Analysis of Garbage Collector Algorithms in Non-Volatile Memory Devices

Ananth Mahadevan[†]$\oplus$
email: {mahadeva}@cse.ohio-state.edu
[†] The Ohio State University, Columbus, OH
$\oplus$ The Samraksh Company, Dublin, OH

February 24, 2013

Contents

1 Abstract

The performance results for five different 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 efficiency. In this paper, we compare the efficiency 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 file system as well as the GC algorithms were coded in Matlab. We compare the performance of five different GC algorithms against three traffic patterns. Our experiments indicate that Round-Robin style algorithms have better efficiency than Generational algorithms.

Keywords

Garbage Collection, Flash memory, Statistical access pattern.

2 Introduction

An important component that has an overhead and affects the performance of the storage system is the Garbage Collector (GC) algorithm. This report quantifies the performance of different GCs against different statistical traffic patterns such as Uniform, Pareto and Bi-Modal. Based on prior experiments, we have observed that the traffic pattern for the data can occur in short bursts or can arrive at regular intervals. This is the reason why we chose to select the afore-mentioned traffic patterns.

Updates to exisitng records cannot happen in-place in Flash devices. The earlier record has to be marked inactive and the newer version of the record has to be written to a new location. After a period of time, the flash becomes full and there is no space left for new records to be written. This is when the GC is invoked whose purpose is to create space for the new records. The space is created by moving active records from one block to another block and erasing the earlier block. When erase operations take place, it has to happen on an entire block and not on individual records. There are several options in selecting a block to be Garbage Collected. It could be the oldest one 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.

But depending on the data pattern, it may so happen that the GC keeps moving around a set of records which is considered “cold”. “Cold” data are those that are never or rarely accessed. Therefore it makes sense to dump this data to a portion of the Flash ear-marked for this purpose. This is the idea behind the Generational algorithms where the entire Flash is divided into different number of generations. We considered different flavors of such Generational algorithms such as 3-Gen – where the flash is divided into three generations; N-Gen – where each of the N-blocks in the Flash is a generation.

For random and uniform data access patterns, we expected the Round-Robin class of algorithms to out-perform the generational algorithms. The intuition behind the generational algorithms is that they 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 Round-Robin algorithms out-perform the rest of the algorithms for all classes of patterns we could throw at it.

A challenge that we faced before starting with our work was the lack of real-world applications (like an app store) that can simulate various access patterns. Therefore we created the benchmarking applications in Matlab. The benchmarking applications were created to simulate patterns of access such as Uniform, heavy-tailed (high locality of reference) and Bi-Modal. The main goal behind this work was to find out the most efficienct GC algorithm for a given traffic pattern. In order to measure efficiency, the parameters we used were costs involved in writing and moving records and erasing blocks. The simulations were repeated for 100,000 times mimicing the life-time of a typical Flash.

In this paper, we present our work in analyzing five different GC algorithms and compare their performances against various parameters. We also present the theoretical model behind our simulation and explain the reason behind the results we have obtained. A major contribution of our work is to measure the performance of Flash devices against traffic patterns that are generally observed in practice.
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.

The report is organized as follows: section 3 gives details on related work. Section 4 mentions about the current state-of-the-art in Flash algorithms and section 5 provides details on the Mathematical models behind our simulations. We conclude by outlining our results in section 6, details about our simulator in section 7 and finally mention about future work in this area.

2.1 Methodology

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

2.2 Implementation details:

Records were generated for different fullness levels. 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.

3 Background

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.

Read time(ms)	**Write time (ms)**	**Erase time (ms)**
4	5	6

There are 2 major Flash devices available today - NAND and NOR. NAND flash 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 flash is organized into blocks and each block is divided into pages. Block size of a typical NAND flash 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, 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 flash can be written-to or erased only a limited number of times (in order of 1 million or 10^6 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 flash, it is first 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 flash is fragmented with valid and invalid data and the flash 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 flash. 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 flash blocks (b) Reduce the amount of time taken to move data and erase blocks.

3.1 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 file system that provides fast write access. A *log structured file 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 file system to operate efficiently, 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 file 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-Benefit analysis and Cost-Age Time (CAT). Greedy algorithms select those data blocks that can yield the most free space, whereas a cost-benefit 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.

4 Modelling

Efficiency of a GC is defined as the ratio of useful write vs. actual write. Useful write is defined as the amount of data passed to the write API of the file system and Actual write is the sum of amount of data moved, blocks erased and the useful write. A good GC is one which is able to find space to write data quickly. It has to do so by

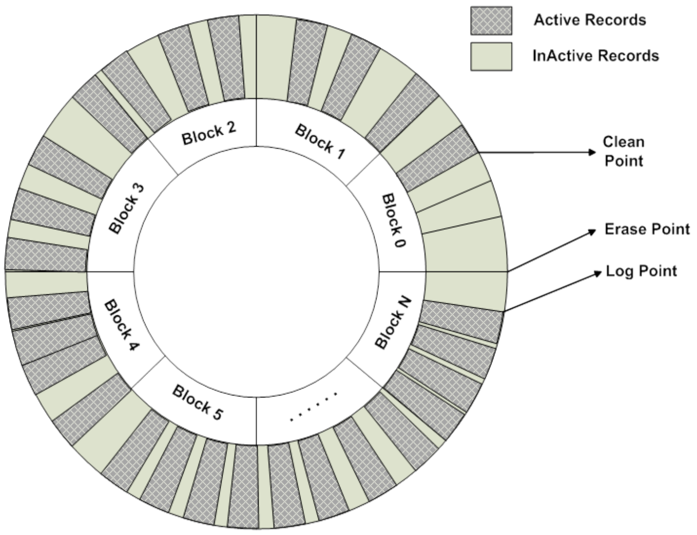


Figure 1: Fragmentation of blocks for FIFE GC during steady state of the Flash device.

spreading the writes evenly across the various blocks in the flash (wear-leveling). Writing too often to a particular block can cause the flash to wear-out quickly.

Another important criterion in choosing a certain GC is the traffic pattern of the data. Data can be written to the flash in short bursts or can be spread out evenly. They tend to follow statistical patterns such as Uniform, Normal or Bi-Modal. For the current project, there are no sets of real-world applications which can be used to test the GC algorithms. So, one of the goals of the project is to create a set of pseudo applications which generates data that follows a Uniform, Normal or a Bi-Modal type of distribution.

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

- **FIFE – First Insert First Erase**: The data is always written starting from the first block and when the flash reaches a pre-defined fullness level, the GC is invoked. The GC compacts the oldest block and keeps moving forward along the blocks until sufficient space is created to store the record. The GC returns an error when even after traversing the blocks, it is not able to find sufficient space.

- **LAC – Least Active Clean**: When the GC is invoked, it finds 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.

- **3-Generation**: In this GC, the entire flash 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 first 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 flash is divided into three generations with a major portion of the blocks in the first generation. The second and third generations have fewer number 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**: All the blocks in the flash is considered as a generation in this algorithm. Cold data is always pushed to the highest possible generation.

- **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 flash when the GC is invoked.

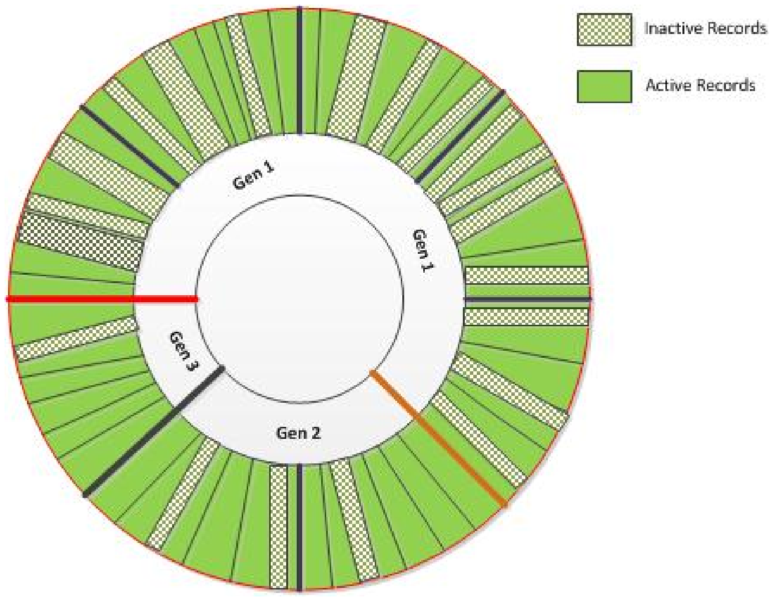
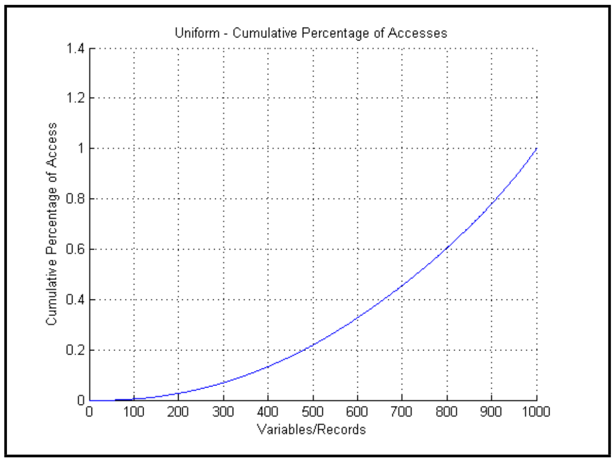
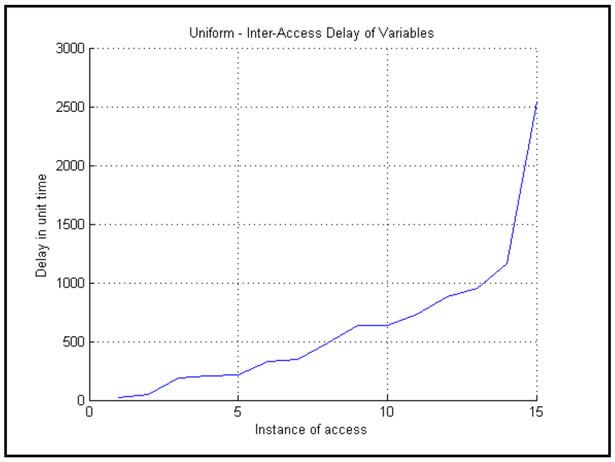


Figure 2: Fragmentation of blocks for 3-Generation GC during steady state of the Flash device.



(a) Cumulative accesses for all variables



(b) Inter-access delay for one variable

Figure 3: Uniform Distribution

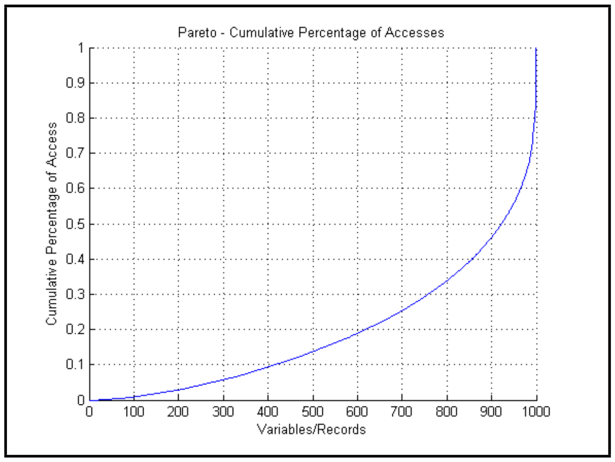
- **Gamma-N-Generation**:

The GC that will be implemented will be the one that has the best efficiency for all three traffic 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.

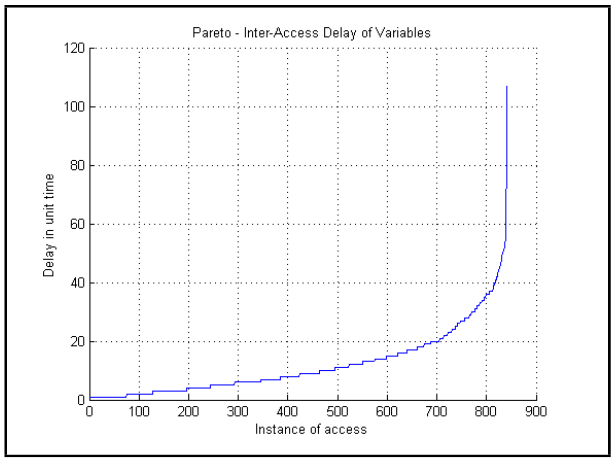
Figure 3a indicates the cumulative percentage of access for all variables. The graph shows a linear rise in the plot and this means that all variables are equally likely to be picked up by the application. The figure 3b on the other hand shows the time between accessing the same variable. Except a sharp rise towards the end, the plot increases linearly. This means that the variable once accessed, will be accessed randomly at another instance in time.

Figure 4a indicates the cumulative percentage of access for all variables. The graph shows an exponential rise in the plot and this means thatl variables towards the end are more likely to be picked up than those at the beginning, by the application. The figure 4b on the other hand shows the time between accessing the same variable.There is a sharp rise towards the end, and rest of the plot is linear. This means that the variable once accessed, may not be accessed at another instance in time except towards the end where it gets accessed much more often.

Figure 5a indicates the cumulative percentage of access for all variables. The graph shows a linear rise in the plot and this means that all variables are equally likely to be picked up by the application.

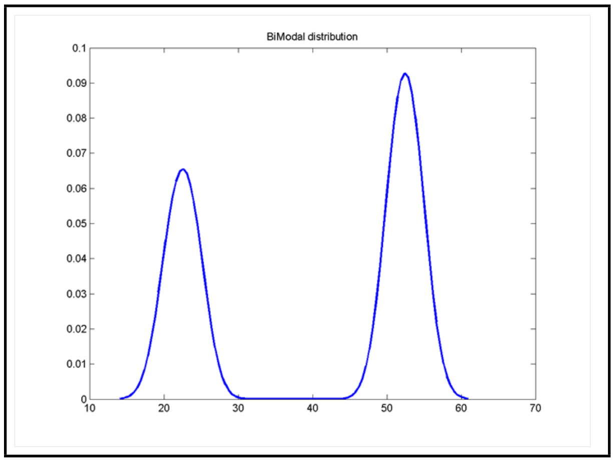


(a) Cumulative accesses for all variables



(b) Inter-access delay for one variable

Figure 4: Pareto Distribution



(a) Cumulative accesses for all variables

Figure 5: BiModal Distribution

5 Theoretical model - FIFE Uniform

Assumptions:

Symbol	**Explanation**	**Assumption**
B	Total blocks in a flash device	128
BlockSize	The size of a block	8192
Count of records per fullness level	N	-
Average records per block	R_b	-
Dummy records per block	dummyRec	1

Table 5 shows the list of variables used.

The device size is, deviceSize = B * BlockSize
The fullness level ranges between 1% and 98%
Average number of records per block,

$$R_b = \frac{BlockSize}{AvgRecSize} - dummyRec \tag{1}$$

Total number of accesses is, (Number of blocks * Records per block) - (Number of records)

$$NumOfAccess = ((B-1) * Rb) - (N-1) \tag{2}$$

NumOfAccess denotes the number of accesses after which a record in a certain block is accessed again. The equation can also be written as: $((B-1) * Rb) - (B-1) * E(ARB_c)$
where $(B-1) * E(ARB_c)$ is the total active records in the flash, which is N.

((B-1) * Rb) gives the total records (both active and inactive) in the entire flash and subtracting N removes the active records and leaves only the inactive. Therefore after going through the entire set of active and inactive records, we access a certain active record (hence the reason behind subtracting (N-1)). The above equation gives the best case scenario where N records are spread evenly across (B-1) blocks.

The expectation for a record to be hit is $\frac{1}{N}$, which is the pdf for Uniform distribution. Let the probability of missing a record be denoted by P_m.

$$Pm = \frac{(N-1)}{N} \tag{3}$$

Let the expectation of active records per block that is going to be cleaned be denoted by $E(ARB_c)$

Expectation is then, $E(ARB_c) = (P_m)^{NumOfAccess} * Rb$

The intuition behind the equation is that, a record is missed for NumOfAccess times before it is hit. Multiplying by Rb gives the relation for all records in a certain block.

The expectation of the average number of records per block can be of the exponential form $1 - e^{\lambda.x}$ where x is the fullness of flash (%). To find λ we can equate the exponential function to R_b when the flash is 100% full. The corresponding equation is:

$$1 - e^{\lambda.x} = R_b$$

$$\lambda = \frac{log(1 - R_b)}{100}$$

where 100 represents the fullness of the flash in percentage.

Expectation can therefore be given as:

$$E(ARB_c) = (1 - e^{\lambda.x})$$

where x is the fullness of flash (%)

For a Uniform Distribution, let the total accesses to the Flash be: 100,000. Out of this, let 1/2 be writes. Hence totalWrites = 50000

The total number of times Garbage Collector(GC) is called decides the total move cost. During one cycle, when active records are moved from one block (x) to another (y), next invocation of the GC is related to the amount of free space in block y. Hence the relation for number of times GC is invoked is:

$$NumberGCRuns = \frac{totalWrites}{(R_b - E(ARB_c))} \tag{4}$$

where $(R_b - E(ARB_c))$ gives the number of free slots where records can be stored.

The fullness of the Flash is given by:

$$FlashFullness = \frac{((N * AvgRecSize)}{deviceSize} \tag{5}$$

The expected move cost is:

$$E(MoveCost) = E(ARB_c) * AvgRecSize * NumberGCRuns \tag{6}$$

The useful write cost is cost involved in writing data to the Flash without any overhead of moving or erasing records:

$$UsefulWriteCost = totalWrites * AvgRecSize \tag{7}$$

The actual write cost is the cost of erasing and moving records in order to write a new record:

$$ActualWriteCost = UsefulWriteCost + ExpMoveCost \tag{8}$$

Efficiency of a Flash device is then:

$$Efficiency = \frac{UsefulWriteCost}{ActWriteCost} \tag{9}$$

6 Theoretical model - FIFE Pareto

Total number of accesses is, (Number of blocks * Records per block) - (Number of records)

$$NumOfAccess = ((B-1) * Rb) - (N-1) \tag{10}$$

Minimum possible value of a variable X below which it will be accessed with increased frequency

$$X_m = 0.2$$

Positive parameter for Pareto distribution is:

$$\alpha = 1.3$$

Average number of records per block can also be represented as,

$$1 - e^{\lambda.x} = R_b$$

$$\lambda = \frac{log(1 - R_b)}{100}$$

where 100 represents the fullness of the flash (%)

The calculated value of λ is used for different substitutions of x (flash fullness) to get the expectation of active records per block that is going to be cleaned.

$$E(ARB_c) = 1 - e^{\lambda.x}$$

The expectation for a record to be hit is the mean of a random variable for Pareto distribution, which is given by:

$$E(X) = \begin{cases} \dfrac{\alpha.X_m}{\alpha - 1} \; if \alpha > 1 \end{cases}$$

Therefore, expectation of a miss for Pareto distribution is:

$$P_m = 1 - \frac{\alpha.X_m}{\alpha - 1}$$

The expectation of active records in the block set to be cleaned can be assumed to be exponentially distributed. Expectation can be given as:

$$E(ARB_c) = (1 - e^{\lambda.x})$$

where x is the fullness of flash.

7 Theoretical model - 2-Gen Uniform

Assumptions:
Average size of a record = s
Size of Gen1 = F_1
Size of Gen2 = F_2

Total records that can be accomodated in Gen 1, $R_{gen1} = \frac{F_1}{s}$

Total records that can be accomodated in Gen 2, $R_{gen2} = \frac{F_2}{s}$

X_1 = Expectation of Active Records in Gen 1
X_2 = Expectation of Active Records in Gen 2

Probability of a miss, $P_m = \frac{(N-1)}{N}$

Records moved over to Gen 2 during GC is δ_2
Records left over in Gen 1 after GC is $\delta_1 = X_1 - \delta_2$

$$X_1 = \delta_1 + \delta_2 \tag{11}$$

$$X_1 = {P_m}^{A_1} * R_{gen1} \tag{12}$$

$$X_2 = {P_m}^{A_2} * R_{gen2} \tag{13}$$

Count of accesses between cleans of Gen 1, $A_1 = \frac{F_1}{s} - \delta_1$
Count of accesses between cleans of Gen 2, $A_2 = \frac{F_2}{s} - \delta_2$

Count of accesses between cleans of Gen 2 is, $A_2 = A_1 * R_{21}$, where R_{21} is factor defined below
Ratio of cleans between Gen 1 and 2 is, $R_{21} = \frac{FreeSpaceRemaininginGen2}{X_1}$

Free Space remaining in Gen 2, $FS_2 = R_{gen2} - X_2$
Therefore, $R_{21} = \frac{FS_2}{X_1}$

8 Performance criteria

There are several measurements that indicate the performance of the GC algorithms. They are plotted and the different graphs that are generated are as follows:

- **Efficiency Vs Fullness (normal scale)**

  Efficiency is the primary parameter for deciding which GC is better than the rest. It denotes the amount of work done in order to write data. Lesser the work done better is the GC.

- **Efficiency Vs Fullness (log scale)**

  This is the log (base 10) representation of the above graph. These denote how fast or slow efficiency of a GC changes.

- **GC time**

  Total time taken by a GC beginning from its invocation till it finds sufficient space to store a record. This is also an important criterion in deciding on a GC.

- **Fullness Vs Actual Write cost**

  Actual write cost is used to calculate the Efficiency of the GC algorithm. Comparing the write cost with the erase cost and the overall efficiency helps in understanding why a given GC performs the way it does.

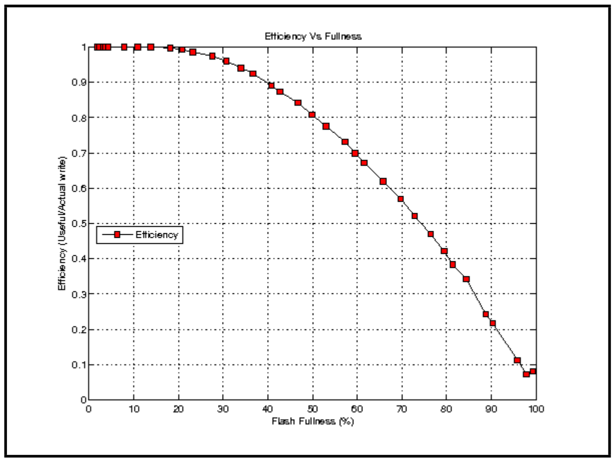
9 Results

9.1 Uniform Distribution

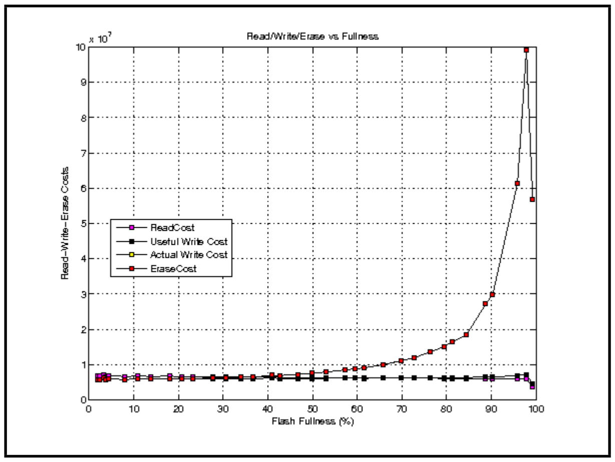
Figure 6a shows a gradual drop in efficiency for the FIFE as fullness increases. This is because as fullness increases the amount of inactive records per block decreases and the GC has to work harder to find space for new records.

The corresponding figure 7a for LAC has almost the same pattern as that for FIFE. The figures that depict the read/write/erase costs show that majority of the work is erasing blocks which contributes the most towards the drop in efficiency.

9.1.1 FIFE



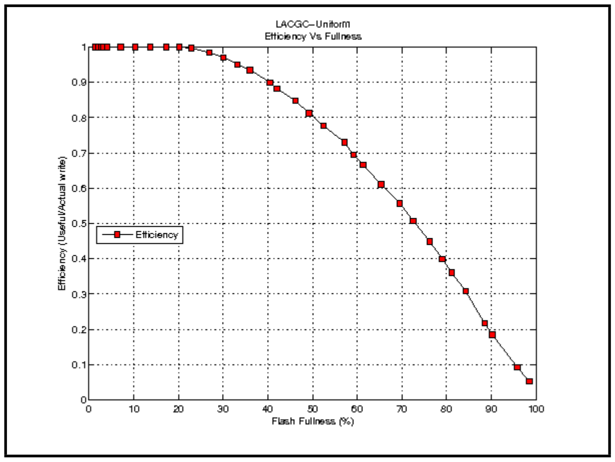
(a) Efficiency Vs Fullness



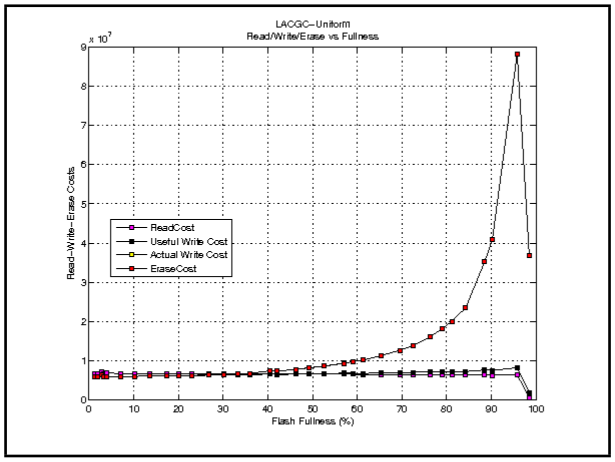
(b) Read, Write, Erase costs Vs Fullness

Figure 6: FIFE - Uniform Distribution

9.1.2 LAC



(a) Efficiency Vs Fullness

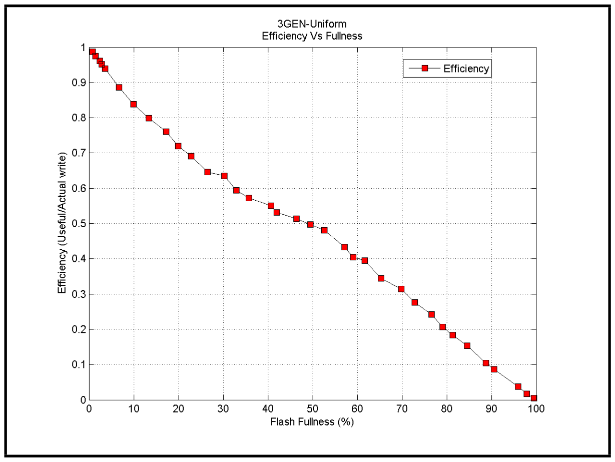


(b) Read, Write, Erase costs Vs Fullness

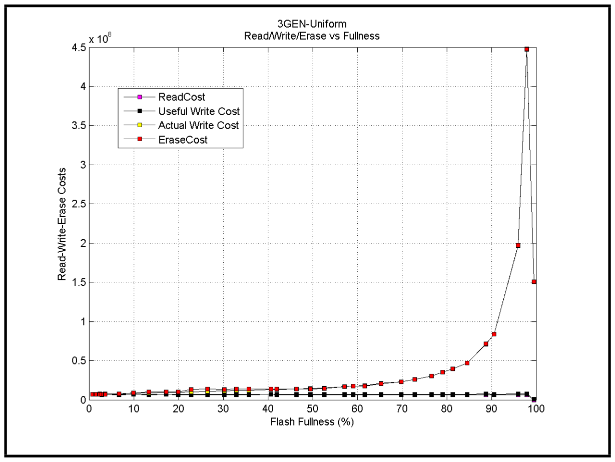
Figure 7: LAC - Uniform Distribution

9.1.3 3-Generation

Figure 8a is almost a linear drop in efficiency. As can be observed, the drop in efficiency is much faster for 3 generation GCs when compared with FIFE algorithms. This is because the amount of blocks in the first generation is less than that in FIFE. Also, when the GC is invoked, it moves all active records from the lower to the higher generations and it has to erase the entire first generation. Even though no effort is spent in moving around active records at regular intervals, the overall effort vs writing a new record is much more than that in FIFE.



(a) Efficiency Vs Fullness

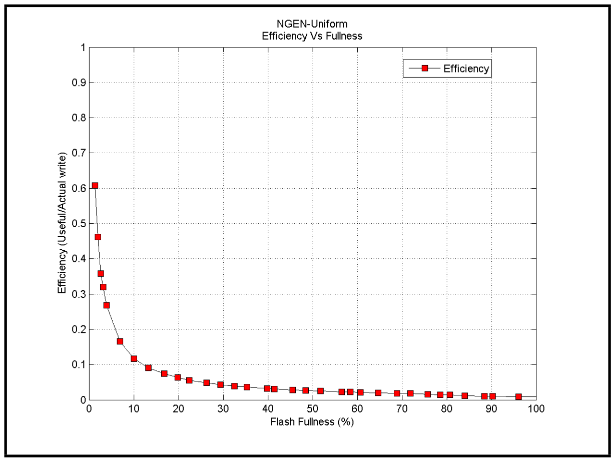


(b) Read, Write, Erase costs Vs Fullness

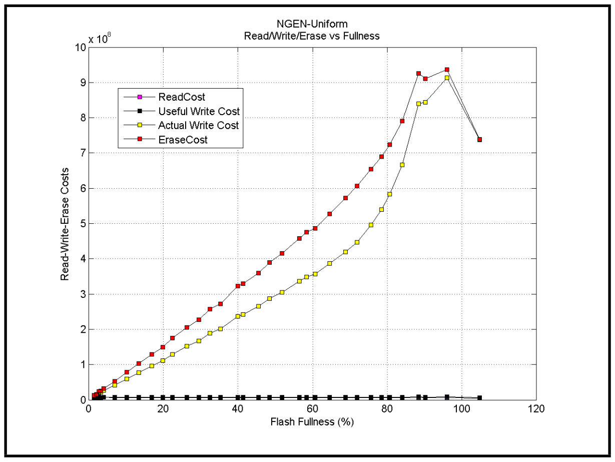
Figure 8: 3Gen - Uniform Distribution

9.1.4 N-Generation

Figure 9a shows a sharp drop in efficiency even at lower fullness levels. This is because only one block is used for writing new records and the GC has to move records to higher generations to make space for new incoming records. Ideally this algorithm should have high efficiency for data with high locality of reference. We conducted a few experiments on this premise by generating data that has high locality of reference. But even in that case, we observed a similar behaviour. This is again due to the fact that there is very little space for writing new records.



(a) Efficiency Vs Fullness



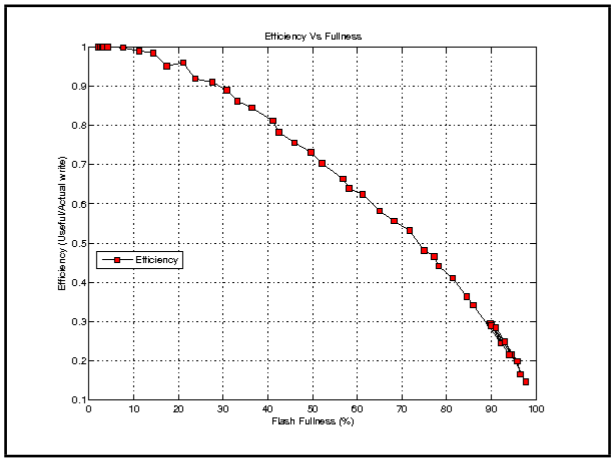
(b) Read, Write, Erase costs Vs Fullness

Figure 9: NGen - Uniform Distribution

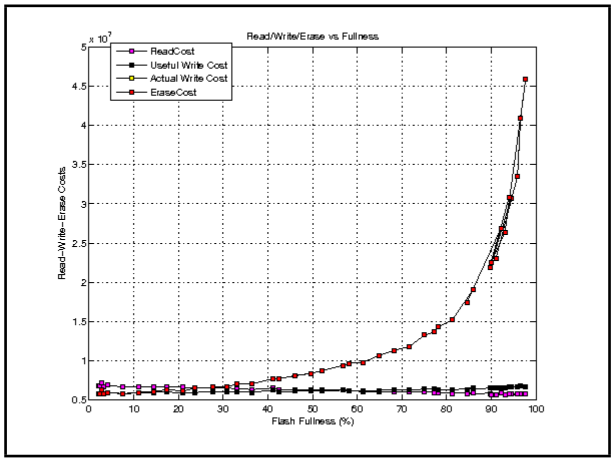
9.2 Pareto Distribution

9.2.1 FIFE

Figures 10a 11a are similar to those for Uniform Distribution, except that at lower fullness levels, the drop in efficiency is much faster. This is due to the cost involved in moving records that are "cold". In Uniform, there is a higher chance for every record to be invalidated, whereas in Pareto, few records are inactivated more often. Pareto distribution has more "cold" data than Uniform and hence contributes more towards the move cost.



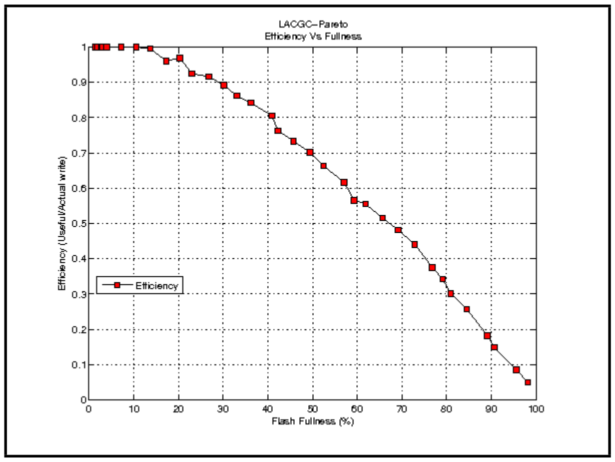
(a) Efficiency Vs Fullness



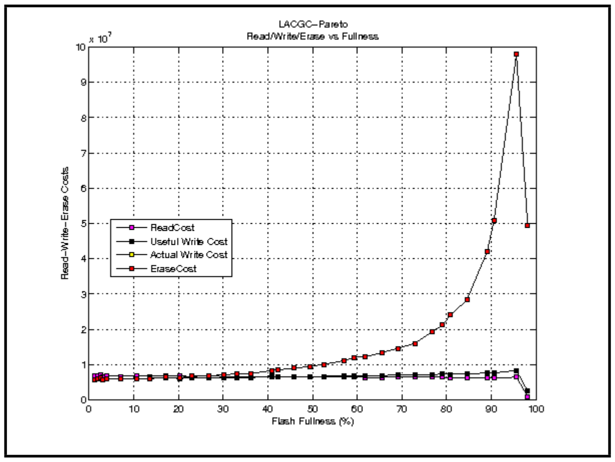
(b) Read, Write, Erase costs Vs Fullness

Figure 10: FIFE - Pareto Distribution

9.2.2 LAC



(a) Efficiency Vs Fullness

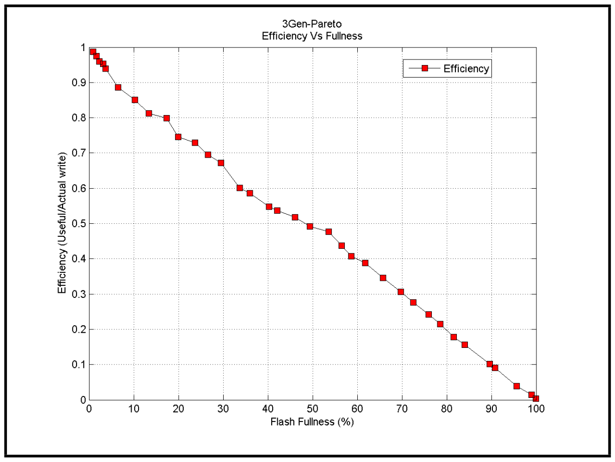


(b) Read, Write, Erase costs Vs Fullness

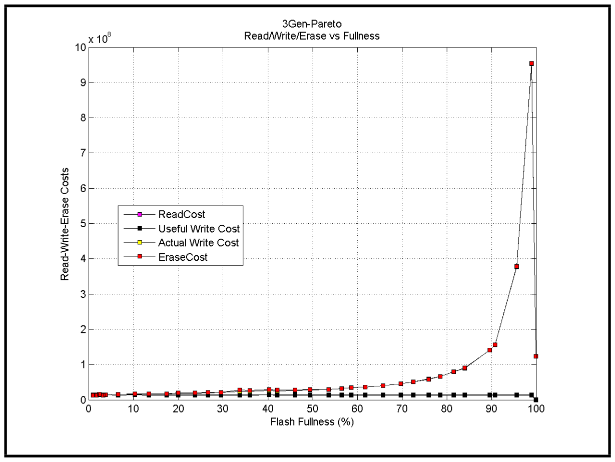
Figure 11: LAC - Pareto Distribution

9.2.3 Generational

Figures 12a 13a 14a show a similar behaviour to that of Uniform distribution. These set of graphs are counter-intuitive and therefore have to be analyzed a little deeper. For instance, in 3-Generation GC, when the first generation becomes full and when the GC is invoked, the active records in Gen-1 are moved to Gen-2. This pattern continues and at steady state, Gen-2 has both active and inactive records and Gen-3 also has the same, but has a higher distribution of active records. Though more space is created during one cycle of GC, the cost per GC over the life-time of the flash is almost the same for Pareto and for Uniform. Therefore, we decided not to consider erase cost, but only the move cost.

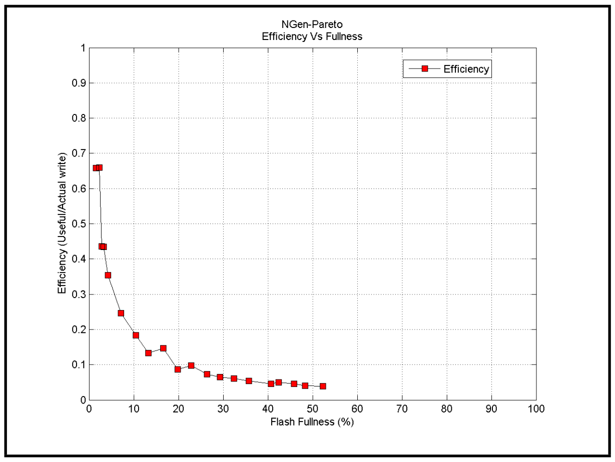


(a) Efficiency Vs Fullness

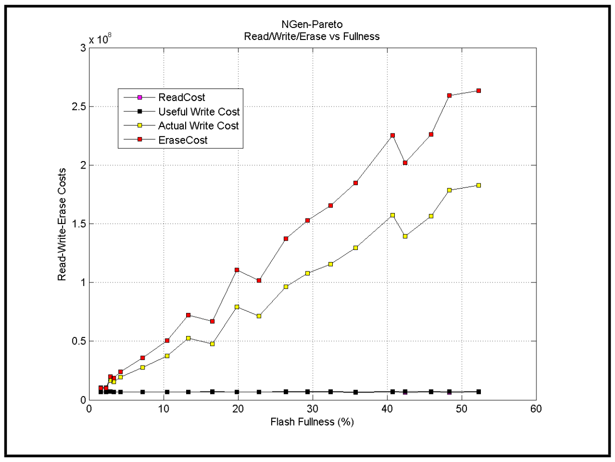


(b) Read, Write, Erase costs Vs Fullness

Figure 12: 3Gen - Pareto Distribution

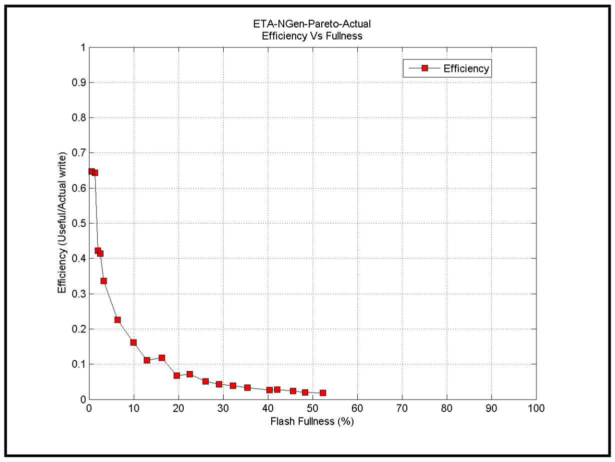


(a) Efficiency Vs Fullness

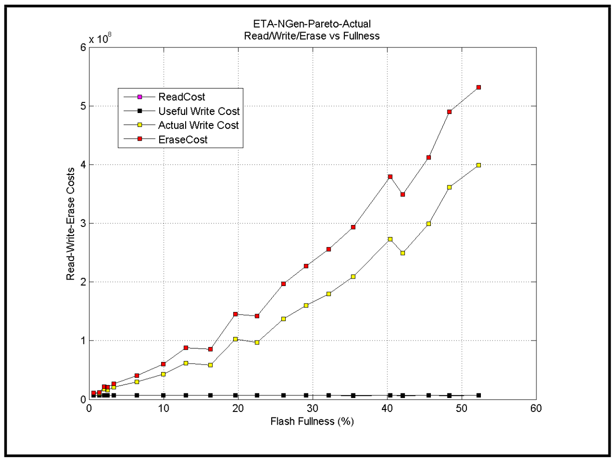


(b) Read, Write, Erase costs Vs Fullness

Figure 13: NGen - Pareto Distribution



(a) Efficiency Vs Fullness



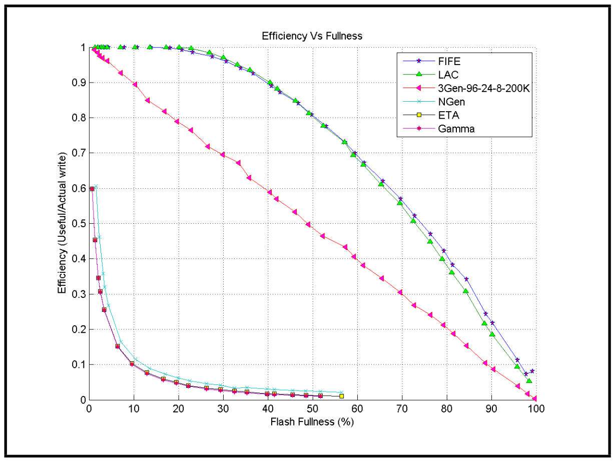
(b) Read, Write, Erase costs Vs Fullness

Figure 14: Eta-NGen - Pareto Distribution

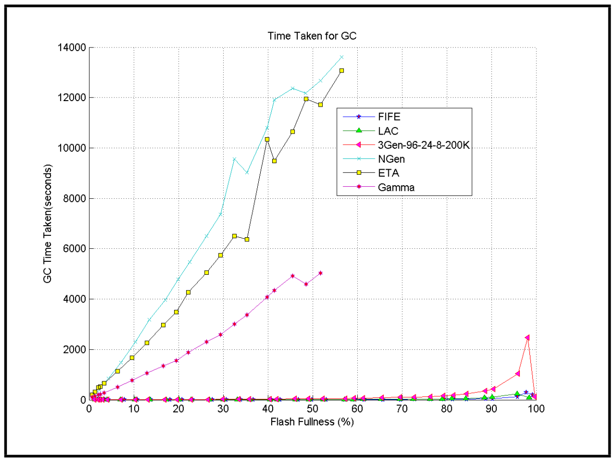
9.3 Efficiency Vs Fullness - Combined Graphs

Figures 15a 16a 17a show the plots for the various GCs. As can be observed, the Round-Robin style of algorithms occupy the top half of the chart and the generational algorithms occupying the bottom half. The 3-Gen algorithm comes in the middle of the chart. The reason behind this has been mentioned in the previous results.

9.3.1 Uniform Distribution



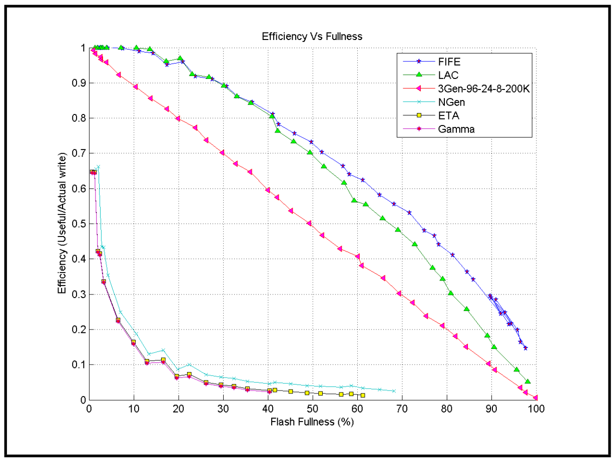
(a) Efficiency Vs Fullness



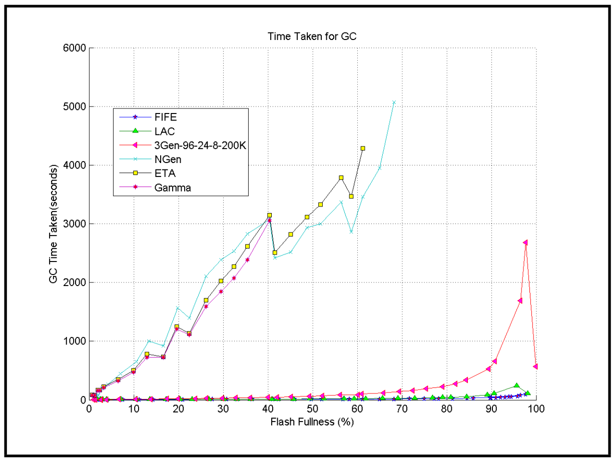
(b) Time Taken by GCs Vs Fullness

Figure 15: Uniform Distribution-Efficiency Vs Fullness for all GCs

9.3.2 Pareto Distribution



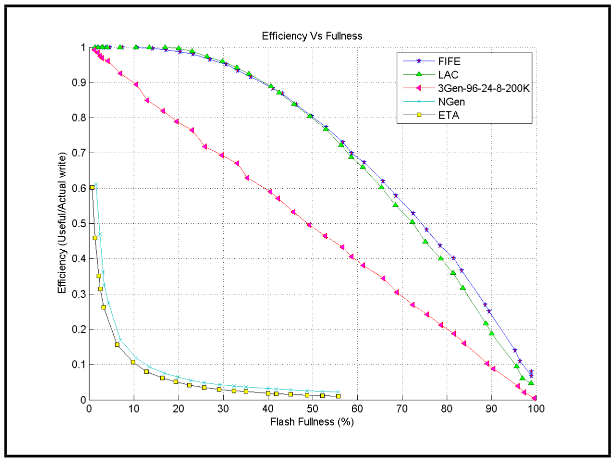
(a) Efficiency Vs Fullness



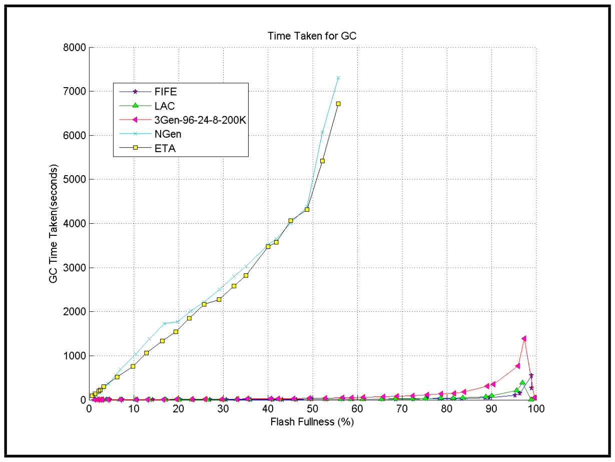
(b) Time Taken by GCs Vs Fullness

Figure 16: Pareto Distribution-Efficiency Vs Fullness for all GCs

9.3.3 BiModal Distribution



(a) Efficiency Vs Fullness

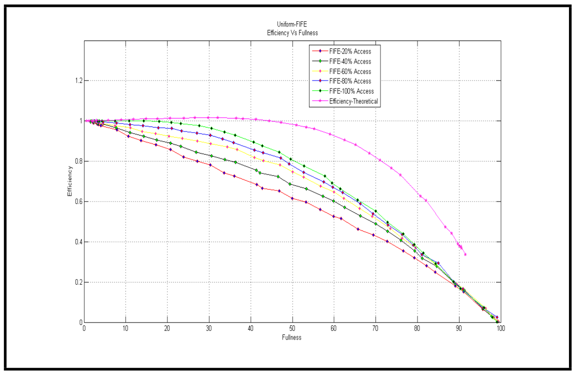


(b) Time Taken by GCs Vs Fullness

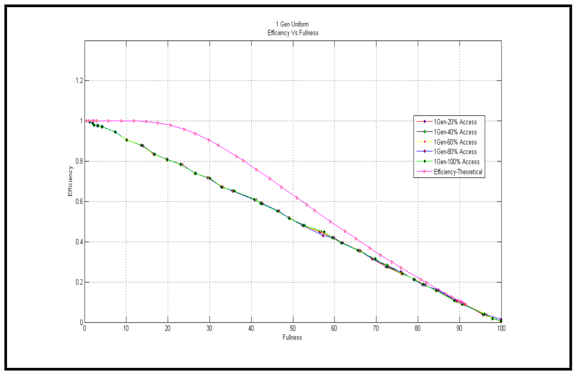
Figure 17: BiModal Distribution-Efficiency Vs Fullness for all GCs

9.4 Theoretical and Simulation results - Uniform Distribution - with different percentages of access

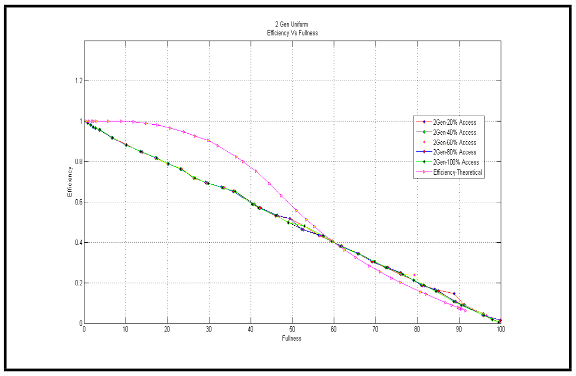
Combination of theoretical and simulation results. Attempts to beat FIFE's efficiency by generational algorithms by accessing 20%, 40%, 60%, 80% and 100% of the records.



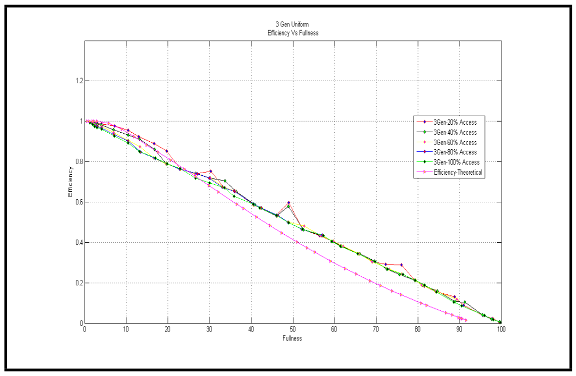
(a) Shows the Efficiency vs Fullness plot for FIFE GC algorithm. It includes the theoretical plot as well



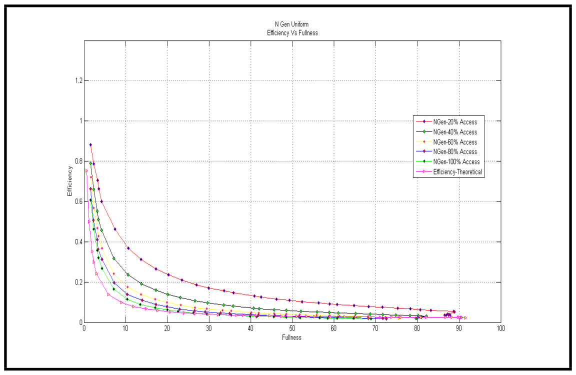
(b) Shows the Efficiency vs Fullness plot for 1Gen GC algorithm. It includes the theoretical plot as well



(c) Shows the Efficiency vs Fullness plot for 2Gen GC algorithm. It includes the theoretical plot as well



(d) Shows the Efficiency vs Fullness plot for 3Gen GC algorithm. It includes the theoretical plot as well



(e) Shows the Efficiency vs Fullness plot for NGen GC algorithm. It includes the theoretical plot as well

Figure 18: Theoretical and Simulation results - Uniform Distribution - with different percentages of access

10 Simulation Setup

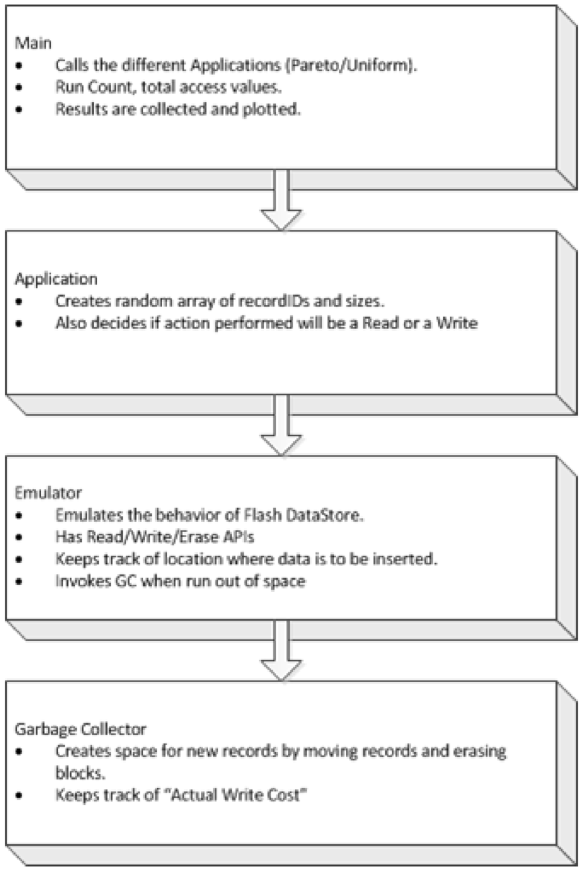

Main
Calls the different Applications (Pareto/Uniform).
Run Count, total access values.
Results are collected and plotted.
Application
Creates random array of recordIDs and sizes.
Also decides if action performed will be a Read or a Write
Emulator
Emulates the behavior of Flash DataStore.
Has Read/Write/Erase APIs
Keeps track of location where data is to be inserted.
Invokes GC when run out of space
Garbage Collector
Creates space for new records by moving records and erasing blocks.
Keeps track of "Actual Write Cost"

Figure 19: Simulator Flow chart

As shown in figure 19, **Main** program controls the number of times the simulation is to be run and which application gets called among other things. It also collects the results returned by the **Application** and generates the graphs. It invokes the **Application** program which contains the implementation for Uniform, Pareto or BiModal type of distributions. Based on the toss of a coin, it decides if the next operation will be a read or a write. Every read and write operation invokes the **Emulator** which contains functionalities such as reading/writing a record, maintaining the indices of the records within the Flash and so on. When space runs out, the **Emulator** invokes the **Garbage Collector** whose primary purpose is to free up space by moving around active records and erasing a block of inactive records. It is successful if it finds sufficient space for the next record to be inserted and it returns an exception back to the Emulator if it is not able to find space in the entire Flash.

11 Related Work

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

This paper compares the Greedy and Cost-Benefit 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-Benefit 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.

11.2 Cleaning Policies in Mobile Computers Using Flash Memory

This paper compares the Greedy, Cost-Benefit 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 flash 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 different 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 inefficiency comes from thinking of hot data as cold (move hot data too soon). For Uniform, FIFE is the best, but for Pareto age definitely has to be considered.

11.3 Questions

What is the recommended algorithm for Pareto distribution? Are any of the generational algorithms good or should they not be used?

Conclusions and Future Work

Flash devices are making a great headway in becoming the storage device of choice in Mobile and Embedded systems. But they still have disadvantages such as a slow write speed which prevent them from being used as the primary storage device replacing RAMs. In order for them to perform optimally, Garbage Collectors are very important. This work analyzed the performance of five different GC algorithms against the most common of the traffic patterns in real-world applications (Uniform, Pareto and BiModal). The results that have come out of this study have gone against the existing results in current literature and hence have the potential to make rapid progress towards using Flash as primary storage.

Future work

Acknowledgements

I am grateful to Dr.Mukundan Sridharan who guided me during the course of this work. Thanks also to Dr.Rajiv Ramnath who during our weekly meetings, guided me in the right direction by asking lot of questions and helping me get the answers for them. My sincere gratitude also to Dr.Kenneth Parker and Dr.Anish Arora for giving me the opportunity to work on this wonderful project. Last but not the least, I am thankful to the professors and friends who helped me when I was stuck at a difficult problem.

References

[1] Aayush Gupta, Youngjae Kim and Bhuvan Urgaonkar. DFTL: a flash translation layer employing demand-based selective caching of page-level address mappings. *ASPLOS '09*, pages 229–240, 2009.

[2] Evgeny Budilovsky, Sivan Toledo and Aviad Zuck. Prototyping a High-Performance Low-Cost Solid-State Disk. *SYSTOR '11*, (13), 2011.

[3] Jonathan Tjioe, Andrés Blanco, Tao Xie and Yiming Ouyang. Making Garbage Collection Wear Conscious for Flash SSD. *Networking, Architecture and Storage (NAS), 2012 IEEE 7th International Conference on*, pages 114–123, 2012.

[4] Toshiba America Electronic Components, Inc. NAND vs NOR Flash Memory Technology Overview. 2011.

[5] Mendel Rosenblum and John K. Ousterhout. The Design and Implementation of a Log-Structured File System. *ACM Transactions on Computer Systems*, 10:26–52, 1991.

[6] Jai Menon and Larry Stockmeyer. An age-threshold algorithm for garbage collection in log-structured arrays and file systems, 1998.

[7] Ohhoon Kwon and Kern Koh. Swap-aware Garbage Collection for NAND Flash Memory Based Embedded Systems. *Computer and Information Technology, 2007. CIT 2007. 7th IEEE International Conference on*, pages 787–792, 2007.

[8] Mei-Ling Chiang, Paul C. H. Lee and Ruei-Chuan Chang. Cleaning policies in mobile computers using flash memory. *Journal of Systems and Software*, 48, 1999.

[9] LI-PIN CHANG and TEI-WEI KUO. Efficient Management for Large-Scale Flash-Memory Storage Systems with Resource Conservation. *ACM Transactions on Storage*, 1:381–418, 2005.

[10] Kenneth W. Parker. Portable electronic device having a log-structured file system in flash memory. *US Patent number: 6535949*, 2003.

[11] Borja Peleato, Rajiv Agarwal and John Cioffi. On The Distribution of Valid Pages with Greedy Garbage Collection for Nand Flash. *Statistical Signal Processing Workshop (SSP), 2012 IEEE*, pages 500–503, 2012.