|  |
| --- |
| 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

Missing intro to problem.

The performance results for \_ve di\_erent Garbage Collection (GC) algorithms for Non-Volatile Memory Devices for three 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 much better in all cases than Generational algorithms. This is counter-intuitive to the existing norms. Invocation of the GC in Flash devices is determined by the fullness of the device. Even at low fullness levels, Generational GCs have very low e\_ciency. In this paper, we compare the e\_ciency and the time taken for the individual GCs at fullness levels ranging from 2% to 98%. Existing research looks into using Flash as storage for data and RAM as cache [1, 2, 3]. We analyze the performance of a Flash when it is used in place of a RAM. A simulator for the Flash \_le system as well as the GC algorithms were coded in Matlab. We compare the performance of \_ve di\_erent GC algorithms against three tra\_c patterns.

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 e\_ciency than Generational algorithms.

Contents

[Summary 1](#_Toc349562044)

[I Introduction 5](#_Toc349562045)

[II Background and Related Work 6](#_Toc349562046)

[II.A Log Structured File System 6](#_Toc349562047)

[II.B Summary of Existing Literature 7](#_Toc349562048)

[Results for RAM 7](#_Toc349562049)

[Results for Non-Volatile Memory 7](#_Toc349562050)

[II.C Algorithms Studied 8](#_Toc349562051)

[First In First Out (FIFE) 8](#_Toc349562052)

[Least Active Clean (LAC) 8](#_Toc349562053)

[Three Generation (3-Gen) 8](#_Toc349562054)

[N-Generation (N-Gen) 9](#_Toc349562055)

[Eta-N-Gneration (E-N-Gen) 9](#_Toc349562056)

[Gamma-N-Generation (G-N-Gen) 9](#_Toc349562057)

[III Analytical Modeling 10](#_Toc349562058)

[III.A Log File System Garbage Collection with Uniform Random Application 10](#_Toc349562059)

[III.B Log File System Garbage Collection with Pareto Application 10](#_Toc349562060)

[III.C Least Active Clean Garbage Collection and Uniform Random Application 10](#_Toc349562061)

[IV Results 11](#_Toc349562062)

[VI Bibliography 13](#_Toc349562063)

# Introduction

# Background and Related Work

Flash memory is a powerful and cost-e\_ective 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 e\_cient 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 ash on the other hand, have 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. |