SOFTWARE—PRACTICE AND EXPERIENCE
Softw. Pract. Exper., **29**(3), 267–290 (1999)

Using Data Clustering to Improve Cleaning Performance for Flash Memory

MEI-LING CHIANG[1], PAUL C. H. LEE[2] AND RUEI-CHUAN CHANG[1,2]*
[1]*Department of Computer and Information Science, National Chiao Tung University, 300, Hsinchu, Taiwan, ROC*
(email: joanna@os.nctu.edu.tw, rc@cc.nctu.edu.tw)
[2]*Institute of Information Science, Academia Sinica, Nankang 115, Taipei, Taiwan, ROC*
(email: paul@iis.sinica.edu.tw)

SUMMARY

Flash memory offers attractive features for storage of data, such as non-volatility, shock resistance, fast access speed, and low power consumption. However, it requires erasing before it can be overwritten. The erase operations are slow and consume comparatively a great deal of power. Furthermore, flash memory can only be erased a limited number of times. To overcome hardware limitations, we use the non-in-place update mechanism that requires a cleaner to reclaim space occupied by obsolete data. To improve cleaning performance and prolong flash memory lifetime, we propose a new data reorganization method. By this method, data in flash memory are dynamically classified and clustered together according to their accessing frequencies. Experimental results show that this clustering technique significantly improved the cleaning performance for a variety of cleaning algorithms. The number of erase operations performed is greatly reduced and flash memory lifetime is prolonged. Even wearing is ensured as well. **Copyright © 1999 John Wiley & Sons, Ltd.**

KEY WORDS: flash memory; cleaning policy; mobile device; consumer electronics; data clustering

INTRODUCTION

Flash memory currently has been widely used in mobile devices, consumer electronics, and embedded applications. Due to its attractive advantages in accessing-speed, shock-resistance, non-volatility, power-consumption, etc., flash memory shows more promise than conventional storage media like EPROM and hard disks [1–6]. However, due to its hardware characteristics, flash memory also asks for specific operations in using it [3].

Three primary operations, the read, write and erase, are defined as being responsible for accessing flash memory. Read operations are most efficient (typically 150–250 nanoseconds/byte) than write (typically 6–9 microseconds/byte) and erase operations. Applications can efficiently read and write data to the addresses within the flash memory. However, for manipulating the erase operations, the whole storage space of flash memory is partitioned into segments† in fixed size. The hardware manufacturers define the segment sizes

*Correspondence to: R.-C. Chang, Department of Computer and Information Science, National Chiao Tung University, 300, Hsinchu, Taiwan, ROC.
Contract/grant sponsor: National Science Council of the Republic of China; Contract/grant number: NSC 88-2213-E-001-016.
†We use ‘segment’ to represent hardware-defined erase block, and ‘block’ to represent software-defined block.

CCC 0038–0644/99/030267–24$17.50
Copyright © 1999 John Wiley & Sons, Ltd.
Received 6 July 1998
Revised 16 November 1998
Accepted 16 November 1998

(typically 64 Kbytes). Each segment is the basic unit in erasing the flash memory. Typically, to erase a segment costs about 0.6–0.8 seconds, which is slow and consumes comparatively a great deal of power.

Another limitation about flash memory is that it cannot be written over existing data unless the whole segment containing that data is erased in advance. The number of times a segment can be erased is also limited typically by 100,000 times to 1,000,000 times. So if the flash memory is not evenly erased, the flash memory will soon be worn out, even though the total utilization of this flash memory is still low. Concluded from above discussions, in order to access flash memory efficiently and be more power-saving, the erase operations should be avoided as much as possible, and should be operated evenly over the whole flash memory. These are the primary principles in implementing flash-memory based storage systems.

Flash-memory based storage systems are commonly implemented in two ways. They are either totally designed from scratch [6,7] or are constructed by using file systems plus new device driver implementations [5,8]. In the later approach, the file system does not care which media it uses to store data, and just generates requests to the drivers in disk block and sector numbers. It is the device driver itself needs to translate the disk block accesses to flash memory addresses. Evidently, no matter which approach is used, it is important to avoid having to erase during every data update. A mechanism, named *non-in-place update* mechanism, is generally used to achieve this expectation [5–10]. Under this mechanism, data updates are written to empty flash memory space and obsolete data are set invalidated as garbage. A software *cleaner* later reclaims garbage by migrating valid data from the segment to be cleaned to another free segment, and then erasing the original segment. With this non-in-place update mechanism, the cleaner has a substantial impact on the number of erase operations induced, which in turn impacts system performance, flash memory lifetime, power consumption, etc [3,4,6,10].

Two major concerns regarding policies in controlling the cleaner are the *segment selection algorithm* and *data redistribution method (or data reorganization method).* The segment selection algorithm determines which segments to be cleaned, while the data redistribution method determines how to migrate valid data in the selected segments. Previous study [10] showed that data redistribution has greatly impacts on cleaning performance. The challenge is how to distribute the data.

Motivation

In this paper, we tried to find an effective data redistribution method to reduce the number of erase operations induced by flash memory cleaner. The simplest way is to copy valid data to another free segment in the same order as they appear in the original segment. But this does nothing contributed to reduce the number of erase operations. If data are migrated in the way that *hot data* (most frequently updated data) are clustered in the same segments, then flash segments will be either full of all hot data or all non-hot data. Because hot data have high possibility to be updated soon to cause the original copy to become garbage, segments containing most of the hot data would soon contain the most amount of garbage [10–12]. To clean these segments then can reclaim the largest amount of garbage. Cleaning is thus less needed. As a result, less erasures are performed and less amount of data are migrated, which significantly reduce the cleaning costs. The remaining problem is how to effectively cluster hot data.

Our approach is to classify data according to their write access frequencies and dynamically cluster them at the time when the data is updated or when the segments are cleaned.

Copyright © 1999 John Wiley & Sons, Ltd.

This approach is low overhead and data classification is fine-grained. Empirical evaluations through practical implementation and simulations showed that this approach significantly reduces the number of erase operations performed. Flash memory lifetime is thus prolonged, throughput is improved, and power is more saving.

RELATED WORK

Many storage systems adopt the non-in-place update scheme that requires garbage collection to reclaim space occupied by obsolete data [5–17]. The Log-Structured File System (LFS) [11–14] is a representative. Since garbage collection in LFS is similar to flash memory cleaning, several cleaning policies proposed in LFS have been employed in flash memory storage systems [5]. The *greedy* policy, which selects the segment with the largest amount of garbage for cleaning, was shown to perform well for uniform accesses but poorly for high localities of reference [6,9–12]. The *cost-benefit* policy, which considers not only the amount of garbage, but also the age of data, was shown to outperform greedy policy for high localities of reference [9–12]. The *age sorting* method [11,12], which sorts data blocks by age before writing them on disks, is used to separate hot data from cold data. Several segments are cleaned at once.

HP AutoRAID [15], a two-level disk array structure, uses the *hole plugging* method in garbage collection. This method reclaims a segment by overwriting its valid data to other segments' *holes* (space occupied by obsolete data). The *adaptive cleaning* policy [16] incorporates this hole plugging into traditional LFS cleaning. This policy adapts to changes in disk utilization by dynamically choosing cost-benefit policy or hole-plugging policy.

Logical Disk (LD) [17] maintains a logical block map to turn an existing file system into an LFS-like file system. LD supports the abstraction of *block lists*, which allows file systems to express the logical relationships among blocks. It provides a data clustering method: when migrating valid blocks from segments to be cleaned, a segment cleaner uses the list information to physically cluster related blocks to improve future read performance. Its clustering effectiveness depends on the correct specifications of block lists.

Douglis *et al.* [4] discussed storage alternatives for mobile computers. They showed that the key to flash memory file systems is erasure management. They also found that flash memory utilization (the percentage of flash memory space occupied by valid data) has substantial impacts: for 90 per cent utilization, energy consumption is increased by 70–190 per cent, write response time is degraded by 30 per cent, and lifetime is decreased by up to a third, as compared with 40 per cent utilization. In addition, at 90 per cent utilization or above, an erasure unit much larger than the file system block size would result in much unnecessary copying.

Microsoft Flash File System (MFFS) [7] provides complete file system capabilities for DOS. MFFS uses linked lists to store and manage data in flash memory. Data are allocated as variable-sized regions instead of fixed-sized blocks. The greedy policy is used in cleaning. Douglis *et al.* [4] reported the poor performance of MFFS when accessing large files. Its write performance degrades linearly with the growth of file size.

M-Systems TrueFFS [8] allows flash memory to emulate hard disks and provides DOS and Windows file compatibility. TrueFFS is a software block device driver to be used with an existing file system. The driver manages flash memory space as fixed-sized blocks and is responsible to translate the file system requests from disk sectors to flash memory blocks. Its garbage collection selects segments with the large amount of garbage, the least number of erasures, and the most static data. It then decides which segments to clean. To ensure the

Copyright © 1999 John Wiley & Sons, Ltd.

evenly reclamation among all segments, a random selection process is also used. TrueFSS uses a statistical approach to wear-leveling.

Linux PCMCIA [18] flash memory drivers [19,20] also uses the greedy policy in cleaning but sometimes chooses to clean the segment that has been erased the fewest number of times. The idea is to avoid concentrating erasures on a few segments.

eNVy [6], a large flash memory-based storage system, provides flash memory as a linear memory array rather than an emulated disk. eNVy uses hardware support of copy-on-write and page-remapping techniques to provide transparent in-place update semantics. Its *hybrid cleaning* policy combines *FIFO* and *locality gathering* to minimize the cleaning costs for uniform access and high locality of reference. Their simulations of a 2-gigabyte eNVy system showed that it could support 30,000 transactions per second using the TPC-A database benchmark.

Kawaguchi *et al.* [5] based on LFS to design a flash-memory based file system. The device driver approach is used, which emulates a hard disk and supports a conventional UNIX file system transparently. They modified LFS's cost-benefit policy but used a different cost measure. During cleaning, the *separate segment cleaning* method is used in data clustering. That is, two segments are used: one for cleaning cold segments and one for cleaning the non-cold segments and writing the data. Hot data are thus less likely to be mixed with cold data. Wear leveling is not implemented in their work. Their evaluations showed that separate segment cleaning performs better than using only one segment for both the data writing and the cleaning operations. Its performance is comparable to the 4.4BSD Pageable Memory Based File System [21].

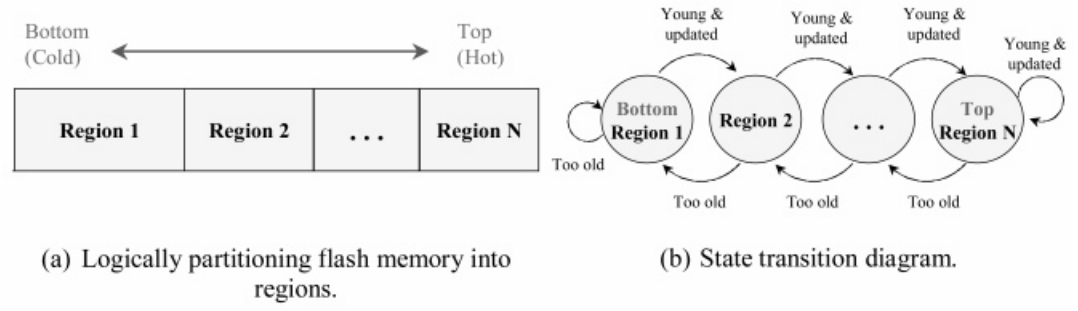
In our early study, the *CAT* policy [9] is proposed to take into account utilization, segment age, and the number of times segments have been erased in selecting segments to clean. Because the number of erase operations performed on individual segments is concerned, flash memory is more evenly worn than greedy policy and cost-benefit policy. Valid blocks in the segments to be cleaned are migrated into separate segments depending on whether the blocks are hot or cold. The CAT policy and the cost-benefit policy were shown to outperform the greedy policy for high localities of reference but do not perform as well as the greedy policy for uniform access [9,10]. Data reorganization was shown to be the most important factor affecting cleaning performance [10].

FLASH MEMORY MANAGEMENT USING DATA CLUSTERING

Data reorganization by separating hot data from cold data can reduce cleaning overhead [3,5,6,10–12]. Previous research [5,6,9,10] reorganizes data only at cleaning time when migrating valid data in the segment that is being cleaned. We propose a new data reorganization method: **DAC** (**D**ynamic d**A**ta **C**lustering) approach. This approach dynamically clusters data not only during segment cleaning, but also during data update. This is motivated by the fact that when data blocks are updated, they are updated to another free flash space. Then hot data and cold data can be separately clustered at this time by updating them to separate flash memory spaces. This approach is detailed below.

The approach logically partitions the flash memory into several *regions*, as shown in Figure 1(a). Each region consists of a set of flash segments that need not physically contiguous. The idea is to cluster data blocks of the similar write access frequencies in the same regions. Only write operations are concerned because read operations do not incur cleaning. Since data access frequencies may change over time, the basic operation is to actively migrate data blocks between regions when their access frequencies change. That

Copyright © 1999 John Wiley & Sons, Ltd.



(a) Logically partitioning flash memory into regions.

(b) State transition diagram.

Figure 1. State machine for data clustering

is, data blocks are moved toward the *Top* region (i.e. the hottest) if their update frequencies increase, whereas they are moved toward the *Bottom* region (i.e. the coldest) if their update frequencies decrease. So regions can be dynamically shrunk or enlarged.

A state machine is used for the region switching. The state machine contains several states and the state transition diagram is shown in Figure 1(b). Each data block is associated with a state indicating the region it resides in. The starting state is '*Bottom region*', where newly created data blocks reside. The state switching occurs only when data blocks are updated or when garbage collection occurs. When a data block is updated, it is 'promoted' to the upper region toward the Top. That is, the obsolete data block in the original region is invalidated as garbage and the update data are written to free space in the upper region. When a segment is selected for cleaning, all of its valid data blocks are 'demoted' to the lower region toward the Bottom. That is, all valid blocks migrate back to the lower region by being copied into free space in the lower region.

Because the degree of hotness for a block is determined by the number of times the block has been updated but degrades as the block's age grows, we thus add an additional criterion for state switching: the time threshold. If a block is to be promoted, it also has to be young for the current region (i.e. the resident time in the current region is smaller than a certain threshold). Otherwise, the update data are written to the free space in the current region. If a block is to be demoted, it also has to be old for the current region (i.e. its resident time exceeds a certain threshold). Otherwise, the block is migrated to the free space in the current region.

By this active data migration between neighboring regions during data updating time and during cleaning time, *Top* region will gather the most frequently updated data during the recent accesses. The closer to the Top region, the hotter the block is; otherwise, the colder it is. Therefore, data blocks of similar write access frequencies can be effectively clustered. Figure 2 shows the detailed operations.

The advantages of DAC method can be summarized as follows. First, data are clustered in a low-overhead way during data updating and during segment cleaning. So complex computations for determining data as hot or cold are not needed whereas they were needed in previous research [5,9,10]. Second, data classification is more fine-grained. Instead of classifying data into hot and cold as in previous research [5,9,10], the DAC approach is more fine-grained since more states of data are allowed depending on the configuration of the state machine. Since data reorganization by separating hot data from cold data can reduce cleaning overhead [3,5,6,10–12], this fine-grained classification is expected to perform better than

Copyright © 1999 John Wiley & Sons, Ltd.

```
Write()
{
    If newly write {
        Allocate a free block in Bottom region;
        Write data into the free block;
    } Otherwise
        /* non-in-place update */
        Mark the obsolete data as invalid;
        If the data block is young to the current region
            Allocate a free block in the upper region;
        Otherwise
            Allocate a free block in the current region;
        Write data into the free block;
    }
}
Cleaning()
{
    Select a victim segment for cleaning;
    For all valid data blocks in the victim segment
    {
        Mark the block as invalid;
        If the data block is old to the current region
            Allocate a free block in the lower region;
        Otherwise
            Allocate a free block in the current region;
        Copy this valid data block into the free block;
    }
    Erase the victim segment;
    Enqueue the victim segment to free segment list;
}
```

Figure 2. Algorithms for write and cleaning

classifying data into only hot and cold. Ideally, more regions are better since data blocks are classified more fine-grainedly. However, too many regions will cause total free flash memory space to be fragmented among more regions since each region needs free space to accept data writing, such that the available free segments that can be assigned to regions are reduced. This fragmentation turns out to cause erasures.

Regarding to the clustering time and clustering method used in other storage systems [5,9–12], Table I summarizes the comparison of various data-clustering methods.

DESIGN AND IMPLEMENTATION OF A FLASH MEMORY SERVER

A flash memory server providing the DAC data clustering was implemented. The server manages flash memory as fixed-size blocks. Every data block is associated with a unique *logical block number*. Since the non-in-place update scheme is used, when data blocks are updated, their physical locations in flash memory change. The server then uses a table-mapping method to map logical block numbers to physical locations.

Copyright © 1999 John Wiley & Sons, Ltd.

Table I. Comparison of various data-clustering methods used in storage systems

Storage systems	LFS [11,12]	Flash memory-based file system [5]	Flash memory server [9,10]	DAC server
Clustering method	*Age Sorting* (i.e. sort data by age)	*Separate Segment Cleaning (Segment-based)* (i.e. classify data into hot and cold)	*Separate Segment Cleaning (Block-based)* (i.e. classify data into hot and cold)	*Dynamic Data Clustering* (i.e. classify data by their update frequencies)
Clustering time	Cleaning	Cleaning	Cleaning	Cleaning and data update time

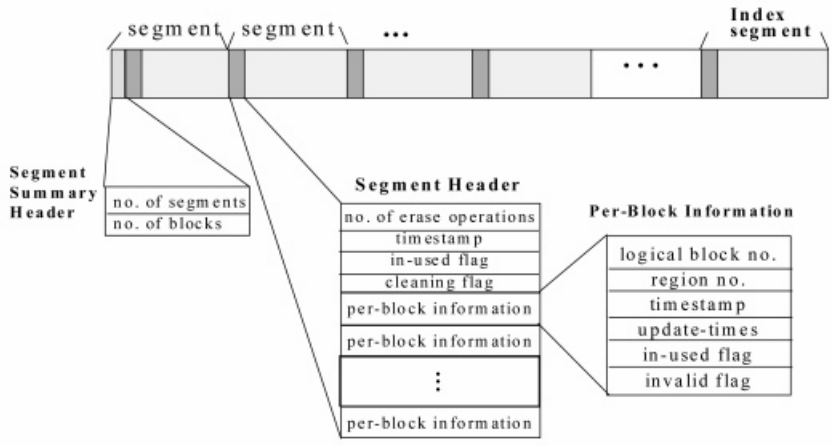


Figure 3. Data layout on flash memory

We first describe data layout on flash memory, then introduce three tables that the server uses to speed up processing. They are translation table, region table, and lookup table. These tables are constructed in main memory during server startup time by obtaining segment information from flash memory and are maintained during runtime. The information stored in the tables is only a copy of information stored in flash memory. Therefore, even if power failures occur, these tables can be reconstructed from flash memory.

Data layout on flash memory

Figure 3 shows the data layout on flash memory. Each segment has a *segment header* to record segment information such as the number of times the segment has been erased, *per block information array*, etc. The per-block information array describes every block in the segment, such as logical block number, region number, the number of times the block has been updated, flags indicating free, valid, or obsolete, etc. The *segment summary header* describes the whole flash memory, including total number of flash segments and total number of blocks per segment. The final segment, *index segment*, keeps track of those *active segments* that are currently used for data writing in each region.

Copyright © 1999 John Wiley & Sons, Ltd.

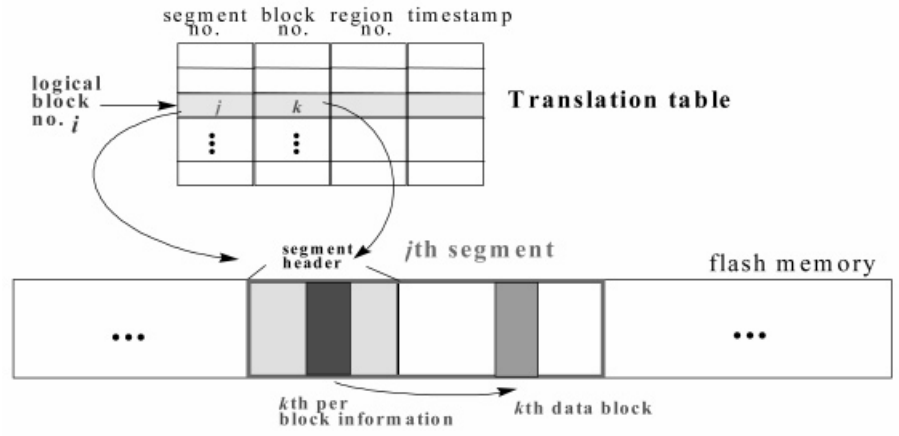


Figure 4. Translation table and address translation

Block-based translation table

The *translation table*, shown in Figure 4, records the physical locations for each logical block. This table is constructed to speed up the address translation from logical block numbers to physical addresses in flash memory. When a block is updated to another new empty block, the old block's per-block information is marked *invalid* and the new block's per-block information records the logical block number. The corresponding translation table entry is also updated to record the current physical location.

Each table entry also contains a region number indicating which region the block belongs to and a timestamp indicating when this block was allocated in the region. The DAC state machine uses this information to decide whether a block's state should be switched.

Region management

The *region table*, shown in Figure 5, keeps track of information for each region, such as the *active segment*, a *region segment list*, etc. The active segment indicates the segment currently used for data writing in the region, while the region segment list keeps track of each segment in the region.

A *free segment list* records the available free segments. Initially, the server reads segment headers from flash memory to identify free segments to construct this free segment list at server startup time. When an active segment is out of free space, a segment taken from the free segment list is used as the active segment. In the meantime, the change of active segment is written to the index segment as an appended log. When the index segment has no free space, it is erased first before wrapping around the log. When the number of free segments falls below a certain threshold, the cleaner is activated to reclaim the garbage.

Segment-based lookup table

The *lookup table*, shown in Figure 6, contains segment information duplicated from segment headers on flash memory. Each table entry also contains a counter for counting the number of valid blocks in the segment. The cleaner then uses this table to speed up the

Copyright © 1999 John Wiley & Sons, Ltd.

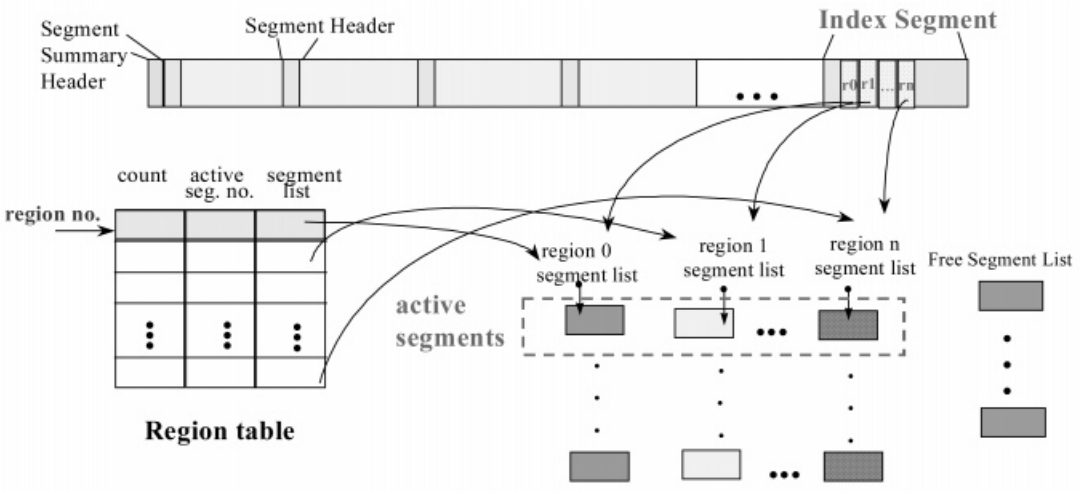


Figure 5. Region table and region segment lists

	Erase count	Timestamp	Used flag	Cleaning flag	Valid blocks count	First free block
	.	.	.	.	.	.
Segment no. *i*						
	.	.	.	.	.	.

Figure 6. Lookup table to speed up cleaning

selection of segments for cleaning. An indicator, indicating the first free block available for writing in the segment, is also included to speed up the block allocation.

SIMULATION STUDIES

Trace-driven simulation was performed to examine the effect of DAC data clustering on cleaning performance. The impact of flash memory utilization, flash memory size, and degree of locality of reference were examined in detail as well. To evaluate the cleaning effectiveness, we first devised several metrics. We then introduce the simulator and traces. Finally, we present simulation results.

Metrics

To measure the amount of work involved in cleaning, we devised a formula to express flash memory cleaning cost. Since, before a segment is erased and reclaimed, valid data in the segment should be migrated by copying to free space in other segments. Thus, the flash memory cleaning cost includes erasure cost and migration cost, which can be expressed as the following formula:

$$CleaningCost_{flash\ Memory} = NumberOfErase * (EraseCostPerSegment + MigrateCost_{valid\ data}) \quad (1)$$

The cost of each erasure on a segment is constant regardless the amount of valid data in the segment [3,22,23], whereas the migration cost is determined by the amount of data migrated

Copyright © 1999 John Wiley & Sons, Ltd.

during cleaning. The larger the amount of valid data, the higher the migration cost. However, the erasure cost dominates the migration cost. In order to provide better performance and prolong flash memory lifetime, the primary goal is to minimize the number of erase operations performed. The secondary goal is to minimize the number of blocks copied during cleaning.

Because the time to write a whole segment is about a constant ratio of the time to erase a whole segment, we can express the ratio of write time to erase time as *WriteToEraseTimeRatio*. The write cost per segment then can be expressed as:

$$WriteCostPerSegment = WriteToEraseTimeRatio * EraseCostPerSegment \tag{2}$$

Then formula (1) can be re-formulated as

$$\begin{aligned} &CleaningCost_{flash\ Memory} \\ &\quad = NumberOfErase * EraseCostPerSegment \\ &\qquad + \frac{TotalNumberOfBlocksCopied}{NumberOfBlocksPerSegment} * WriteCostPerSegment \\ &\quad = NumberOfErase * EraseCostPerSegment \\ &\qquad + \frac{TotalNumberOfBlocksCopied}{NumberOfBlocksPerSegment} * WriteToEraseTimeRatio * EraseCostPerSegment \\ &\quad = EraseCostPerSegment * \\ &\qquad \left(NumberOfErase + \frac{TotalNumberOfBlocksCopied}{NumberOfBlocksPerSegment} * WriteToEraseTimeRatio \right) \end{aligned} \tag{3}$$

Since erasure cost per segment is constant, we then use the following simplified formula derived from formula (3) as the metric to compare the cleaning effectiveness of various cleaning policies:

$$\begin{aligned} &SimplifiedCleaningCost_{flash\ Memory} \\ &\quad = NumberOfErase + \left(\frac{TotalNumberOfBlocksCopied}{NumberOfBlocksPerSegment} * WriteToEraseTimeRatio \right) \end{aligned} \tag{4}$$

Since another important goal for flash memory storage systems is wear leveling, the *degree of uneven wearing* [10] that indicates the variance of wearing for flash segments is also used as a metric. Because the segment header of each flash segment has recorded the number of times the segment has been erased, we created a utility to read them out from flash memory segments and compute the standard deviation of these numbers as the degree of uneven wearing. The smaller the standard deviation, the more evenly the flash memory is worn.

Simulator

Our flash simulator completely simulated the flash memory server except that it stores data in a large memory array instead of flash memory. The simulator accepts the parameters as shown in Table II. To demonstrate the DAC data clustering is very effective in reducing the cleaning costs for various cleaning policies, the following three segment selection algorithms are also implemented:

(a) greedy policy (**Greedy**)
The cleaner selects the segment with the largest amount of invalid data for cleaning.

Copyright © 1999 John Wiley & Sons, Ltd.

Table II. Simulator parameters

Flash size	Flash memory size.
Flash segment size	The size of an erasure unit.
Flash segment lifecycle	Program/erase cycles.
Flash block size	The block size that the server maintains.
Flash storage utilization	The amount of initial data, relative to flash size. Data are preallocated in flash memory at the start of simulation.
Number of regions	Number of regions (i.e. the states of the DAC state machine).
Segment selection algorithm	Algorithms to select segments for cleaning.

(b) cost-benefit policy [5] (**Cost-benefit**)
The cleaner chooses to clean segments that maximize the formula: $a*(1-u)/2u$, where u is flash memory utilization and a (age) is the time since the most recent modification.
(c) CAT policy [10] (**CAT**)
The cleaner chooses to clean segments that minimize the formula: $u/((1-u)*a)*t$, where u is utilization, a is segment age, and t is the number of times the segment has been erased.

For simplicity, the simulator assumes each request can be finished before arrival of next request. The number of erase operations performed on segments, the number of blocks copied, the simplified cleaning cost, and the degree of uneven wearing are reported for each simulation.

Traces

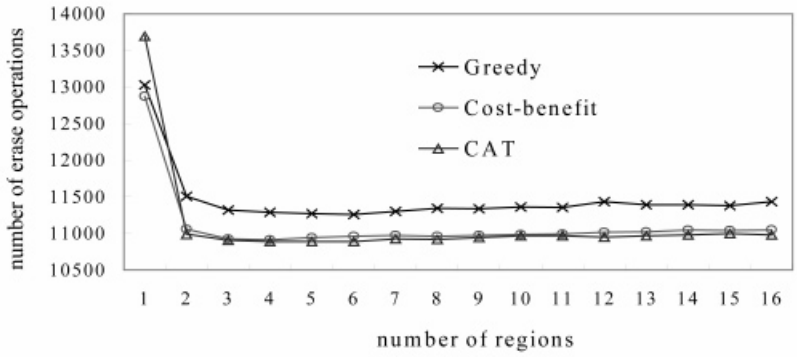
Ruemmler and Wilkes [24,25] have collected the disk-level traces of HP-UX workstations at Hewlett-Packard Laboratories. The traces cover two months of activities on three different systems: a time-sharing computer, a file server, and a personal workstation. The personal workstation (**hplajw**) was used mainly for email and document editing. Since the usage behaviors of personal computers are likely to be similar to what would be used on mobile computers, hplajw traces were often used in simulations of mobile computers [4,10,26–28], where flash memory products are typically applied. So our simulations used the hplajw traces to drive the simulator.

The traces exhibit high locality of reference, where 71.2 per cent of writes were to metadata [24]. Since flash memory capacity is currently small, we do not expect flash memory to contain swap space, so traces from swap partition were excluded. This exclusion should not have much impact on the correctness of results since swap partition occupies only 7.1 per cent of the writes in the traces [24]. In total, 1331 Mbytes of data were written.

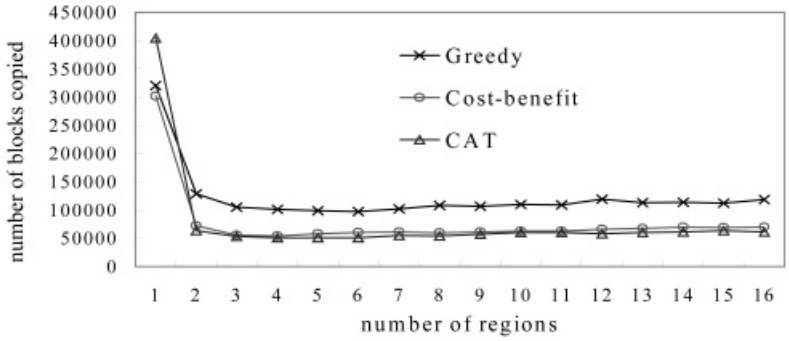
Simulation results for hplajw traces

Though flash memory capacity is currently small, the capacity is increasing as hardware technologies advance. So we simulated the flash memory as large as the hard disk used in the hplajw (i.e. 278 Mbytes). Because of the need of extra space for segment headers and cleaning, the simulated flash memory was 283 Mbytes with 128-Kbyte erase segments. The server maintains data in 1-Kbyte blocks. Because the typical time to write a segment is 0.4–0.6 seconds and to erase a segment is 0.6–0.8 seconds [22,23], we used 0.75 as the ratio of write time to erase time in calculating the cleaning cost.

Copyright © 1999 John Wiley & Sons, Ltd.



(a) Number of erase operations.



(b) Number of blocks copied during cleaning.

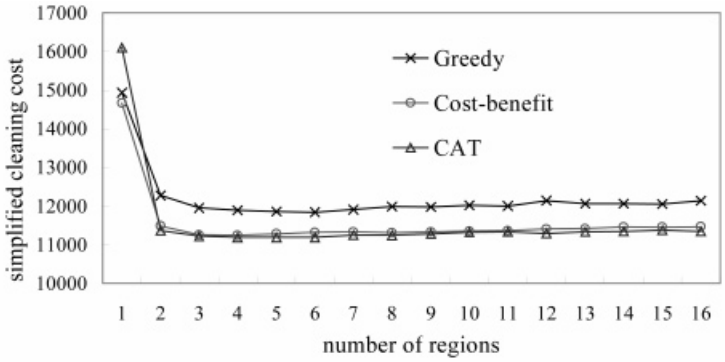
Figure 7. Effect of DAC data clustering. The flash memory was 283 Mbytes with 128-Kbyte erase segments and the block size was 1 Kbytes. The flash memory utilization was set to 85 per cent initially and hplajw traces were used

The effect of DAC data clustering

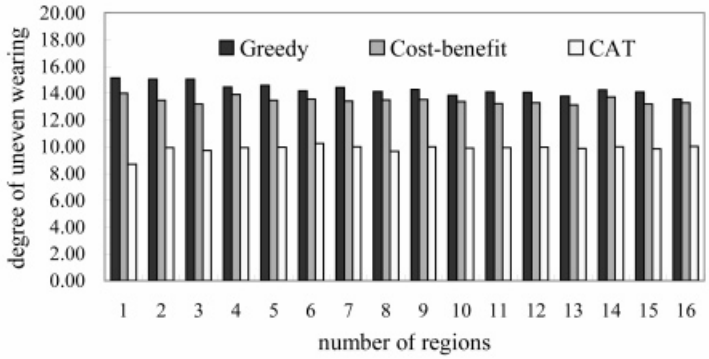
We first measured the effect of DAC data clustering for various cleaning policies. The number of regions (i.e. the number of states in the DAC state machine) ranged from 1 to 16. Because the time threshold for state switching will affect the performance, to fairly compare the effectiveness of various cleaning policies, the time threshold was not set. That is, a block's state is promoted every time it is updated and demoted every time the segment it belongs to is selected for cleaning. The effect of time threshold was measured in the next simulation. Because cleaning activities were low under low utilization, in order to measure the cleaning overhead, flash memory utilization was initially set to 85 per cent by writing enough blocks in sequence to fill the flash memory to 85 per cent of flash memory space, then hplajw traces were used.

Figure 7 shows the results. When DAC data clustering was not used (i.e. the number of regions is 1), each policy incurred high cleaning cost. However, when DAC data clustering was used (i.e. the number of regions is more than 1), the numbers of erase operations performed and blocks copied for each policy were significantly reduced. Cleaning cost was

Copyright © 1999 John Wiley & Sons, Ltd.



(c) Cleaning cost.



(d) Degree of uneven wearing

Figure 7. Continued

thus greatly reduced. For example, CAT incurred 19.7–20.5 per cent fewer erase operations, 76–87.4 per cent fewer blocks copied, and 29.3–30.5 per cent less cleaning cost, as shown in Figures 7(a)–(c). Among all algorithms, CAT performed best and Greedy performed worst.

The results also show that for this workload, the number of erasures did not have prominent variance when the number of regions was greater than 3. This is because the further reduction of erasures is little as the number of regions is increased, while increasing the number of regions causes the effect of fragmentation of free space among more regions. This fragmentation turns out to incur erasures. Therefore, the number of regions was set to 4 in the rest of simulations.

Figure 7(d) shows that CAT performed best in the wear leveling whereas flash memory was worn unevenly for Greedy and Cost-benefit. This is because only CAT formula takes even wearing into account in selecting segments to clean.

Varying the time threshold for state switching

The *time threshold for state switching* is an additional parameter to control whether a block's state must be promoted when the block is updated or must be demoted when the segment the block belongs to is selected for cleaning. Figure 8 shows the results of varying

Copyright © 1999 John Wiley & Sons, Ltd.

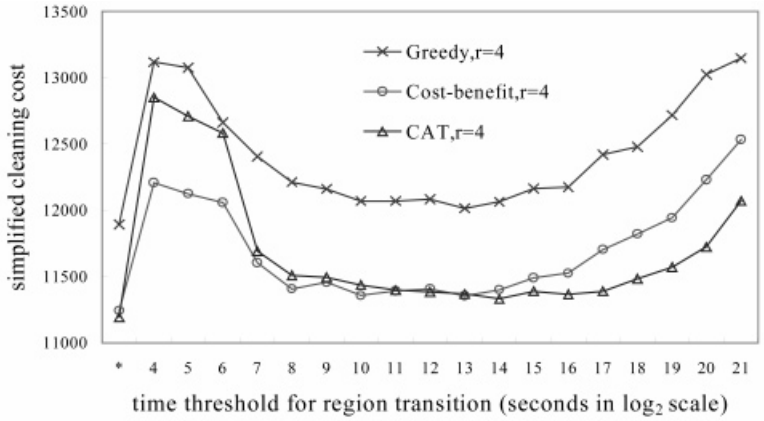


Figure 8. Effect of varying the time threshold for state switching in the DAC state machine. The flash memory was 283 Mbytes with 128-Kbyte erase segments and the block size was 1 Kbytes. The flash memory utilization was set to 85 per cent initially and hplajw traces were used

the time threshold. We found that under this workload, for a 4-state DAC state machine, not to set the time threshold (i.e. labeled '*' in the x-axis) is better than to set it. However, in our experiences with the other workloads (e.g. workloads in the simulations for varying localities of reference), setting the threshold can further reduce cleaning overhead. Therefore, whether setting the time threshold for region transition improves performance depends largely upon workloads and data access behaviors.

Varying flash memory utilization

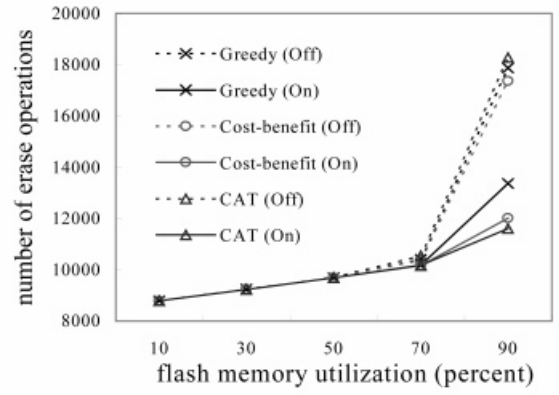
Since our DAC data clustering significantly improved performance under high utilization, we wanted to find out how performance varies under various degrees of utilization. Therefore, before each run of simulation, we wrote enough blocks to sequentially fill the flash memory till the desired level of utilization. Then hplajw traces were used in the simulation. The DAC state machine was configured with 4 states.

Figure 9 shows that as the utilization increased, performance degraded for each policy. The degradation is because more space was occupied by valid data and then more cleaning was needed in order to reclaim free space. Therefore, the effect of different cleaning methods was especially prominent for higher utilizations. For example, at utilizations above 70 per cent, each policy degraded dramatically when DAC data clustering was not used, whereas each policy degraded gradually when DAC data clustering was used. At 90 per cent utilization, the reduction of cleaning costs when using DAC data clustering were 33.8 per cent for Greedy, 41.8 per cent for Cost-benefit, and 48.5 per cent for CAT.

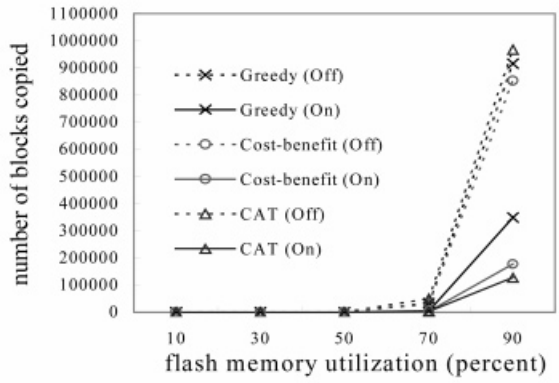
Varying flash memory size

In order to know the impact of flash memory size on the cleaning, the traces were preprocessed to map to flash memory space before simulation. Figure 10 shows that as the size of flash memory increased, each policy performed better since more free space was left

Copyright © 1999 John Wiley & Sons, Ltd.



(a) Number of erase operations.



(b) Number of blocks copied during cleaning.

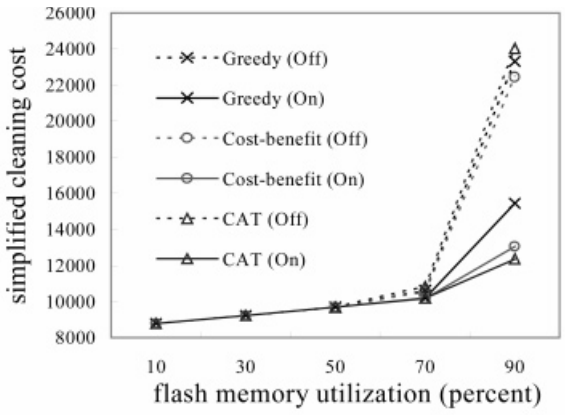
Figure 9. Varying flash memory utilization. Policies used with DAC data clustering were labeled 'On' in the legends; otherwise, they were labeled 'Off'. The flash memory was 283 Mbytes with 128-Kbyte erase segments. Block size was 1 Kbytes and hplajw traces were used

and then cleaning was less needed. However, performance for each policy depended largely on flash memory size when DAC data clustering was not used, whereas each policy performed well for various sizes when DAC data clustering was used (4-state DAC state machine was used in this simulation).

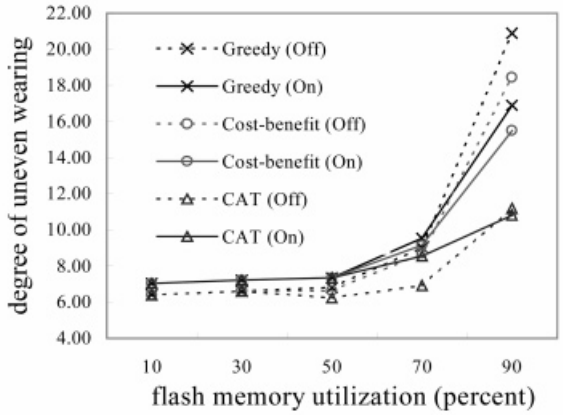
Varying degree of locality

Because HP traces exhibit high localities of reference, we wanted to find out whether DAC data clustering performs well for various localities of reference. A workload generator was created to generate workloads for different localities of reference, which was based on the *hot-and-cold* workload used in the evaluation of Sprite LFS cleaning policies [11,12]. The generated workload was 14-day write references. In total, 192-Mbyte data were written to flash memory in 4-Kbyte units. The arrival rate of requests is Poisson distribution.

Copyright © 1999 John Wiley & Sons, Ltd.



(c) Cleaning cost.

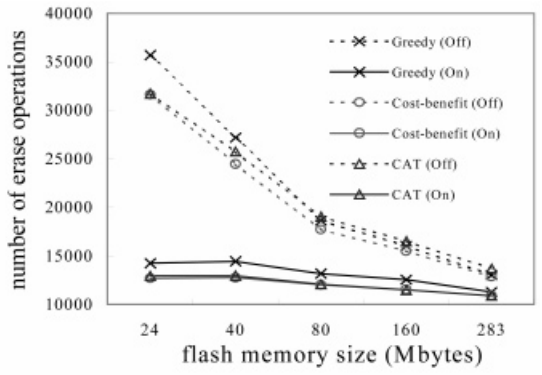


(d) Degree of uneven wearing.

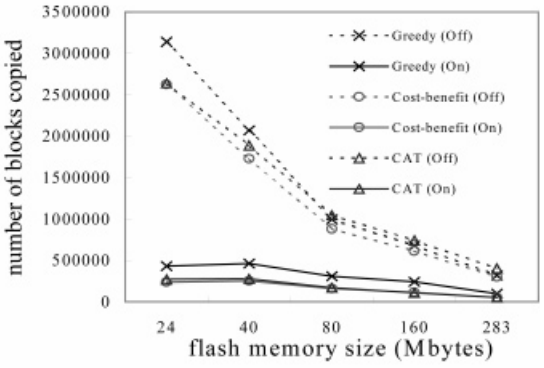
Figure 9. Continued

Figure 11(a) shows the results. We used the notation 'x/y' for locality of reference, in which x% of all accesses go to y% of the data while (1-x)% goes to the remaining (1-y)% of data. As the locality increased, when DAC data clustering was not used, the cleaning costs for each policy were greatly increased. This is because hot data and cold data were mixed together, which increased the cleaning overhead. When DAC data clustering was used (4-state DAC state machine was used in this simulation and the time threshold for state switching was not set), the cleaning costs for each policy were dramatically decreased. This is because hot data were successfully separated from cold data, such that cleaning costs were largely decreased. This effect is especially prominent for higher localities of reference. The reduction of cleaning costs when DAC data clustering was used were 1.9–28.5 per cent for Greedy, 0.5–61.5 per cent for Cost-benefit, and 0.8–65.6 per cent for CAT. This shows that DAC data clustering can significantly reduce cleaning costs by effective data clustering.

Copyright © 1999 John Wiley & Sons, Ltd.



(a) Number of erase operations.



(b) Number of blocks copied during cleaning.

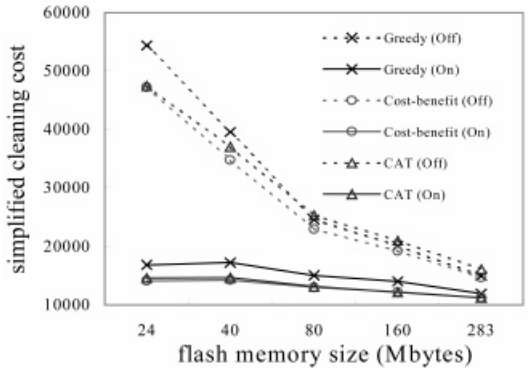
Figure 10. Varying flash memory size. Policies used with DAC data clustering were labeled 'On' in the legends; otherwise, they were labeled 'Off'. The segment size was 128 Kbytes and the block size was 1 Kbytes. The flash memory utilization was set to 85 per cent initially and hplajw traces were used

Figure 11(b) shows that setting the time threshold for state switching is beneficial for this kind of workloads. When the threshold was set to 84 minutes (arbitrarily chosen), cleaning costs can be further reduced as compared with those when the time threshold was not set: 0.3–12.5 per cent for Greedy, 0.4–12.5 per cent for Cost-benefit, and 0.6–12.5 per cent for CAT.

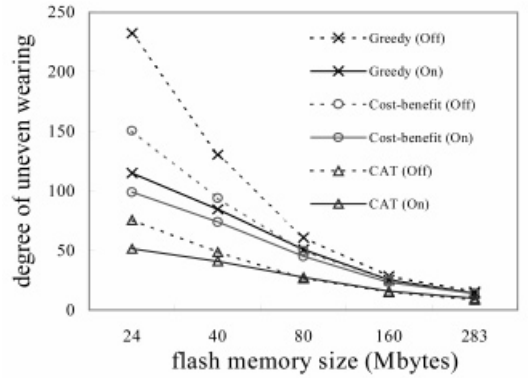
PERFORMANCE EVALUATIONS

A flash memory server utilizing DAC data clustering was implemented on Linux in GNU C++. Table III summarizes the experimental environment. In order to measure the effectiveness of DAC data clustering for various cleaning policies, three policies were also implemented in the server: greedy policy (**Greedy**), cost-benefit policy [5] (**Cost-benefit**), and CAT policy [9,10] (**CAT**).

Copyright © 1999 John Wiley & Sons, Ltd.



(c) Cleaning cost.



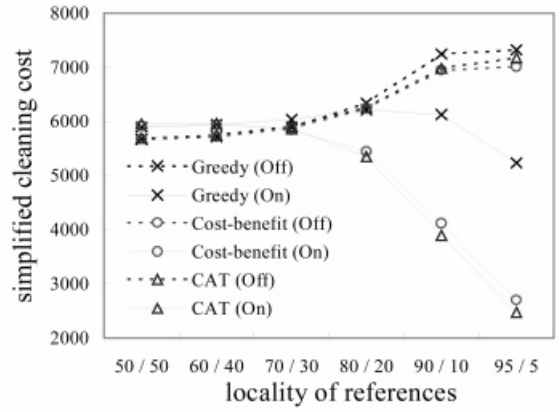
(d) Degree of uneven wearing.

Figure 10. Continued

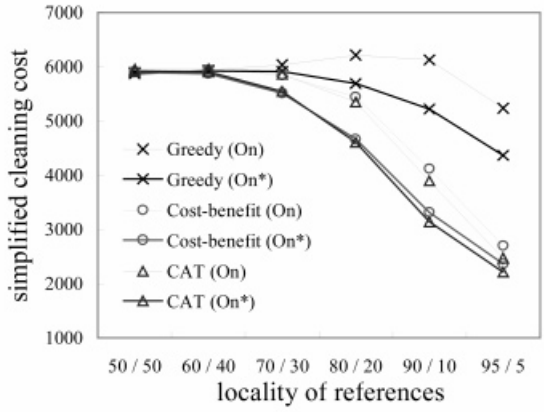
Because we wanted to know whether DAC data clustering performs well for combination of different workloads, a synthetic workload combining random access and locality access was created. The workload contained 4-phase data accesses: the first and the third phases were locality access in which 90 per cent of accesses were to 10 per cent of data; the other phases were random access. Since read operations do not incur cleaning, the workload focused on data updates that incurred invalidation of old blocks, writing of new blocks, and cleaning. In each phase, 40-Mbyte data were written to flash memory in 4-Kbyte units. Totally, 160-Mbyte data were written.

To initialize the flash memory, enough blocks were written in sequence to fill the flash memory to 90 per cent of flash memory space. Benchmarks were created to overwrite the initial data according to the synthetic workload. The block size that the server managed is 4 Kbytes. The number of states with which DAC state machine was configured ranged from 1 to 4. The time threshold for state switching was set to 30 minutes. In each run, the number of erase operations performed, the amount of blocks copied, and the average throughput were

Copyright © 1999 John Wiley & Sons, Ltd.



(a) Using DAC data clustering to reduce cleaning cost.



(b) Effect of setting time threshold for state switching.

Figure 11. Varying localities of reference. Policies used with DAC data clustering were labeled 'On' in the legends when the time threshold for state switching was not set, were labeled 'On' when the time threshold was set; otherwise they were labeled 'Off'. The flash memory was 24 Mbytes with 128-Kbyte segments and the block size was 4 Kbytes. The flash memory utilization was set to 85 per cent initially and the workload for locality of reference was used*

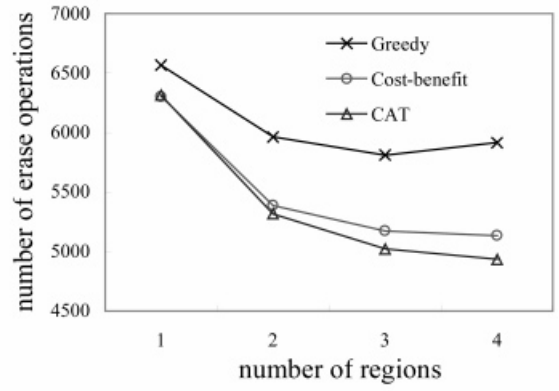
measured. The throughput was obtained from dividing 160 Mbytes by the elapsed time for the synthetic workload.

Figure 12 shows that applying DAC data clustering (i.e. the number of regions is more than 1) is beneficial for each policy. Large amounts of erase operations and blocks copied were reduced and the average throughput was largely increased as well. For example, CAT incurred 15.8–21.83 per cent fewer erase operations, 19.96–27.55 per cent fewer blocks copied, and 16.33–22.56 per cent less cleaning costs. The throughput improvement was 18.89–27.05 per cent. Figure 12(e) shows that flash memory was more evenly worn for CAT.

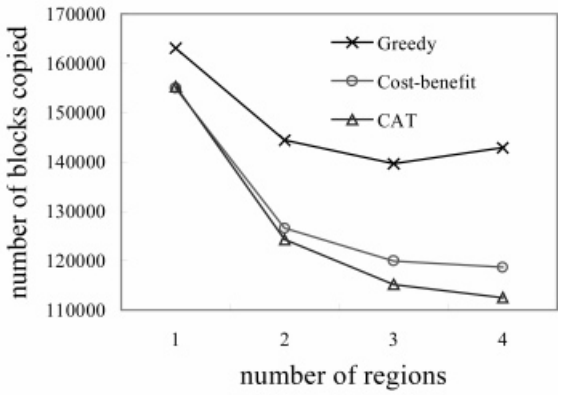
Copyright © 1999 John Wiley & Sons, Ltd.

Table III. Experimental environment

Hardware:
PC: Pentium 133 MHz, 32 Mbytes of RAM
PC Card Interface Controller: Omega Micro 82C365G
Flash memory: Intel Series 2+ 24Mbyte Flash Memory Card [23] (segment size:128 Kbytes)
Software:
Operating system: Linux Slackware 96 (kernel version: 2.0.0, PCMCIA package version [19,20]: 2.9.5)



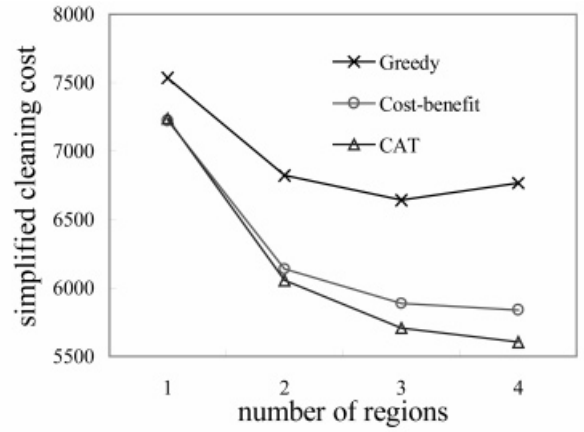
(a) Number of erase operations.



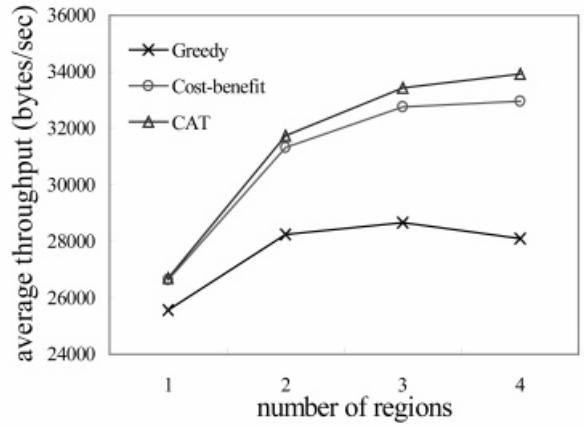
(b) Number of blocks copied during cleaning.

Figure 12. Performance results of DAC data clustering for synthetic workload

Copyright © 1999 John Wiley & Sons, Ltd.



(c) Cleaning cost.



(d) Average throughput.

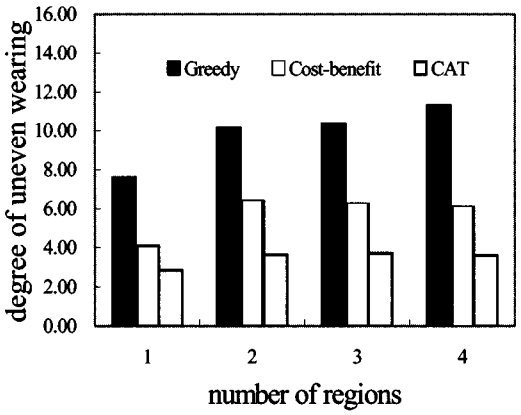
Figure 12. Continued

The above results demonstrate using DAC data clustering that effectively clusters data according to write access frequencies can substantially reduce cleaning overhead. The results also show that applying an effective cleaning policy can further reduce the cleaning overhead. CAT performed best among all policies.

Discussions for time and space overheads

In our implementation of the server, the region table, translation table, and lookup table are constructed in main memory. They are used to speed up processing. To store these tables requires a substantial amount of main memory: 12 bytes per region, 13 bytes per block, and

Copyright © 1999 John Wiley & Sons, Ltd.



(e) Degree of uneven wearing.

Figure 12. Continued

17 bytes per segment. However, it is a trade-off between space consumption and performance. Because current flash memory capacity is still small, the space overhead is limited. For example, for a 24-Mbyte flash memory with 128-Kbyte segments, 4-Kbyte blocks, and four regions, these tables take up 78 Kbytes of main memory.

CONCLUSION

Flash memory is expected to be largely used as the increase of capacity and the decrease of price. Large erases and writes will be created and wear leveling will be very important. Effective cleaning policies help to maximize the flash memory lifetime, improve system performance, and reduce power consumption. In this paper, we have presented a data reorganization technique for clustering frequently accessed data to improve cleaning performance. Data are clustered dynamically according to their write frequencies.

We have detailed the implementation of a flash memory server utilizing the proposed data clustering technique. Performance is evaluated through implementation and trace-driven simulations with a variety of cleaning algorithms. Experiments with synthetic workloads showed that by applying the proposed method, cleaning costs for various cleaning policies were significantly reduced by 9.5–22.56 per cent and throughputs were improved by 9.9–27.05 per cent. Simulations with real-world traces showed that cleaning costs were reduced by 17.8–30.5 per cent. Flash memory lifetime is thus extended, system throughput is largely improved, and flash memory is evenly worn. In examining the degree of wear-leveling and exploring the impacts of flash memory utilization, flash memory size, and degree of locality of reference, the proposed method all performed well.

Several factors are important in determining how well the DAC data clustering will work in a given environment, such as the configuration for the number of states in the DAC state machine and the setting of the time threshold for state switching. In our experience, these factors are highly dependent on workload. We also noticed that different segment selection algorithms perform differently for the same setting of factors.

The proposed method can not only be used in flash memory, it can also be used in other applications that can benefit from data clustering or need segment cleaning. For

Copyright © 1999 John Wiley & Sons, Ltd.

example, in our evaluations and simulations, the number of blocks copied during cleaning was significantly reduced. This result suggests that applying the proposed method is also beneficial to segment cleaning in the log-based disk systems, which have no erasure cost and consider only reducing the number of blocks copied to improve cleaning performance. The other promising application is applying the DAC data clustering to cluster hot disk data together near the center of disk to reduce seek times in disk storage systems. Clustering frequently accessed data can reduce seek times and improve disk performance has been shown in many researches [29–34].

ACKNOWLEDGEMENTS

We would like to thank John Wilkes at Hewlett Packard Laboratories, who graciously made the HP I/O traces available for our simulations and provided valuable comments. We would also like to thank David Hinds for his valuable comments and help on our work. The referees' precious comments helped to improve this paper. This research was supported in part by the National Science Council of the Republic of China under grant no. NSC 88-2213-E-001-016.

REFERENCES

1. M. Baker, S. Asami, E. Deprit, J. Ousterhout and M. Seltzer, 'Non-volatile memory for fast, reliable file systems', *Proceedings of the 5th International Conference on Architectural Support for Programming Languages and Operating Systems*, Boston, MA, October 1992, pp. 10–22.
2. R. Caceres, F. Douglis, K. Li and B. Marsh, 'Operating system implications of solid-state mobile computers', *Proceedings of the Fourth Workshop on Workstation Operating Systems*, Napa, CA, October 14–15 1993, pp. 21–27.
3. B. Dipert and M. Levy, *Designing with Flash Memory*, Annabooks, 1993.
4. F. Douglis, R. Caceres, F. Kaashoek, K. Li, B. Marsh and J. A. Tauber, 'Storage alternatives for mobile computers', *Proceedings of the First Symposium on Operating Systems Design and Implementation (OSDI)*, 1994, pp. 25–37.
5. A. Kawaguchi, S. Nishioka and H. Motoda, 'A flash-memory based file system', *Proceedings of the 1995 USENIX Technical Conference*, New Orleans, LA, January 16–20, 1995, pp. 155–164.
6. M. Wu and W. Zwaenepoel, 'eNVy: A non-volatile, main memory storage system', *Proceedings of the Sixth International Conference on Architectural Support for Programming Languages and Operating Systems*, San Jose, CA, October 1994, pp. 86–97.
7. P. Torelli, 'The microsoft flash file system', *Dr. Dobb's Journal*, 62–72 (February 1995).
8. R. Dan and J. Williams, 'A TrueFFS and Flite technical overview of M-systems flash file systems', 80-SR-002-00-6L Rev. 1.30. http://www.m-sys.com/tech1.htm, March 1997.
9. M. L. Chiang, P. C. H. Lee and R. C. Chang, 'Managing flash memory in personal communication devices', *Proceedings of the 1997 International Symposium on Consumer Electronics (ISCE'97)*, Singapore, December 2–4 1997, pp. 177–182.
10. M. L. Chiang and R. C. Chang, 'Cleaning policies in mobile computers using flash memory', *Journal of Systems and Software* (accepted).
11. M. Rosenblum and J. K. Ousterhout, 'The design and implementation of a log-structured file system', *ACM Trans. Computer Systems*, **10**(1), 26–52 (February 1992).
12. M. Rosenblum, 'The design and implementation of a log-structured file system', *PhD Thesis*, University of California, Berkeley, June 1992.
13. M. Seltzer, K. Bostic, M. K. McKusick and C. Staelin, 'An implementation of a log-structured file system for UNIX', *Proceedings of the 1993 Winter USENIX Conference*, January 1993, pp. 307–326.
14. T. Blackwell, J. Harris and M. Seltzer, 'Heuristic cleaning algorithms in log-structured file systems', *Proceedings of the 1995 USENIX Technical Conference*, New Orleans, LA, January 16–20 1995, pp. 277–288.
15. J. Wilkes, R. Golding, C. Staelin and T. Sullivan, 'The HP AutoRAID hierarchical storage system', *ACM Trans. Computer Systems*, **14**(1), 108–136 (February 1996).

Copyright © 1999 John Wiley & Sons, Ltd.

16. J. N. Matthews, D. Roselli, A. M. Costello, R. Y. Wang and T. E. Anderson, 'Improving the performance of log-structured file systems with adaptive methods', *Proceedings of the Sixteenth ACM Symposium on Operating System Principles*, Saint Malo, France, October 5–8 1997.
17. W. de Jonge, M. F. Kaashoek and W. C. Hsieh, 'The logical disk: A new approach to improving file systems', *Proceedings of 14th Symposium on Operating Systems Principles*, 1993, pp. 15–28.
18. D. Anderson, *PCMCIA System Architecture*, MindShare, Inc. Addison-Wesley, 1995.
19. D. Hinds, 'Linux PCMCIA HOWTO', http://hyper.stanford.edu/~dhinds/pcmcia/doc/PCMCIAHOWTO.html, v2.5, February 19 1998.
20. D. Hinds, 'Linux PCMCIA Programmer's Guide', http://hyper.stanford.edu/~dhinds/pcmcia/doc/PCMCIA-PROG.html, v1.38, February 4 1998.
21. M. K. McKusick, M. J. Karels and K. Bostic, 'A pageable memory based file system', *USENIX Conference Proceedings*, Anaheim, CA, Summer 1990, pp. 137–144.
22. Intel, *Flash Memory*, 1994.
23. Intel Corp., 'Series 2+ Flash Memory Card Family Datasheet', http://www.intel.com/design/flcard/datashts, 1997.
24. C. Ruemmler and J. Wilkes, 'UNIX disk access patterns', *Proceedings of the 1993 Winter USENIX*, San Diego, CA, January 25–29 1993, pp. 405–420.
25. C. Ruemmler and J. Wilkes, 'A trace-driven analysis of disk working set sizes', *Technical Report HPL-OSR-93-23*, Hewlett-Packard Laboratories, Palo Alto, CA, April 5 1993.
26. K. Li, 'Towards a low power file system', *Technical Report UCB/CSD 94/814*, University of California, Berkeley, CA, May 1994.
27. K. Li, R. Kumpf, P. Horton and T. Anderson, 'A quantitative analysis of disk drive power management in portable computers', *Proceedings of the 1994 Winter USENIX*, San Francisco, CA, 1994, pp. 279–291.
28. B. Marsh, F. Douglis and P. Krishnan, 'Flash memory file caching for mobile computers', *Proceedings of the 27 Hawaii International Conference on System Sciences*, Maui, HI, 1994, pp. 451–460.
29. S. Akyurek and K. Salem, 'Adaptive block rearrangement', *ACM Trans. Computer Systems*, **13**(2), 89–121 (1995).
30. S. Akyurek and K. Salem, 'Adaptive block rearrangement under UNIX', *Software—Practice and Experience*, **27**(1), 1–23 (January 1997).
31. C. Ruemmler and J. Wilkes, 'Disk shuffling', *Technical Report HPL-91-156*, Hewlett-Packard Laboratories, Palo Alto, CA, October 28 1991.
32. C. Staelin and H. Garcia-Molina, 'Clustering active disk data to improve disk performance', *Technical Report CS-TR-283-90*, Department of Computer Science, Princeton University, September 1990.
33. C. Staelin and H. Garcia-Molina, 'Smart filesystems', *Proceedings of the 1991 Winter USENIX*, Dallas, TX, 1991, pp. 45–51.
34. P. Vongsathorn and S. D. Carson, 'A system for adaptive disk rearrangement', *Software—Practice and Experience*, **20**(3), 225–242 (1990).

Copyright © 1999 John Wiley & Sons, Ltd.