operation manager of ITEX utilizes LF policy.

This phenomenon is expected because LF policy can reduce the probability that NFSDs execute long extra phases, since LF policy selects extra operations requiring longest time first. On the other hand, SF policy severely suffers from high L_{peak} in the region of large L_R values, since the background operation manager has fewer chances to execute expensive extra operations, as larger L_R values are used. Hence, long L_{peak} of SF policy induces freezing phenomenon explained in Section 6.2.

6.4.3 *Impact of L_R*

As shown in Fig. 10, efficiency represented by I_{EA} becomes optimal around the point $L_R = 10\%$. The existence of the optimal point means that the L_R-based aggressiveness control of the background operation manager offers a trade-off between N_E and latency improvement.

Fig. 10 shows that SF policy outperforms LF policy for almost L_R values, unlike L_{peak} as shown in the previous section. This is explained by the fact that I_{EA} is dominated by L, not L_{peak}. Since SF policy executes more extra operations than LF policy does with the same given idle-time, over-provisioning blocks have more room for updated data. Therefore, L of SF policy is shorter than that of LF policy. To be specific, SF policy has 1% better average latency than LF policy at $L_R = 10\%$. However, at the same time, it experiences 10% worse L_{peak} than LF policy, thus we select LF policy for the background operation manager.

7 Conclusion and Future Work

In this paper, we proposed an extended and aggressive method to exploit the idleness of NFSDs. The experimental results obtained with a representative hybrid-mapped FTL indicate that the proposed method exhibits much better performance than previous schemes that have naive solutions to decide when and how idle periods are used. Quantitatively, in terms of the average write latency, the performance of the proposed scheme is 74% better than an existing method (FlexFS) and is within 9% of what is obtained with optimal cases through a time consuming off-line simulation. For the performance improvement, the proposed method spends only 2.9% larger erase count.

In the future, there are a few ways to extend the proposed method. First of all, our decision module can be improved by taking into account the characteristic of idle-time in addition to execution time of operation. It may give more accurate and faster T_{TO} adaptation. Additionally, power consumption is also another candidate to be considered by the decision module or background operation manager. ITEX may be extended to reduce or minimize power consumption by choosing proper idle periods and executing operaions.

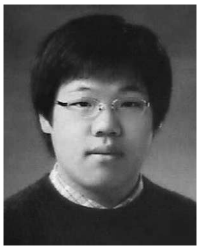
Acknowledgment

This work (Grant No. C0146555) was supported by Business for Cooperative R&D between industry, Academy, and Research Institute funded Korea Small and Medium Business Administration in 2013, by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MEST) (No. 2011-0027625), by Samsung Electronics.

References

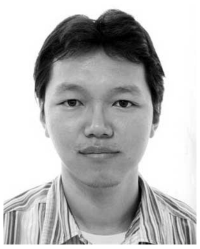
[1] G. Eskelsen, A. Marcus, and W. K. Ferree, *The Digital Economy Fact Book*, 10th ed. Washinton, DC, USA: The Progress & Freedom Foundation, 2009.

[2] H. Kim and S. Lee, "A new flash memory management for flash storage system," in *Proc. Comput. Softw. Appl. Conf.*, Oct. 1992, pp. 284–289.
[3] M. L. Chiang and R. C. Chang, "Cleaning policies in mobile computers using flash memory," *J. Syst. Softw.*, vol. 48, no. 3, pp. 213–231, 1999.
[4] J. Kim, J. M. Kim, S. H. Noh, S. L. Min, and Y. Cho, "A space-efficient flash translation layer for compactflash systems," *IEEE Trans. Consum. Electron.*, vol. 48, no. 2, pp. 366–375, May 2002.
[5] S.-W. Lee, D.-J. Park, T.-S. Chung, D.-H. Lee, S. Park, and H.-J. Song, "A log buffer-based flash translation layer using fully-associative sector translation," *ACM Trans. Embed. Comput. Syst.*, vol. 6, no. 3, p. 18, Jul. 2007.
[6] J.-U. Kang, H. Jo, J.-S. Kim, and J. Lee, "A superblock-based flash translation layer for nand flash memory," in *Proc. 6th ACM IEEE Int. Conf. Embedded Softw. (EMSOFT'06)*, 2006, pp. 161–170.
[7] H. Cho, D. Shin, and Y. I. Eom, "Kast: K-associative sector translation for nand flash memory in real-time systems," in *Proc. Des. Autom. Test Eur. Conf. Exhib. (DATE'09)*, 2009, pp. 507–512.
[8] H. Kwon, E. Kim, J. Choi, D. Lee, and S. H. Noh, "Janus-ftl: Finding the optimal point on the spectrum between page and block mapping schemes," in *Proc. 10th ACM Int. Conf. Embedded Softw. (EMSOFT'10)*, 2010, pp. 169–178.
[9] E.-Y. Chung, L. Benini, A. Bogliolo, Y.-H. Lu, and G. D. Micheli, "Dynamic power management for nonstationary service requests," *IEEE Trans. Comput.*, vol. 51, no. 11, pp. 1345–1361, Nov. 2007.
[10] Y.-H. Lu and G. De Micheli, "Adaptive hard disk power management on personal computers," in *Proc. 9th Great Lakes Symp. Very Large Scale Integration (VLSI'99)*, Mar. 1999, pp. 50–53.
[11] A. Gupta, Y. Kim, and B. Urgaonkar, "DFTL: A flash translation layer employing demand-based selective caching of page-level address mappings," *SIGPLAN Not.*, vol. 44, pp. 229–240, Mar. 2009.
[12] Y. J. Seong, E. H. Nam, J. H. Yoon, H. Kim, J. Y. Choi, S. Lee et al., "Hydra: A block-mapped parallel flash memory solid-state disk architecture," *IEEE Trans. Comput.*, vol. 59, no. 7, pp. 905–921, Jul. 2010.
[13] S. Lee, K. Ha, K. Zhang, J. Kim, and J. Kim, "Flexfs: A flexible flash file system for MLC nand flash memory," in *Proc. 2009 Conf. USENIX Annu. Tech. Conf.*, San Diego, CA, USA, 2009, pp. 9–9.
[14] S. Choudhuri and T. Givargis, "Real-time access guarantees for nand flash using partial block cleaning," in *Proc. 6th IFIP WG 10.2 Int. Workshop Softw. Technol. Embedded Ubiq. Syst. (SEUS'08)*, 2008, pp. 138–149.
[15] M. Jung and J. Yoo, "Scheduling garbage collection opportunistically to reduce worst-case I/O performance in solid state disks," in *Proc. Int. Workshop Softw. Support Portable Storage (IWSSPS'09)*, 2009.
[16] T. Simunic, L. Benini, P. Glynn, and G. De Micheli, "Dynamic power management for portable systems," in *Proc. 6th Annu. Int. Conf. Mobile Comput. Netw. (MobiCom'00)*, 2000, pp. 11–19.
[17] L. Rabiner, "A tutorial on hidden Markov models and selected applications in speech recognition," *Proc. IEEE*, vol. 77, no. 2, pp. 257–286, Feb. 1989.
[18] Hynix Semiconductor Inc., *8Gbit (1Gx8bit) NAND Flash Memory*. Hynix Semiconductor Inc., 2008 [Online]. Available: http://www.datasheetlib.com/datasheet/801671/hy27ug088g5b_hynix-semiconductor.html
[19] J.-U. Kang, J.-S. Kim, C. Park, H. Park, and J. Lee, "A multi-channel architecture for high-performance nand flash-based storage system," *J. Syst. Archit.*, vol. 53, no. 9, pp. 644–658, Sep. 2007.
[20] S.-H. Park, S.-H. Ha, K. Bang, and E.-Y. Chung, "Design and analysis of flash translation layers for multi-channel nand flash-based storage devices," *IEEE Trans. Consum. Electron.*, vol. 55, no. 3, pp. 1392–1400, Aug. 2009.
[21] M. Russinovich. (2006). *DiskMon for Windows v2.01* [Online]. Available: http://technet.microsoft.com/en-us/sysinternals/bb896646.aspx



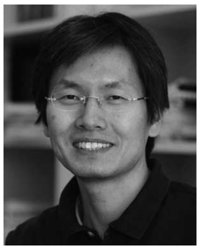
Sang-Hoon Park received the BS degree in electrical and electronic engineering from Yonsei University, Seoul, Korea, in 2009. Currently, he is a PhD candidate at Yonsei University. His research interests include System on Chip, NAND flash based mass storage architecture, and system architecture.

Dong-gun Kim received the BS degree in electrical and electronic engineering from Yonsei University, Seoul, Korea, in 2010. Currently, he is a MS candidate at Yonsei University. His research interests include System on Chip, NAND flash based mass storage architecture, and system architecture.



Kwanhu Bang received the BS degrees in computer science and electronic engineering and the MS degree in electrical and electronic engineering from Yonsei University, Seoul, Korea, in 2006 and 2008, respectively. Currently, he is a Ph.D. candidate in the School of Electrical and Electronic Engineering, Yonsei University. His research interests include biocomputation, flash memory applications, and system-level low-power design.

Hyuk-Jun Lee received the BS degree in computer engineering from University of Southern California, Los Angeles, CA, in 1993, and the MS and PhD degrees in electrical engineering from from Stanford University, Stanford, CA, in 1995 and 2001, respectively. From 2001 to 2011, he served as a senior engineer in routing technology group at Cisco System, San Jose, CA, where he participated in developing CRS-1 and CRS-3. Currently, he is an assistant professor with the Department of Computer Science and Engineering, Sogang University, Seoul, Korea. His research interests include embedded systems, bio-computing, low-power design, and memory architectures.



Sungjoo Yoo received PhD degree from Seoul National University, South Korea, in 2000. From 2000 to 2004, he worked as a researcher with TIMA laboratory, Grenoble, France. From 2004 to 2008, he was with Samsung System LSI. He is an assistant professor with the Department of EE, POSTECH, Korea, since 2008. His research interests include software, architecture and RTL design for low power SoC, and memory and storage hierarchy from cache, DRAM, phase-change RAM to solid state disk.

Eui-Young Chung received the BS and MS degrees in electronics and computer engineering from Korea University, Seoul, Korea, in 1988 and 1990, respectively, and the PhD degree in electrical engineering from Stanford University, Stanford, CA, in 2002. From 1990 to 2005, he was a principal engineer with SoC R&D Center, Samsung Electronics, Yongin, Korea. Currently, he is a professor with the School of Electrical and Electronics Engineering, Yonsei University, Seoul, Korea. His research interests include system architecture, bio-computing, and VLSI design, including all aspects of computer-aided design with the special emphasis on low-power applications and flash memory applications.