A Dynamic-Voltage-Adjustment Mechanism in Reducing the Energy Consumption of Flash Memory for Portable Devices[1]

Li-Pin Chang, Tei-Wei Kuo, and Shi-Wu Lo
{d6526009,ktw,d89015}@csie.ntu.edu.tw
Department of Computer Science and Information Engineering
National Taiwan University
Taipei, Taiwan 106, ROC
Fax:+886-2 23628167

ABSTRACT

With the increasing popularity of portable devices, there is a strong demand of power management mechanisms for microprocessors and other subsystems. Advanced microprocessor and flash memory chips design now supports several voltage levels and allows more intelligence in managing energy consumption to lengthen the operating time of portable devices. In this paper, we propose a dynamic-voltage-adjustment method to reduce the energy consumption of a flash memory storage system, depending on the system workload. The usefulness of the proposed method is demonstrated by a series of experiments over a realistic setup, for which we have very encouraging results.

Keywords: request scheduling, flash memory, energy consumption, storage systems, portable devices, real-time performance.

[1] This paper is an extended version of a conference paper appeared in the *IEEE International Conference on Consumer Electronics*, 2001 [14].

1. Introduction

With the increasing popularity of portable devices, there is a strong demand of energy management mechanisms for microprocessors and other subsystems. Advanced microprocessor and flash memory chips design now supports several voltage levels and allows more intelligence in managing energy consumption to lengthen the operating time of portable devices. The goal is to let a portable device operate at a low-voltage level, whenever it is possible, to save battery power. An intelligent energy management mechanism should be able to deliver reasonable system performance at the lowest possible energy consumption.

Recently researchers have started investigating voltage-clock-scaling scheduling algorithms for microprocessors, e.g., [1,2,7,8,9]. The main idea is to let a microprocessor operate at a low energy consumption mode to fully utilize any possible microprocessor idle time, while the system is not heavily loaded. A reasonable system performance is delivered. In a portable device, flash memory also contributes a significant portion of energy consumption. For example, a typical 16MB flash memory consumes 264mW for page writes, where an ARM10 processor consumes 275mW when running at 390MIPS. But, little work is done in the power management of flash memory. In particular, Douglis, et al. [5] provided a series of energy consumption measurement for flash memory under different percentages of capacity utilization. While the size of flash memory in a system is increasing, the issue for proper management of energy consumption becomes very critical.

In this paper, we propose a dynamic-voltage-adjustment method to manage the energy consumption of flash memory storage systems. When the system is heavily loaded, a reasonable portion of operations will be performed at a high voltage level to deliver a reasonable performance. However, when the system workload is low, the proposed mechanism will try to service requests at a low voltage level without sacrificing the system performance. We show that the scheduling with an energy consumption constraint problem is NP-Complete. We then propose an efficient on-line scheduling algorithm with an objective to satisfy the response time of requests and, at the same time, to minimize the energy consumption of flash memory. The strength of the proposed mechanism is evaluated by a series of experiments over realistic traces and typical flash memory characteristics, for which we have very encouraging results.

The rest of the paper is organized as follows: Section 2 summarized the related work. Section 3 presents the system model and the system architecture for flash-memory file systems. Section 4 proposes our dynamic-voltage-adjustment method in managing the energy consumption of flash memory storage systems. We also show the NP-Completeness of the problem and propose an O(n) on-line approximate algorithm. The optimality of the algorithm in meeting request deadlines (in a restrictive sense for energy consumption) is also shown. Section 5 provides our experimental results over realistic workloads, for which we have very encouraging results. Section 6 is the conclusion.

2. Related Work

The fundamental theory behind the recently proposed voltage-clock-scaling techniques is the power consumption model of CMOS. The power dissipation of CMOS consists of switching power, short-circuit power, and leakage power. Chandrakasan et al. [17] showed that the switching power dominates the total power consumption, and it could be represented as:

$$P=\alpha C_L V_{DD}^2 f \quad (1)$$

where αC_L is the effective switched capacitance, V_{DD} is the supply voltage, and f is the clock frequency. In Equation (1), we observe that the energy consumption is proportional to the frequency and also the square of the supplied voltage. The equation suggests the adjustment of voltage is the most efficient way to reduce the energy consumption. Meanwhile, with the demanding of a variable voltage control, Namgoong et al. [15] presented a highly-efficient DC-DC bulk-converter switching regulator with a variable voltage output. The proposed variable voltage control was efficient in voltage adjustment.

Mark, et al. [7] is among the pioneers who investigated the possibility of having the processor operating at a low-power-consumption mode, while the system still guarantees the real-time response of requests (in terms of deadline satisfaction). They also proposed a variable-voltage scheduling method based on the fact that the energy consumption is proportional to the square of the supplied voltage, as described in Equation (1). The energy consumption is effectively reduced, while no deadline violation occurs. They [7] defined a new CPU performance metric, named MIPJ, which denotes MIPS per Watt (note that the time factor is eliminated). They indicated that although the energy consumption decreases linearly with the frequency, the reducing of the frequency won't reduce the total energy consumption (or MIPJ) because it takes a longer time to complete the computation of jobs when the frequency is low. In other words, MIPJ remains with the changing of the frequency. Although MIPJ remains, Nakamoto, et al. [2] pointed out that the adjustment of the frequency is still useful for a battery-powered device, because of the property of Lithium-Ion batteries. A Lithium-Ion battery does not have a linear relationship between the discharging current and the battery life. They measured and proposed the discharging property of a battery. It can be modeled approximately by the following formula:

$$I^{\alpha} \cdot t = C_b \quad (2)$$

where I is the discharging current, t is the usage time, $\alpha > 1$ is a constant, and C_b is the amount of the battery capacity. Obviously, the relationship of I and t is not linear. Because $P = I \cdot V$ and Equation (1), the relationship of the frequency f and the discharging current I is linear. Nakamoto concluded that the reducing of f effectively lengthens the endurance of the battery, because the reducing of the frequency also reduces the discharaging current. With a smaller discharging current and the property shown in Equation (2), more "energy" can be drained from the batteries.

Recently researchers started noticing that there are still many power-hungered devices in portable devices. Hong, et al. [9] proposed an integrated variable-voltage control mechanism for the micro-processor and the cache sub-system. Li [18] surveyed the energy consumption of disk, and he used a spin-off technique to substantially reduce the energy consumption. Douglis et al. [5] conducted a series of experiments on storage devices for mobile computer, such as a linear flash memory card and a PCMCIA flash disk. The most important conclusion of their work was that the energy consumption of a flash memory storage device depends on the capacity utilization of the device. They reported that the energy consumption of a flash memory with a 95% capacity utilization is increased by 70%~190%, compared to a flash memory with 40% capacity utilization. The rationale behind the results comes from the needs for garbage collection. In Section 3.1 we shall illustrate the garbage collection behavior of a falsh memory, and how it has impacts on the system workload.

3. System Model

3.1 Flash Memory Characteristics

Flash memory is a non-volatile, write-once, and bulk-erase memory device. A flash memory chip consists of many fixed-sized blocks, where a block is the smallest erase unit. The size of a block varies from 8Kbytes to 64Kbytes, depends on the hardware design. Furthermore, a block consists of many fixed-sized pages, where a page is the smallest unit of a read / write operation. Typically, a page is 512 bytes large, which is identical to a disk sector. However, flash memory is free from any seek penalty since it is a random-access memory device. It is extremely fast on data retrieval. Because flash memory is write-once and bulk-erase, no in-place updating of data in a block is allowed unless an (bulk) erase on the block is performed first. The erase operations are relative slow, and each block on flash memory has an erase cycle limit, which ranges from 100,000 to 1,000,000. A worn-out block will suffer from frequent write errors. As a result, flash-memory-based storage systems should try not to overwrite data on flash memory. Instead, a new version of data is usually written at any available space, and the old copy of the data is then invalidated. Flash memory Translation Layer (FTL) is introduced to emulate a block device for flash memory [4,11,20,28] so that users/applications have transparent access of data at dynamically allocated space. Note that available space is created by erasing and recycling data that are no longer used. Such activities are called garbage collection in flash memory.

The garbage collection policy may choose to recycle unused space on a particular block while there are still few valid data on it. Therefore, valid data must be copied to somewhere else before the block erase takes place. The garbage collection consists of a series of reads, writes, and erases. These internally generated requests are referred as “internal requests” in this paper, and requests received from the file system are referred as “external requests”. Garbage collection is one of the major issues for flash memory management, and prior researches had studied this topic extensively [4,10,11,16]. A well-designed garbage collection policy should

V_{cc} = 5 v, Block Size = 64KB				
V_{pp}	Block Write		Block Erase	
	Power Consumption	Performance	Power Consumption	Performance
5v	375mW	0.5s	250mW	0.4s
12v	540mW	0.4s	480mW	0.3s

Table 1(a).
Performance and power consumption of a typical
NOR flash memory [20] that supports 12v and 5v.

Block Size = 16KB, Page Size = 512B						
V_{cc}	Page Read		Page Write		Block Erase	
	Power Consumption	Performance	Power Consumption	Performance	Power Consumption	Performance
3.3v	330mW	0.3ms	264mW	0.9ms	264mW	1.6ms
5v	700mW	0.182ms	600mW	0.546ms	600mW	0.975ms

Table 1(b).
Performance and power consumption of a typical
NAND flash memory [22,23] that supports 3.3v and 5v.

*Measured from a NAND flash memory evaluation board.

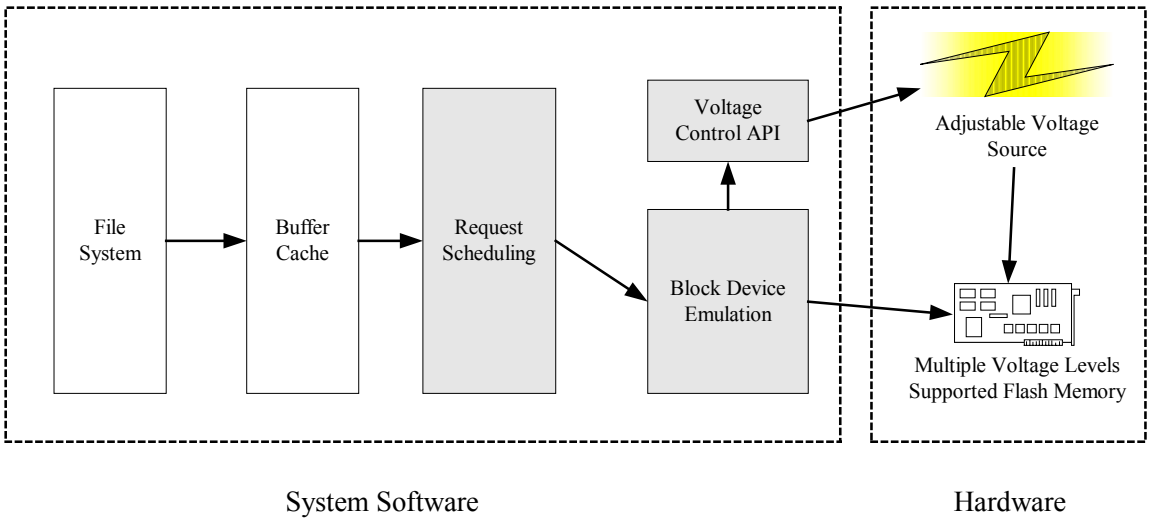


Figure 1. System Architecture.

reduce the number of data copyings and, at the same time, reduce the number of erases. In the experiment setup of this paper, we used the greedy reclaiming policy [4], which always selects the block that has the largest number of invalidated data to reclaim.

There are two major different architectures in flash memory design [27]: NOR and NAND. NOR [20] flash memory is designed for EEPROM replacement. On the other hand, NAND [16,17,19] / AND [18] flash memory is specifically designed for data storage. NAND flash memory has a higher density and is block-oriented. In this paper, we consider NAND flash memories in constructing the storage system. Advanced flash memory design now supports multiple voltage levels. The characteristics of the flash memory are summarized in Table 1 (note that NOR flash memories can also support multiple voltage levels). As it shown in Table 1, flash memories have higher performance when higher voltage supplied. For example, as pointed out in Equation (1), NAND flash memory consumes 600mW for page writes on 5v, and consumes 264mW on 3.3v. The power consumption is indeed proportional to the square of the supplied voltage. The usefulness of energy saving can be demonstrated by a simple calculation: the energy consumption of a page write is 0.6*0.000546 = 0.3276mJ on 5v, and 0.264*0.0009 = 0.2376 mJ on 3.3v, which is 72% of that on 5v. The simple calculation illustrates the usefulness of voltage adjustment on energy saving.

3.2 System Architecture

Workloads of a typical I/O subsystem are more unpredictable, compared with CPU workloads. Even though applications may issue I/O requests periodically, the requests still arrive at the corresponding block devices in a batch. The write burst could easily flood the request queue of the system and deteriorate the response time of requests quickly. Fortunately, the phenomenon may be resolved by the bandwidth reservation concept up to a certain degree [19]. The sources of unpredictability also include those from the occasional delay by the hardware, such as reading error, writing error, and calibration. Because it is not the focus of this paper, we refer interested readers to [19] for details.

Many embedded systems are performance-sensitive. The performance of a storage system can be measured by the response time of requests. For the purpose of our mechanism, we assign a deadline to each request as a soft bound for its response time and to reflect the performance

requirements of the system. Note that request deadlines are soft deadlines, where a similar concept could be found in the I_2O specification [26]. In other words, the system shall try to fulfill soft deadlines. The percentage of deadline violations somehow reflects the satisfaction level of the system performance requirements.

The system architecture of the flash memory storage system is illustrated in Figure 1: The file system issued requests, and the requests are queued in the request queue. A proper mechanism should schedule requests and adjust requests' voltage levels to satisfy the performance requirements under a limited amount of energy consumption. The mechanism for block device emulation translates requests into flash memory operations and execute them according to their voltage assignment.

In this paper, we have the following assumptions:

A1: Requests can not be preempted by each another.
A2: Each entire request is serviced at the same voltage level.
A3: Voltage-switching introduces negligible overhead [1,2,7,8,9].

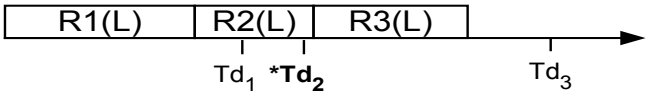
The assumption A1 and A2 are very reasonable because I/O requests can hardly be preemptible. Since the adjustable voltage can be generated efficiently, the voltage-switching overhead is not considered in this work.

4. Algorithm

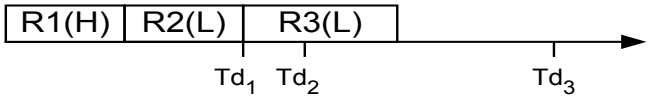
In this section, we first define the scheduling problem and show a basic polynomial-time algorithm. We then prove the NP-Completeness of the problem and propose an efficient approximate on-line algorithm, which has a linear time complexity.

4.1 The Basic Algorithm

Each request R_i is associated with an arrival time Ta_i, a completion time Tc_i, and a deadline Td_i. The time between Ta_i and Tc_i depends on the system workload and the supplied voltage. Let *S*



(a) Requests are all serviced at the low voltage level
(deadline of R2 is missed)



(b) R_1 is serviced at the high voltage level
(all deadlines are satisfied)

Figure 2. Concept of voltage-adjustment in scheduling.

= $\{R_1, R_2 \ldots R_n\}$ be a collection of pending requests for flash memory. The goal of request scheduling with voltage adjustment is to provide requests a good response time and, at the same time, minimize the total energy consumption.

We propose to first schedule requests in S in an earliest deadline first (EDF) order, where EDF is an optimal real-time non-preemptive scheduling algorithm if all requests are ready at time 0 [12,13,29]. Under EDF, a request with an earlier deadline is assigned a higher priority. We first generate an EDF schedule in polynomial time. In the schedule, requests are initially serviced at the low voltage level. If there is any request that can not meet its deadline, we will raise the voltage level of the first k requests until all requests meet their deadlines, or all requests are already serviced at the high voltage level. The number k must be minimized. The voltage adjustment have a time complexity $O(n^2)$, where n is the number of pending requests. Therefore, the basic algorithm has a polynomial time complexity.

We shall illustrate the idea by an example, shown in Figure 2. Suppose that there are three pending requests R_1, R_2, and R_3. with deadlines Td_1, Td_2, and Td_3, where $Td_1 < Td_2 < Td_3$. Figure 2(a) shows that R_2 will miss its deadline if all requests are serviced at the low voltage level. As shown in Figure 2(b), when the first pending request, i.e., R_1, is serviced at the high voltage level, all requests will meet their deadlines.

4.2 The Complexity of Request Scheduling with Minimal Energy Consumption

Definition: Request Scheduling with an Energy Consumption Constraint

> *Suppose a flash memory supports* $\boldsymbol{m}$ *operating modes in handling a request. Let* $\boldsymbol{S} = \{R_1, R_2, \ldots, R_n\}$ *be a set of pending requests, where each request* R_i *is associated with a pair* (Γ_i, Td_i)*, where* $\boldsymbol{Td_i}$ *is the deadline, and* $\boldsymbol{\Gamma_i} = \{(e_{i,1}, c_{i,1}), \ldots, (e_{i,m}, c_{i,m})\}$*.* $e_{i,j}$ *and* $c_{i,j}$ *are the energy consumption and computation time under the corresponding operating mode. Given an energy consumption bound* $\boldsymbol{E} \in Z^+$*, does there exist a schedule and a operating mode assignment* $\boldsymbol{A}$ *for each request such that all deadlines are satisfied, and the total energy consumption is no more than* $\boldsymbol{E}$*.*

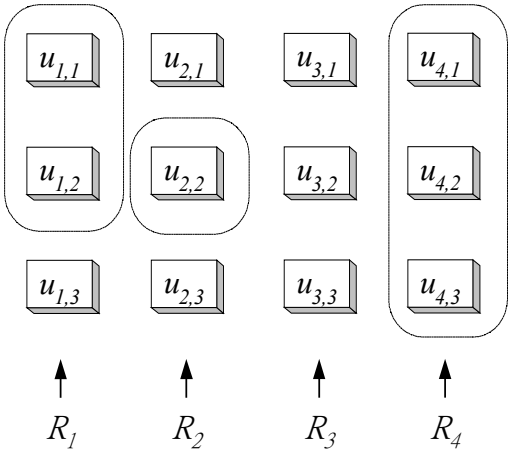
Theorem 1*: The Request Scheduling with an Energy Consumption Constraint problem is NP-complete.*

Proof: The Request Scheduling with an Energy Consumption Constraint problem is a NP problem because there exists a non-deterministic polynomial-time algorithm that can guess and verify a solution in a polynomial time. The NP-Hardness of the problem can be shown by a reduction from a well-known NP-Complete problem, e.g., the knapsack problem [18], as follows:

The knapsack problem is defined as follows: Given a finite set U of items, the size and the weight of u are denoted as $s(u)$ and $v(u)$ for each item $u \in U$, respectively, where s(u) and v(u) are both positive integers. The knapsack problem is to determine whether there exits a subset $U' \in U$ such that the following constraints are satisfied:

$$\sum_{u \in U'} s(u) \leq B \tag{3}$$

$$\sum_{u \in U'} v(u) \geq K \tag{4}$$



$U = \{u_{1,1}, u_{1,2}, u_{1,3}, u_{2,1}, u_{2,2}, u_{2,3}, u_{3,1}, u_{3,2}, u_{3,3}, u_{4,1}, u_{4,2}, u_{4,3}\}$

U is partitioned into four subsets, and each subset has three elements. Therefore we have **four requests** and **eight operating modes** for each request.

$U_1=\{u_{1,1}, u_{1,2}, u_{1,3}\}$ $U_1'=\{u_{1,1}, u_{1,2}\}$
$U_2=\{u_{2,1}, u_{2,2}, u_{2,3}\}$ $U_2'=\{u_{2,2}\}$
$U_3=\{u_{3,1}, u_{3,2}, u_{3,3}\}$ $U_3'=\{\}$
$U_4=\{u_{4,1}, u_{4,2}, u_{4,3}\}$ $U_4'=\{u_{4,1}, u_{4,2}, u_{4,3}\}$

$U' = \{u_{1,1}, u_{1,2}, u_{2,2}, u_{4,1}, u_{4,2}, u_{4,3}\}$

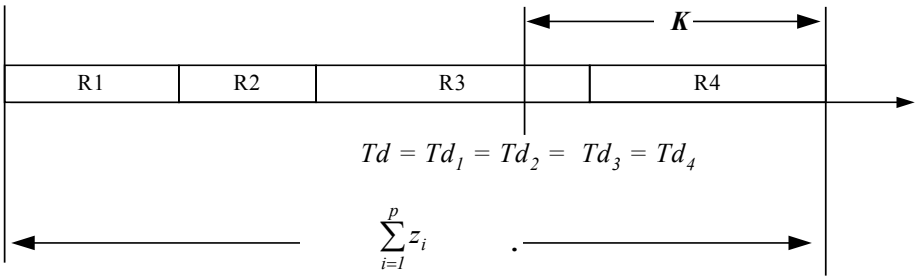
Figure 3. The reduction of a kanpsack problem into a request scheduling with an energy consumption constraint problem

The knapsack problem can be reduced to the request scheduling with an energy consumption constraint problem in a polynomial time as follows:

First we transform the items of an instance of the knapsack problem to a set of pending requests of a scheduling problem as follows: Let $|U| = C$, i be an integer ($1 \le i \le p$), and q be an arbitrary integer which can divide the size of U and $\lceil log|U| \rceil \ge q \ge 1$. We randomly and exclusively partition U into p subsets $U_1, U_2, U_3 \ldots U_p$, where $|U_i| = q$ for each i. Note when $|U| = 1$, $q = 1$ and the problem can be solved easily. After the partitioning, we have $U_i = \{u_{i,1}, u_{i,2}, \ldots, u_{i,q}\}$ for each i and $U = U_1 \cup U_2 \cup \ldots\ldots \cup U_p$. Now let $n = p$ and request R_i correspond to U_i for each i, and we have a set of pending requests $\{R_1, R_2, \ldots, R_p\}$. The transformation is illustrated in Figure 3.

We now create Γ_i of each request R_i as follows: We first have two sets of numbers $b_i = \{b_{i,1}, b_{i,2}, \ldots, b_{i,2^q}\}$ and $k_i = \{k_{i,1}, k_{i,2}, \ldots, k_{i,2^q}\}$ for each request R_i as follows:

Because each $U_i = \{u_{i,1}, u_{i,2}, \ldots, u_{i,q}\}$ has q items, there are 2^q combinations in selecting the q items. Let a tuple $D_{i,j} = (I_{i,j,1}, \ldots, I_{i,j,q})$ denote the selection of the q items in U_i, where $I_{i,j,k} = 1$ means that $u_{i,k}$ is selected, and $I_{i,j,k} = 0$ means that $u_{i,k}$ is not selected. R and V are two functions of $D_{i,j}$ such that $R(D_{i,j}) = \sum_{I_{i,j,k}} I_{i,j,k} * u(u_{i,k})$ and $V(D_{i,j}) = \sum_{I_{i,j,k}} I_{i,j,k} * v(u_{i,k})$, respectively. $b_i = \{b_{i,1}, b_{i,2}, \ldots, b_{i,2^q}\}$ and $k_i = \{k_{i,1}, k_{i,2}, \ldots, k_{i,2^q}\}$ are equal to $\{R(D_{i,1}), \ldots, R(D_{i,2^q})\}$ and $\{V(D_{i,1}), \ldots, V(D_{i,2^q})\}$, respectively. Note that each $k_{i,j}$ and $b_{i,j}$ corresponds to the same $D_{i,j}$. The creation of each b_i and k_i can be done in a $O(q*2^q)$ time. Because the number of requests is bounded by C ($C = |U|$), the time complexity for the creation of all b_i and k_i is $O(C*q*2^q)$. Furthermore, since $\lceil log|U| \rceil \ge q \ge 1$, the time complexity for the creation of all b_i and k_i is



K is the "speed-up" required to satisfy the deadlines when R1, R2, R3, and R4 are serviced at the maximum computation time.

Figure 4. Reducing Inequation (4) into the deadline satisfaction constraint.

$O(C^2 \log C)$, i.e., a polynomial-time complexity. Note that the mapping of each $b_{i,j}$ to the selection combination can be saved without consuming a lot of space (i.e., being bounded by $O(C^2)$).

The next step is to construct a request scheduling with an energy consumption constraint problem based on b_i and k_i: Let $m=2^q$ be the number of supported operating modes, and $m=2^q=|b_i|=|k_i|$ for all i. Let $Y=\{y_1, y_2, y_3, ..., y_p\}$ be the set of minimal energy consumption needed to service each request R_i, where y_i, which corresponds to R_i, could be any positive integer constant. We define the energy consumption needed to service request R_i at the *j-th* operating mode be $e_{i,j} = y_i + b_{i,j}$. As reader may notice, we create Y because $b_{i,j}=0$ when all elements in $D_{i,j}$ are zero (that is, not selected).

The computation time of each R_i at each operating mode is created as follows: Let $Z=\{z_1, z_2, z_3, ..., z_p\}$ be the set of the maximum computation time to service each request R_i, where $z_i > max(k_{i,j})$ for all j, and z_i is any positive integer constant. Now we define the computation time $c_{i,j}$ to service the request R_i at the *j-th* operating mode. Let $c_{i,j} = z_i - k_{i,j}$, where $k_{i,j}$ can be considered as the speed-up obtained from servicing R_i at the *j-th* operating mode.

Let $E-\sum_{i=1}^{q} y_i = B$. Suppose that U' is a solution to the knapsack problem, Inequation (3) can be re-written as follows:

$$\sum_{u \in U'} s(u) \le E - \sum_{i=1}^{p} y_i \tag{5}$$

Let all requests be associated with the same deadline T_d, that is, $Td_1 = Td_2 = ... = Td_p = Td$, and K in Inequation (4) be the "speed-up" required to satisfy to the deadline T_d when every request is serviced at the maximum computation time , as illustrated in Figure 4. Obviously, $(\sum_{i=1}^{p} z_i) - Td = K$. Inequation (4) can be re-written as follows:

$$\sum_{i=1}^{p} \sum_{u \in U_i'} v(u) \ge (\sum_{i=1}^{p} z_i) - Td \tag{6}$$

Since U is partitioned into $U_1, U_2, U_3, ..., U_p$, U' can also be partitioned into

$U_1', U_2', U_3', ..., U_p'$, where U_i' is a subset of U_i for each i. Therefore Inequation (5) becomes:

$$\sum_{i=1}^{p}(y_i + \sum_{u \in U_i'} s(u)) \leq E \quad (7)$$

Let $A=\{a_1, a_2, ..., a_p\}$ be the operating mode assignment for each request. When we substitute $\sum_{u \in U_i'} s(u)$ with the corresponding elements in b_i, Inequation (7) can be further re-written as follows:

$$\sum_{i=1}^{p}(y_i + b_{i,a_i}) \leq E \quad (8)$$

Obviously the E is the desired energy consumption bound, and the left-hand-side of Inequation (8) is the energy consuimption of all requests with a set of operating mode assignments A.

We then substitute $\sum_{u \in U_i'} v(u)$ with the corresponding elements in k_i,. Repeat the substution as shown above, Inequation (6) becomes:

$$\sum_{i=1}^{p}(z_i - k_{i,a_i}) \leq Td \quad (9)$$

Inequation (9) denotes that all requests serviced under the operating mode assignment A should be completed no later than the deadline T_d. This is the deadline satisfaction constraint.

With the above reduction procedure, we successfully reduce a knapsack problem into a request scheduling with an energy consumption constraint problem. If there exists an algorithm that can find a voltage assignment A to solve the scheduling problem, then the corresponding knapsack problem can be solved. Since the scheduling problem is a NP problem, the scheduling problem is NP-Complete. □

Corollary 1: *The request scheduling with an energy consumption constraint problem remains NP-Complete even if the flash memory has only two operating modes available.*

Proof: The corollary directly follows from Theorem 1, given that $|U_i| = 1$ for each i. Note that $\lceil log|U| \rceil \geq q \geq 1$, $2^{1+\log U} = 2U \geq 2^{\lceil \log|U| \rceil} \geq m$, and the sizes of b_i and k_i are still bounded by $2|U|$. □

4.3 An On-Line O(n) Approximate Algorithm

Because the problem for request scheduling with an energy consumption constraint is NP-complete, an efficient approximate solution must be found. While the CPU workloads are more predictable in many systems [1,2,8,14], it is hard to know the I/O requesting patterns in advance for many applications. Therefore, it would be impractical to use an off-line scheduling algorithm in the system implementation. In this section, we proposed an efficient linear-time on-line algorithm based on the basic algorithm in the Section 3.1: Given a set of pending requests with given voltage level assignments and a newly arrived request, the on-line algorithm should try to meet requests' deadlines and minimize the energy consumption. Flash memory with two voltage levels is considered in this paper.

```
1 On-Line-Schedule(S,Ri)
2  {
3     (M, L, S) = insertByEDF(S,Ri);
4
5     While true Do {
6           Let RL be the first request in L;
7           Let RM be the first request in M;
8
9           If(RL or RM does not exist)
10             Return S;     /* Complete */
11
12          If((the deadline of RM is satisfied) ||
13             (RL is scheduled behind RM)) {
14             Remove RM from M
15             Continue;
16          }
17
18          Switch the service of RL to the high-voltage;
19          Remove RL from L;
20       } /* End While; */
21  }
```

Notation definition:

M = $\{R_{M1}, R_{M2}, \ldots, R_{Mx}\}$ is a collection of deadline-missing requests sorted in an EDF order.
L = $\{R_{L1}, R_{L2}, \ldots, R_{Ly}\}$ is a collection of low-voltage requests sorted in an EDF order.

*** Requests in *M*, *L* preserve original order in *S***

Figure 5. O(n) On-Line-Schedule algorithm

During run-time, new requests may arrive at the system at any time. When a new request R_i arrives at the system, we first insert R_i into the original schedule S according to the order of their deadlines. The insertion of R_i has an O(n) time complexity, where n is the number of pending requests. The insertion may introduce some deadline violations, which may or may not exist already. After the insertion, the algorithm will try to satisfy the missed deadlines by servicing some requests at the high voltage level. Note that all requests in the queue are ready. If a request which was originally serviced at the low voltage level is now serviced at the high voltage level, and a speed-up of K time units is obtained, then all subsequent requests are also speeded up by K time units. The algorithm, as shown in Figure 5, is illustrated as follows:

When a new request R_i arrives, it is inserted into the original schedule S according to the order of their deadlines (Step 3). There might exist a collection of requests that miss their deadlines, i.e., M in Figure 5, with their tardy time known, where the tardy time of a request is the amount of time that the request misses its deadline. The insertion process has a time complexity $O(n)$. After the insertion, the algorithm will then try to re-satisfy missed deadlines by voltage adjustment. From Steps 5 to 20, in order to re-satisfy a particular deadline violation, the algorithm adjusts all low-voltage-serviced requests that are precedent to the deadline violation to high voltage in turn, until the deadline is satisfied. If a deadline is still not satisfied after all precedent requests are serviced at the high voltage, then the algorithm turns to process next

deadline violation. The proposed voltage adjustment algorithm is a greedy method, and it may not guarantee minimal energy consumption. (Note that internal requests mentioned in Section 3.1 could participate in the voltage adjustment and scheduling process.)

The time complexity of the on-line approximate algorithm is *O(n)*, provided that queues **L** and **M** are maintained all the time. The on-line algorithm can be modified to support *c* different voltage levels, as follows: In Step 18, the voltage level of R_L is raised by one level at a time. In Step 19, R_L is removed from *L* only if R_L is already serviced at the highest voltage level. The time complexity of the revised algorithm remains *O(c*n) = O(n)* for voltage assignment.

Lemma 1 If there exists an optimal schedule that can satisfy all requests' deadline, then the proposed on-line approximate algorithm can also satisfy all of the deadlines.

Proof: Suppose that there exists an optimal solution that could not only minimize the energy consumption but also satisfy the deadlines of all requests. Let *S* be the order of requests in the optimal solution. Because EDF is optimal in meeting requests' deadlines when all requests are ready at time *0* [12, 13], requests in *S* can be re-ordered into an EDF schedule, and the deadlines of all requests are still satisfied. As astute readers may notice, the only difference between the optimal schedule after reordering and the schedule under the proposed on-line algorithm is that the on-line approximate algorithm may unnecessarily raise the supplied voltages of many requests before R*i*, compared to the optimal solution. □

5. Experimental Results

4.1 Overview

The capability of the proposed mechanism is evaluated over a typical flash memory which supports two voltage levels, and the characteristics of the flash memory can be found in Table 1.(b). The workload traces were gathered by emulating web-surfing applications over a portable device, with a 50% capacity utilization initially. As mentioned in Section 3.1, a greedy reclaiming policy [4] was adopted in the experiments. The characteristics of the traces, the internal garbage collection activities, and the configuration of the flash memory were described in Table 2.

There were two types of metrics used in the experiments: energy consumption and performance. The total energy consumption for each experiment was measured in Joule. In order to evaluate the energy efficiency of the proposed mechanism, the number of requests serviced at the high voltage level was measured. The performance of the proposed mechanism was measured in terms of the average response time, the accumulated tardy time, and the number of deadline violations, where each (external) request was associated with a deadline to reflect the expected response time. We are interested in the performance of the proposed mechanism under limited energy consumption. We evaluated the mechanism under three voltage adjustment policies: dynamic adjustment policy, high voltage policy (5v), and low voltage policy (3.3v).

Traces' Characteristics	
File system	FAT32
Applications	Web Browser & Email Client
Sector Size	512 bytes
Duration	3.3 hours
Read / Write Ratio	48 / 52
Mean Read Size	8.2 sectors
Mean Write Size	5.7 sectors
Inter-Arrival Time	Mean: 32 ms Std. Dev. : 229 ms
Bytes Written	18.284MB
Bytes Read	23.651MB

Garbage Collection Activities	
Block Erases	2,798
Live Pages Copied	6,798

Flash Memory Characteristics	
Capacity	16 Mbytes
Block Size	16 Kbytes
Page Size	512 bytes
Initial utilization	50%

Table 2: The Characteristics of Traces and Flash Memory

5.2 Performance Evaluation over Realistic Traces

In this part of experiments, we evaluated the proposed mechanism over realistic traces. Two parameters were used to manage the workload and performance requirements of the traces: inter-arrival time and deadlines of requests. The smaller the inter-arrival time of requests, the heavier the system workload was. On the other hand, when a request was given a more urgent deadline, the system had a higher pressure to service the request in a shorter time frame. The experiments under different inter-arrival times were to evaluate the performance and behavior of the proposed mechanism under different system workloads. The experiments under different deadline/response-time requirements were to evaluate the proposed mechanism under different performance requirements.

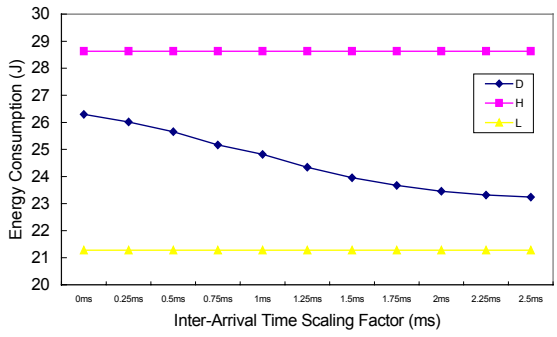


Figure 6(a). Energy Consumption

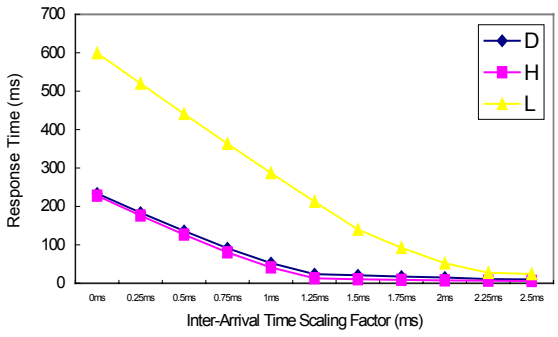


Figure 6(b). Average Response Time

5.2.1 Performance Evaluation of Request Workloads

In this part of experiments, the inter-arrival time of requests was controlled by a parameter called the *scaling factor*, which ranged from *0* ms to *2.5* ms. When the scaling factor was *x* ms, the inter-arrival time of two consecutive requests was enlarged by *x* ms. In other words, the larger the scaling factor, the lighter the workload is. The original inter-arrival time of the traces can be found in Table 2. Figures 6.(a) and 6.(b) show the energy consumption and the average response time of the traces under different voltage adjustment methods. Here the X-axis denotes the scaling factor of the inter-arrival time in ms, and the Y-axis of Figure 6.(a)/(b) denotes the total energy consumption (in Joule) and the average response time (in ms) of each (external) request. "D", "H", and "L" denote the voltage adjustment policies: dynamic adjustment policy, high voltage policy (5v), and low voltage policy (3.3v). The response time requirements of requests were set as 40ms (i.e., deadlines).

It was shown in Figure 6.(a) that the dynamic adjustment policy could substantially decrease the energy consumption when the system workload was not heavily loaded. More requests were serviced at the low voltage level since more flexibility in scheduling was allowed. When the system workload was heavy, the dynamic adjustment policy tended to service more requests at the high voltage level because of performance requirements. It was clear that the dynamic adjustment policy was effective in energy saving. Figure 6.(b) shows the average response time of (external) read/write requests under the same configuration. Note that the response time of the requests due to garbage collection was not included in the performance indices because users of flash memory would not be able to observe the response time of such requests. However, the energy consumption of traces should and did include that for garbage collection because the energy consumption of traces should include everything, such as those caused by garbage collection. In Figure 6(b), it was surprised to see that the average response time of requests under the dynamic adjustment policy was very close to that under the high

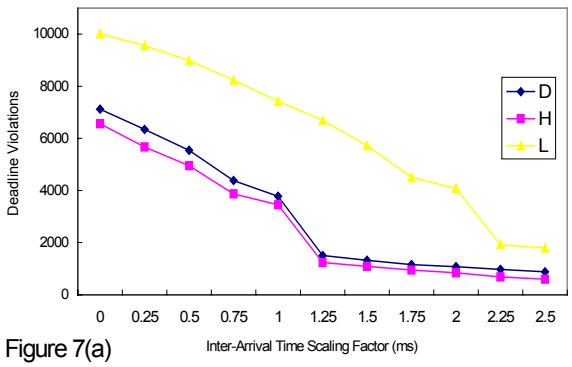


Figure 7(a)

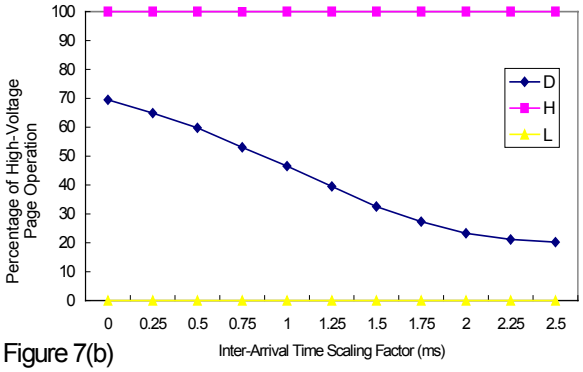


Figure 7(b)

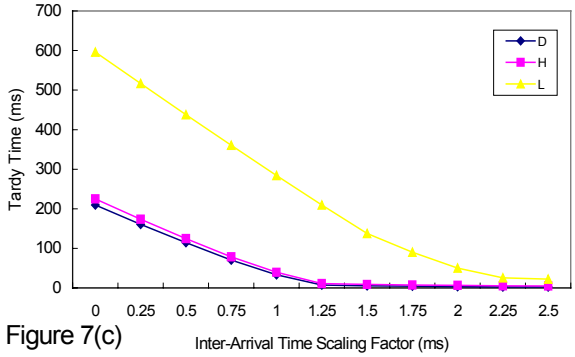


Figure 7(c)

Figure 7(a). Number of Deadline Violations
Figure 7(b). Percentage of High Voltage Page Operations
Figure 7(c). Total Tardy Time

voltage method. The performance of the dynamic adjustment policy was almost as the same as that of the high voltage, except that the dynamic adjustment policy consumed much less energy. The dynamic adjustment policy greatly outperformed the low voltage method. As the system workload dropped, the energy consumption of traces under the dynamic adjustment policy was quickly approaching that under the low voltage method.

Figure 7.(a) shows the number of deadline violations over different system workloads. As we expected, the dynamic adjustment policy eliminated nearly the same number of deadline violations serviced by high voltage policy. In the Figure 7.(a) we also observed the deadline violations decreased rapidly when system was not heavily loaded. The performance requirements were efficiently satisfied by the dynamic adjustment policy but much less energy was consumed. Figure 7.(b) showed the percentage of high voltage page operations. Note that a request may consist of a series of page operations, where the page size is 512 bytes. For example, a 40KB write request is identical to 80 page writes. Figure 7.(b) shows that the percentage of high voltage operations significantly dropped from 70% to 20% when the system workload was relaxed. Clearly, the dynamic adjustment policy had a significant portion of requests being serviced at low voltage. Figure 7.(c) shows the total tardy time. Note that the total tardy time was the sum of all deadline-missing requests' tardy time.

5.2.2 Performance Evaluation of Various Performance Requirements

In this part of experiments, we varied the deadline of read / write requests to observe the efficiency of the dynamic adjustment policy in the satisfaction of the performance

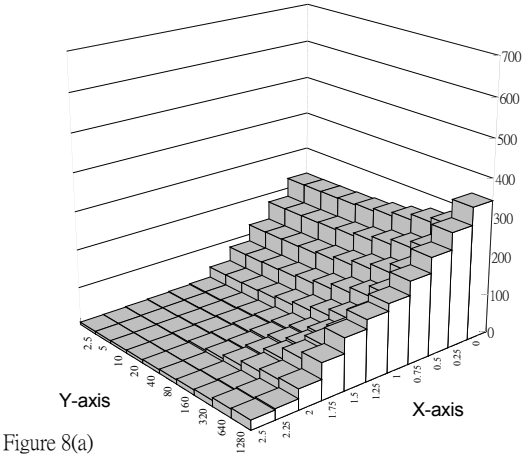


Figure 8(a)

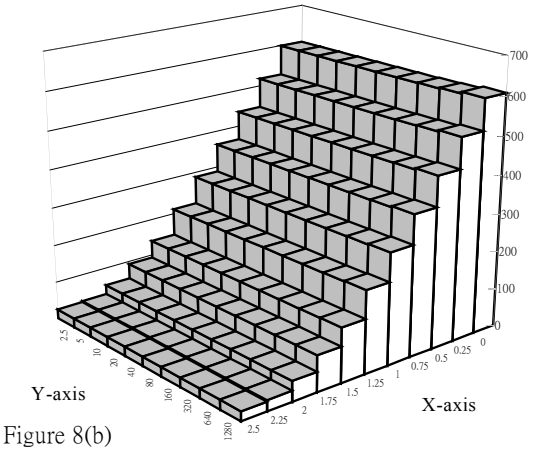


Figure 8(b)

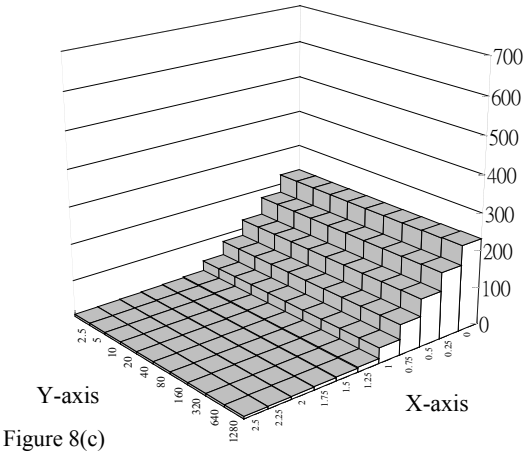


Figure 8(c)

Figure 8. Average Response Time

(a) Dynamic-Voltage-Adjustment
(b) Services at a High Voltage
(c) Services at a Low Voltage

X-axis denotes inter-arrival time scaling factor (ms).
Y-axis denotes different deadlines.
Z-axis denotes average response time (ms).

requirements. Requests were given different deadlines to reflect different performance requirements (and different scheduling windows, i.e., deadlines), where a smaller scheduling window imposed more stringent requirements for system in servicing requests. The deadline was increased from 2.5ms to 1280ms, and stepped by doubling. In this part of experiment, we assigned the same deadline for both read and write requests. We present experimental results with different deadlines and inter-arrival times to show the overall behavior of the dynamic adjustment policy.

Figure 8. shows that the average response time increased when deadlines became less urgent. When requests had less urgent deadlines, the dynamic adjustment policy would tend to increased the average response time accordingly, and the energy consumption was reduced. A similar phenomenon could also be observed in Figures 9.(b) and 9.(c). Note that the variations of deadlines had no impacts on the energy consumption under the high voltage policy and low voltage policy since no voltage adjustments were allowed. Figure 9 showed that the dynamic adjustment policy could significantly reduce the energy consumption when the performance requirements were less stringent. Figure 9.(a) shows that the impacts of deadline variation were very similar to the variation of the system workload. The energy consumption of the high voltage policy and low voltage policy didn't change when the system workload changed, as shown in Figures 9.(b) and 9.(c).

The number of deadline violations, the number of high voltage page operations, and the total tardy time are shown in Figure 10.(a), 10.(b), and 10.(c), respectively. The results are very encouraging. As shown in Figure 10.(a), the dynamic adjustment policy could meet the deadlines of requests as those serviced by the high voltage level. In other words, the system

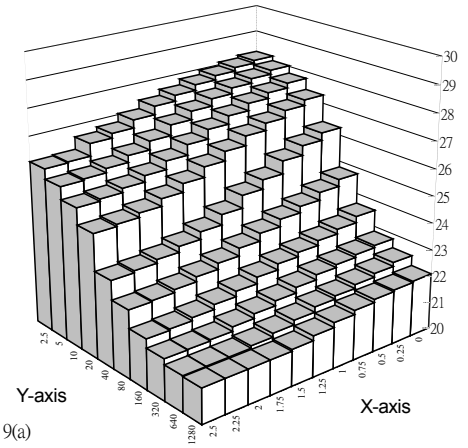


Figure 9(a)

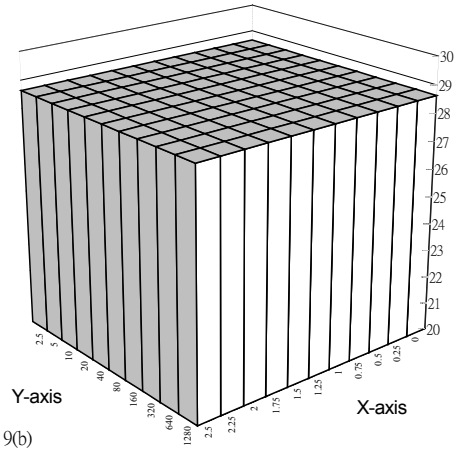


Figure 9(b)

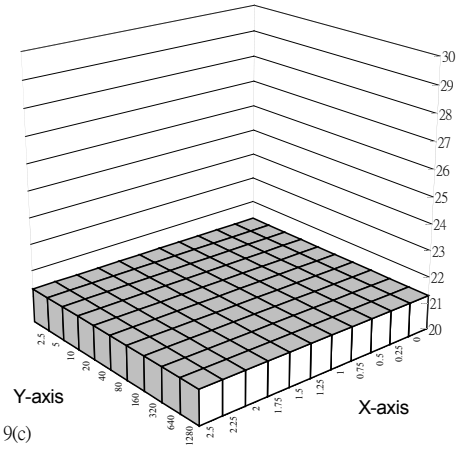


Figure 9(c)

Figure 9. Energy Consumption

(a) Dynamic-Voltage-Adjustment
(b) Services at a High Voltage
(c) Services at a Low Voltage

X-axis denotes inter-arrival time scaling factor (ms).
Y-axis denotes different deadlines.
Z-axis denotes total energy consumption (Joule).

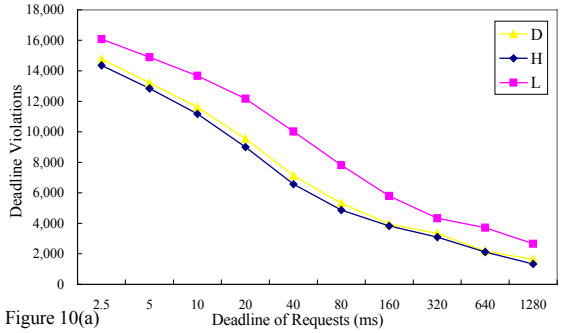


Figure 10(a)

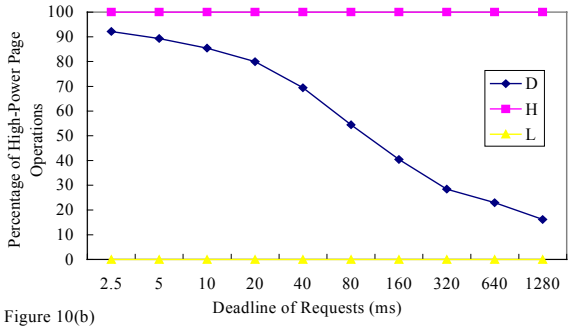


Figure 10(b)

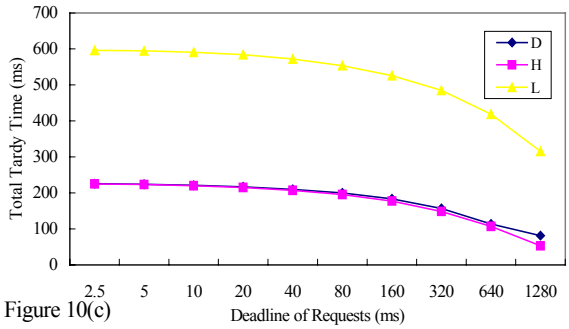


Figure 10(c)

Figure 10(a). Number of Deadline Violations.

Figure 10(b). Percentage of High-Voltage.

Figure 10(c). Total Tardy Time.

performance was fulfilled with a much less energy consumption! The same phenomenon could also be observed for the total tardy time, as shown in Figure 10.(b). The percentage of page operations serviced at the high voltage level quickly decreased whenever possible, as shown in Figure 10.(c).

6. Conclusion

In this paper, we propose a methodology for dynamic voltage adjustment of flash memory storage systems to manage the energy consumption and, at the same time, to meet performance requirements. A scheduling framework with a voltage adjustment mechanism is proposed. When the system is heavily loaded, a reasonable portion of operations will be performed at a high voltage level to deliver a reasonable performance. When the system workload is low, the proposed mechanism will service requests at a low voltage level without sacrificing the system performance. We show that the scheduling with an energy consumption constraint problem is NP-Complete. We also propose an efficient on-line scheduling algorithm with an objective to satisfy the response time of requests and, at the same time, to minimize the energy consumption of flash memory. The strength of the proposed mechanism is evaluated by a series of experiments over realistic traces and a typical multi-voltage-capable NAND flash memory, for which we have very encouraging results.

For future research, we shall further investigate joint scheduling of microprocessors and flash memory under dynamic voltage adjustment. We shall also investigate cache-related behavior and multiple resources management for energy consumption over portable devices. We believe that more research in this direction may prove being very rewarded.

REFRERENCES

[1] C.M. Krishna and Y.H. Lee, “Voltage-Clock-Scaling Adaptive Scheduling Techniques for Low Power in Hard Real-Time Systems,” *IEEE Real-Time Technology and Applications Symposium*, May 31-June 2, 2000.

[2] Y. Nakamoto, Y. Tsujino, and N. Tokura, “Real-Time Task Scheduling Algorithms for Maxumum Utilization of Secondary Batteries in Portable Devices,” *International Conference on Real-Time Computing Systems and Applications*, December 2000.

[3] A. Molano, K. Juvva, R. Rajkumar, “Real-Time Filesystems: Guaranteeing Timing Constraints for Disk Accesses in RT-Mach,” *Proceedings of the 18th IEEE Real-Time Systems Symposium*, June 1997.

[4] A. Kawaguchi, S. Nishioka, and H. Motoda, “A Flash Memory based File System,” *Proceedings of the USENIX Technical Conference*, 1995.

[5] F. Douglis, R. Caceres, F. Kaashoek, K. Li, B. Marsh, and J.A. Tauber, “Storage Alternatives for Mobile Computers,” *Proceedings of the USENIX Operating System Design and Implementation*, 1994.

[6] C.L. Liu and J.W. Layland, “Scheduling Algorithms for Multiprogramming in a Hard Real Time Environment,” *Journal of the ACM* , 1973.

[7] M. Weiser, B. Welch, A. Demers and S. Shenker, “Scheduling for Reduced CPU Energy,” *Proceedings of the USENIX Operating System Design and Implementation*, 1994, pp. 12-23.

[8] F. Yao, A, Demers and S. Shenker, “A Scheduling Model for Reduced CPU Energy,” *Proceedings of the 36th IEEE Symposium Foundations of Computer Science*, 1995, pp. 374-382.

[9] Inki Hong, Darko Kirovski, Gang Qu, Miodrag Potkonjak, and Mani B. Srivastava, “Power Optimization of Variable-Voltage Core-Based Systems,” *IEEE Transaction On Computer-Aided Design of Integrated Circuits and Systems*, Vol. 10, No. 12, December 1999.

[10] K. Han-Joon, and L. Sang-goo, “A New Flash Memory Management for Flash Storage System,” *Proceedings of the Computer Software and Applications Conference*, 1999.

[11] M. L Chiang, C. H. Paul, and R. C. Chang, “Manage flash memory in personal communicate devices,” *Proceedings of IEEE International Symposium on Consumer Electronics ’97*, 1997.

[12] Stankovic, J.A.; Spuri, M.; Di Natale, M.; Buttazzo, G.C. , “Implications of Classical Scheduling Results for Real-Time Systems,” *IEEE Computer*, vol 28, issue 6, 1995.

[13] Michael R. Garey, David S. Johnson, “Computers and intractability”, 1979.

[14] Li-Pin,Chang, Tei-We Kuo, Shi-Wu Lo, “A dynamic-Voltage-Adjustment Mechanism in Reducing the Energy consumption of Flash Memory for Portable Devices,” *IEEE International Conference on Consumer Electronics*, 2001.

[15] W. Namgoong, M. Yu, and T. Meng, ”A high-efficiency variable-voltage CMOS dynamic DC-DC switching regulator,” *Proceedings of IEEE International Solid-State Circuits Conference*, 1997

[16] Michael Wu, Willy Zwaenepoel, “eNVy: A Non-volatile, Main Memory Storage System”, *Proceedings of Architectural Support for Programming Language and Operating System*,

1994.

[17] A. P. Chandrakasan, S. Sheng, and R. W. Broderson, “Low-power CMOS digital design,” *IEEE Journal on Solid-State Circuit*s, vol. 27, no. 4, pp.473–484, 1992.

[18] Kester Li, Roger Kumpf, Paul Horton, and Thomas Anderson, “A Quantitative Analysis of Disk Drive Power Management in Portable Computer”, *Proceedings of Winter 1994 USENIX Conference*, 1994

[19] Anastasio Molano, Kanaka Juvva, and Ragunathan Rajkumar, “Real-Time Filesystems: Guaranteeing Timing Constraints for Disk Accesses in RT-Mach,” *Proceedings of Real-Time System Symposium,* 1997.

[20] SSFDC Forum, “SmartMedia™ Specification,”, 1999.

[21] Compact Flash Association, “Compact Flash™ 1.4 Specification,” 1998.

[22] Hitachi, “Hitachi HN29W12814A AND-type Flash Memory Datasheet,” 1999

[23] M-System, “DiskOnChip Millennium Datasheet,” 2000

[24] Intel, “28F016S5 5-Volt FlashFile Flash Memory Datasheet”, Intel Corporation, 1999

[25] Samsung Electronics Company, “SmartMedia™ White Paper,” 2000

[26] I_2O Special Interest Group, “I_2O specification 2.0”, http://www.intelligent-io.com

[27] B. Dipert, and M. Levy, “Designing with Flash Memory”, Annabooks, 1994

[28] NAND Flash memory Translation Layer (NFTL), Linux MTD project

[29] Kevin Jeffay, Donald F. Stanat, and Charles U. Martel, “On Non-Preemptive Scheduling of Periodic and Sporadic Tasks”, *Proceedings of Real-Time System Symposium*, 1991