國立交通大學

資訊科學與工程研究所

碩 士 論 文

應用於矽態硬碟上的低成本以 LBA 為主的平均磨損演算法

A Low-Cost LBA-Based Wear Leveling Algorithm for Solid-State Disk

研 究 生：黃千庭

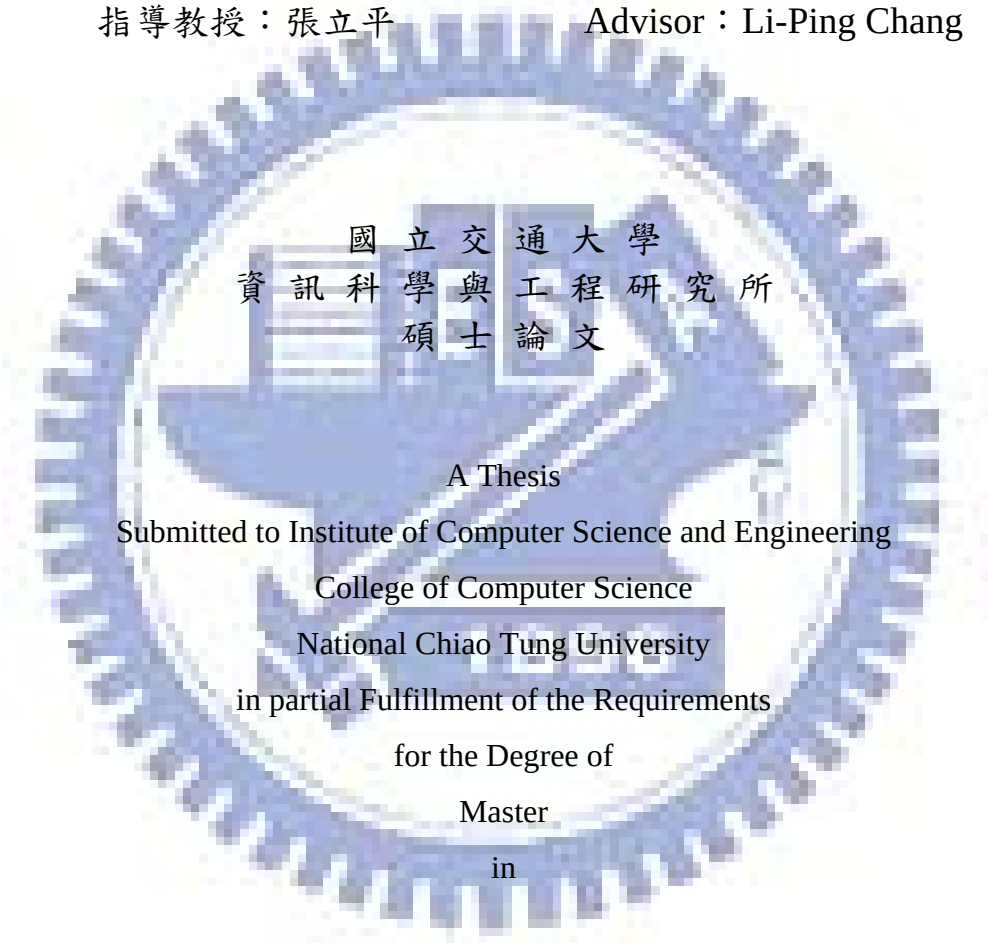
指導教授：張立平　教授

中 華 民 國　九 十 七　年 七 月

應用於矽態硬碟上的低成本以 LBA 為主的平均磨損演算法
A Low-Cost LBA-Based Wear Leveling Algorithm for Solid-State Disk

研 究 生：黃千庭　　Student：Chien-Ting Huang

指導教授：張立平　　Advisor：Li-Ping Chang



國 立 交 通 大 學

資 訊 科 學 與 工 程 研 究 所

碩 士 論 文

A Thesis

Submitted to Institute of Computer Science and Engineering

College of Computer Science

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master

in

Computer Science

June 2006

Hsinchu, Taiwan, Republic of China

中華民國九十七年七月

應用於矽態硬碟上的低成本以 LBA 爲主的平均磨損演算法

學生：黃千庭　　指導教授：張立平

國立交通大學資訊科學與工程研究所碩士班

摘　　要

Endurance 為 NAND Flash Memory 一個相當嚴重的缺點。然而有個新的技術與新的應用的出現，使得 Endurance 更加的嚴重，即 SSD(Solid-State Disk)和 MLC(Multi-Level Cell)。而為了減輕 Endurance 的問題，因此有一個機制來解決此問題，即 wear leveling algorithm。目前已有相當多的 Wear leveling algorithms，每一個 wear leveling 的效果、overhead 以及耗用的 RAM space 都不同。有些 wear leveling 效果雖然很好，但是耗用的 RAM space 過多，而有些則是 overhead 過高，因此在實際系統使用上皆不太適合。因此我們此篇 paper 則是要設計出一個 Low-cost 的 wear leveling algorithm，不同於以往的 wear leveling algorithm，我們的 wear leveling algorithm 主要是處理異常磨損的區塊，並以 LBA(Logical Block Address)取代以往常使用的 PBA(Physical Block Address)，如此非常的省資源並且不會有太高的 overhead，亦有不錯的效果。在最後的實驗，可看出在相同的 Standard Deviation 之下，我們的 wear-leveling 的 Mean 較低，即 overhead 低，而在相同的 Mean 之下，我們的 wear-leveling 的 Standard Deviation 較低，即 wear leveling 的效果較好。

關鍵字：NAND 快閃記憶體(NAND Flash Memory)，平均磨損演算法(Wear leveling algorithm)，矽態硬碟(Solid-State Disk)。

A Low-Cost LBA-Based Wear Leveling Algorithm for Solid-State Disk

student：Chien-Ting Huang Advisors：Dr.Li-Ping Chang

Institute of Computer Science
National Chiao Tung University

ABSTRACT

Endurance is a serious drawback of NAND flash memory. As the development of technology, MLC NAND flash memory and Solid-State Disk appear, make endurance worse and worse. In order to alleviate the problem of endurance, we should have a mechanism to solve it, called wear leveling. There are already a lot of wear leveling algorithms. All of them have difference effective, cost and the usage of RAM space. Some of them may have better effective, but using a lot of RAM space or get higher cost such that cannot be implemented on real system. In this paper we propose a low-cost wear leveling algorithm. Different with other wear leveling algorithm, the main idea is to attend to the prodigiously erasure of blocks. We use LBA (Logical-Block-Address) instead of the common used PBA (Physical-Block-Address). We will save more resource and have low cost, and get better effective. After experiments, we can see that at the same standard deviation, we have lower cost than other wear leveling algorithms, and vise versa.

Keyword: NAND flash memory, Wear leveling algorithm, Solid-State Disk

誌　　謝

終於抵達寫致謝詞的這一刻，這意味著研究所生涯即將正式的畫上句點，回顧兩年來的經歷，內心充滿幸福與感謝，首先誠摯的感謝指導教授張立平老師，老師悉心的教導使我得以了解嵌入式系統的深奧，不時的討論並指點我正確的方向，使我在這些年中獲益匪淺。因為有你的體諒及幫忙，使得本論文能夠更完整而嚴謹。

二年的求學生涯裡，讓我學習了不少新的知識和待人接物的方法，非常感謝，每個老師的敦敦教誨，勤而不懈的教導我，讓我能夠一路走來都很順利。

再來，感謝在這段期間，許辰暉、林松德、鄭家明、許蕙茹同學的幫忙，使得我能順利走過這兩年。實驗室的郭郡杰、蘇宥全、黃士庭、林明毅學弟和廖秀芬學妹們當然也不能忘記，你/妳們的幫忙及搞笑我銘記在心。

研究口試期間，感謝謝仁偉老師、陳雅淑老師和楊家玲老師不辭辛勞細心審閱，不僅給予我指導，並且提供寶貴的建議，使我的論文內容可以更臻完善， 在此由衷的感謝。

感謝系上諸位老師在各學科領域的熱心指導，讓我增進各項知識範疇， 在此一併致上最高謝意。

最後，謹以此文獻給我摯愛的雙親。

目　　錄

表　　目　　錄

圖　　目　　錄

一、Introduction

由於 NAND Flash Memory 的硬體特性優勢：耐震以及體積小，使得大部份的嵌入式系統大都採用 NAND Flash Memory 做爲其儲存系統。因此 NAND Flash Memory 的應用相當的廣泛，如：行動電話、多媒體播放器（PMP）、數位相框等。隨著技術的進步，NAND Flash Memory 的容量也跟著變大，因此衍生出了一個新的應用以取代行動電腦內的硬式磁碟（Hard Drive Disk），固態硬碟-Solid-State Disk(SSD)，用以取代傳統的硬式（Hard Disk）。

當 NAND Flash Memory 空間不夠時，則必須將一些 invalid 的 Data 刪除掉，藉以回收區塊（區塊），而由於 NAND Flash Memory 的特性，必須將區塊抹除後才能再寫入 Data 到該區塊中，因此回收一個區塊就要對該區塊做 Erase，但 NAND Flash Memory 有個特別的 Issue，即它有 Endurance 的問題，即 NAND Flash Memory 中的每一個區塊的寫入次數有限。在 NAND Flash Memory 中，當出現一個區塊毀損，則其它區塊會加速磨損，進而造成大部份的區塊在短時間內皆毀損。因此爲了避免某些區塊過度的被使用，則必須平均的使用每一個區塊，因此而有了 wear leveling algorithm。Wear leveling 主要是爲了平均使用每一個區塊，以長久維持所有區塊都在可用狀況下，如此可延長 NAND Flash Memory 的壽命。

NAND Flash Memory 的 Endurance 的問題愈來愈嚴重。主要是有兩種原因：（1）NAND Flash Memory 容量愈來愈大，新的應用出現，如：SSD，而這種應用主要爲將作業系統裝置在 SSD 上，而它的 Access Pattern 不同於一般的使用，如：數位相機對儲存系統的 Access 大都是 Sequential Access。作業系統的對儲存裝置的 Access 則通常會有 Locality 的問題，即可能在某個時間內會對同一地方做多次存取，由於此種特性，使得爲了減少空間回收時的成本，會傾向抹除特定幾個區塊，此種運作方式相當不利於 NAND Flash Memory，會造成 NAND Flash Memory 提早毀損。（2）新的 NAND Flash Memory 的出現-MLC(Multi-Level Cell) NAND Flash Memory。原本的 NAND Flash 爲 SLC(Single-Level Cell) NAND Flash Memory，MLC 利用將記憶胞(Memory Cell)電位系分爲多位階達成倍增儲存密度的目的，因此 MLC 相對於 SLC 的優勢爲容量大、成本低，但也使得 MLC Endurance 的問題更嚴重。SLC 的 Endurance 爲 100,000(100K) cycles[5]，而 MLC 的 Endurance 爲 10,000(10K) cycles[6]，差距約爲 10 倍。因此若爲了節省成本而使用 MLC，則會面臨更大的 Endurance 的問題，因此 wear leveling 則相對愈來愈重要。

過去一些 wear-leveling[1,2,3,7]的做法如：Static wear leveling[7]，Static wear leveling 主要概念即爲避免不常被更新的資料長期的置於某些區塊上，以致於那些實體區塊一直未被使用到，但是我們認爲應注意的是更新頻率高的 Data，主要是這種 Data 是造成區塊快速磨損的主因，並且在實際的 workload 中更新頻率高的 Data 比較少，它們會快速地將少數區塊磨損，而大部分區塊則不被磨損，因此應該是去阻止區塊被快速磨損，而不是像 Static wear leveling 在避免某些區塊一直未使用到，將此想法稱爲 Lazy wear leveling。一些 wear-leveling algorithm[3,4]，如：Dual Pool algorithm for wear leveling 即是此種概念，但 Dual Pool algorithm for wear leveling 所需要的資源甚多（如：需要五個 Priority Queues），因此實際實作的可能性很低。

我們根據 Realistic workload 去分析，發現更新頻率高的 Data 比更新頻率低的 Data 少得多，因此我們認爲 wear leveling 應該以不同的方法去解決，而提出一個不同於以往的解決 wear leveling 的新想法。由於更新頻率高的 Data 遠少於更新頻率低的 Data，根據此特點我們可利用非常少的 RAM Space 資源即可追蹤大部份更

新頻率高的 Data，並根據此來實現 Lazy wear leveling。由於這些更新頻率高的 Data 會造成某些區塊快速的磨損、老化，而其它的區塊則不被磨損，因此爲了阻止某些區塊快速的磨損，Lazy wear leveling 主要目的在於將更新頻率高的 Data 放入到年輕的區塊中，以阻止已過度磨損的區塊持續磨損。在以往的方法中，追蹤更新頻率高的 Data 的方法都是去追蹤常常被 erase 的 Physical Block (Dual Pool algorithm for wear leveling)，因爲會造成區塊常常被 erase 的原因主要是因爲該區塊中存有更新頻率高的 Data。不過當 Wear leveling 開始動作去改變區塊抹除的狀況，會使得大部份的區塊被抹除的次數會慢慢被調和，使得我們追蹤的對象變得很多，使得必須要用到相當多的 RAM space 資源才能追蹤到更新頻率高的 Data。不過我們發現，因爲存取區域性（Access locality）中，經常存取的 LBA(Logical Block Address)非常少，而且經常存取的 LBA 集合改變並不頻繁，即使 Wear leveling 開始動作去改變區塊抹除的狀況，也不會影響到追蹤的對像，因此我們發現追蹤 LBA 事實上比追蹤 PBA（Physical Block Address）更省系統資源，而且效果並不比他差。

最後我們對此篇 paper 所提出的方法做了一些實驗，測試在不同的情況下，所得出的實驗結果。最後都達到了不錯的效果，在使用 wear-leveling 後 standard deviation 會下降，即大部份區塊的 Erase Cycle 差距皆不大。在與 static wear leveling 比較時，在相同的 standard deviation 之下，static wear leveling 的 mean 較高，而在相同的 mean 之下，static wear leveling 的 standard deviation 較高。因此不管是 performance 和 overhead 皆比 Static wear leveling 表現的還好。而我們的方法僅用一個固定大小的 Priority Queue 以及一個 Bitmap 達成，使用的資源相較於[3]則少了非常的多，用較少的資源可達到不錯的效果。

二、Problem definition / Related work

2.1 Flash Geometry

NAND Flash Memory 的組成如圖 1，主要是由多個區塊所組成，每一個區塊又是由多個 Pages 所組成。NAND Flash Memory 的一次寫入單位(Write Unit)則爲一個 Page，每一個 Page 會有一個 Spare Area，用以描述 Page 中的 Data，即 Metadata。而刪除單位(Erase Unit)則爲一個區塊，即每次執行 Erase 時都是以區塊爲單位。不同的 NAND Flash Memory 的組成可能不同。如[9] 爲 SLC-small block，表示其 NAND Flash Memory 中一個 Block 有 32 個 Pages，一個 Page 爲 512 Bytes，一個區塊則爲 16K Bytes。[5]爲 SLC-large block，表示其 NAND Flash Memory 中一個 Block 有 64 個 Pages，一個 Page 爲 2K Bytes，一個區塊則爲 128K Bytes。NAND Flash Memory 在更新資料時與一般的區塊 Device 不同，一般的 Block Device，如：Disk，在更新某部份的資料後可直接寫回其原本的實體位置，但是 NAND Flash Memory 無法將更改完後的資料直接寫入至該資料原本所在的實體位置，必須再額外找空閒的 Page 寫入，此動作即稱爲 Outplace-Update。而原本的 Data 則視爲失效（Invalid），但也因爲如此，NAND Flash Memory 中常常會存有很多 invalid data，因此 NAND Flash Memory 常常一段時間即要清除那些 invalid data，以空出空間提供其他 Data 儲存，此動作稱之爲 Garbage Collection。

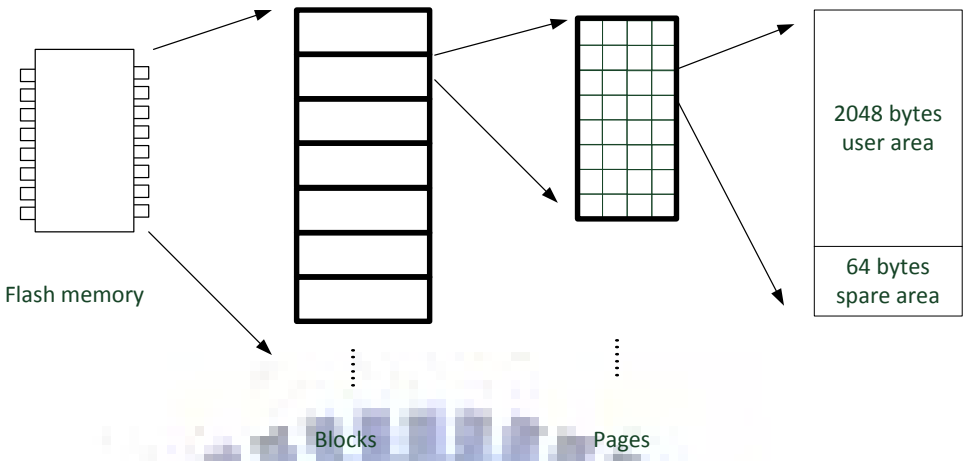


圖 1：Flash Geometry

由於 SSD 與 Hard Drive Disk 的硬體架構的不同，因此以往的 File systems，如：ext2、ext3、NTFS、FAT 等皆無法直接在 NAND Flash Memory 上使用，但 SSD 的應用主要在於取代傳統硬碟，因此在 SSD 上所使用的作業系統的種類相當多，相對會使用到的 File system 則會依所使用的作業系統不同而不同，因此爲了能讓 SSD 能適用於現有的 For Block Device 的 File system，因而有了一個 Translation Layer，如：NFTL(NAND Flash Translation Layer)，即可在 SSD 上面使用現有的 File system（如圖 2）。NFTL 包含了(1)Update Policy，(2)Address Translation 以及(3)Garbage Collection。在我們實驗中主要就是以 NFTL 爲底主，在其架構下設計我們的 wear-leveling algorithm。

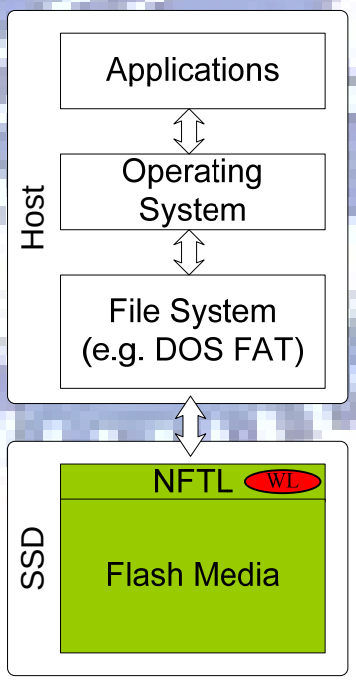


圖 2：Structure of SSD

2.2 Wear-Leveling Concept

由於 NAND Flash Memory 有 Endurance 的問題，因此爲了要延長 NAND Flash Memory 的壽命，主要概念即是平均的使用 NAND Flash Memory 中的每一個區塊。當有一個壞區塊的出現則會加速其他區塊損毀，因爲當可用區塊數量變少後，系統則會花相當多的成本在內部管理，例如回收可用的空間，進而加速其他區塊的磨損，因此要盡可能延長第一個壞區塊的出現，這種延長第一個壞區塊出現的方法即稱爲 wear leveling。

我們可將 Data 分爲兩種類型：更新頻繁的 Data，稱爲 Hot Data，另一即爲更新不頻繁的 Data，稱爲 Cold Data。由於 Hot Data 的存在，常常會造成某些 Block 經常被 Erase，因而逐漸老化，這些老化的區塊稱爲 Old Block，其它少被 Erase 的區塊稱爲 young Block。Hot Data 的存在造成區塊之間 erase 的不平均，使得少部份區塊的磨損程度遠高於其他 Block，如圖 3，藍色的爲放置 Cold Data 的區塊，橘色的則爲放置 Hot Data 的區塊，由於 Hot Data 常常更新，更新的資料寫入其他空的區塊中，而 Garbage Collection 主要就是要回收那些存有 invalid data 的區塊，因此會回收那些存有 Hot Data 的區塊。如此 Hot Data 就會在幾個固定的區塊中來回走動，造成固定幾個區塊的磨損情形比其它區塊嚴重。爲了要有方法能夠平均的使用每一個區塊，而這種方法即稱爲 wear leveling，其目的就是延長 NAND Flash Memory 的壽命。

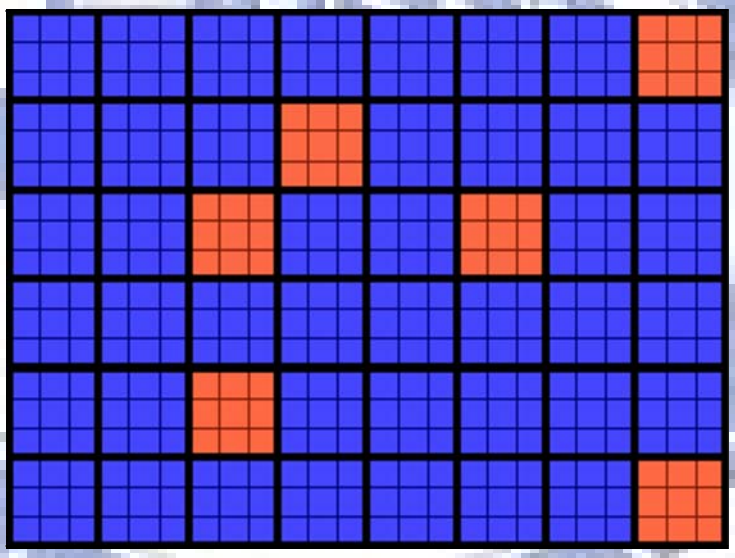


圖 3:Hot/Cold Regulation

現有的 Wear leveling 方法相當多，如[1,2,3,4,7,8]，效果大都相當不錯，wear leveling 設計的好與不好，可以用三個標準來評斷：effective、overhead 以及耗用的 RAM space。如圖 4，三種不同 wear leveling 的效果，(a)即爲不做 wear leveling 或 wear leveling 效果不好的情況下，而(b)和(c)則爲所有區塊都平均磨損，但差別在於(c)的 overhead 相當高，容易造成區塊提早損毀，因爲(b)的 wear leveling 爲效果較好的。

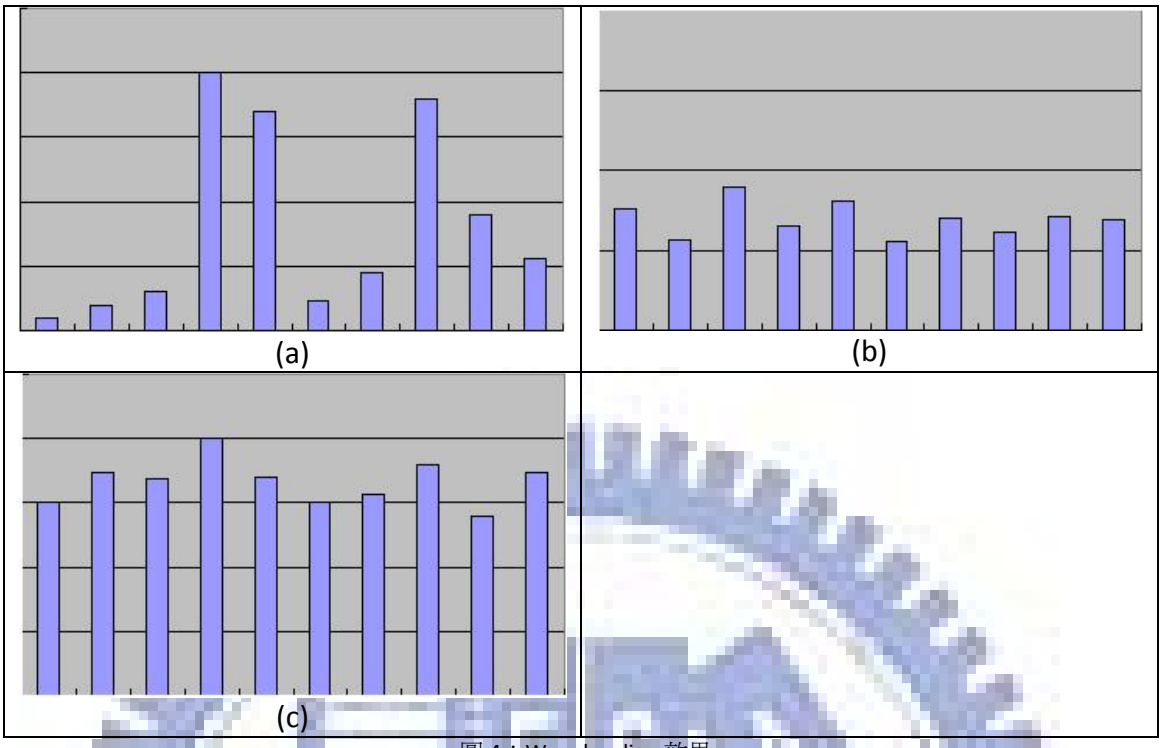


圖 4：Wear leveling 效果

在(a)中爲未啓用 wear leveling 的情況下，可觀察出區塊抹除極不均勻。(b)則爲啓用了 wear leveling，使得大部份的區塊抹除狀況相當的平均，且 overhead 也較低，此爲較好的 wear leveling algorithm。(c)中大部份的區塊抹除狀況亦相當平均，但是 overhead 較高，亦會使得區塊較快損毀。

另外一個設計上的 issue 即爲 Resource，要達到 wear-leveling 的方法相當多，在設計上通常會使用到許多的資料結構，但是 RAM 空間有限，因此在設計上，想達到不錯的效果所使用的資料結構可能就會相當多，使得實際上無法使用。

2.3 Related Work

此篇 paper 所提出之 wear leveling 方法主要在於一個可節省 RAM space 並達到一個不錯的效果，以便能使用在一般的 NAND Flash Memory[5,6,9]以及 SSD 上，我們所提出之方法不僅可在一般 pure SLC SSD、pure MLC SSD 上使用，亦適用在[11]中所提出的 Hybrid SSD 中。

目前已知的 Wear leveling algorithm 已相當的多，但大部份的處理方法皆不大相同，下列將目前已提出之 wear leveling algorithm 約略分爲兩種類型：

- 在 Garbage Collection 時啓動 Wear leveling，如表 1：

表 1：Garbage Collection 相關之 Wear leveling

Name	Wear leveling Policy	Reference	Problem
Kim and Lee	Erase the highest score that calculate in function f(x) among all blocks.	[16]	infeasible, ineffective
CAT	Erase the highest score that calculate in function f(y) among all blocks.	[17]	infeasible, ineffective

- 不在 Garbage Collection 時啓動 Wear leveling，如表 2：

表 2：Garbage Collection 正交之 Wear leveling

Name	Wear leveling policy	Reference	Problem
Static wear leveling	Moving cold data to another disk when necessary.	[7,8,20]	Ineffective.
Turn-Based Selection	In every x time of Garbage Collection, system will choose y blocks to be recycled randomly.	[13,14]	Ineffective.
Dynamic wear leveling	Update data to a new free space.	[4,10,18,19]	Ineffective.
Hot-Cold Swapping	The difference of the erase cycle of the oldest and youngest block is more than threshold.	[1,2,3,15,16]	Pathological behaviors on the oldest block.

我們所設計出的是一個 Low-cost 的 wear leveling algorithm，在已提出的 wear leveling algorithm 中，亦有針對如何去降低 cost 的方法，如：[1]中即是在 RAM 中僅記錄一個實體區塊的實際 Erase cycle，而其它的實體區塊的 Erase cycle 都是相對於該區塊的 Erase cycle 差值，如此即可用少數的 Bit 即可記錄每一個區塊的 Erase cycle，但此方法仍需記錄所有區塊的 Erase cycle，仍耗用不少空間。而在[7]中則是不記錄每個區塊的 Erase cycle 在 RAM 中，而是將區塊分成數個 Groups，而每一個 Group 會有一個 Summary Information，判斷是否做 wear leveling 或 Victim Group 皆是以每個 Group 的 Summary Information 爲主要判斷資訊，影響此方式的最大因素即是 Group size 如何設定及 Summary Information 如何決定，Group size 愈大，雖然可節省更多的 RAM 空間，但 wear leveling 效果相對的即會變更差，而在 Group 中的區塊可能會有極端現象（如 Oldest 區塊及 Youngest 區塊在同一個 Group 中），因此如何取得 Summary Information 必須避免被極端現象所影響會是個問題。在[8]中所提的方法僅利用一個 Bitmap，而不需記錄到區塊的 Erase cycle，利用 Bitmap 將 Cold Data 移到其他區塊中的方法，雖然可節省相當多的空間，但因它未積極的將 Cold Data 移到 Old 區塊中，所以 wear leveling 效果並不是很好。

在以往的 wear leveling 方法中，大部份都是把 Hot Data 放到 Young 區塊中，或把 Cold Data 放到 Old 區塊中，如：[1,2,3,7,8,15,16,17]，而這些方法中擷取 Hot Data 的方式都是依區塊的 Erase cycle 的高低與否來判斷該區塊內的 Data 是否爲 Hot，但這種方式有個缺點，即當 wear leveling 啓動後，大部份的區塊的 Erase cycle 差距不大，因此很難再以此方式判斷 Hot/Cold Data，而我們的方法即不是以此做爲 Hot Data 的判斷依據，是以 LBA 的觀點來判斷是否爲 Hot Data，Hot Data 通常不容易變動，且以 LBA 的觀點即使 wear leveling 啓動了，亦不會影響 Hot/Cold Data 的判斷。

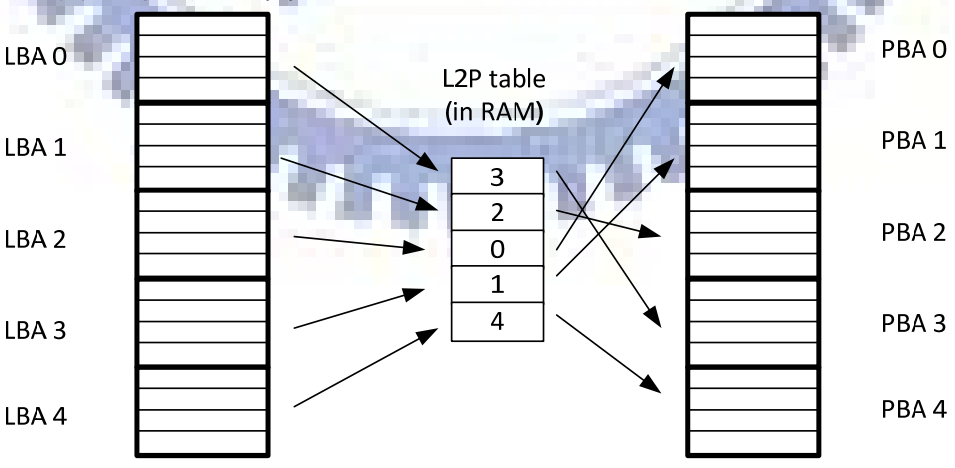
三、Low-Cost LBA-Based Wear-Leveling Algorithm

3.1 NFTL(NAND Flash Translation Layer) & Garbage Collection Policy

3.1.1 NFTL(NAND Flash Translation Layer)

我們的 wear leveling 主要是建構在 NFTL 之上，因此 wear leveling 方法與其運作習習相關，因此在這一節主要是說明 NFTL 的基本原理。爲了讓 NAND Flash Memory 可套用現有 Block-Device 的 File systems，因此要有一個機制能做出轉換的動作，即是將 Operating System 傳下來的資訊轉成適用於 NAND Flash Memory 的方式。因此 M-System 所提出了一個 NFTL[10]架構，可適用在一般的 NAND Flash Memory 及 SSD 上面，而讓它們可直接套用現有的 File systems。在我們的方法中主要也是以 NFTL 爲主要架構，在此描述 NFTL 的架構以及我們所做的一些修改。

NFTL 會在 RAM 中儲存兩張 Tables，其中一個爲 Mapping Table，或稱爲 Logical-To-Physical Table(L2PTable，圖 5(a)中間的圖)，主要是 Block-Level Mapping，L2Ptable 即是記錄 Logical Block 所指到的 Physical Block 的 Address。另一張 Table 即爲 Log Block Table(LBTable，圖 5(b)中右邊的圖)。NFTL 主要透過 L2Ptable，將 Logical Block 對應到其 Physical Block。當 Operating System 傳下來 Read Request 或 Write Request，會將其帶有的 LBA(Logical Block Address)透過 Logical-to-Physical Table 的查詢，找出 PBA(Physical Block Address)，以便做寫入或讀取的動作。Logical-to-Physical Table 中會讓每一個 LBA 都有對應的實體區塊，稱之爲 Data Block 或 Primary Block。系統中有一個 Write quest 到達，透過 Logical-to-Physical Table(L2PTable)查到對應的 PBA，該 PBA 即爲該 LBA 的 Data Block，在找到 Data Block 後則會寫入資料，但若 Data Block 已存在同樣的資料，表示該 Write Request 爲 Data 的 Update，此時即會向系統要一個空的實體區塊，系統則會在現有的可用區塊（稱爲 Spare Blocks）中配給一個空的實體區塊，並將更新的資料寫在該實體區塊內，這種實體區塊稱爲 Log Block 或 Replacement Block，並會利用 Log Block Table(LBTable)記錄每一個 LBA 的 Log Block 的 PBA。當一個 Log Block 被寫完之後則再向系統要求一個 Log Block，並會串接在原本的 Log Block 後面，若該 LBA 相當的 Hot，則該 LBA 會串上相當多的 Log Block，這一整個串起來的 Log Block 稱爲 Block chain。每一個 LBA 皆有其 Block Chain，其架構可參考圖 5(b)。



(a)

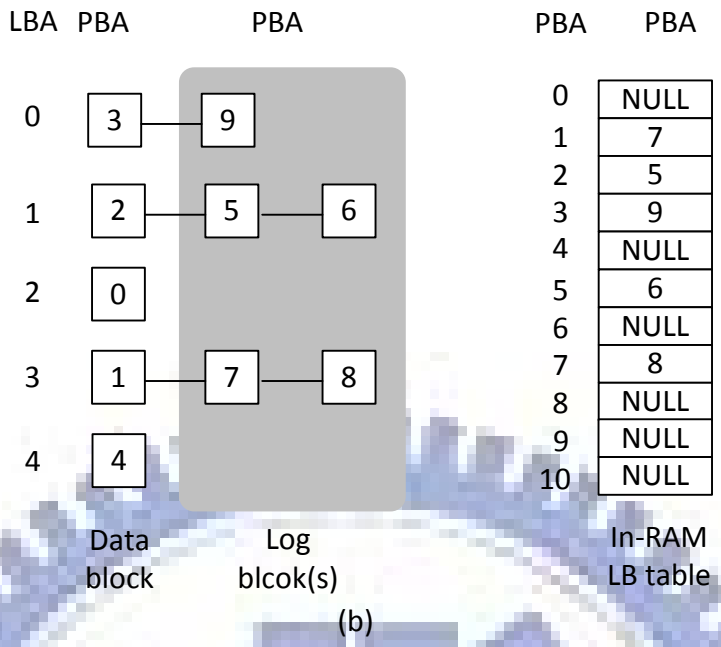


圖 5：NFTL mapping

(a)爲 NFTL 的 L2Ptable 的 mapping，即由一個 L2Ptable 記錄 Logical Block 對應到哪一個 Physical Block。(b)爲 Block chain 的記錄方式，即利用一個 LBtable，記錄每一個 Block Chain 的詳細資訊，如 Logical Block 1 的 Block Chain，即是先用 L2Ptable 找到對應的 Physical Block 2，再透過 LBtable，找出 PBA 2 所指的 PBA 爲何，即爲其 Log Block，如此找到所指的 PBA 爲 NULL 爲此，即可知其 Block Chain 爲 2→5→6。

在 NFTL 中，當可用的空間不夠時，則會執行 Garbage Collection 以回收可用空間，NFTL 的 Garbage Collection 即是挑選一個 Block Chain，將該 Chain 摺疊合併起來，將整個 Block Chain 中的有效資料(valid data)複製到一個新的區塊，而這新的區塊即爲該 Block Chain 的新 Data Block，而整個 Chain 中所有區塊會回收至系統中成爲新的可用區塊，其運作可參考圖 6。

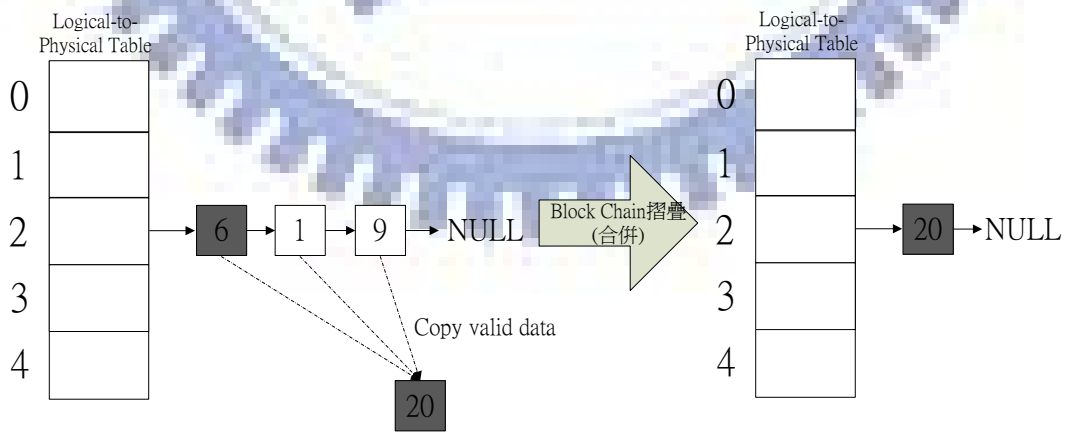


圖 6：Block chain folding

當 Garbage Collection 運行時，會挑選 Block Chain 來回收，當挑選好 Block Chain 之後，會依序複製該 Chain 中的 Valid Data 到一個新的區塊中，如圖中，挑選到的是 LBA 2 的 Block Chain，然後先從 Block 6 中每個 Page 檢

查是否爲 valid data，若是就複製到新區塊區塊 20 中，依序複製完區塊 6→1→9 到新區塊 20 後，即將區塊 6、1、9 回收到系統中成爲新的可用區塊。

3.1.2 Garbage Collection Policy

Garbage Collection 的設計與 wear leveling 習習相關，通常 Garbage Collection 做的好，其每個實體區塊磨損的程度則會相當不平均，反之則會相當平均。如何設計一個 wear leveling algorithm，會與 Garbage Collection 的設計有相當關係。因此在此小節則說明及探討我們在 NFTL 中 Garbage Collection Policy 的設計。

Garbage Collection 主要做來回收可用空間，當系統空間不夠時，即會執行 Garbage Collection，以便把實體區塊回收回來供它人使用。但當 Garbage Collection 執行相當的頻繁時，會造成系統負荷加重，使得 Overhead 上升。如何挑選適當的 Block Chain 以回收最多的可用空間並可降低 Garbage Collection 執行次數即是一個問題。

爲了避免 Garbage Collection 執行頻率過高，最好的方法則是每次能回收最多的空間回來，以避免 Garbage Collection 運作頻繁，因此在 Garbage Collection 的設計上希望能一次回收最多的空間。NFTL 原本的設計中是以 Greedy 的方式，即在所有的 Block Chain 中挑選最長的 Block Chain 來回收，因爲回收最長的 Block Chain 可一次回收最多的實體區塊，若一次可回收的相當多的實體區塊，Garbage Collection 的運作即不會頻繁，整體耗費成本最低，但這種方法在僅於當所有實體區塊空間都充份使用時才是最好的（如圖 7(a)），因爲 Block Chain 的長度可盡可能的拉長，則在回收時可一次回收最多的空間。但是我們發現，在實際運作上發現，所有的實體區塊空間並未都充份使用，這是因爲系統中有一些 cold data 的存在，使得很多實體區塊的使用率並不高，且 Cold Data 又遠多於 Hot Data，使得這些 Cold Data 佔據了系統中不少的實體區塊，當 Garbage Collection Policy 採用 Greedy 時，這些被 Cold Data 所佔據的實體區塊並不會被 Garbage Collection 所回收，因而長時間佔用實體區塊，使得系統中可用的實體區塊大幅下降，而 Hot Data 的 Block Chain 一直無法拉至最長就迅速的被 Garbage Collection 所回收，也會造成 Hot Data 的 Block Chain 最後一個 Log Block 常常才使用了一點空間，就被回收，而浪費了大部份 Log Block 的空間。大部份的 Hot Data 都在競爭僅剩的可用實體區塊，使得系統中可用空間會常常不夠，而一直執行 Garbage Collection 回收空間，整體的 Overhead 會變得相當的高，如圖 7(b)。例如：當系統中的 Spare Block=6，Hot LBA = 2，則此兩個 Hot LBA 平均長度可以長到 3，但是當其他 Cold LBA 佔去幾個 Spare Blocks 時，會使得這兩個 Hot LBA 平均長度無法再長到 3，因此會造成 Hot LBA 長度還未到達平均長度時即引起 Garbage Collection 而回收，原本可在 Hot LBA 達到長度 3 時才執行 Garbage Collection 回收區塊，但現在卻在 Hot LBA 達到長度小於 3 時就要執 Garbage Collection 回收區塊，因此造成 Garbage Collection 相當頻繁，系統的 Overhead 因此上升。

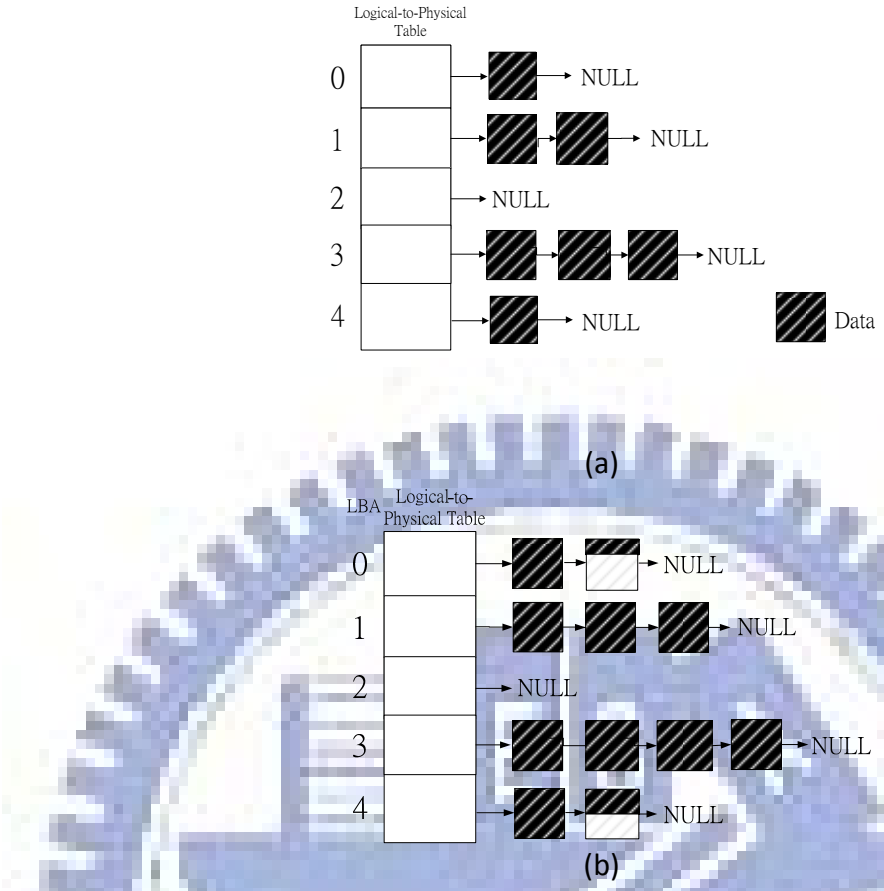


圖 7：NFTL garbage collection 喜好

(a)中大部份的實體區塊都是充份使用的，直接回收最長的 Block Chain，如：LBA 5 的 Block Chain，則一次可回收三個實體區塊回來，如此可用實體區塊變多，也使得 Garbage Collection 次數變少，系統的 Overhead 亦降低。(b)中存在著一些 Cold Data，佔著一些實體區塊，使得這些實體區塊無法給 Hot Data 使用，原本有八個實體區塊可供使用，被 Cold Data 佔走三個後，Hot Data 就只能競爭剩下的五個，因而 Garbage Collection 次數相當頻繁。

由於 Greedy 無法將一些被 Cold Data 所佔據的實體區塊做有效的回收，使得系統可用區塊長期被佔用而大幅減少，造成 Garbage Collection 的運作頻繁，系統 Overhead 因而大幅上升。因此爲了回收真正有效的空間回來，應該是要去回收被 Cold Data 所佔據的 Block Chain 中最長的，避免實體區塊被 Cold Data 長時間佔用不還，而不一定是所有 Block Chain 中最長的。因此在 Garbage Collection Policy 上我們則不採用 Greedy，而是另一種可有效回收區塊的方法，稱爲 Bitmap Garbage Collection。

爲了回收最多最有效的空間回到系統，我們則利用一個 Bitmap，如圖 8，該 Bitmap 每一個 Bit 對應到一個 LBA，記錄該 LBA 是否被 Update 過，若有則對應的 Bit 設爲 1，否則爲 0。當執行 Garbage Collection 時則從 Logical-to-Physical Table 從頭往下掃描 k 個 LBA，當遇到對應的 Bit 爲 1 則改爲 0 並跳過，表示給予該 Block Chain 一個機會不回收它，在這 k 次中找出對應的 Bit 爲 0 且 Block Chain 最長的出來回收，如此便可回收最多且有效的空間回來。圖六即示意 Bitmap Garbage Collection 如何挑選合適的 Block Chain。

Hot Data 會常常做 Update 的動作，造成 Block Chain 愈串愈長，但在我們的 Garbage Collection Policy 中並不會回收到 Hot Data 的 Block Chain，如此會造成 Hot

Data 的 Block Chain 愈串愈長，而使得可用實體區塊相對減少，爲了避免此種情形，我們會在系統中設定 Block Chain 最大可長長的長度爲何，當超過系統預設的長度即自動回收該 Block Chain。

因此在我們的設計中主要在兩種情況時會執行 Garbage Collection，(1)當某個 Block Chain 超過系統設定的長度時即自動回收該 Block Chain，以避免 Block Chain 因爲 Hot Data 的關係愈串愈長，(2)即是當空間不夠時，則透過 Bitmap Garbage Collection Policy 挑選一個適當的 Block Chain 來回收空間。兩種 Garbage Collection Policy 的實驗數據可參考 Section 4.2.1 的實驗結果，發現利用 Greedy Garbage Collection 會因爲可用實體區塊減少，因而 Total Erase Cycle 上升，約爲 Bitmap Garbage Collection 的四倍。

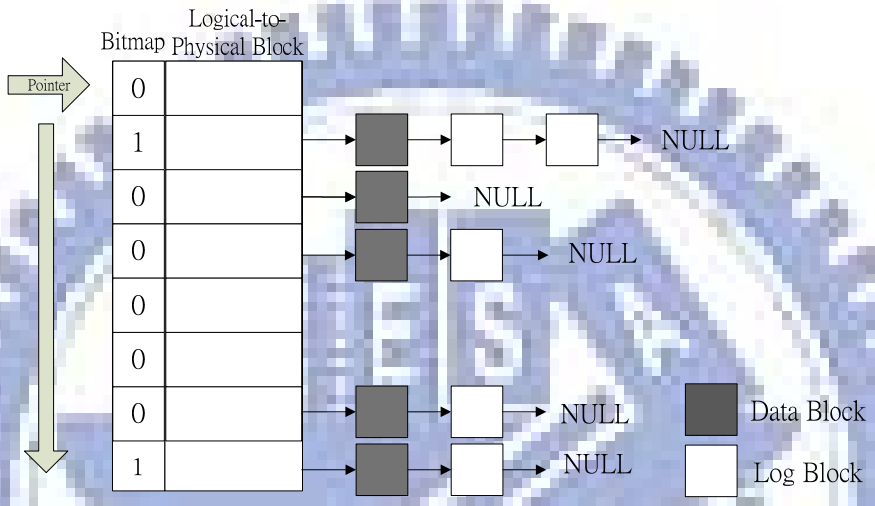


圖 8：Bitmap garbage collection

利用 Bitmap 來判斷是否爲低利用率的 Data（Cold Data），以便 Garbage Collection 挑選低利用率的 Block Chain 來回收。

3.2 A Lazy wear leveling algorithm: Concept

Static wear leveling[8]是一個常用的 wear leveling algorithm，此種方法主要的對像爲 Cold Data，它的運作方式是將 Cold Data 從某些區塊中移走，避免 Cold Data 長期放在某些實體區塊上，但我們根據 Realistic workload 的分析，得知 Cold Data 佔所有 Data 的比例相當高，因此若採用 static wear leveling 會移走相當多的 Cold Data，但這些 Cold Data 又未必會移到適當的區塊中（例如 Old Block），因此會有許多額外不必要的資料搬動，例如：搬移 Cold Data 到其它 Young Block 中。因此我們反向操作，即是將重點擺在 Old Block 身上，因爲 Hot Data 遠少於 Cold Data，因此 Old Block 數量相對比 Young Block 少得多。依此特性，追蹤 Old Block 不需要耗用太多的資料結構，所以我們將重點擺在 Old Block。我們的 lazy wear leveling algorithm 的設計就是以 Old Block 爲主要對像。Static wear leveling 是避免 Cold Data 存在某些區塊中太久，而我們的 Lazy wear leveling 則是避免 Hot Data 存在 Old Block 中太久，造成 Old Block 持續磨損，因此要將 Hot Data 移到其他實體區塊上，並將 Cold Data 移到 Old Block 中。

Lazy wear leveling algorithm 的主要對像爲 Old Block，因此我們需要一些資料結構將 Old Block 記錄下來。根據已提出的方法中，大部份的 wear leveling algorithm 大部份都是將每個實體區塊的 Erase cycle 記錄在 RAM 中，然後根據 Erase cycle 的

高低來找出 Old Block，但是此種方法會耗用相當多的 RAM 空間，一般實作上不太適用。Lazy wear leveling algorithm 則採用一個固定大小的 Priority Queue 來記錄 Old Block，如圖 9 中，我們在 Priority Queue 中利用[12]中所提的 LRU(Least Recently Used) Replacement 來記錄最近常常被 Erase 的實體區塊，因爲該實體區塊最近常常被 Erase，因此實體區塊有相當大的機會會是 Old Block。因爲 Old Block 相當的少，因此 Priority Queue 的容量不需要太大即可記錄大部份的 Old Block。

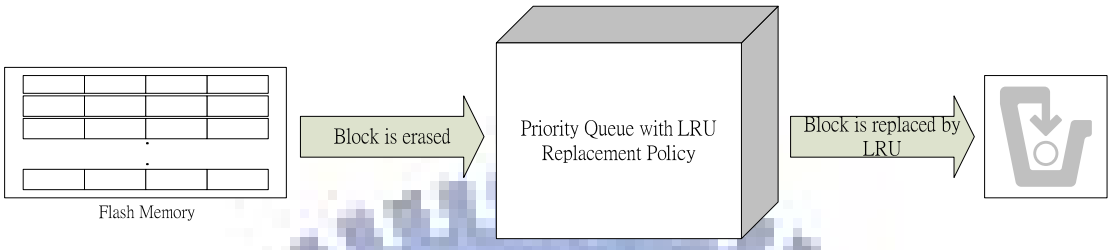


圖 9：Priority Queue with LRU Replacement Policy on PBA

爲了避免 Hot Data 在 Old Block 內，使得 Old Block 磨損的愈來愈嚴重，因此我們需要將 Hot Data 從 Old Block 中移走，並將 Cold Data 放入 Old Block 去冷卻它。由於我們的 Lazy wear leveling algorithm 是不記每個實體區塊的 Erase cycle，因此我們也不根據此來判斷實體區塊內的 Data 是否爲 Cold Data。在判斷實體區塊中的 Data 是否爲 Cold Data 的方法上，我們使用了一個 Bitmap 來判斷，如圖 10，每一個實體區塊會一一對應到 Bitmap 中每一個 Bit，當該實體區塊所對應的 Bit 爲 0 則表示該實體區塊未曾被 Erase 過，表示該實體區塊內的 Data 必爲 Cold Data，反之則表示被 Erase 過，如圖 10。在 Bitmap 中找尋 Cold Data 時會 Round-Robin 的方式開始搜尋 Bit 爲 0 的 Physical Block，直到搜尋到最後一個 Bit 後，再將所有 Bit 重置成 0，下一次再重頭開始搜尋。

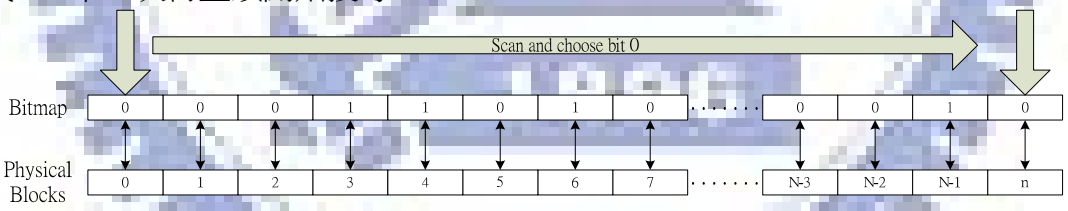


圖 10：使用 Bitmap 找尋 Young Block

3.3 A LBA-Based Lazy Wear Leveling: Design

Lazy wear leveling algorithm 即是利用 Priority Queue with LRU Replacement Policy 與 Bitmap 來達成，即透過 Priority Queue 找出 Old Block，並將 Hot Data 移走，並把由 Bitmap 所挑選的實體區塊中的 Data 移到 Old Block 中。

Lazy wear leveling 會在 Garbage Collection 時觸發。當系統執行 Garbage Collection 時，在挑選 Block Chain 回收後，會先檢查 Block Chain 每一個要回收的實體區塊是否有記錄在 Priority Queue 中，若存在 Priority Queue 中表示該實體區塊可能爲 Old Block，再做下列公式的檢查：

$$\text{Actual Erase Cycle} \geq \text{Average Erase Cycle+TH},\begin{cases}\text{TH: Threshold}\\ \text{Average Erase Cycle: } \text{Total Erase Cycle} \big/ \text{Total Block Number}\\ \text{Actual Erase Cycle: Actual Erase Cycle of the block}\end{cases} \quad (1)$$

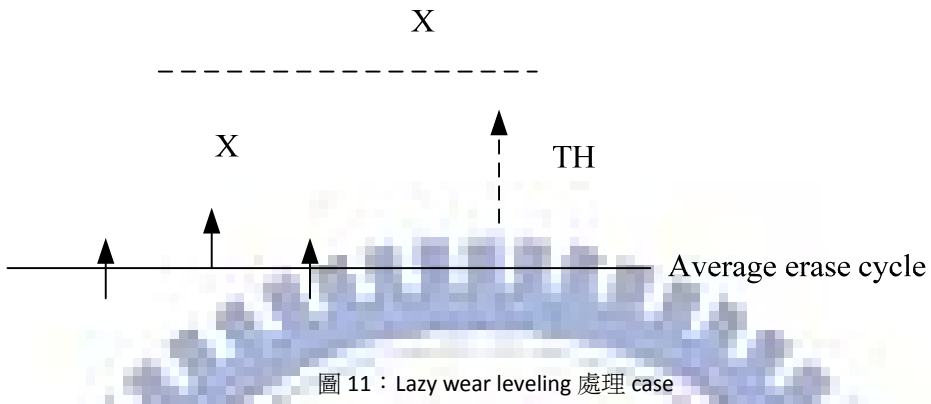


圖 11：Lazy wear leveling 處理 case

公式即是檢查實體區塊是否高於 Average erase cycle+TH，即 Old Block，如圖 11 中高於 TH 的實體區塊，我們主要就是要冷卻這些過於磨損的實體區塊。當找出 Old Block 後先將 Hot Data 從 Old Block 中移到其他實體區塊中，再透過 Bitmap 找尋存有 Cold Data 的實體區塊，找到後將實體區塊中的 Cold Data 移到 Old Block 中。

在做了實驗之後，發現其效果並未良好，後來發現其不成功的主要原因爲當 wear leveling 執行後，會使得原本最近被 Erase 的實體區塊已不再被 Erase，而不被 Erase 的實體區塊開始被 Erase，造成 Priority Queue 內的記錄對像會轉換，因開始被 Erase 的實體區塊卻未必是 Old Block，並且在 wear levelingn 運作後，其他的實體區塊也會開始的老化，因此 Old Block 會愈變愈多，因此要記錄 Old Block 則會需要更大的 Priority Queue 才能完全記錄下來，如此會耗用較多的 RAM 空間，無法達到 Low cost。針對此情形。因此我們提出了以 LBA 以代 PBA 的方式，即不再記錄最近常常被 Erase 的實體區塊，而是記錄最近常常寫到的 Logical Block 爲哪些。利用 LBA 的方式則不會有 PBA 的缺點，由於 Old Block 是因爲 Hot Data 所造成的 Old Block 必會存在 Hot LBA 的 Block Chain 中。而且在 wear leveling 運作後，LBA 是不會受到其影響的，Hot LBA 依舊是 Hot LBA，因此我們的 Priority Queue 即可利用少少的空間即可達成。實驗數據可參考 Section 4.4 的實驗結果。

根據上一小節，我們將方法略更改爲利用 Priority Queue with Replacement Policy 去記錄最近常常被 Write 的 LBA，如圖 12，近期常常被 Write 的 LBA 必定是 Hot Data。而 wear leveling 亦是在 Garbage Collection 時觸發，當挑選好要回收的 LBA 的 Block Chain 時，會先檢查該 LBA 是否在 Priority Queue 中，再根據欲回收的區塊檢查公式是否成立，若成立即表示其爲 Old Block，當兩者同時成立則執行 wear leveling，將 Old Block 中的 Data 移到其他區塊上，再透過 Bitmap 找尋 Cold Data，將 Cold Data 移到 Old Block 中，wear leveling 即完成。實驗後其效果相當不錯，實驗數據可參考第四章的實驗結果。

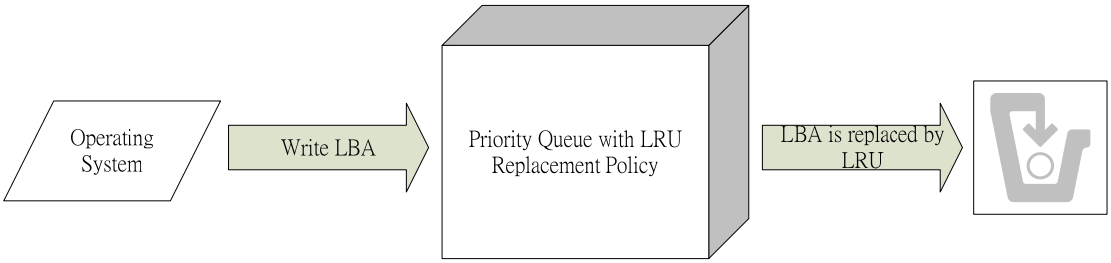


圖 12：Priority Queue with LRU Replacement Policy on LBA

3.4 Implementation Issues

我們的 wear leveling 是一種 low-cost 的 wear leveling algorithm，希望能盡可能的減少 RAM 的使用空間，因此我們希望在現有的架構下，利用其他已知的資訊來達到相同的效果，並捨棄某些重覆的資訊，如此可減少 RAM 的使用空間。在此章節討論幾種方法取代用來尋找 Cold Data 的 Bitmap，可省下一個 Bitmap 的空間，亦可尋找到 Cold Data，完成 wear leveling。

在前面的章節中的 wear leveling 中所提到的 bitmap，主要用途是用來尋找 Cold Data，Cold Data 的特性即爲不常 Update，利用此特性我們覺得可以使用其他現有的資訊來尋找 Cold Data，而不使用 Bitmap。而在現有的資訊中，我們發現有三種方式可以搜尋到 Cold Data：(1)使用 Priority Queue 來判斷是否爲 Cold，(2)依序檢查每一個 LBA 的 Block Chain，若長度爲零（即只有 Data Block），則表示爲 Cold Data，(3)利用 Bitmap for Garbage Collection，此 Bitmap 爲當該 LBA 有 write 時，則 Bit 設爲 1，未 write 則 Bit 爲 0，因此當 Bit 爲 0 時，其 Data 必爲 Cold Data。這幾種尋找 Cold Data 的方法雖然比不上使用 Bitmap 精確，但不失爲減少 RAM 使用空間的一種方法。

第一種方式是以 Priority Queue 來判斷 Data 是否爲 Cold。我們的 Priority Queue 主要是用來儲存 Hot LBA，因此不在 Priority Queue 的通常可以當作是 Cold LBA，亦即 Cold Data。我們使用 Round-Robin 的方式從頭一一的檢查每一個實體區塊，當檢查一個實體區塊時找到其所配置的 LBA，再查詢 LBA 是否存在 Priority Queue 中，若不存在，則表示爲 Cold LBA，表示實體區塊內所儲存的爲 Cold Data。當 wear leveling 執行時，利用此方式挑選 Cold Data，再將 Cold Data 移到由 Priority Queue 所挑選出來的 Old Block。此種方式雖然節省了一個 Bitmap，但是卻額外多出了 Priority Queue 的搜尋時間，可能會拖慢一些系統速度。在當 Priority Queue 不夠大時，會無法記錄到所有的 Hot LBA，因此也會造成在判斷是否爲 Cold Data 時發生誤判的情況。

第二種方式是利用 Block Chain 的長度來判斷是否爲 Cold Data 的依據。當 Block Chain 長度爲 0 時(該 Block Chain 無 Log Block)，表示其對應的 LBA 幾乎未曾有 Update，即該 LBA 爲 Cold LBA，其 Data 爲 Cold Data。當 wear leveling 執行後，利用 Priority Queue 挑選出 Old Block，再以 Round-Robin 的方式從頭開始檢查每一個 LBA 的 Block Chain 長度，當檢查到 Block Chain 長度爲 0 的 LBA，則視爲 Cold Data，將該 LBA 中的 Data 移到 Old Block 中，而 Old Block 則變成該 LBA 的新 Data Block。此種方式雖然可尋找到 Cold Data，但是仍會有許多 Cold Data 會被其視爲非 Cold Data，因爲許多的 LBA 可能會有些許數量的 Update，但可能一、兩次 Update 即不再 Update 了，但是 Block Chain 的長度卻不爲 0。這種 Data 亦爲 Cold Data，但透

過此方式在判斷是否爲 Cold Data 的過程中，會將其誤判爲非 Cold Data。

第三種方式則是利用前面章節所提到的 Garbage Collection 所使用的 Bitmap，當 Bitmap 中的 bit 爲 0，則表示其對應的 LBA 未曾被 Update，反之則曾經被 Update。Wear-leveling 運作時要尋找 Cold Data 則以 Round-Robin 的方式從 LBA 0 開始，檢查 LBA 對應的 Bit 是否爲 0，若爲 0 則表示爲 Cold LBA，亦爲 Cold Data，而此 LBA 的 Block Chain 未必爲 0，因此我們先將該 LBA 的 Block Chain 中有效的 Data 移到 Old Block 中，再把 Block Chain 回收起，Old Block 則成爲該 LBA 的新的 Data Block。這種方式如前一種方式的缺點一樣，即是會將 Cold Data 誤判會非 Cold Data 的情形發生。

這三種方式皆可取代 Bitmap 而達到搜尋到 Cold Data 的目的，省下 RAM 空間的耗用，可參考 Section 4.7 的數據，這幾種方式取代 Bitmap 後的效果與使用 Bitmap 並不會差太多。

3.5 How to store erase cycle of all blocks

我們的 Wear leveling algorithm 需要計算 Average Erase Cycle 以及得知實體區塊的 Actual Erase Cycle，因此在我們的 wear leveling 使用的預設前提之下，是會知道所有實體區塊的 erase cycle，即每一個實體區塊的 Actual Erase Cycle 皆可查詢到，並可計算出 Average Erase Cycle。我們需要把所有實體區塊的 Erase Cycle 儲存在 Flash Memory 上，而如何將實體區塊的 Erase Cycle 儲存起來，可利用現有的方法達到儲存的目的。

(1)利用一個實體區塊儲存所有實體區塊的 Erase Cycle，會先在 RAM 中記錄，然後定時更新至實體區塊中，(2)在 M-System 的 NFTL 中，使用的方式即是每一個實體區塊的 Erase Cycle 存放在該實體區塊中第一個 Page 的 Spare Area 中，需要支援 Partial Page Write 的 NAND Flash Memory 才行，但是現在大部份的 NAND Flash Memory 皆不支援 Partial Page Write，因此可行性低，(3)將每一個實體區塊的 Erase Cycle 固定存放在該區塊的第一個 Page 中，此方式會浪費掉每一個實體區塊的一個 Page，(4)同方法(2)，但不需要 Partial Page Write 的支援，在回收實體區塊時不將實體區塊抹除，而直接回收到系統中，待系統分配出可用實體區塊時再從該區塊第一個 Page 的 Spare Area 讀出 Erase Cycle，再抹除實體區塊，將 Data 寫入該實體區塊時一并將最新的 Erase cycle 寫入第一個 Page 的 Spare Area，如此即可記錄正確的 Erase cycle。

四、Experimental Results

4.1 Environment set up

在我們的實驗中，所使用的 NAND Flash Memory 規格如表 3，所有的實驗皆在此 NAND Flash Memory 的規格下完成。在實驗測試的過程中，我們會在 Windows XP 上蒐集爲期一個月中 Operating System 對 Hard Disk 所下 Write/Read 指令，將這些指令對各個 wear leveling algorithm 進行測試。

我們的 wear leveling algorithm 主要比較對像爲 Random wear leveling[13]、Static wear leveling[8]以及 Group-Based wear leveling[7]。Random wear leveling 即每 Threshold 次的 Garbage Collection 即 Random 挑選 k 個實體區塊來 Erase。，這幾種 wear leveling algorithm 皆與我們的 wear leveling algorithm 建置在相同的 NFTL 架構下，以便在同等條件下比較出各 wear leveling 的效果。在實驗中，我們會測試每一

個 wear leveling algorithm 的 standard deviation 和 mean，其中 standard deviation 是用來測試 wear leveling 的效果如何，磨損程度是否平均，standard deviation 愈小則效果愈好，表示抹損愈平均，mean 則表示 wear leveling 所花費之成本，mean 愈高則成本愈高。

我們會在不同數量的 Spare Block 情況下測試各 wear leveling algorithm，觀察當系統可用區塊減少，對 wear leveling algorithm 的影響。而在本篇提出的 wear leveling 會測試在不同的 Spare Block 數、Priority Queue Size 以及 Threshold 不同之下的數據比較，以得知在何種情況下、哪種條件的 wear leveling 效果較好，或者 Overhead 較低。

表 3：NAND Flash Memory 規格

Page Size	2K Bytes
Block Size	128K Bytes
Pages Per Block	64
Total Physical Blocks	172,032
Total Size	20G Bytes

4.2 Realistic Workload Analysis & NFTL Garbage Collection Policy Analysis

4.2.1 Realistic Workload Analysis

我們在 Windows XP 上擷取 Operating System 對 Hard Disk 所下的 Write/Read 指令，爲期一個月，然後針對擷取下來的資訊做分析。分析結果如表 4，對同一個 LBA 寫入次數愈多的數量愈少，反之則愈多，表示大部份的 Data 都僅寫入數次，即 Cold Data 較多，只有少數的 Data 寫入次數較高，即 Hot Data 較少，可知 Hot Data 的數量遠少於 Cold Data 的數量。因此我們的 lazy wear leveling algorithm 以 Hot Data 爲主要對像，因爲它的數量較少，可用較少的 RAM Space 即可記錄大部份的 Hot Data，以表四來觀察，若我們認爲對同一 LBA 寫入次數超過 100 即爲 Hot Data，則我們只用一個大小爲 2048 的 Priority Queue 即可記錄了所有的 Hot Data，我們的 Priority Queue Size 也是以此分析決定的。

表 4：Disk trace 分析

Total Writes(次數)：1,772,339	
範圍(次數)	LBA 個數
0~100	162,208
101~1000	1,505
1001~10000	119
10001~30000	6
>=30001	2
	Total：163,840 個 LBAs

4.2.2 NFTL Garbage Collection Policy Analysis

此節要討論 NFTL Garbage Collection Policy，在 Section 3.1.2 所討論的 Garbage Collection Policy 中的兩種方法：一個爲 Greedy Garbage Collection，另一個則爲 Bitmap Garbage Collection。而我們在這裡測試了下面幾個 Garbage Collection Policy：

P1：Bitmap Garbage Collection

P2：Greedy Garbage Collection

P3：Recycle overrun block chain

表 5 爲使用上列三種方法以及混合使用所產生的 Total Erase cycles 及 Average Erase cycles。所表可知當使用 P2 時，會因爲區塊被 Cold Data 佔據，使得系統可用實體區塊大幅減少，造成 Garbage Collection 相當頻繁，因而使得 Total Erase cycles 相當高，即使加上 P3 後效果亦無改善，大部份長的 Block Chain 都會被 Garbage Collection 所回收，比較不會因 overrun 而回收。而 P1 的主要特點爲回收被 Cold Data 佔據的區塊，效果比 P2 改善甚多，與 P1+P3 的差別在於 P1 中的 Block Chain 長度不限制，因此 Hot Data 的 Block Chain 會長得相當長，亦耗用不少的區塊，在加上 P3 的 Policy 後亦可改善此缺點，效果上有所改善，可知在 Garbage Collection Policy 的設計上，應該回收的對象應該爲被 Cold Data 佔據的 Block Chain，而非最長的 Block Chain。

表 5：Garbage Collection Policy 比較

	Total Erase cycles	Average Erase cycles
P1	485,100	2.82
P2	1,596,387	9.28
P1+P3	416,418	2.42
P2+P3	1,596,323	9.28

此表爲 P1、P2、P3、P1+P3 以及 P2+P3 的實際 Total Erase cycles，可觀察出 P2 造成的 overhead 相當得高，因爲它無法真正回有有效的區塊，因此造成 Garbage Collection 相當頻繁且 overhead 高。在使用 P1 後 Total erase cycles 降低 P2 的約四分之一。

4.3 PBA-Based Lazy Wear-Leveling v.s. LBA-Based Lazy Wear-Leveling

在此節中針對我們的 Lazy wear leveling 中 Priority Queue 的對像不同做比較，一個爲 PBA-Based 的，利用 Priority Queue 記錄最近常常被 Erase 的區塊 PBA。另一個則爲 LBA-Based，利用 Priority Queue 記錄最近常常 Write 的 LBA。在圖 13 中，可看出每一個 Threshold 的 Standard Deviation 在 Priority Queue on LBA 皆比 Priority Queue on PBA 低，並且在 Mean 相同時，Priority Queue on LBA 的 Standard Deviation 亦較低，wear leveling 效果較好。如 Priority Queue on PBA 在 TH4 時與 Priority Queue on LBA 在 TH12 時的 Mean 皆約爲 24.3，但 Priority Queue on LBA 的 Standard Deviation 約爲 14，Priority Queue on PBA 的 Standard Deviation 約爲 17，Priority Queue on LBA 的效果優於 Priority Queue on PBA。

Priority Queue on PBA 效果較 Priority Queue on LBA 差，主要原因爲在 Section 3.3 中所提到的使用 PBA-Based 時會有一些問題，因此效果較 LBA-Based 的差。

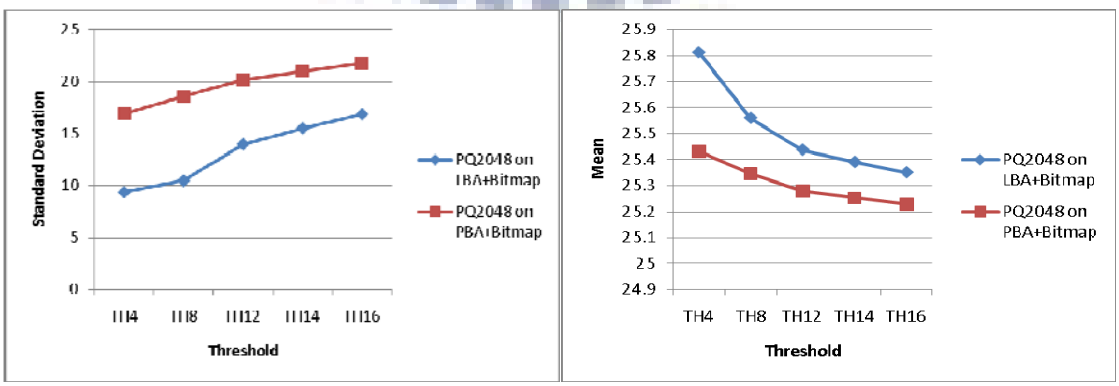


圖 13：Priority Queue on PBA v.s. Priority Queue on LBA

4.4 The effect of priority queue size, threshold

在此節我們說明不同的 Priority Queue Size 以及不同的 Threshold 所造成的影響。在實驗中的 Priority Queue Size 分別是 2048、1024 和 512 和 Threshold 分別爲 4、8、12、14、16 的情形下的 wear leveling 效果。其結果爲圖十，觀察出不管 Spare Block 個數爲多少，Priority Queue Size 愈大，其 Standard Deviation 愈低，效果也愈好。主要是因爲 Priority Queue Size 愈大，其擷取到的 Hot Data 愈精確，比較不會有將 Hot Data 誤判爲 Cold Data 的情形。Threshold 愈大則 Standard Deviation 也愈高，反之則愈低。Threshold 主要是用來當作一個門檻值，即當一個區塊的 Erase Cycle 高於平均 Erase Cycle 加上 Threshold 即認爲是 Old Block。Threshold 愈低，表示愈多區塊會被視爲 Old Block 而執行 wear leveling 將 Cold Data 放入 Old Block 中，使得大部份區塊的 Erase Cycle 的差距範圍大概在 0 至 Threshold 之內，效果較好，但也使得一直執行 wear leveling 將 Cold Data 放到 Old Block 中，雖然效果好，但是成本會相對的提高。

在圖 14 中，可看出 Threshold 愈低，mean 愈高，即成本較高。Threshold 愈低表示所執行的 wear leveling 次數愈多，因此效果也較好（即 Standard Deviation 愈低），但所花費的成本也較高（即 mean 愈高）。Wear leveling 次數愈多會使得每個區塊的磨損程度愈平均，但也因爲 wear leveling 次數過多，會花費額外搬移資料及磨損區塊的成本更多，因此 mean 亦會愈高。

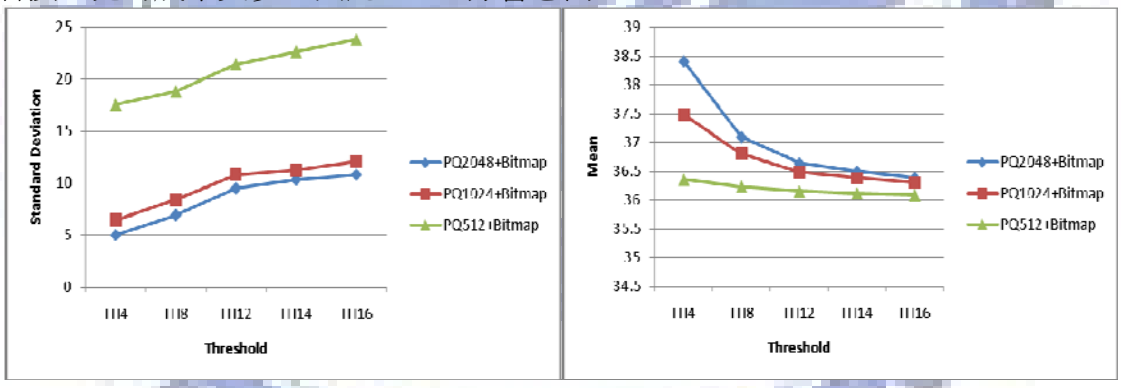


圖 14：LBA-Based wear leveling 在不同的 Priority Queue Size 與 Threshold 的比較。

4.5 Wear leveling algorithm comparison

此節要針對 LBA-Based lazy wear leveling、Static wear leveling[8]、Random wear leveling[13]以及 Group-Based wear leveling[7]的效果以及所花費的成本比較。觀察每一 wear leveling algorithm 在執行了 Windows XP Disk Trace 後的 Standard Deviation 和 Mean。最後觀察在長期使用的過程中，standard deviation 是否會收斂亦或發散，若爲收斂則 wear leveling 長期使用的效果都很穩定，若是發散則表示在長期使用該 wear leveling 之下，效果會隨時間愈久愈差。

在圖 15 中是在使用上述四個 wear leveling algorithm 執行 Windows XP Disk Trace 在不同的 Threshold 下所得出的數據。可看出大部份的 wear leveling algorithm 在 Threshold 愈大的情況下，standard deviation 愈高，mean 愈低。

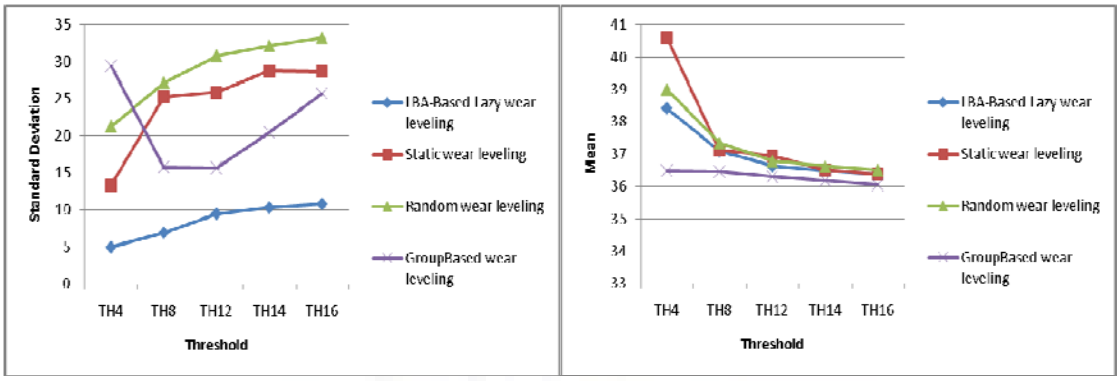


圖 15：各 wear leveling 的 standard deviation 與 mean 趨勢圖

接下來我們針對在相同的 mean 和相同的 standard deviation 情況下做 wear leveling 的效果比較。表 6 爲在相同的 Mean 下，對各 wear leveling algorithm 的 standard deviation 比較，即 wear leveling 效果的比較。可觀察出在相同的 Mean 之下，可看出不管 Spare Block 的數量多少，LBA-Based lazy wear leveling 的效果皆優於其他的 wear leveling。表 7 爲在相同的 Standard Deviation 下，對各 wear leveling algorithm 的 Mean 比較，即 wear leveling 所花費的額外成本比較。亦可觀察出 LBA-Based wear leveling 的成本較其他 wear leveling algorithm 低。因此不管在效果和額外花費的成本下，LBA-Based wear leveling 皆比其他 wear leveling algorithm 表現還優異。

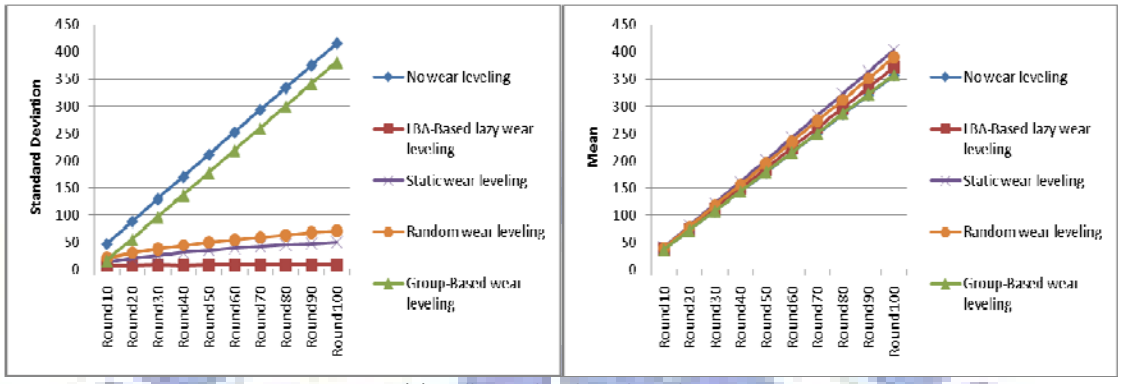
表 6：各 wear leveling 在相同的 Mean 下，standard deviation 的比較

Mean	Standard Deviation			
	Static wear leveling	LBA-Based lazy wear leveling	Random wear leveling	Group-Based wear leveling
≒25.65 (Spare Block=8192)	16.81789	9.888734	23.52941	15.2917
≒30.42 (Spare Block=4096)	22.90512	12.06228	28.86379	15.2478
≒36.61 (Spare Block=2048)	27.00441	10.6309	32.15556	29.46214

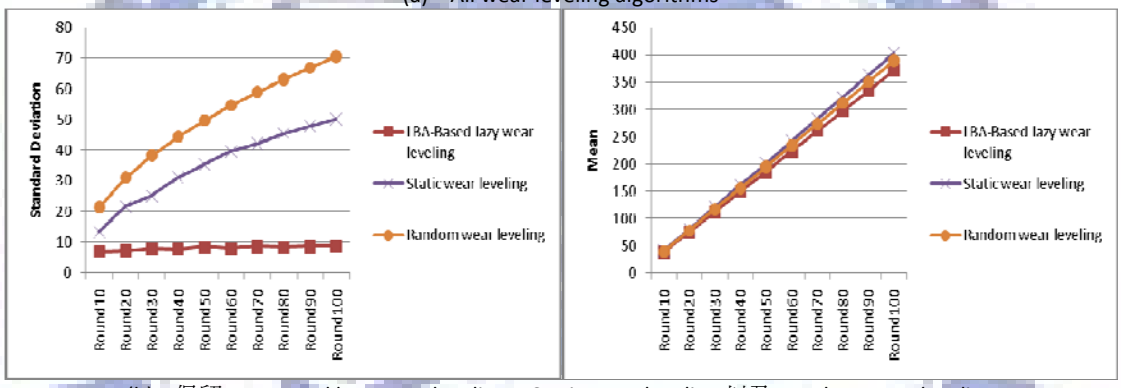
表 7：各 wear leveling 在相同的 Standard Deviation 下，mean 的比較。

Standard Deviation	Mean			
	Static wear leveling	LBA-Based lazy wear leveling	Random wear leveling	Group-Based wear leveling
≒16.83 (Spare Block=8192)	25.6562	25.50735	26.91419	25.69794
≒12.63 (Spare Block=4096)	33.39208	30.39008	32.41787	30.59791
≒10.84 (Spare Block=2048)	40.60057	36.3809	39.00024	36.46858

接下來我們將實驗的次數拉長，即長時間來觀察 wear leveling 是否收斂。圖 16 中爲不同 Spare Block 數量下的 standard deviation 及 mean 的趨勢發展，在不執行 wear leveling 的情況下，區塊的磨損程度會相當不平均，因此必不會收斂，但是其他的 wear leveling algorithm，包括 Static wear leveling、Random wear leveling 以及 Group-Based wear leveling 皆不會收斂，時間愈長效果愈差，而 LBA-Based lazy wear leveling 則可以一直保持在水平，是收斂的情況，即效果一直都不錯，不會因爲時間拉長就使得 wear leveling 效果變差。



(a) All wear leveling algorithms



(b) 保留 LBA-Based lazy wear leveling、Static wear leveling 以及 Random wear leveling

圖 16：各 wear leveling 的長期使用趨勢圖

Static wear leveling 不收斂主要原因在分析完 Oldest Block 在執行整個系統中的一些運作後（參考附錄），得知是因爲其處理方式主要是將 Cold data 從某些區塊中移走，在移走的同時，會將 Cold data 放到某些區塊上，而這些區塊可能會是 old block，但 cold data 可能仍帶有一些 update，也因爲如此這也會造成在 Garbage Collection 時，會回收到該 Old Block。在 static wear leveling 中會將 Cold data 放到 Old block 只有 2 種情形：1.wear leveling 、2.write request。但是 static wear leveling 的這兩種情形並沒有積極的把 cold data 放到 old block 中，因此 old block 有很大的機率會一直存放在 Hot data 或 Not-so hot data(即會有數次的 update)中，造成 Old Block 會持續一直磨損而停不下來，因此無法有效的降低 Old block 的磨損情形，造成 static wear leveling 無法收斂的情況。

圖 17 爲所有區塊最後的 Erase cycle 的分散圖，可看出 LBA-Based wear leveling 的分散最爲平均。

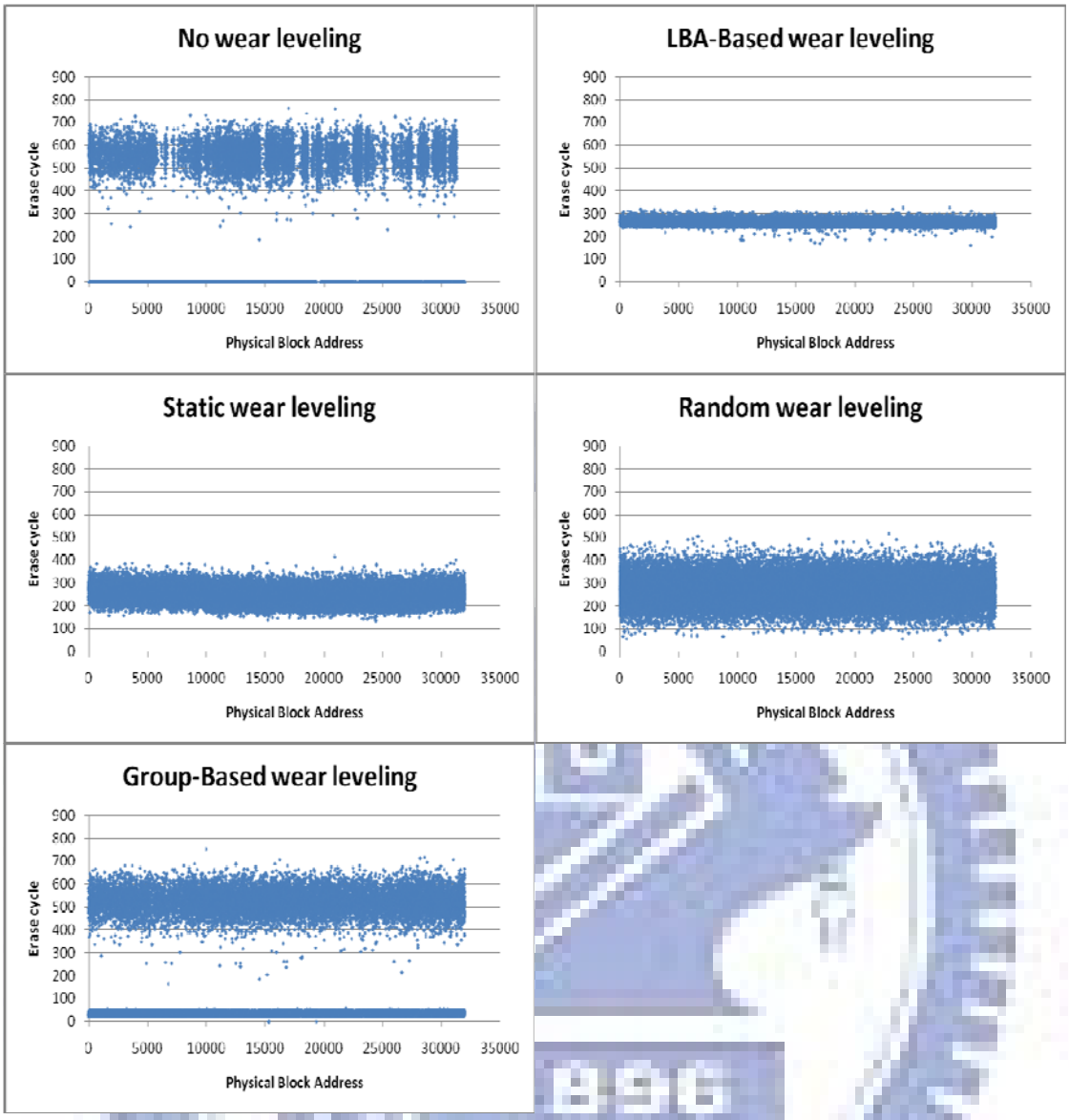


圖 17：各 Wear leveling final distribution

4.6 The effect of amount of spare blocks

此節說明在不同的 Spare Block 數量對 wear leveling 的影響。圖 18 爲 No wear leveling、LBA-Based wear leveling、Static wear leveling、Random wear leveling 與 Group-Based wear leveling 之下的趨勢圖，在觀察此趨勢圖發現，除了 LBA-Based wear leveling 外，當 Spare Block 數量愈少，Standard Deviation 愈高，表示區塊磨損愈不平均。但是在 LBA-Based wear leveling 的情況下，Spare Block 愈少 Standard Deviation 愈低，這是因爲當 Spare Block 數量愈少，LBA-Based wear leveling 會執行的愈頻繁，反而使得 Standard Deviation 愈低，但所花費的成本就比較高，而其他的 wear leveling algorithm 並不會更低，反而更高，主要是設計上的問題。

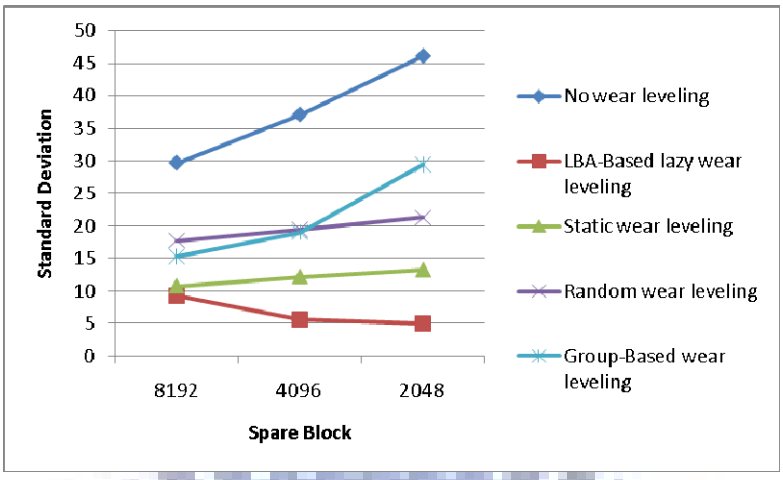


圖 18：不同數量 Spare Block 下，使用各種 wear leveling algorithm 的趨勢圖。

4.7 Lazy wear leveling implementation issues analysis

在設計 Lazy wear leveling 時，需要利用 Bitmap 找出 Cold Data，但我們希望能把 Bitmap 移掉，利用現有的資訊達到相同的效果，亦可再省下一些空間。在 Section 3.4 則討論到這方面的 Implementation Issues，有三種取代方式可找出 Cold Data：(1)利用 Priority Queue 來尋找 Cold Data，(2)Block Chain 長度爲 0 表示爲 Cold Data，(3)利用 Garbage Collection Bitmap 來判斷，若爲 0 表示爲 Cold Data。圖 19 則爲使用這上述那幾種方法所產生的 Standard Deviation 及 Mean，從圖中可觀察出使用這幾種方法對 wear leveling 影響不大，差距都不會太大。透過這幾種方式既可省去一個 Bitmap，效果亦不會比使用 Bitmap 差，若實作上需要極精簡 RAM 空間，則可使用這幾種替代方法。

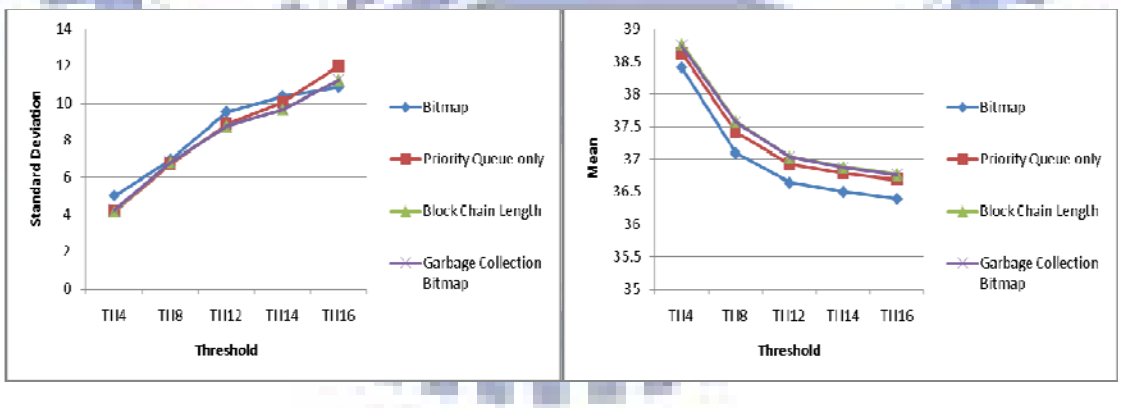


圖 19：各種取代 bitmap 方法的趨勢圖

4.8 Memory Requirement

在此章節中要探討使用不同的 Priority queue 和不同的 implementation 所耗用的 RAM space。假設我們一個 LBA 需要 2 Bytes 來表示，SSD 中有 16K 個區塊，則各種不同的方法所耗用的 RAM space 如表 8：

表 8:Memory Requirement

方法	The size of Priority queue	Contributed by	RAM space
Lazy wear-leveling with bitmap	2048	Priority queue, Bitmap	2048 * 2 Bytes + 16k bits = 6K Bytes
Lazy wear-leveling with GC bitmap	2048	Priority queue*	2048 * 2 Bytes = 4K Bytes
Lazy wear-leveling with block chain length	2048	Priority queue**	2048 * 2 Bytes = 4K Bytes
Lazy wear-leveling with priority queue only	2048	Priority queue	2048 * 2 Bytes = 4K Bytes
*GC Bitmap 不列入 Wear-leveling 所耗用的 RAM space **記錄 Block chain length 不列入 Wear-leveling 所耗用的 RAM Space			

五、Conclusion

SLC NAND Flash Memory 本身已存在 Endurance 的問題，而隨著科技的進步，容量更大，成本更低的 NAND Flash Memory: MLC 的出現，然而雖然 MLC 有上述之優點，但是 MLC 的 Endurance 問題卻又比 SLC 更加嚴重許多。而現在又將 NAND Flash Memory 進一步的應用，即 SSD 的出現以取代現有的 Hard Drive Disk，而將 NAND Flash Memory 作爲系統之用會因爲其 Access Pattern 而造成 Endurance 更加嚴重。Wear-leveling algorithm 則是用以解決此 Endurance 的問題，因此如何設計出能夠達到解決 Endurance 且成本又低則相當重要。本篇 paper 主要提出一個新的與以往不同的 wear-leveling algorithm，將重點擺在 Old Block 以及 Cold Data 身上，並且改以往方式以 PBA 的方式轉爲由 LBA 切入以找到 Old Block。如此我們僅需使用一個固定大小的 Priority Queue 並加上一個 Bitmap 找出 Cold Data 即可達到不錯的 wear-leveling 效果。在最後我們的實驗結果中與其他不同的 wear-leveling 做比較與觀察，可知我們的 wear-leveling 在效果上（standard deviation）、overhead（Mean）都比其他的 wear-leveling 表現的效果較好，所耗用的 RAM Space 亦不多。

參 考 文 獻

[1] S. Park. K-leveling: An efficient wear-leveling scheme for flash memory. In UKC ’05: Proceedings of The 2005 US-Korea Conference on Science, Technology, and Entrepreneurship, 2005.

[2] Richard John Defouw, Boulder, CO(US); Thai Nguyen, Thornton, CO(US). Storage Technology Corporation. LouisVille, CO(US). Method and system for improving usable life of memory devices using vector processing. United States Patent, (US 7139863 B1), NOV.21, 2006.

[3] Li-Pin Chang , " On Efficient Wear-Leveling for Large-Scale Flash-Memory Storage Systems," the 22nd ACM Symposium on Applied Computing (ACM SAC), 2007.

[4] Carlos J. Gonzalez, Los Gatos, CA(US), Kevin M. Conley, San Jose, CA(US). SanDisk Corporation, Milpitas, CA(US). Automated wear leveling in non-volatile storage systems. United States Patent, (US7120729 B2), Oct.10, 2006

[5] Samsung Electronics Company, “K9NBG08U5M 4Gb * 8 Bit NAND Flash Memory Data Sheet".

[6] Samsung Electronics Company, “K9GAG08U0M 2Gb * 8 Bit NAND Flash Memory Data Sheet (Preliminary)".

[7] Dawoon Jung, Yoon-Hee Chae, Heeseung Jo, Jin-Soo Kim, and Joonwon Lee, “A Group-Based Wear-Leveling Algorithm for Large-Capacity Flash Memory Storage Systems,” CASES, 2007, pp:160-164

[8] Y.-H. Chang, J.-W. Hsieh, and T.-W. Kuo. “Endurance enhancement of flash-memory storage systems: An efficient static wear leveling design,” the 44th ACM/IEEE Design Automation Conference (DAC), San Diego,California, USA, June 2007

[9] Samsung Electronics Company, “K9F1208X0C 64M * 8 Bits NAND Flash Memory Data Sheet”.

[10] M-Systems, Flash-memory Translation Layer for NAND Flash (NFTL)

[11] Li-Pin Chang , "Hybrid Solid-State Disks: Combining Heterogeneous NAND Flash in Large SSDs," the 13th Asia and South Pacific Design Automation Conference (ASP-DAC), 2008. (best-paper nominee, 10 out of 122 accepted papers/350 submissions

[12] Silberschatz, Galvin, Gagne, “Operating System Concepts,”

[13] D. Woodhouse, “JFFS: The Journalling Flash File System,” Proceedings of Ottawa Linux Symposium, 2001.

[14] C. Manning and Wookey, “YAFFS Specification,” Aleph One Limited, 2001.

[15] Li-Pin Chang and Tei-Wei Kuo, "Efficient Management for Large-Scale Flash-Memory Storage Systems with Resource Conservation," ACM Transactions on Storage, Volume 1, Issue 4, 2005.

[16] H. J. Kim and S. G. Lee, “An Effective Flash Memory Manager for Reliable Flash Memory Space Management," IEICE Transactions on Information and System, 2002.

[17] M. L. Chiang, Paul C. H. Lee, and R. C. Chang, “Using Data Clustering To Improve Cleaning Performance For Flash Memory," Software - Practice and Experience, 1999.

[18] "SmartMedia™ Specification", SSFDC Forum, 1999.

[19] “Sandisk Flash Memory Cards Wear Leveling", http://www.sandisk.com/Assets/File/OEM/WhitePapersAndBrochures/RS-MMC/WPaperWearLevelv1.0.pdf, 2003.

[20] M-Systems, “TrueFFS® Wear-Leveling Mechanism”

附　錄　一

Static wear-leveling oldest block trace

Block 124352 is allocated to LBA 124352 as DataBlock
Block 124352 is blockSwitched(shoulded be erased) in LBA 124352
Block 124352 is allocated to LBA 124355 as logBlock
Block 124352 is blockSwitched(shoulded be erased) in LBA 124355
Block 124352 is allocated to LBA 124360 as logBlock
Block 124352 is blockSwitched(shoulded be erased) in LBA 124360
Block 124352 is allocated to LBA 124363 as logBlock
Block 124352 is blockSwitched(shoulded be erased) in LBA 124363
Block 124352 is allocated to LBA 124366 as logBlock
Block 124352 is blockSwitched(shoulded be erased) in LBA 124366
Block 124352 is allocated to LBA 124373 as logBlock
Block 124352 is folded in LBA 124373 block chain,so in SB chain
Block 124352 is allocated to LBA 76308 as logBlock
Block 124352 is folded in LBA 76308 block chain,so in SB chain
Block 124352 is allocated to LBA 22506 as logBlock
Block 124352 is folded in LBA 22506 block chain,so in SB chain
Block 124352 is allocated to LBA 76308 as logBlock
Block 124352 is folded in LBA 76308 block chain,so in SB chain
Block 124352 is allocated to LBA 146267 as logBlock
Block 124352 is folded in LBA 146267 block chain,so in SB chain
Block 124352 is allocated to LBA 78076 as logBlock
Block 124352 is folded in LBA 78076 block chain,so in SB chain
Block 124352 is allocated to LBA 55646 as logBlock
Block 124352 is blockSwitched(shoulded be erased) in LBA 55646
Block 124352 is allocated to LBA 55649 as logBlock
Block 124352 is folded in LBA 55649 block chain,so in SB chain
Block 124352 is allocated to LBA 22771 as logBlock
Block 124352 is folded in LBA 22771 block chain,so in SB chain
Block 124352 is allocated to LBA 143668 as logBlock
Block 124352 is folded in LBA 143668 block chain,so in SB chain
Block 124352 is used to store data from Block 65812 by WL in LBA 159407
Block 124352 is choosed to erased in WL,and put it into LBA 159407
Block 124352 is allocated to LBA 41703 as DataBlock in foldChain
Block 124352 is folded in LBA 41703 block chain,so in SB chain
Block 124352 is allocated to LBA 41804 as DataBlock in foldChain
Block 124352 is folded in LBA 41804 block chain,so in SB chain
Block 124352 is allocated to LBA 41830 as DataBlock in foldChain
Block 124352 is folded in LBA 41830 block chain,so in SB chain
Block 124352 is allocated to LBA 41891 as DataBlock in foldChain
Block 124352 is folded in LBA 41891 block chain,so in SB chain
Block 124352 is allocated to LBA 42249 as DataBlock in foldChain
Block 124352 is folded in LBA 42249 block chain,so in SB chain
Block 124352 is allocated to LBA 18530 as logBlock

Block 124352 is blockSwitched(shoulded be erased) in LBA 18530
Block 124352 is allocated to LBA 76309 as logBlock
Block 124352 is folded in LBA 76309 block chain,so in SB chain
Block 124352 is allocated to LBA 35911 as logBlock
Block 124352 is folded in LBA 35911 block chain,so in SB chain
Block 124352 is allocated to LBA 22506 as logBlock
Block 124352 is folded in LBA 22506 block chain,so in SB chain
Block 124352 is allocated to LBA 76308 as DataBlock in foldChain
Block 124352 is folded in LBA 76308 block chain,so in SB chain
Block 124352 is allocated to LBA 4801 as logBlock
Block 124352 is folded in LBA 4801 block chain,so in SB chain
Block 124352 is allocated to LBA 144404 as logBlock
Block 124352 is folded in LBA 144404 block chain,so in SB chain
Block 124352 is allocated to LBA 144441 as DataBlock in foldChain
Block 124352 is folded in LBA 144441 block chain,so in SB chain
Block 124352 is allocated to LBA 79304 as logBlock
Block 124352 is folded in LBA 79304 block chain,so in SB chain
Block 124352 is allocated to LBA 12883 as logBlock
Block 124352 is folded in LBA 12883 block chain,so in SB chain
Block 124352 is allocated to LBA 79168 as logBlock
Block 124352 is folded in LBA 79168 block chain,so in SB chain
Block 124352 is allocated to LBA 76329 as logBlock
Block 124352 is blockSwitched(shoulded be erased) in LBA 76329
Block 124352 is allocated to LBA 43008 as logBlock
Block 124352 is folded in LBA 43008 block chain,so in SB chain
Block 124352 is allocated to LBA 75013 as logBlock
Block 124352 is blockSwitched(shoulded be erased) in LBA 75013
Block 124352 is allocated to LBA 99957 as DataBlock in foldChain
Block 124352 is blockSwitched(shoulded be erased) in LBA 99957
Block 124352 is allocated to LBA 99960 as logBlock
Block 124352 is folded in LBA 99960 block chain,so in SB chain
Block 124352 is allocated to LBA 100013 as DataBlock in foldChain
Block 124352 is folded in LBA 100013 block chain,so in SB chain
Block 124352 is allocated to LBA 100044 as DataBlock in foldChain
Block 124352 is blockSwitched(shoulded be erased) in LBA 100044
Block 124352 is used to store data from Block 148222 by WL in LBA 34758