|  |
| --- |
| The Samraksh company |
| Effect of Garbage Collection Algorithms on the Efficiency of Non-Volatile Memory |
| Report |
|  |
| **Mukundan Sridharan and Ananth Mahadevan** |
| **March 1, 2013** |

|  |
| --- |
|  |

# Summary

Non-Volatile Memory Devices (Flash), though have many advantages, they still have a few problems such as the inability to update data in place. This necessitates the need for a Garbage Collector (GC) that can collect active data and create space by erasing the Flash blocks. But this is a very costly operation and increases the write latency thereby lowering the efficiency of the Flash device. The frequency at which the GC is invoked by the underlying File System depends on the data’s traffic pattern as well as the fullness of the device. It is therefore important to study different GC algorithms for different traffic patterns and at varying fullness levels in order to find the most efficient one for a particular situation.

The performance results for five different GC algorithms for Flash Devices for three traffic/access patterns are presented in this report. The access patterns include a long-tailed distribution as well as Uniform distribution. The results indicate that Round-Robin style GC algorithms perform better in all scenarios than Generational algorithms. This is counter-intuitive to the existing norms. Even at low fullness levels, Generational GCs have very low efficiency. In this paper, we compare and contrast the efficiency and the time taken for the GCs at fullness levels ranging from 2% to 98%. A simulator for the Flash file system as well as the GC algorithms was coded in Matlab and simulations were performed for 100,000 cycles which is typically the life-time of a Flash device.

The major goals of our work are:

* To find out if Flash can work as a good primary storage system.
* To create performance benchmarks and understand which Garbage Collection algorithm is better.
* To create statistical models to test the GC algorithms.

Main results:

* LAC seems to perform much better than expected
* Generational algorithms have very poor efficiency on NVM (but what about RAM)
* Our experiments indicate that Round-Robin style algorithms have better efficiency than Generational algorithms.

Contents

[Summary 1](#_Toc349633261)

[I Introduction 5](#_Toc349633262)

[I.A Methodology: 6](#_Toc349633263)

[I.B Implementation details: 6](#_Toc349633264)

[II Background and Related Work 8](#_Toc349633265)

[II.A Log Structured File System 8](#_Toc349633266)

[II.B Summary of Existing Literature 9](#_Toc349633267)

[Existing Literature on RAM Garbage Collection 9](#_Toc349633268)

[Existing Literature on Non-Volatile Memory Garbage Collection 9](#_Toc349633269)

[III Methodology and Algorithms 11](#_Toc349633270)

[III.A Analytical Modeling 11](#_Toc349633271)

[III.B Matlab Flash Simulation 11](#_Toc349633272)

[III.C Application Classes 11](#_Toc349633273)

[III.D Algorithms Studied 11](#_Toc349633274)

[First In First Out (FIFE) 11](#_Toc349633275)

[Least Active Clean (LAC) 11](#_Toc349633276)

[Three Generation (3-Gen) 11](#_Toc349633277)

[N-Generation (N-Gen) 12](#_Toc349633278)

[Eta-N-Gneration (E-N-Gen) 12](#_Toc349633279)

[Gamma-N-Generation (G-N-Gen) 12](#_Toc349633280)

[IV Analytical Modeling 13](#_Toc349633281)

[IV.A Log File System Garbage Collection with Uniform Random Application 13](#_Toc349633282)

[IV.B Log File System Garbage Collection with Pareto Application 13](#_Toc349633283)

[IV.C Least Active Clean Garbage Collection and Uniform Random Application 13](#_Toc349633284)

[V Results 14](#_Toc349633285)

[VI Conclusions 15](#_Toc349633286)

[VII Bibliography 16](#_Toc349633287)

# Introduction

Non-Volatile Memory Devices are increasingly being used as primary storage devices. They have many advantages such as retaining data even during loss of power and low latency. They have high storage capacities and can be placed under two major categories – NAND or NOR. NAND flashes have larger storage capacities than NOR and are typically used to store data. NOR flashes on the other hand are used to execute code in embedded devices. Unlike RAM, where data can be over-written, flash devices are plagued by the inability to update data in-place. This means that when data has to be updated, the earlier one is marked invalid and the newer one is written to an empty region. This can quickly fill up the flash and hence mandates the need for a proper data management policy. This is to ensure that write operations can happen with as minimum of a latency as possible and thereby increase the overall performance. For purposes of this paper, data will be referred to as a Record.

Flash devices are typically divided into multiple blocks with records written to these blocks by a File System. File systems manage the Read/Write operations for an application and are responsible to keep the time required for such operations to a minimum. An important component that has an overhead and affects the performance of the File System is the Garbage Collector (GC). GCs are responsible for collecting active records from blocks, moving them to blocks with sufficient space and erasing the blocks from which data was moved. This erase operation clears up space for the File System to add new records. But the move and erase operations are costly and depending on the underlying algorithm can improve or degrade the performance. The frequency with which the GC is invoked depends on the data traffic pattern. Data can be generated by the application in periods of short bursts or can be distributed uniformly throughout the life-cycle of the application. Data can be classified as either “Hot” or “Cold”. “Hot” data are those that are frequently updated and “Cold” data are either rarely or never accessed at all. A good GC algorithm needs to take this into consideration to provide good performance. This report quantifies the performance of different GC algorithms against different traffic patterns.

In order to simulate the above-mentioned traffic patterns we created applications that generate data that follow Uniform, Pareto (long-tailed) and BiModal distributions. As mentioned previously, move and erase operations are costly. Erase operations cannot happen on individual records and has to happen on entire blocks. A proper GC algorithm can make or break the over-lying application. There are several options to select a block to be Garbage Collected. It could be the one that was written to first or one that yields the maximum space. We considered both of these options and call the algorithms FIFE - First Insert First Erase and LAC - Least Active Clean. LAC is a type of Greedy algorithm that always chooses the block that has the least amount of active records and hence contributes the lowest to the move cost. Based on the “hotness” or “coldness” of the data, the entire flash can be divided into different regions with cold data occupying a certain portion and the hot data the other portion. Such algorithms are called Generational algorithms and are quite popular. We created 3 different generational algorithms – 3-Gen, N-Gen and Eta-N-Gen. In 3-Gen, the flash is divided into 3 generations with hot data occupying the 1st generation and cold data, the lower two. Eta-N-Gen and N-Gen algorithms consider all blocks of the flash to be a generation with the coldest data occupying the last generation.

In this paper, we present our results of analyzing and measuring the performance of five different GC algorithms for three different data patterns. The performance is measured using various parameters such as write cost, erase cost and Efficiency. We also present the theoretical model behind our simulation and explain the reason behind the results we have obtained. For Uniform, Pareto and Bimodal data access patterns, LAC and FIFE outperform the Generational algorithms. We intuitively believed that the generational algorithms would be efficient for a class of data access patterns such as those with high localities of reference. But as the results in latter half of the paper show, our intuition is incorrect and the FIFE and LAC algorithms out-perform the rest of the algorithms for all classes of patterns we could throw at it.

The report is organized as follows: section 2 gives details on related work. Section 3 mentions about the algorithms used and our experimental methodology. Section 4 provides details on the Mathematical models behind our simulations and we conclude by outlining our results in section 5.

## Implementation details:

The GC and the applications were implemented in Matlab 2011b. The tests were run on the Ohio Super Computer center’s cluster called Oakley. The fullness level ranges from 2% to 98% and the simulations were run for 100,000 read/write accesses per fullness level. The simulations are done for all three traffic patterns for all five GC algorithms. We used the rand function in Matlab which generates numbers with Uniform distribution. For Pareto, we added a weight (that followed a long-tail distribution) to the numbers which decides the usage of a record.

We simulated an application which is both equally read and write dominant. We also tested an application which has only writes. After the records are generated, based on a coin toss, it is decided whether the next operation will be a read or a write. Every time the GC is accessed, details such as amount of bytes moved, blocks erased, are captured. These details are then used to plot the required graphs.

# Background and Related Work

Flash memory is a powerful and cost-effective solid-state non volatile storage technology that is widely being used in mobile devices and other embedded devices. Compared to traditional Hard Disk Drives, they have low power consumption and a small size. Embedded devices are constrained by power and low memory capacity. Hence there is a need to have a very efficient primary storage system that allows fast read and write operations and thereby less power consumption. Flash devices generally have fast read accesses but have very slow write accesses. There are 2 major Flash devices available today - NAND and NOR. NAND ash has a very small cell size and is mainly used for storage of large amounts of data as its cost-per-bit is very low compared to NOR [4]. NAND ash is organized into blocks and each block is divided into pages. Block size of a typical NAND ash is 16KB and the page size can be 512B (32 pages in a block). Read and Write operations happen take place on pages whereas erase happens on blocks. NOR flash on the other hand, has individual cells connected in parallel which allow it to achieve random access. This enables it to achieve short read times and allow individual bits to be set (called reprogramming). This has the advantage that it can execute code, a feature called eXecute-In-Place (XIP).

Every block on a NAND ash can be written-to or erased only a limited number of times (in order of 1 million or 106 cycles). Writing to or erasing a block beyond this limit can result in \wearing" out of the Flash, which can lead to write failures or can return invalid data for read operations. Data cannot be written over already written areas (called in-place update). Data can only be written to areas that have already been erased. Therefore if some data has to be updated on the ash, it is \_rst written to a new area and the old data is marked invalid. This is called out-of-place-update. After many cycles of writes, the entire ash is fragmented with valid and invalid data and the ash quickly runs out of space for new data. This is when the Flash invokes the Garbage Collector whose task is to collect all valid data in a block, write it to a new location and erase the old block. This paves the way for new data to be written to the ash. But this operation of moving data to a new location and erasing a block is costly and has to be kept to a minimum. Based on the above points, some of the challenges for a good GC algorithm is: (a) To maintain \wear-levelling" of the ash blocks (b) Reduce the amount of time taken to move data and erase blocks.

## Log Structured File System

Due to increasing capacity and reduction in accessing times of RAM, reads have become quite fast and writes take up bulk of the time in a typical embedded device. Hence there is a need for a \_le system that provides fast write access. A log structured \_le system is one such system that writes data sequentially in a log-like manner [5]. This reduces the write time as there is no structure other than logs in a device to be maintained and thereby reducing the amount of data seeks. But in order for a log-structured \_le system to operate e\_ciently, it needs to have large amounts of free space to write new data. LFS also allow fast recovery from crashes which is not present in traditional \_le systems as they have to scan the entire set of data in order to build the index.

The Log File System that we implement makes use of three pointers which facilitate the write operations. Log pointer always points at a location where the write operation can take place. Clean Pointer indicates the location in the Flash until which there are no active records and the Erase pointer which always moves one block at a time points at the block which was last erased. There are three major GC algorithms that have been studied for Log-Structured File Systems - Greedy, Cost- Bene\_t analysis and Cost-Age Time (CAT). Greedy algorithms select those data blocks that can yield the most free space, whereas a cost-bene\_t algorithm selects blocks based on the free space as well as the age of the segment [6, 7]. CAT reduces the erase operations by segregating the hot from the cold data [8]. A major advantage of this approach is that it takes wear-levelling into account before cleaning a block.

In Solid State Devices such as NAND and NOR based Flash, new data is always written out-of-place. In a Log-Structured File System, this reduces the amount of free space and a Garbage Collector algorithm is invoked which defragments the device by moving all valid data together and erasing the invalid data. This is a critical factor in the performance and life-time of a Flash device.

## Summary of Existing Literature

### Existing Literature on RAM Garbage Collection

### Existing Literature on Non-Volatile Memory Garbage Collection

An Age-Threshold Algorithm for Garbage Collection in Log-Structured Arrays and File Systems

This paper compares the Greedy and Cost-Bene\_t algorithm with their algorithm called "Age-Threshold algorithm". Greedy considers those segments for Garbage Collection that has the least amount of active records or in other words can yield the maximum amount of space. Cost-Bene\_t considers those segments that will yield the maximum space but at the same time, are above a certain age. Age-Threshold considers those segments that are above a threshold. Age is determined by means of a clock. When data is moved from one segment to another by GC, the age of the new segment is the old + 1. But when a segment is erased and then new data is written, the age starts from 0.

Cleaning Policies in Mobile Computers Using Flash Memory: This paper compares the Greedy, Cost-Bene\_t with their algorithm - CAT (Cost Age Time). CAT algorithm claims to reduce the number of erase operations performed on a block while evenly wearing out the ash at the same time. It considers the age of the data, the cleaning cost and the number of times a segment has been erased. A segment is chosen such that the formula CleaningCost\_1=Age\_Countofcleaning is minimized. It also considers di\_erent ways of redistributing data (within the same block and across several blocks). The algorithm is compared for data patterns such as those with varying localities of reference, uniform access and so on.

Grouping data across segments based on their hotness degree is similar to generational algorithms. There are algorithms on opposite ends of the spectrum. One one end there are algorithms that claim longer the age of a block, the more likely to have inactive data. The other end of spectrum, algorithms claim that longer the data is not accessed, higher they can be moved to generations.

In generational, the problem is that there is a large amount of hot data that is being marked as cold. Very soon after the data is moved to a higher generation, data tends to get inactivated. This results in wasted move cost. Most of the ine\_ciency comes from thinking of hot data as cold (move hot data too soon). For Uniform, FIFE is the best, but for Pareto age de\_nitely has to be considered.

# Methodology and Algorithms

## Analytical Modeling

Overview of Modeling

## Matlab Flash Simulation

Overview of Matlab implementation

## Application Classes

Uniform and Pareto

## Algorithms Studied

Five di\_erent GC algorithms are considered which has been described below. The \_rst two are round-robin style of GCs while the other three are generational type of algorithms.

### First In First Out (FIFE)

FIFE { First Insert First Erase: The data is always written starting from the \_rst block and when the ash reaches a pre-de\_ned fullness level, the GC is invoked. The GC compacts the oldest block and keeps moving forward along the blocks until su\_cient space is created to store the record. The GC returns an error when even after traversing the blocks, it is not able to \_nd su\_cient space.

### Least Active Clean (LAC)

LAC { Least Active Clean: When the GC is invoked, it \_nds out the block that has the least amount of active records and compacts that block. The compacted block is then used for the next write operation. Figure 1 shows the state of the Flash at steady state for the FIFE GC. The inactive records are spread evenly throughout the device. Also shown are the Log, Clean and Erase points.

### Three Generation (3-Gen)

Three Generation: In this GC, the entire ash is divided into three generations. The ratio of the blocks in each generation can be 12:3:1 or 4:3:1. Data is always written to the \_rst generation and when it becomes full, the active records are moved to the 2nd generation. When the 2nd generation becomes full, its active records are moved to the 3rd generation. This allows the GC to separate the hot from the cold data. The intuition is that this reduces the amount of movement of the active records which is a draw-back of the round-robin algorithms.

Figure 2 shows the state of the Flash at steady state for the FIFE GC. The ash is divided into three generations with a major portion of the blocks in the \_rst generation. The second and third generations have fewer numbers of blocks. We experimented with block distribution ratios of 12:3:1 and 4:3:1. There are more inactive records in the 1st generation than in the other two generations.

### N-Generation (N-Gen)

N-Generation: All the blocks in the ash is considered as a generation in this algorithm. Cold data is always pushed to the highest possible generation.

### Eta-N-Gneration (E-N-Gen)

Eta-N-Generation: This is similar to the N-Gen algorithm, except that the amount of data that is moved is decided by a factor called Eta. Eta denotes the fullness level of the ash when the GC is invoked.

### Gamma-N-Generation (G-N-Gen)

The GC that will be implemented will be the one that has the best e\_ciency for all three tra\_c patterns. Depending on the outcome of this project, a single GC might be implemented or a hybrid approach will be chosen where one type of GC is used for certain fullness levels and another type for other levels of fullness.

# Analytical Modeling

## Log File System Garbage Collection with Uniform Random Application

## Log File System Garbage Collection with Pareto Application

## Least Active Clean Garbage Collection and Uniform Random Application

# Results

# Conclusions

# Bibliography

|  |  |
| --- | --- |
| [1] | "NAND vs NOR Flash Memory Technology Overview," Toshiba America Electronic Components, Inc., 2011. |