國立交通大學

資訊科學與工程研究所

碩士論文

平均磨損演算法的自我調整策略

On-Line Self Tuning for Wear-Leveling Algorithms

研究生　　:黃莉君

指導教授: 張立平　教授

中華民國 九十九 年 七月

平均磨損演算法的自我調整策略

On-Line Self Tuning for Wear-Leveling Algorithms

研 究 生：黃莉君　　Student：Li-Chun Huang

指導教授：張立平　　Advisor：Li-Pin 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 2010

Hsinchu, Taiwan, Republic of China

中華民國九十九年七月

平均磨損演算法的自我調整策略

學生：黃莉君　　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

平均磨損演算法能夠使快閃記憶體上的區塊能夠平均被磨損，解決快閃記憶體的 endurance 問題。但是區塊的磨損程度的平均與否大多是由平均磨損演算法內所使用的 Threshold 值所控制。但是平均磨損演算法的效果在不同的軟硬體條件可能會不相同，而且不同的環境對於平均磨損演算法的需求也一定不相同，使用一個固定的 Threshold 值來並沒有辦法發揮演算法的最佳效能。因此本研究針對平均磨損演算法在不同的軟硬體條件之下，提出了一個創新的方法，結合了實際的測量和理論的推導，該方法可以在短時間就能快速估計出成本和效果的對應關係，為平均磨損演算法決定出適合當時環境的 Threshold 值，使其達到一個效果與成本之間的良好平衡狀態。在最後的實驗中，可以看出我們所提出的方法所估計出的結果比其它的對照方法都還要接近實際情況且執行快速，因此能讓平均磨損演算法隨時都發揮最佳的效果，使快閃記憶體的壽命有效延長。

關鍵字: NAND 快閃記憶體(NAND Flash Memory) ，平均磨損演算法(wear leveling algorithm)，線上調整(On-Line tuning)

On-Line Self Tuning for Wear-Leveling Algorithms

Student：Li-Chun Huang Advisors：Dr.Li-Pin Chang

Institute of Computer Science

National Chiao Tung University

ABSTRACT

Wear leveling algorithm makes all the blocks of NAND flash memory can be erased evenly. The evenness of blocks is always controlled by the Threshold value defined in wear leveling algorithm, because Threshold controls the frequency to invoke wear leveling algorithm. The performance of wear leveling varied from different software of hardware conditions, so a fixed Threshold value can not satisfied all the cases. Therefore, we propose an approach which combines the realistic measurement and analysis model to calculate the relation of overhead and Threshold of wear leveling rapidly, and choosing an ideal Threshold value which is most suitable for the environment. The experiment result shows our approach is more accurate and faster than other methods for deciding an ideal Threshold value .With self –tuning mechanism, the performance of wear leveling improves, and it effectively extends the life of NAND flash memory.

Keyword: NAND flash memory, Wear leveling algorithm, On-Line tuning

誌　　謝

研究所兩年的學習生涯轉眼間就過去了，在這當中許多人的幫助和關懷都使我銘記在心。

首先要感謝的是張立平教授，每當我研究遇到困難時總是耐心的給予協助，並且引導我至正確的思考方向，使我在知識方面增長了許多。兩年來的相處也讓我覺得教授總是非常關心學生，對我來說就如同是非常親切的良師益友一樣。

再來，感謝郭郡杰學長在我遇到問題時總是不厭其煩的幫助我。還有同學黃義勛，黃偉杰，郭晉廷，非常高興兩年來可以在研究上一起努力以及一起度過許多歡樂的日子，也謝謝學弟妹們在生活中大大小小的幫忙。還要感謝室友張筱苑總是一直給我鼓勵，讓我有勇氣面對所有難過的事。

最後感謝我摯愛的家人，謝謝你們一直以來的支持，才讓我有力量往自己的夢想邁進!

目錄

圖目錄

表目錄

On-Line Self Tuning for Wear-Leveling Algorithm

1. Introduction

NAND 快閃記憶體 因為它有體積小，耐震以及存取快速的優勢，所以現今被廣泛的使用在各種嵌入式裝置中。而隨著製程技術的進步，快閃記憶體的容量越來越大，故以快閃記憶體為基礎的固態硬碟有逐漸取代傳統硬碟的趨勢，開始被廣泛使用到筆記型與個人電腦,甚至是大型資料庫的應用場合。

快閃記憶體的存取是以頁(page)為單位，而抹除是以區塊(block)為單位，由於要更新一個頁的資料前，必須要先將該頁抹除，而一次抹除必須要抹除一個區塊，所以快閃記憶體的更新策略不能像一般硬碟一樣更新在資料原本的位置，而是採用 out place 的更新，更新前的舊資料將會被視為 invalid，因此會造成快閃記憶體 的空間會被一些 invalid data 給佔據。所以當快閃記憶體可用空間不足的時候，必須要盡量刪除那些 invalid data 回收區塊，而回收的過程中便會對那些區塊做 erase 的動作。每一個區塊可以被 erase 的次數是有上限的，超過上限即會造成區塊的毀損，進而使得其他區塊加速毀損而縮短快閃記憶體壽命，使得快閃記憶體面臨了 endurance 的問題。

而現今 endurance 的問題變得越來越嚴重，原因在於現今的技術可以讓快閃記憶體電位分成多個位階，讓每個 cell 可以儲存的 bit 數越來越多，出現了 MLC 快閃記憶體甚至是 TLC 快閃記憶體，由於他們具有容量大且成本低的優勢，目前漸漸成為市場的主流，但是 MLC[1] TLC 也因為電位分的較細所以比起 SLC[2] 來說 endurance 的問題更加惡化。而且隨著快閃記憶體的容量變大，新的應用出現，例如固態硬碟。一般用於例如記憶卡上的 access pattern 多半是循序的存取，而固態硬碟 的應用和普通硬碟一樣，如將作業系統裝置在 SSD 上，其 access pattern 會比較隨機且具有 locality 會一直存取同一個區域，而造成某些區塊被抹除的次數特別高而提早毀損。為了解決 endurance 問題，我們需要 wear leveling 來讓每一個區塊可以平均的被使用，延後第一個毀損區塊出現的時間，延長快閃記憶體的壽命。

Wear leveling 的目的是為了要平均快閃記憶體每個區塊被抹除這的次數，但是 wear leveling 不可能做到使所有區塊的抹除次數完全一樣，所以對於 wear leveling 演算法來說，多半會具有一個 Threshold 值來定義怎樣的狀態是平均來決定是否需要啟動 wear leveling。過去的一些 wear leveling 的演算法 例如 static wear leveling[3]，用 Threshold 值來定義一段時期內的是否密集的寫入某一些相同的 block，來判斷平均與否。而 group wear leveling[4] 和 lazy wear leveling[5]都是以控制 block 和平均的差值在 Threshold 範圍內來達到平均狀態。

上述的方法的 Threshold 值是需要高專業人士或是事先對 workload 已相當瞭解的人來調整的。使用同樣的 Threshold 值在不同的環境下不一定會產生相同的效果，例如裝置作業系統的固態硬碟和用於儲存多媒體檔案的記憶卡，前者的 access pattern 有明顯的 locality，可能會造成某些區塊一直被抹除，而後者是屬於循序的 access pattern，會比較平均的抹除各個區塊，因此後者對 wear leveling 的需求比前者小了很多，所以如果一律都使用前者所設定的 Threshold 值，可能會對後者的例子造成許多不必要的 wear leveling 成本。就算是同一個快閃記憶體也有可能接觸到各種不同類型的 access pattern。再者固態硬碟的製作也會搭配到不同的 flash geometry，這樣又要一一調整。所以我們需要一個能夠依照環境變化自我調整 Threshold 值的機制，才能夠使 wear leveling 在無時無刻都能夠發揮最佳的效果。

要挑選最適合的 Threshold 值，最簡單的就是使用在將當下環境之下測試完所有 Threshold 值的暴力法，它雖然單純且準確，但是因為需要大量的測試數據，所以將會讓一次自我調整耗費太多時間，依照我們的實驗結果，一次調整可能需要耗費將近一年的時間，所以這使得暴力法非常不實用。如果試圖減少數據量或是縮短測試時間，則會因為實驗的過程不夠精準，而使得到的則會讓實驗數據變的悖離真實情況而失真，即使選出的 Threshold 值也不具意義。

上述的討論展現了準確度和效率無法兩全其美的問題，因此我們提出了一個方法只需要少量的實際數據測量就能夠估計出 Threshold 值和它所付出的成本的關係曲線，解決了暴力法的缺點。藉著這樣的方法，我們實驗結合了實際數據測量和理論推演。我們先推導出 wear leveling 的成本-TH 函數，再去測量少量的實際數據，並使用這些數據來調整函數使其可以逼近真實的成本和 Threshold 關係。因為有理論函數的輔助，所以可以縮短測量的時間，但還是有實際數據的測量，所以依舊可以兼顧準確度。因此能夠在短時間內找出最適當的 Threshold 值，達到自我調整適應調整的目的。讓 wear leveling 可以在任何情況都可以用合理的成本發揮最佳的效能，增進快閃記憶體壽命。

我們做了一系列的實驗，首先分析了在不同的軟硬體情況下對 wear leveling 成本及效果的影響大不同，證實了自我調整適應機制的重要性。再來我們使用本篇所提出的方法找出最適合的 Threshold 值，從實驗結果顯示，使用我們的方法所找出的 Threshold 值比起其他對照組的方法，更加貼近使用長時間暴力法的所得到的結果，但是所使用的時間卻比暴力法短了很多，讓 wear leveling Threshold 值自我調整適應的機制，可以真正的被應用在實際產品中。

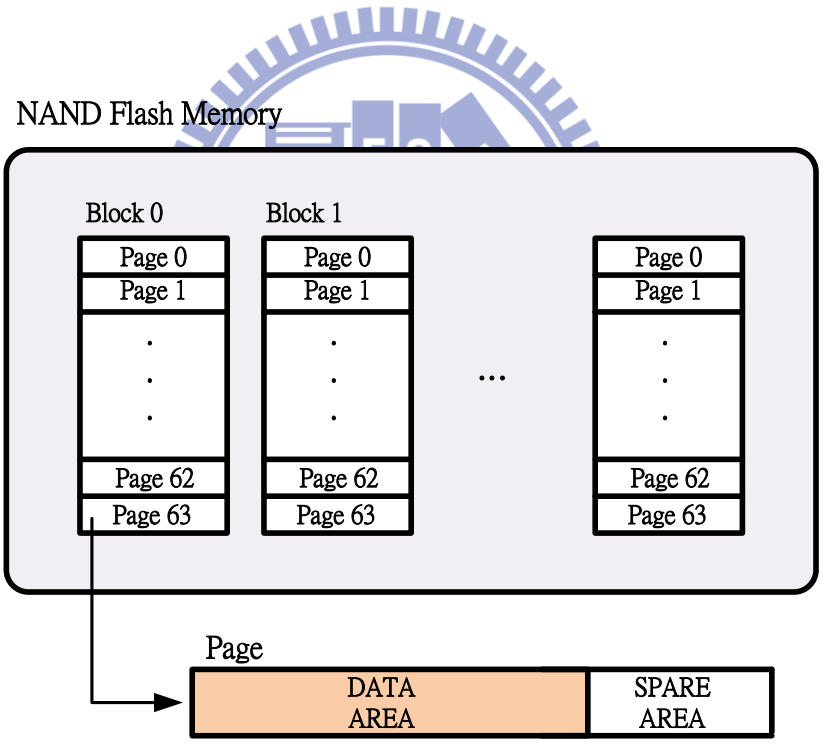
本論文的架構如下：第二章介紹了 NAND flash memory 的相關背景知識，以及論

文的動機。第三章則為做法的詳細說明。第四章是實驗結果的呈現，包含了本篇論文方法和其他對照組的比較，以及各種會影響 wear leveling 效果的因素說明。最後是本篇論文的總結。

2.Related Work

2.1 Flash Geometry

一個快閃記憶體的晶片內部被畫分成多個大小相同的區塊(block)，每個區塊內部又再被細分成多個大小相同的頁(page) ，例如典型的頁和區塊大小為 2KB 和 128KB，每一個頁會有一個 Spare Area，用以描述 Page 中的 Data，即 Metadata 如圖(1)。快閃記憶體具有 read/write/erase 的單位和時間不對稱的獨特物理特性，它讀寫的單位為頁，而 erase 的單位為區塊，而讀取一個頁的時間約為 20 微秒，寫入一個頁的時間約為 200 微秒，而 erase 需要最長的時間約為 1.5 毫秒，而快閃記憶體要寫入一個頁之前，必須確保它是乾淨無資料的，所以和傳統硬碟不同，快閃記憶體若要更新資料並不會將更新的資料寫到原本的位置，而是會去挑選一個乾淨的頁寫入，稱為 out place update，並將原本的舊資料變成 invalid data。當可用的空間不足時，快閃記憶體需要透過 erase 區塊這個動作來回收被 invalid data 所佔據的空間，這個動作就稱為 Garbage Collection[6]。

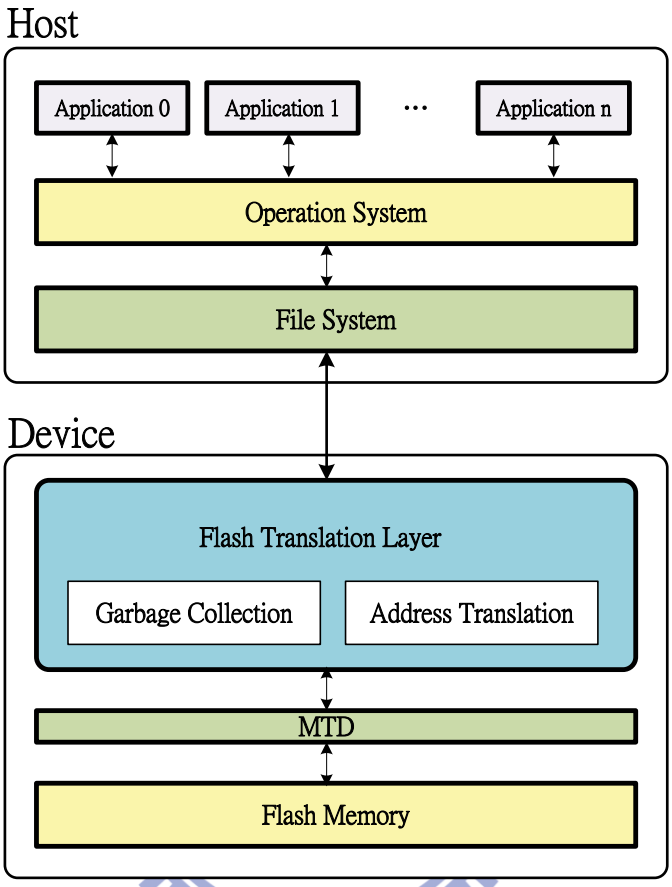


圖(1) flash geometry

2.2 FTL

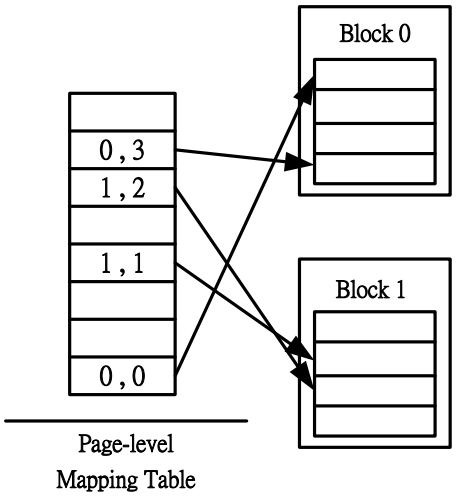
由於快閃記憶體是採用 out place update ，一筆資料真正位置會隨著更新而一直變動，所以原本常用於一般硬碟的上的檔案系統，例如 NTFS 或是 FAT32 並沒有辦法直接使用在快閃記憶體上，所以必須透過 flash translation layer(FTL)[7,8]

來將快閃記憶體儲存裝置模擬成與硬碟相容的裝置，讓使用者可以在上面運行硬碟常用的檔案系統，如圖(2)。FTL 最主要的兩個功能即為位置的轉換和 Garbage collection。

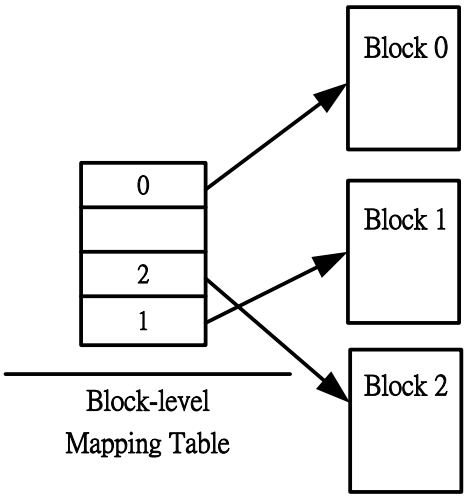


圖(2)系統架構

由於快閃記憶體寫入的單位是 page，而 erase 的單位是 block，所以也衍生出了 page level 和 block level 兩種不同的位置轉換對應。Page level 的對應是屬於 fine grain 的對應方式如[9][10]就採用此對應方式，它可以充分的使用快閃記憶體的空間，可將一個由 host 端傳來的 logical address 對應到任何一個快閃記憶體的實體page上，但是它必須記錄每個page對應的位置，所以page level 對應的mapping table 需要很大的空間 ，見圖(3.a)，所以 page level 對應比較少用在實際的產品上。



圖(3.a)　page level mapping



圖(3.b) block level mapping

為了解決 page level 對應所使用的 mapping table 太大的問題，我們需要 coarse grain 的對應方式，block level 對應即是以 block 為單位來對應，大幅縮小的 mapping table 的大小，見圖(3.b)。block level 對應會將 logical address 切割成兩部分，一個是 logical block address，另一個是 block offset。Logical block address 會對應到一個快閃記憶體的實體 block，該 block 稱為 data block，而 block offset 會指出要將該 address 對應到 data block 的哪一個 page。如果寫入 data block 時已經有 data，這時會配置對應的 log block 來提供更新資料的寫入同樣的 block offset 位置。但是 block level 的對應當有較小資料寫入時，會有空間碎裂和區塊使用率低落的問題。

因此還有結合 block level 和 page level 對應的 hybrid 的對應方式。在 data block 上的對應是採 block level 的對應，而更新 data block 的資料時則會採用 page level 對應的方式，來改善 log block 使用率低的問題。另外也有其他的更具彈性 data log block 對應的方式，可自由調整 data block 和 log block 的對應比例[11][12]。

本篇論文中所使用的 FTL Block Chain [5]，FAST[13]，NK[14]都是 hybrid 的對應方式，原始資料和 data block 的對應皆是採用 Block level 的對應方式，而更新的資料則會以 page level 的對應方式儲存在 log block 上，他們的不同之處就在於 data block 和 log block 的對應比例。Block Chain 為一個 log block 只能接受一個 data block 的更新資料，一個 data block 可擁有固定數量的 log block 來存放更新資料。FAST 則是所有讓所有的 data block 共用所有的 log block，一個 log block 可接受來自各個 data block 的更新資料。而 NK 則可以有彈性的調整 data block 和 log block 的對應關係為 N:K。

2.3 wear leveling

由於快閃記憶體有 endurance 的問題，每個區塊可以承受的被 erase 的次數是有上限的，一般來說 SLC 快閃記憶體可承受被 erase 的次數為 100000 次，而 MLC 快閃記憶體約為 10000 次而已，若是有一個區塊毀損，則會使整個快閃記憶體的其他區塊加速毀損，所以為了延後第一個區塊毀損的時間，延長快閃記憶體的壽命，所以需要 wear leveling 來使每個區塊被使用的程度能夠達到平均。而各個 wear leveling 的做法和對平均的定義都各不相同，我們可以從下表(1)中看出

表中所用到的縮寫:
WL: wear leveling
GC: Garbage Collection
TH: Threshold

WL 名稱	啟動時機	WL 策略
Ban's algorithm[15]	經過 TH 次的 erase 或是 write request 啟動 WL	選一個 block 來 erase
Static wear leveling[3]	當時間間隔內被寫入的區塊密集程度大於 TH 值	將 cold data 搬走
Group wear leveling[4]	當寫入 Update block 時，發現其 ec 和平均 ec 次數最少的 group 的差值大於 TH 值	做 hot cold data 的交換
Lazy wear leveling[5]	當 GC 時，發現欲回收的 block 其 ec 的差值和平均大於 Threshold 值	將 cold data 移入 old block
Dual-Pool Algorithm [16]	當 hot cold pool 各自的 head block 差值超過 TH	Hot cold data 的交換

表(1)wear leveling 方法

2.4 Problem Formulation

從上一節談到了許多不同的 wear leveling 的方法，我們可以看到其中有很多都是使用 Threshold 值來當作偵測目前各區塊被抹除的次數是否平均的標準，如果此

標準成立就會觸發 wear leveling。但是因為 wear leveling 的效果會因為在不同的 workload，FTL，flash geometry，over-provision 比例之下而不同，而且不同的軟硬體環境對 wear leveling 的需求可能也會不同，所以如果一律使用固定的 Threshold 值並沒有辦法讓 wear leveling 發揮最好的效果。

在本篇的研究我們使用上述所提到的 wear leveling 成效最佳的 lazy wear leveling 為研究的基礎，提出一個方法可以讓 wear leveling 的 Threshold 值可以依照環境需求自我調整到最適當的狀態。原本最簡單準確的方法就是使用在當下的環境一一測試所有 Threshold 值的暴力法，但是暴力法存在極大的缺點讓它無法實際應用。

我們計算一次使用暴力法調整 Threshold 值所使用的時間，假定我們要測試的 Threshold 值的範圍在 4~64 之間，而每一個值至少需要 8GB 的資料才能，所以我們需要至少 480GB 的資料量才能夠完成一次 Threshold 值的調整，但是一般使用者每個月所存取的資料量大約也才 50GB，所以使用暴力法來調整 Threshold 值需要將近一年的時間才能完成，但是如果我們試圖降低測試時使用的資料量測量的結果就會像呈現完全失真的狀態。

所以為了解決暴力法無法兼顧準確度和效率的缺點，我們提出的方法必須要做到以下三點，(1)能夠針對 wear leveling 的行為做分析，建立不同的 Threshold 與成本的關係函數來減少測量的時間，必須兼顧實用性和準確性(2)找出適當的 Threshold 值能夠同時有效的平均區塊間的抹除次數，且能夠控制成本在合理的範圍內(3)能夠週期性的啟動才能即時配合環境的變動來調整 Threshold 到最適當的數值。

3.On-Line Self Tuning

3.1 Lazy Wear Leveling

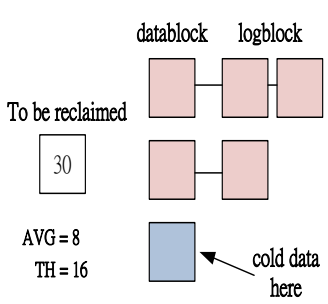
Lazy wear leveling 是我們實驗所採用的 wear leveling 方法，為一種針對被 erase 很多次的 old block 來做追蹤的 wear leveling 方式。Lazy wear leveling 主要的概念就是當他偵測到 old block 時，就將 cold data 移到 old block 上，以阻止 old block 繼續被磨損下去，藉此解決 block 之間磨損不均的現象。因為在一般的 workload 中，cold data 的比例比 hot data 來的高，因此只追蹤被 hot data 磨損的 old block 的 lazy wear leveling 是一種很節省成本的 wear leveling algorithm。

Lazy wear leveling 挑選 cold data 的方式是去找出沒有被更新過的 logical block，因為一個 logical block 如果沒有被更新，幾乎就可以認定該資料為 cold data。而挑選 old block 的方式則是在啟動 Garbage Collection 時，檢查欲回收的 block 的 erase cycle 是否高於 average erase cycle + Threshold，如果條件成立了即為 old block，此時 lazy wear leveling 會避免 old block 再繼續被 erase 後回收，而是找出 cold data 並將 cold data 放入 old block 上，而由原本存放 cold data 的 block 代替 old block 被回收。

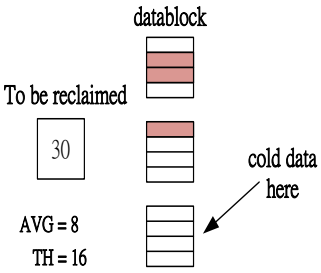
Lazy wear leveling 為一種可適用於各種 FTL 的 wear leveling algorithm。執行的過程就如下圖(4)的例子所示，原本被 garbage collection 選來回收的 victim block ，因為它的 erase count 為 30 大於平均的 erase count 加上 TH 值，因此他會被 lazy wear leveling 當作 old block。當 FTL 為 BC 時，見圖(4.a)，我們就直接選擇後面並沒有接 log block 的 data block 作為 cold data 人選，由該 data block 代替 old block 回收，原本的 old block 變成 data block。

當 FTL 為 FAST 時，見圖(4.b)，圖中著色的部分為 invalid data，所以我們找 block 內所有 page 皆為 valid 的 data block 作為 cold data 的選擇。其餘的步驟和 BC 都還是相同。

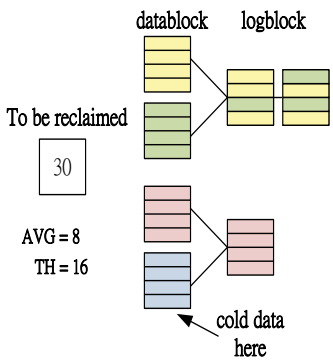
當 FTL 為 NK 時，見圖(4.c)，log block 內 page 的各種顏色顯示出該 page 是來自哪一個 data block 的更新資料。雖然 NK 也具有像 BC 一樣的 log block chain，但是其 log block 所寫入的 page 可能有來自多個 data block 的更新，見圖中黃色和綠色的部分，也有可能只來自單一一個 data block 的更新，見圖中的紅色和藍色的部分。所以這邊我們不以 log block chain 的長度作為 cold data 選擇的依據，我們還是選擇 page 全為 valid page 的 data block 為 cold data。



圖(4.a) LWL 執行--BC



圖(4.b)LWL 執行--FAST



圖(4.c) LWL 執行--NK

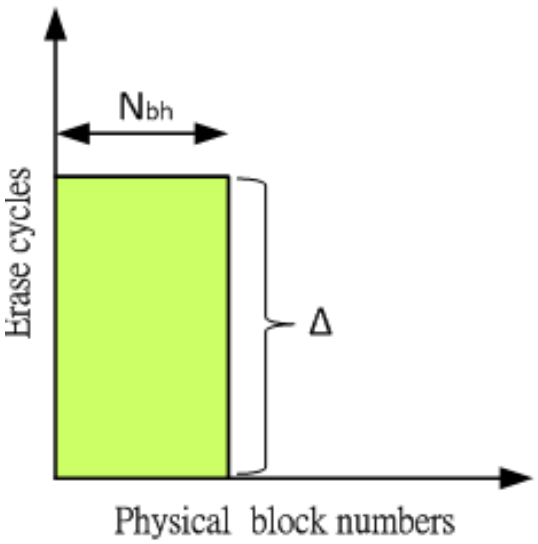
3.2 Overhead Analysis

假設一個 NAND flash memory 的 chip 中共有 N_b 個 physical block，假設每一次寫入 flash 的資料的大小都正好是 block size 的倍數，每次寫入也都會對齊 block 的位置，且 workload 是非常一致的只會更新一樣的邏輯位置。假設其中 flash 有 N_{bc} 個 physical block 是對應到不會被更新的邏輯位置，也就是不會被更新的 cold block 數量有 N_{bc} 個，所以會因為一直被寫入而磨損的 physical block 則有 $N_b - N_{bc} = N_{bh}$ 個。

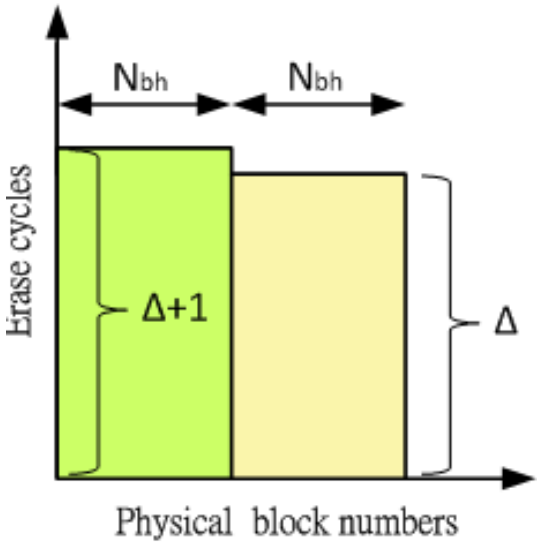
設函數 *f(x)*為沒有使用 lazy wear leveling 時，Garbage Collection 會造成的 total erase count。*x* 為 workload 中持續被寫入的 x 個 logical block，我們結合上一段所述，設定 $x = i * Nbh * \Delta$，其中 i 為一個非負的整數，Δ 為 Threshold。因為沒有 lazy wear leveling 的介入，所以我們可以得到以下式子

$$f(x) = x$$

為了分析上的方便，我們先簡單的修改一下 lazy wear leveling 的做法。原本 old block 的判定是和平均的 erase count 比較，現在先將他修改為和擁有最小的 erase count 的 block 比較。從下圖(5.a)可以看出在 lazy wear leveling 啟動前，Garbage Collection 會將 erase count 平均的分布在一直持續被寫入的 N_{bh} 個 block 上，總共造成了 $N_{bh} * \Delta$ 次的 erase counts。



圖(5.a) LWL 沒有介入前



圖(5.b) LWL 轉移 erase 到其他區塊

在這時候如果這 N_{bh} 中的 block 其中有一個又因為 Garbage Collection 而 erase，此時就會造成 erase count 達到 Δ+1，而使 lazy wear leveling 開始啟動。所以說這 N_{bh} 個 block 如果 erase count 都達到 Δ+1，則會引起 N_{bh} 次的 lazy wear levling 。因為 lazy wear leveling 的介入會造成 logical block 和 physical 的 block 的 remapping，讓原本被一直持續被 erase 的 N_{bh} 個 block 會停止繼續被 erase，見圖(5.b)綠色部分，取而代之由另外一批 N_{bh} block 開始被 Garbage Collection erase 見圖(5.b)黃色部分。

如果將 lazy wear leveling 的加入考慮，我們將之前的 *f(x)*修改成以下式子

$$f'(x) = x + x/\Delta \quad = \quad x + i*N_{bh}$$

但是實際的 wear leveling 結果並不會完全就像分析的一樣單純，所以我們將式子加入了一個變數 K。

$$f'(x) = x + i*N_{bh}*K$$

K 代表著各種影響 wear leveling 效果的因素，例如不同的 workload，FTL，flash geometry 等等。舉例來說，不一定每一種 workload 都有如此明顯的 locality，有時候 hot cold data 的界限比較難判定時，wear leveling 可能會產生誤判的現象，造成一些不必要的 erase，這時後就會讓 K 值大些 。又或者在某些軟硬體條件之下本來就會使區塊 erase 分布比較平均，這時後 K 值就會比較小。

我們定義 *g(Δ)*為 overhead funtion，可以計算出在不同的 Δ 之下，lazy wear leveling

所耗費的成本(overhead ratio)。

$$g(\Delta) = (f'(x) - f(x)) / f(x) = i*Nbh*K / i*Nbh*\Delta \quad = \quad K/\Delta$$

因為 lazy wear leveling 原本的定義是和平均 erase count 而非和最小 erase count 比較，所以我們再將 overhead fuction 修改成

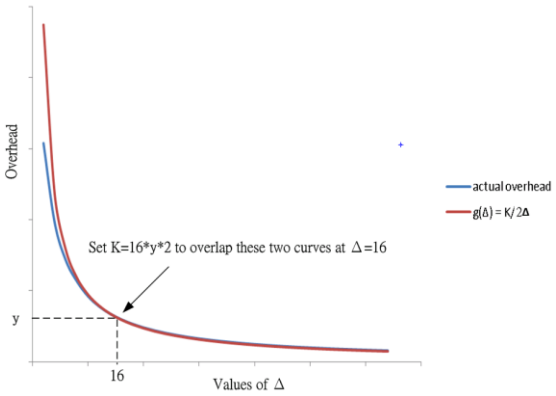
$$g(\Delta) = K/2\Delta$$

3.3 Self Tuning Concept

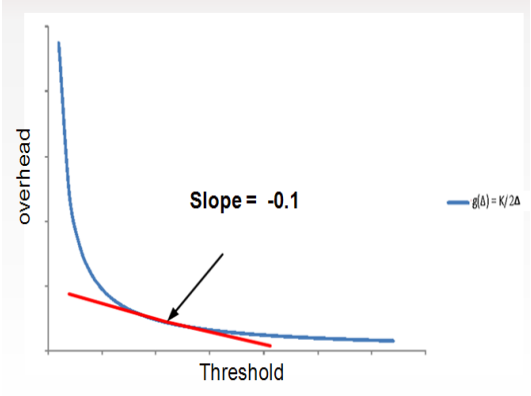
當得到 overhead function 為 $g(\Delta) = K/(2\Delta)$時，就可以用此公式為輔助來完成快速的自我 Threshold 值的調整。

要先建立出屬於當下環境的 overhead function。由上一節所述，我們已經得到 overhead function 的雛形為 $g(\Delta) = K/(2\Delta)$時，所以我們真正需要實際量測的就只有該 function 的 K 值，大大的降低了原本使用暴力法所需要的資料量。

測量 K 值的方法，首先我們先設定 Δ 為 16 並讓 wear leveling 跑到 overhead 比例呈現穩定的狀態，選擇 Δ 為 16 乃是因為根據我們的實驗結果，Δ=16 是最適合的設定。我們假設 y 為 Δ 為 16 時所偵測出來的 overhead ratio。再帶入 g(16) = y，就可求出 K = y*2*16 見，下圖(6.a)。當我們測量出 K 值時，就可以利用 $g(\Delta) = K/(2\Delta)$的式子去描繪出模擬的 wear leveling 成本和 Δ 的關係曲線，下圖(6.a)(紅線)，並利用他去逼近實際的 overhead ratio，下圖(6.a)(藍線)。因為 $g(\Delta) = K/(2\Delta)$在 Δ 很小時，會讓 $g(\Delta)$的值上升的很快，所以估計出來的 overhead ratio 會偏高一點。

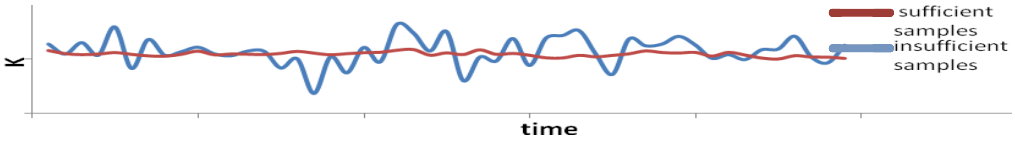


圖(6.a) K-estimation



圖(6.b) TH 選擇

當測量 K 值時，我們需要有足夠的寫入資料量才能夠讓量出來的 K 值是有意義的，下圖是在一個不停重複跑的 workload 下，使用不同的資料量所測出的 K 的差異。



圖(7)資料量與 K 值

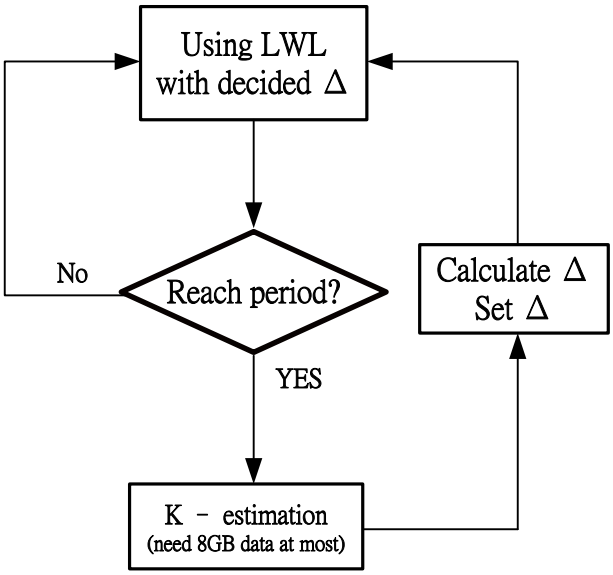
在同樣一個 workload 之下，如果我們每次測量 K 值時有提供足夠的寫入資料量讓他達到 overhead ratio 呈現穩定，所測量出的 K 值就會像上圖紅色線所示，呈現相當平穩的狀態。反之，就會像藍線一樣動盪不定，每次測出來的值都不同。而根據我們的實驗結果觀察，大概只要提供 8GB 左右的寫入資料量就能得到穩定的 K 值。

當得到 K 值以後，我們就可以描繪出在當下環境下 wear leveling 的 Threshold 與其付出成本的模擬曲線，就可以依照曲線的形狀，找出一個具有 wear leveling 效果但是卻不需要付出太高成本的的 Threshold 值，並將目前的 Threshold 值調整為選出來的值。因為當 Δ 很小時，overhead 會上升的很快，所以再挑選適當的 Threshold 時，我們選擇曲線斜率為-0.1 的部分，如圖(6.b)。由圖中可見超過這個斜率之後，overhead 幾乎呈現急遽成長。

所以總結以上所述，wear leveling 自我調整 Threshold 的步驟就如下。首先，先測量當 Δ=16 時所造成的 overhead。再來，利用所量出的 overhead 得到 K 值，並建構出 g(Δ)曲線來預估其他 Δ 所造成的 overhead。最後再選擇曲線斜率為-0.1 的 Δ 為理想的 Threshold 值，並且要週期性的執行自我調整 Threshold 值，才能夠隨時確保 wear leveling 能夠有效率的執行，延長 NAND flash memory 的壽命。

3.4 Implementation Issues

Wear leveling 的 Self-Tuing 的機制執行的流程就同下圖(8)所示。我們修改原本使用固定 Threshold 值的 lazy wear leveling，每經過一段固定的週期，週期可以設定為快閃記憶體每寫入定量的資料，就會呼叫 K-estimation。進入 K-esitimation 程序後，它會暫時將 wear leveling 的 Threshold 值設定為 16，並且統計出 Threshold 值為 16 時，wear leveling 所帶來的額外的 erase count，也就是 wear leveling 在 Threshold = 16 時的成本，在依照上節所述的求出 K 值。K-esitimation 的統計時間是非常快速的，只需要 8GB 以下的資料寫入就可以得到結果。



圖(8) self-tuning 流程

得到 K 值後就可直接 $g(\Delta) = K/(2\Delta)$得到成本-TH 的模擬關係曲線。計算出建議的 Threshold 值過程也相當快速，只須帶入簡單的微分公式求出斜率為-0.1 的 Δ 值。

$$g'(\Delta) = -2K/(\Delta^{-2}) = -0.1$$

$$20K = \Delta^2$$

$$\Delta = \sqrt{(20K)}$$

最後再變更 Lazy wear leveling 的 Threshold 值設定為所求出的 Δ 值，即為完成一次的 self-tuning 的執行區間。

K-esitamation 所需要的成本是相當低的，我們僅需要幾個額外的變數來計算是否累積到定量的寫入資料，儲存 K 和 Δ 值，以及記錄 wear leveling 所帶來額外的 erase count 和 overhead，若每個變數使用 4 byte 也僅需要 20 byte 的記憶體空間。至於 K-esitamation 蒐集完資料後，計算時間只需複雜度 O(1)就可完成，最佳 Threshold 值的決定也可在 O(1)間完成，所以 self-tuning 的機制幾乎不會帶來多餘的成本。

4. Experiment Result

4.1 Experiment Setup and Metrics

我們的實驗所使用的 NAND flash memory 是以 4 個 channels 的架構為基礎，假設 erase 時所有的 channel 會一起動作，而存取則不需要所有 channel 一起運作，規格如下表(2)所示:

Page Size	4K Bytes
Block Size	512K Bytes x4 = 2M Bytes
Pages Per Block	128
Total Size	20G Bytes
Over-Provision	1.25%

表(2)快閃記憶體規格

而所使用的 FTL 為一個 log block 只能對應到一個 data block 的更新的 Block Chain。在實驗測試的過程中，我們使用的 workload 是從筆電所擷取下來的一個月內的 read/write 存取指令，環境如下表(3)所示:

OS	Windows XP
File system	NTFS
Total writes	1772339
Disk volume size	20GB

表(3)workload 資訊

我們的比較對象包括了以下表格所列出的的方法，而比較的結果我們將會使用圖示以及與比較對象和暴力法的平方和差距的方式來呈現。

名稱	方法
暴力法	在固定環境下，將所有的 Threshold 值一個一個逐一分別測量，每一個 Threshold 值使用 20G 的資料測量
Stepping - 1G/4G	在固定環境下，將所有 Threshold 值由大到小連續測量。 每一個 Threshold 值使用 1G/4G 的資料測量
Partition - 2/4/8 parts	將 NAND flash memory 分割成同等大小的 2/4/8 等份，每一個部分測量不同的 Threshold 值，每一個等份使用的資料量為 20G

表(4)實驗對照組列表

實驗的內容包含了在不同 Workload，FTL，Flash Geometry 和 Over-provision 的比例下，會分別對 lazy wear leveling 產生怎樣不同的影響，而本篇 paper 的方法又會以這些影響為依據，找出最適合當下環境的 Threshold 值。在後面的章節中，我們會依照這些因素對 wear leveling 影響力的大小呈現實驗結果。

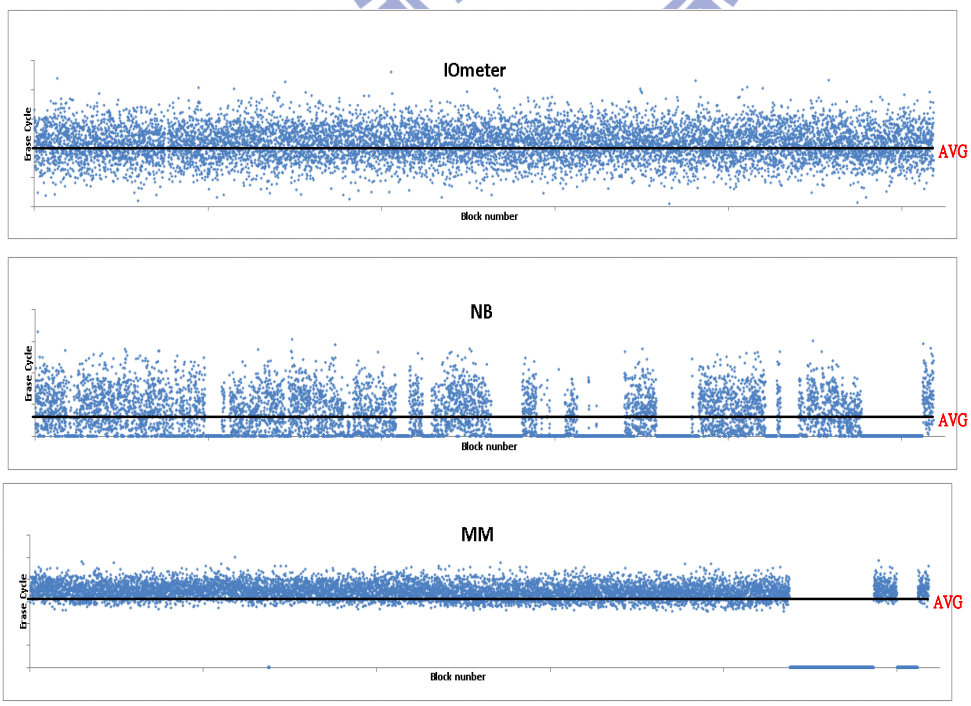
4.2 Using Different Host Workloads

4.2.1 Workload Introduction

不同的 Workload 會對 wear leveling 有不同的影響，所以我們的實驗選擇了 NB，IOmeter[17]，Multimedia，這三種特性相差甚大的 workload，來囊括所有快閃記憶體在使用時可能會遭受到的各種存取案例。

不同的 Workload 因為存取的特性不同，所以會在快閃記憶體上所造成的現象也不同，下圖(9)是三種 Workload 在沒有 wear leveling 干涉下，造成的快閃記憶體上 block 的 erase count 分布圖。IOmeter 是一個非常隨機的 access pattern，它會非常分散但是平均的將資料寫入不同的位置，所以會讓快閃記憶體各個 block 的 erase count 分布呈現常態分布的現象，沒有 block 的 erase count 會特別高或是特別低。

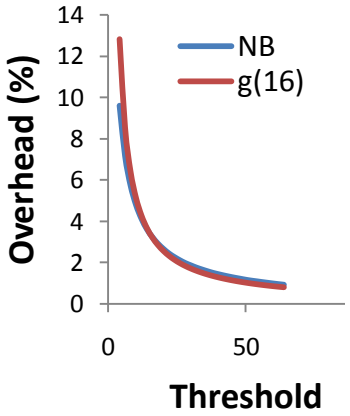
NB trace 有明顯的 locality，所以有些位置會一直重複的被寫入，而有些位置都沒有被寫到，因此會有 erase count 分布非常不平均的現象。Multimedia 則是一個比較循序的 access pattern，除了一些因為作業系統所造成的少數一直寫入的 hot data 和一部分沒有被寫到的區塊之外，在沒有 wear leveling 的介入下，它幾乎呈現非常平均且集中的 erase cycle 分布。



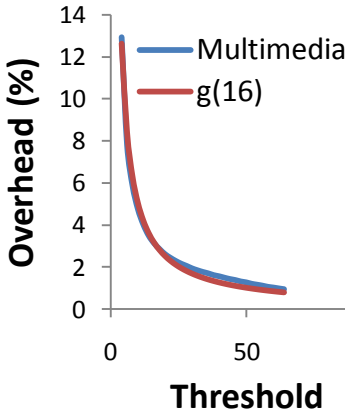
圖(9)不同 workload 的 erase count 分布圖

4.2.2 **Workload Experiment Result**

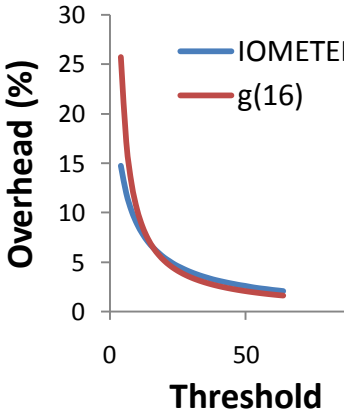
在本節中，我們先討論不同的 access pattern 會對 wear leveling 的效果產生不同的影響以及比較不同的實驗結果。下圖(10)是我們使用 K-estimation 所模擬出的成本-TH 的關係圖和暴力法的比較。



圖(10.a)NB



圖(10.b) Multimedia



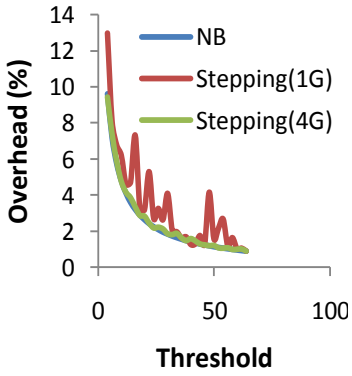
圖(10.c)IOmeter

Trace	RSS of K-estimation
NB	13.4399
Multimedia	1.967
IOmeter	163.167

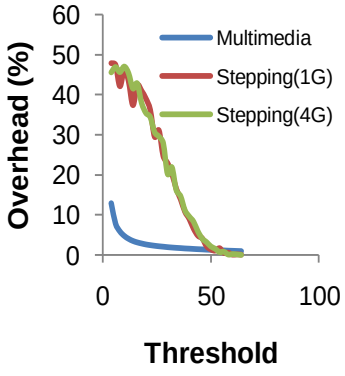
表(5)K-estimation 的 RSS 值

可以看出，K-estimation 所模擬出的成本-TH 曲線在直接觀察下，其實和暴力法的曲線看起來並沒有相差太多，造成的 RSS 也數值也很小，見表(5)，唯一差距較大的是 IOmeter，但是我們之前有提過當 Threshold 值很小時，K-estimation 所估計的 overhead 會偏大，此部分所佔的誤差佔了全體誤差的 75%以上，所以捨去端點部分不看，K-estimation 的模擬結果依舊是相當接近。

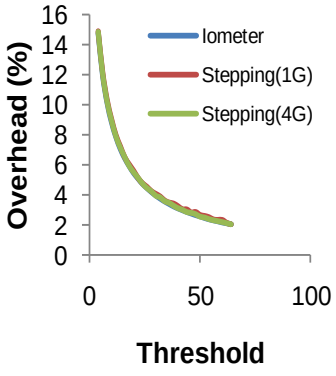
下面圖示是使用 Stepping 所測量出的結果



圖(11.a)NB



圖(11.b)Multimedia



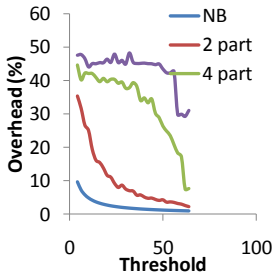
圖(11.c) IOmeter

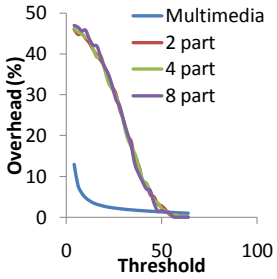
Trace	RSS of Stepping(4G)
NB	0.41
Multimedia	17591
IOmeter	0.33

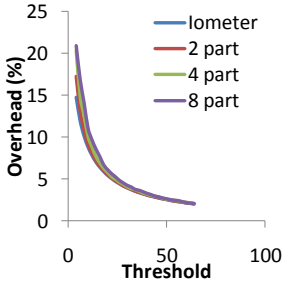
表(6)Stepping 的 RSS 值

使用 Stepping 的結果中，可直接看出當測量的資料量不足時，測量出的曲線就很容易動盪而不穩定，所以可以明顯看出 Stepping - 1G 的曲線跟 Stepping - 4G 比起來就顯得非常不穩定，跟暴力法相差甚大，所以難以用來決定最佳的 Threshold 值。而 Stepping - 4G 雖然表現的很好，在 RSS 值在 NB 和 IOmeter 上均小，見表(4)。但是它具有兩項缺點，第一，Stepping - 4G 所需要的時間非常的長，因為它執行一次所需要的資料量至少要 250GB 以上，而 K-estimation 則只需要 8GB 左右，Stepping - 4G 幾乎不具有實用的價值。第二，Stepping 對於循序檔案的測量效果表現得很差，從圖(11.b)中就可以看出它的曲線跟實際情況根本嚴重失真，造成如此結果的原因就在於 Stepping 是在執行中連續測量 Threshold 值，所以前一個 Threshold 值的測量結果就有可能會影響到下一個 Threshold 值的結果。這個現象在 Multimedia 中特別的嚴重，因為 Multimedia 的寫入是很循序的，所以大部分的 block 它的 erase count 都是很均勻的一起上升著，就算是有 wear leveling 的介入也不會改變這個趨勢，而且之前見圖(9)就已經看到 Multimedia 它的 erase count 分布非常集中，所以隨著 Stepping 測量的 Threshold 值越來越小，當測量的 Threshold 值進入 erase count 分布很集中的地帶時，反而會造成 wear leveling 太容易引發而產生高 overhead。

最後是使用 Partition 所得到的實驗結果。







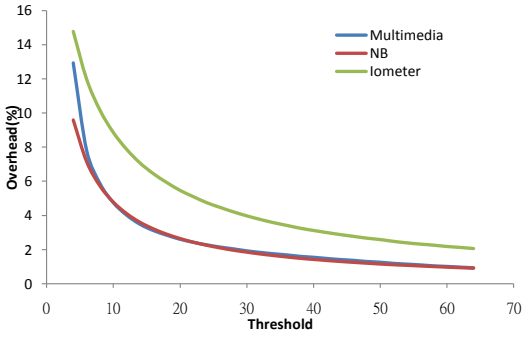
圖(12.a)NB　　圖(12.b) Multimedia　　圖(12.c)IOmeter

從實驗的圖表中，可見 Partition 在 Multimedia 時依舊延續了上一段所提到的缺點，所以 Partition 在 Multimedia 上幾乎有和 Stepping 一樣情況的誤差。Partition 在 NB 時我們從圖(12.a)中可以看到，當我們將快閃記憶體切成越多部分，測量出的曲線誤差就越大，原因就在於因為 NB 這個 trace 具有高度 locality，當切成越多等份測量時，就越容易受到 NB 的 locality 影響，造成每個 Threshold 值的測量環境差異很大，失去公平性。同理，因為 IOmeter 完全沒有 locality 的問題，所以測量出來的結果不管將快閃記憶體切成多少等份結果都相差不大。

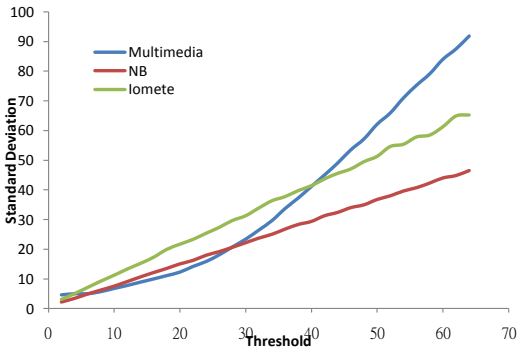
再來表(7)是我們使用 K-estimation 在不同 worload 下所得到的 Threshold 值。以使用暴力法測量 IOmeter，Multimedia，和 NB 所測量到的成本-TH 關係圖以及標準差-TH 關係圖，見圖(13)。

Workload	K	Threshold
NB	1.0268	23
Multimedia	1.0092	23
Iometer	2.06	32

表(7)不同 workload 之 K 值及 TH



圖(13.a)不同 workload 成本-TH 關係圖



圖(13.b)不同 workload 標準差-TH 關係圖

從圖(13.b)中可以看出 TH 跟標準差大致上就是呈現正比的關係，這與 lazy wear leveling 對平均的定義有關，所以基本上 Threshold 值越小的話，block 之間的 erase count 分布也就越平均。

從圖(13.a)中，可以看到 IOmeter 的 wear leveling 成本是高於另外兩者的，原因就在於在上節所提到的，IOmeter 的 erase count 的分布幾乎是呈現常態分布，所以 erase count 高於平均值加上標準差的比例約為 17%，所以很容易就可以引發 wear leveling。但是 IOmeter 並沒有所謂 hot cold data 的分別，所以 wear leveling 很難介入這麼平均的隨機寫入，做了效果沒有很好，反而造成很高的成本，因此 K-estimation 所測量出的 K 值較大，找出的 Threshold 值也偏大如表(7)所列，如此可避免引發過的無謂的 wear leveling。

NB trace 有明顯的 locality 而非常不平均，但是在 trace 中其實 hot data 佔的比例是比較少的的，所以會引發 wear leveling 的區塊跟 IOmeter 比起來較少，又它的 hot cold data 是很明顯的，因此不需付出太多成本，wear leveling 的效果就很好，所以 K-estimation 找出的理想 Threshold 值約為 23，正好是處在成本暴增之前的點。

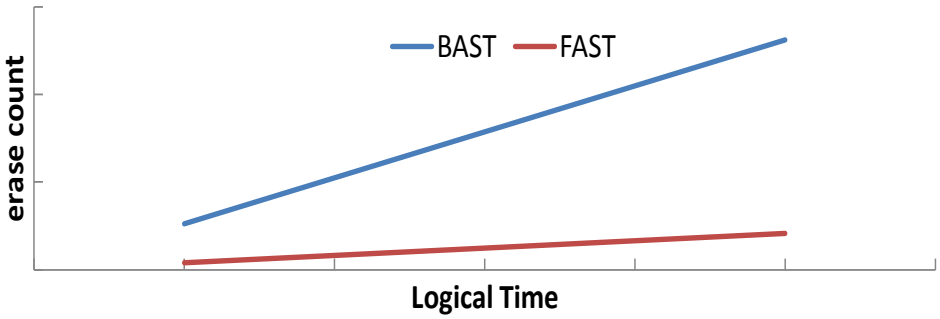
至於 Multimedia trace，因為他的 erase cycle 分布很集中且平均，所以太大的 Threshold 值並不會啟動 wear leveling ，而太小的 Threshold 值則會大幅引發 wear leveling 。他大部分為循序寫入，只有少部分的 hot data，所以 hot data 很容易辨識，原本 wear leveling 應該會很好做，但是 Multimedia 常常會有大範圍的循序寫入，因此又會破壞了之前 wear leveling 的將 hot cold 資料交換的結果，，因此整體的效果和 NB 看起來差不多。

4.3 Different FTL algorithms

此節我們將針對不同的 FTL 來觀察它對 wear leveling 效果的影響。
首先我們先介紹實驗中所使用的 FTL 的不同之處。

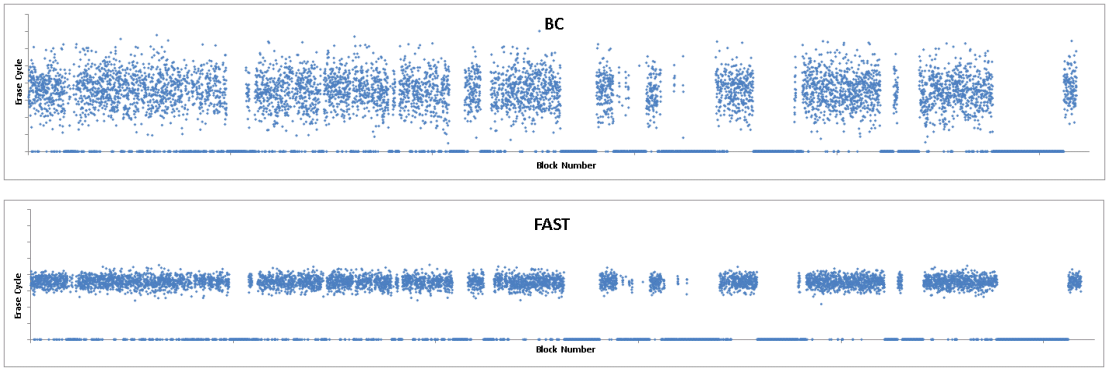
BC 和 NK 這兩種 FTL 之下的結構是很相似的，唯一的不同就只有 NK 的每一個 log block 可以吸收來自兩個 data block 的更新資料，增加些許的 log block 使用率。所以在此我們就針對 BC 和 FAST 的部分來做比較。

FAST 的 log block 是可以以容納所有 data block 的更新資料，所以他的區塊的使用率較佳，比較不會引發 Garbage Collection，所以在同樣的 trace 之下，FAST 的區塊被 erase 的速率比 BC 還慢了許多，平均 erase count 也會較小，見圖(14)。



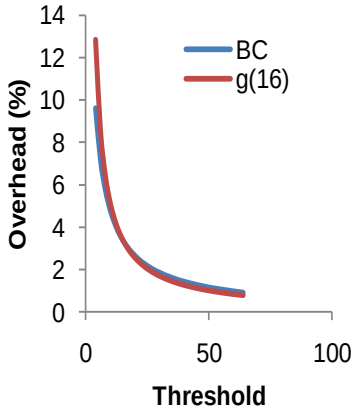
圖(14)BC 和 FAST　erase count 成長速度比較

又再者，我們觀察在沒有 wear leveling 介入時，我們若使 FAST 和 BC 兩者有相同的 erase count，則他們所造成的區塊 erase count 分布圖也不相同，從下圖(15)中可看到 BC 的 erase count 分布較 FAST 比起來比較凌亂不集中，因此可見 FAST 的 Garbage Collection 的效果是比較好的，它本身的結構就可以讓 block 間的 erase count 分布比較平坦。

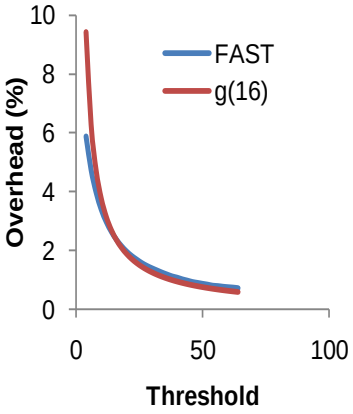


圖(15)不同 FTL 的 erase count 分布圖

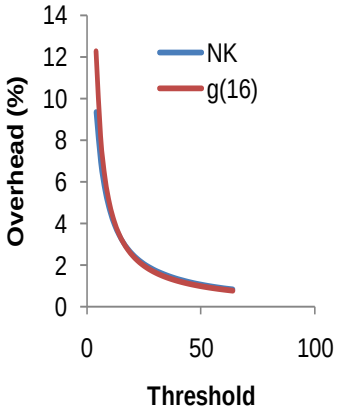
下圖(16)是我們使用 K-estimation 所模擬出的成本-TH 的關係圖和暴力法的比較。



圖(16.a)　BC



圖(16.b)FAST



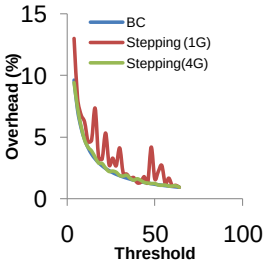
圖(16.c) NK

FTL	RSS of K-estimation
BC	13.43
FAST	15.95
NK	10.67

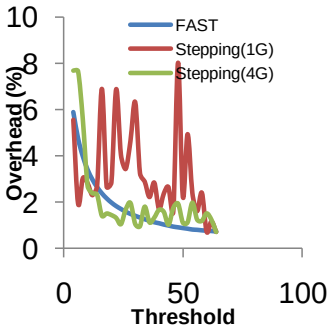
表(8)K-estimation RSS 值

由實驗結果的圖(14)中可以直接看出模擬曲線除了接近 Threshold 值很小的端點部分之外，和暴力法的曲線幾乎都是重合的，而且表(8)的 RSS 在 3 個 FTL 下也都很接近，若我們將端點部分的誤差扣除，BC，FAST，NK 的 RSS 僅剩下 2.94，3.44，2.13 而已，所以使用 K-estimation 的結果來代替實際測量結果是相當適合的。

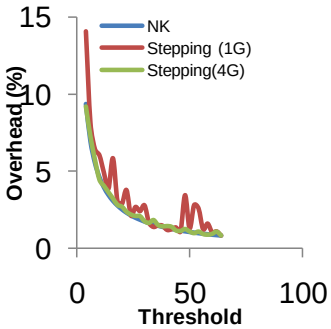
下面圖示是使用 Stepping 所測量出的結果



圖(17.a)BC



圖(17.b)FAST

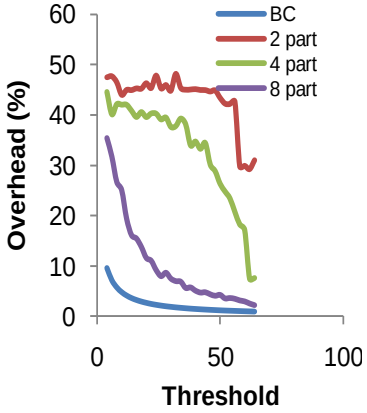


圖(17.c) NK

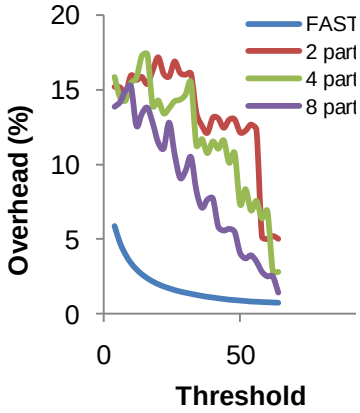
從 Stepping 的結果還是可明顯看出，Stepping 依舊存在著上一節所提過的相同問

題，當測量的資料量不足的時候，測量出的曲線就會動盪不穩定而難以用來決定最適合的 Threshold 值，而當給定足夠的測量的資料量又會造成一次調整 Threshold 值得時間太久而不實用。

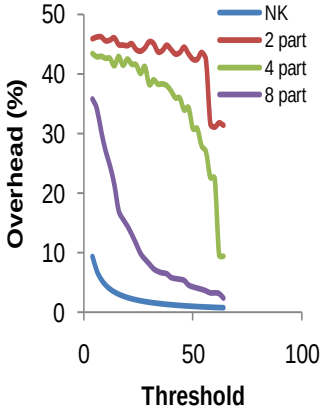
最後是 Partition 的測量的結果。



圖(18.a)BC



圖(18.b)FAST



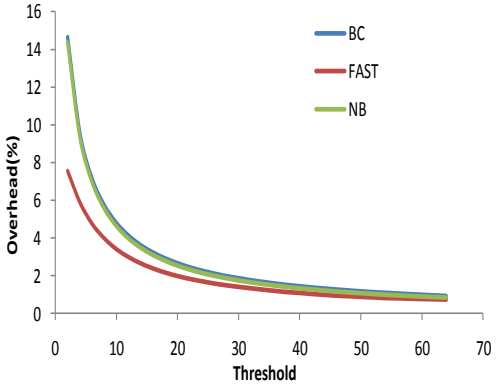
圖(18.c) NK

也依舊可以看出上一節所說的，若我們將快閃記憶體分成越多部分分別測量 Threshold 值，則越容易受到 workload 的 locality 的影響，而且切成越多部分也代表著用來測量的資料量越少，所以 Partition 的結果也呈現出將快閃記憶體切成越多部分就和暴力法的結果相差越遠的趨勢。

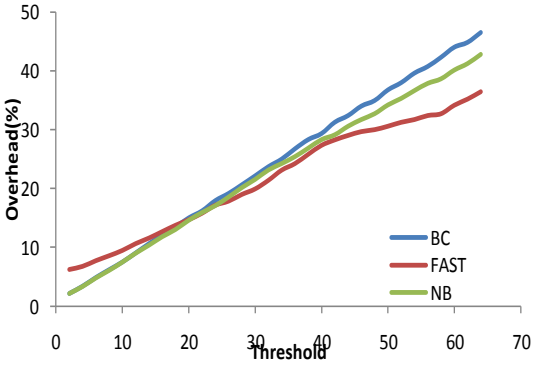
再來是我們使用 K-estimation 在不同 FTL 下所得到的 Threshold 值，見表(9)。以及用來對照用的使用暴力法測量 BC，FAST，和 NK 所測量到的成本-TH 關係圖以及標準差-TH 關係圖，見圖(19)。

FTL	K	Threshold
BC	1.1468	23
FAST	0.7536	19
NK	0.9826	22

表(9)不同 FTL 的 K 值及 TH



圖(19.a)不同 FTL 的成本-TH 關係圖



圖(19.b)不同 FTL 的標準差-TH 關係圖

從圖(19.a)來看的話，可以看到在 BC 和 NK 這兩種 FTL，因為他們結構很相似，所以我們使用 K-estimation 所選出來的 K 值和 Threshold 值也很相近。

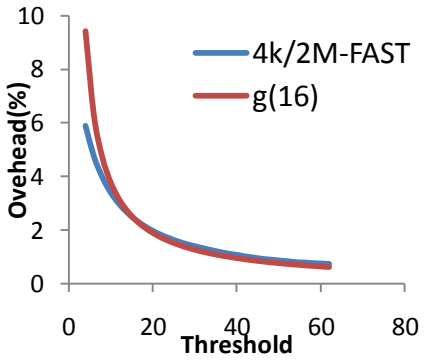
至於 FAST 的差別較大，我們可以明顯看出他的 wear leveling 的成本是低於其他兩者的。會造成如此現象的原因，是因為前面所提到的， FAST 的 block 被 erase 的速率是比 BC 還慢了許多，平均 erase count 也會較小。因此我們在同樣環境之下，若使用相同的 Threshold 值，則 FAST 所引發的 wear leveling 比例必定會比較少，所以成本也會比較低。另外 FAST 的 Garbage Collection 的效果比較好，讓 block 間的 erase count 分佈本身就比較平坦，因此會讓 wear leveling 的執行比較有效率，overhead 也較低。

因此綜合以上的觀察，我們知道 FAST 可以啟動 wear leveling 但比較不會引起過高的 overhead，因此 K estimation 所決定出的 K 值較小，選出的 Threshold 值也較小，可以讓 FAST 積極一點的啟動 wear leveling。

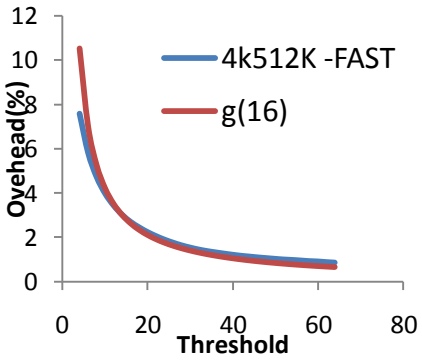
4.4 Using Different Flash Geometry

此節我們將針對不同的 Flash Geometry 來觀察它對 wear leveling 效果的影響。下面的圖分別為 K-estimation，Stepping，Partition 使用不同的 Flash geometry 的實驗結果， Page/Block 的設定分別為 4KB/512KB 以及 4KB/2MB。4k/2M 的設定亦可以視為 page 大小為 4KB，block 大小為 512KB 且具有 4 個非獨立 channel 的 NAND flash memory，另外在此節使用的 FTL 為 FAST。

K-estimation

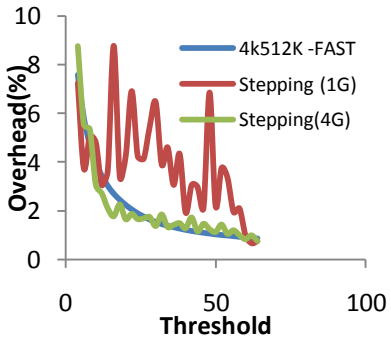


圖(20.a)4KB/512KB

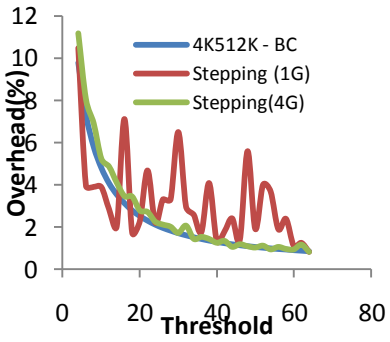


圖(20.b)4KB/2MB

Stepping

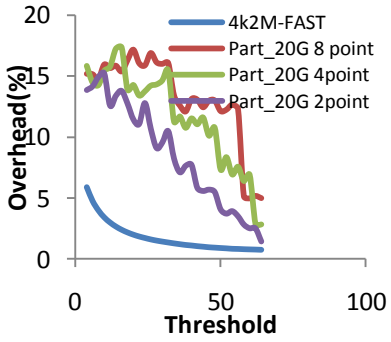


圖(21.a)4KB/512KB

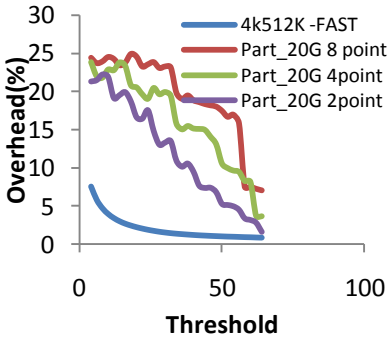


圖(21.b)4KB/2MB

Partition



圖(22.a)4KB/512KB



圖(22.b)4KB/2MB

從實驗結果的圖中來看，Stepping 不論是在哪一種 Flash geometry 下，當測量的資料量給的越少時，會造成曲線越動盪而失準。所以 Stepping -1G 幾乎都只能看出趨勢但卻不準確，而 Stepping - 4G 則測量時間太久。而 Partition 則會因為 workload 本身具有 locality 的問題，所以把快閃記憶體切成越多部分就越會受到這個因素影響，且測量的資料量也越少，所以就越來越不準。

而 K-estimation 跟上面兩個方法比起來的話，模擬出的成本-TH 關係曲線就接近多了，唯一的缺點就是會將 Threshold 值較小的成本估計的較高一些，但是其實我們很少會使用到這麼小的 Threshold 值，因為他會帶來太大的成本。我們可以在從下表(10)更詳細的看到 K-estimation 所造成的估計上的差異，其實結果並不會相差太大，而且若我們將 Threshold 極小時所造成的差異不計的話，和平方和的差距幾乎都降到 3 以下，其實模擬出的曲線還是相當準確的。

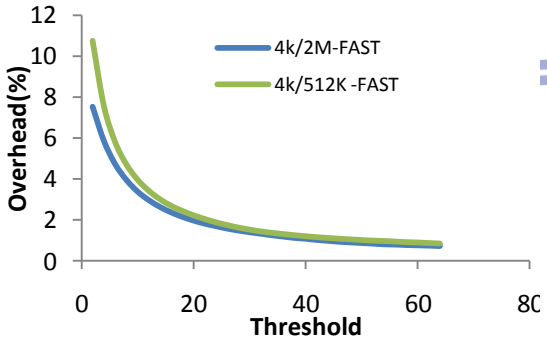
Geometry	RSS
4KB/512KB	11.14
4KB/2MB	15.95

表(10)不同 geometry 的 RSS 值

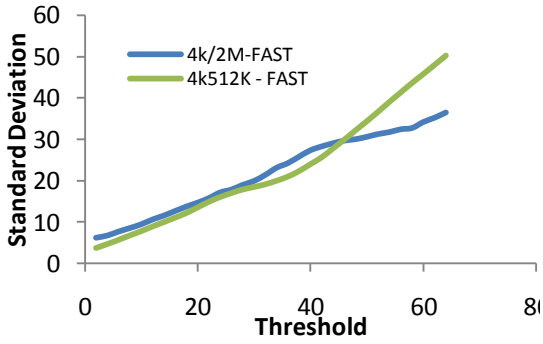
再來是我們使用 K-estimation 在不同 Flash geometry 下所得到的 Threshold 值，見表(11)。以及使用暴力法測量不同的 Flash geometry 所得到的成本-TH 關係圖以及標準差-TH 關係圖，見圖(23)。

Geometry	K	Threshold
4KB/2MB	0.7536	19
4KB/512KB	0.8424	21

表(11)不同 flash geometry 的 K 值和 TH



圖(23.a)不同 geometry 的成本-TH 關係圖

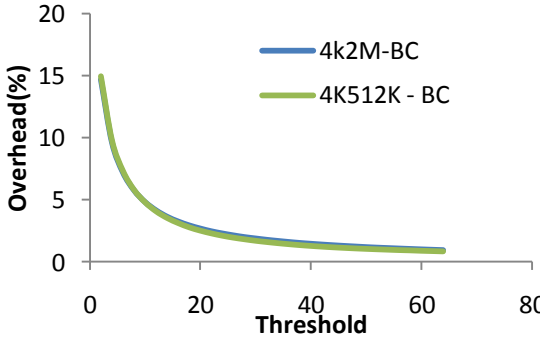


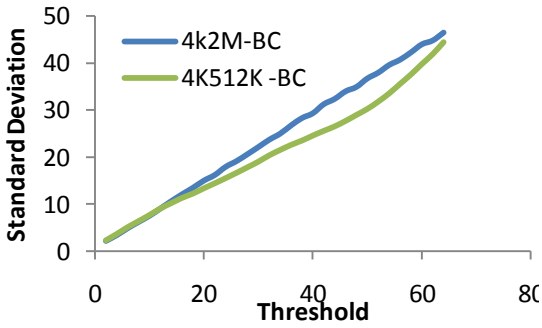
圖(23.b) 不同 geometry 標準差-TH 關係圖

從圖(23.a)中可以觀察出，block 的大小是會影響 wear leveling 的效果的，block 越大，則 wear leveling 的成本會比較低。原因在於在對一個 block size 為 2MB 的 NAND flash memory 做一次 erase 操作，都相當於一次對四個 block size 為 512KB 的 NAND flash memory 做 erase 操作，相形之下也會比較平均，因此 block size 較大會比較少引發 wear leveling。

當我們使用 K-estimation 測量 K 值的時候，若 block size 較大則 wear leveling 成本較低，不會引起非常大的 overhead，因此我們會測得比較小的 K 值，算出的 Threshold 值也較小，使 wear leveling 可以比較積極的引發。

另外我們再列出不同的 Flash Geometry 在 BC 下使用暴力法所測得的成本-TH 關係圖以及標準差-TH 關係圖，見下圖(24)。





圖(24.a)不同 geometry 的成本-TH 關係圖 使用 BC

圖(24.b) 不同 geometry 的標準差-TH 關係圖 使用 BC

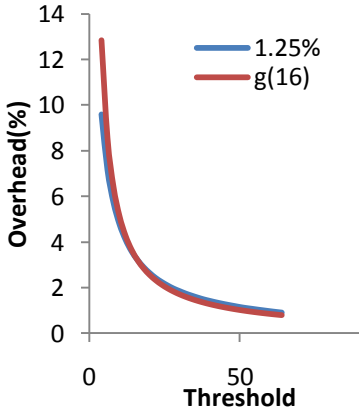
實驗結果，我們發現在 BC 下 flash geometry 的影響力降低了，從圖中 4kB/512kB 和 4KB/2MB 的測量結果幾乎是一樣的。造成這個現象的原因就在於 BC 的 block thrashing 現象。BC 因為他本身架構是一個 log block 只能對應到一個 data block 的更新資料，所以他會因為 log block 使用率較低而有 log block 不足的現象，而這種現象會因為 block size 越大變的越嚴重，因此會導致不停的引發 Garbage Collection，抵銷了大 block size 比較平均的情況，才會讓 flash geometry 幾乎沒有影響力。

4.5 Using Different Over –Provision Ratio

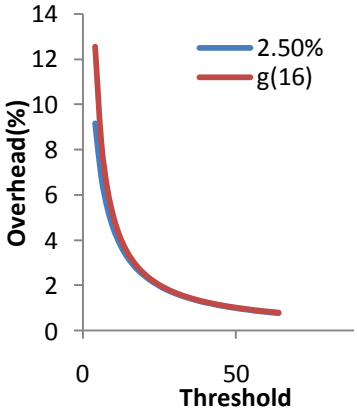
最後本節我們將針對不同的 over-provision 比例來觀察它對 wear leveling 效果的影響，我們選用的 FTL 為 BC，實驗中設定拿來當 log block 的空間分別為 0.25G，0.5G 以及 1G，也就是 over-provision 的比例分別為 1.25%，2.5%，5%。

下面的圖分別為 K-estimation，Stepping，Partition 使用不同的 over-provision 做出的模擬成本-TH 關係圖。

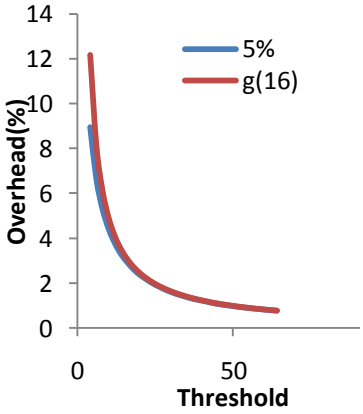
K-estimation



圖(25.a)1.25%

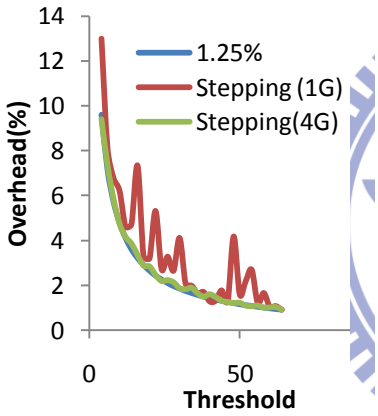


圖(25.b)2.5%

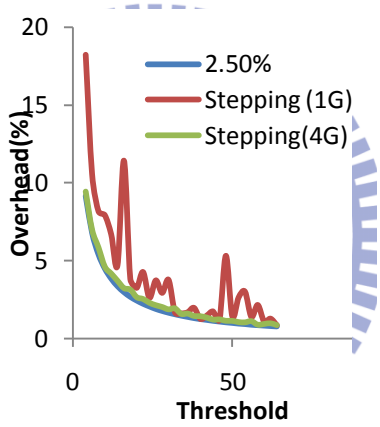


圖(25.c) 5%

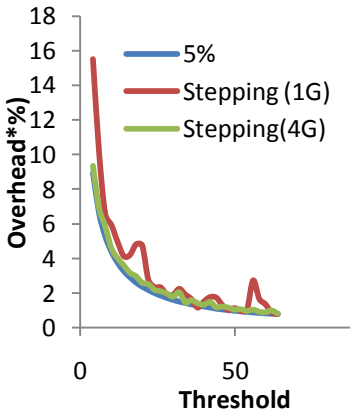
Stepping



圖(26.a)1.25%

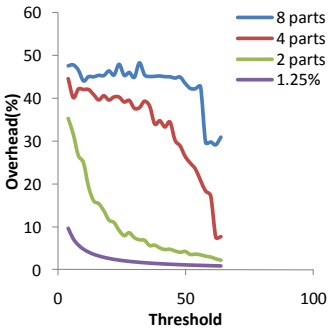


圖(26.b)2.5%

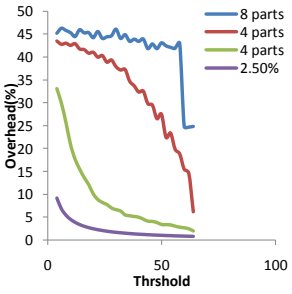


圖(26.c)5%

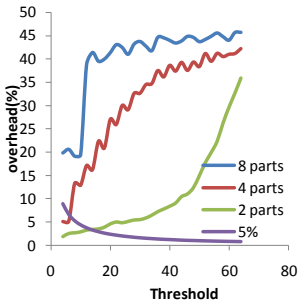
Partition



圖(27.a)1.25%



圖(27.b)2.5%



圖(27.c)5%

我們從上圖看到，不論是哪一種方法的實驗結果，都可看出 over-provision 的影響幾乎都很小。而 K-estimation 與暴力法的曲線在直接觀察圖表下，也比大部分的對照組，除了測量時間超久而不實用的 Stepping - 4G，都還接近暴力法的曲線。

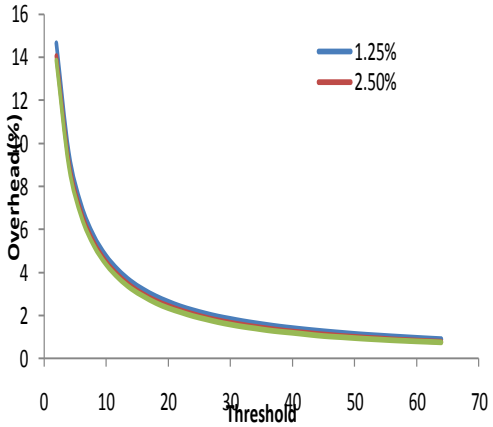
K-estimation 和暴力法曲線的 RSS 如下表(12)所示

Over provision ratio	RSS
1.25%	13.4399
2.5%	15.3926
5%	13.64

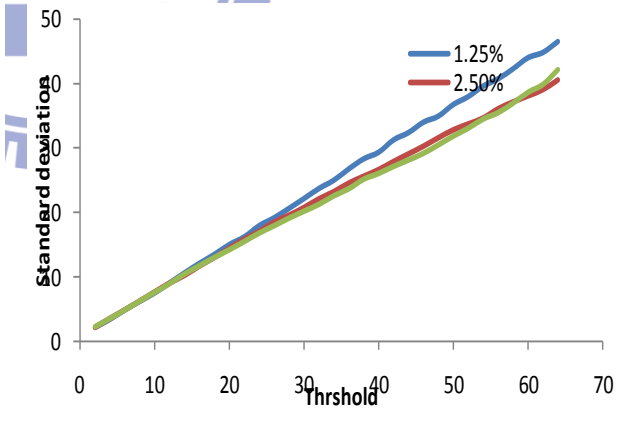
表(12)不同的 over-provision 比例的 RSS 值

造成誤差的部分主要也都在 Threshold 值極小的時候，在 Threshold 值為 4 時所造成的誤差佔了平方和的 75%以上，但是用該 Threshold 值所花費的成本其實已經太高了，我們一般並不會使用到這麼小的 Threshold 值，所以這部份的誤差並不會影響 K-estimation 的結果。

下圖(28)是我們使用暴力法所測量出來的成本和標準差與 Threshold 的對照圖，以及表(13)使用 K-estimation 所求出的 K 值和 Threshold。



圖(28.a)不同 over-provision 成本-TH 關係圖



圖(28.b)不同 over-provision 標準差-TH 關係圖

over-provision ratio	K	Threshold
1.25%	1.02628	23
2.5%	1.00508	22
5%	0.9732	22

表(13)不同 over-provision 比例的 K 值和 TH

由圖(28.a)來看，不管 over-provision 的比例為多少，所呈現出來的曲線幾乎都是一樣的，雖然 over-provision 比例較低時，spare block 的數量較少，會造成 Garbage Collection 的活動會比較活躍，加重了區塊間磨損不均的程度。但是 Workload locality 並沒有改變，over-provision 比例的高低並不會影響 wear leveling 去找 cold data 冷卻 old block 的效果，所以 wear leveling 的成本並沒有差太多。對照我們使用 K-estimation 所所測得的 K 值也都非常接近，建議的 Threshold 值也都差不多，見上表(13)。

4.6 Discussion

總結上述的實驗結果，我們將影響 wear leveling 的因素依照其影響力來排序，而影響力最大的為 workload，由實驗的觀察中我們可以得知，當 workload 沒有明顯的 temporal locality 時，例如實驗中 IOmeter，因為本身並沒有很明顯的 hot cold data 的區別，因此 要設定比較高的 Threshold 值來避免 wear leveling 太過於積極的介入，造成過高的成本耗費。

再來次要的影響因素是 FTL，根據實驗結果中的 FAST 和 BC 的比較，可以總結出以下結論。Garbage Collection 較有效率的 FTL 會讓 wear leveling 執行起來比較有效率，耗費的成本比較低。而 FAST 就具備此條件，因此我們可以將 FAST 的 Threshold 設定的比較小一點，可以使其達到非常好的效果也不會耗費太大的成本。

至於 wear leveling 受到 Flash Geometry 的影響就比較小，僅限於在區塊使用率比較高的 FTL，如 FAST。在 FAST 下，區塊越大會使區塊間較平均，wear leveling 成本較小，因此可以使用較小的 wear leveling 值來增加 wear leveling 的效果。而 over-provision 的比例對於 wear leveling 就幾乎沒有影響 ，所以我們就可以知道，我們不需要針對這個因素為 Threshold 值多做調整，節省了測量的時間。

5 Conclusion

快閃記憶體有 endurance 的問題，為了要使各個區塊被磨損的次數能夠平均，進而延長快閃記憶體的壽命，所以 wear leveling 是不可缺少的，怎樣的情況算是平均，多數的 wear leveling 多會有一個 Threshold 值來定義。根據實驗觀察，wear leveling 的效果會因為不同的 workload，FTL，flash geometry 影響，由其 workload 有可能隨著時間變動，要有隨時自我調整 Threshold 值的機制才能將 wear leveling 的效果發揮到最好。

本篇 paper 提出一個能夠兼顧時間和效能的 wear leveling 自我調整機制，推導出 wear leveling 的成本和 Threshold 的關係函數 $g(\Delta) = K/2\,\Delta$，所以只需實際測量 K 值即可估計出全部的 Threshold 和其對應的成本，並且利用此關係曲線來決定理想的 Threshold 值，隨時將 Threshold 值控制在成本爆增之前，以避免 wear leveling 帶來太多的額外成本。根據我們的實驗結果，本篇 paper 所提出的方法曲線都幾乎比其它方式更接近實際的成本-Threshold 關係，且需要實際測量的資料量也比其他方法都小，所以能夠真正做到 Threshold 值的即時調整。

在未來，我們還希望可以繼續改進 wear leveling 的分析模組，使其可以更準確的貼近實際的測量情形，改善 Threshold 值太小時成本會被估計的本較高的缺點。另外也希望對於各種會影響 wear leveling 的因素再增加更多的測試案例，以及繼續找出其他還會影響 wear leveling 的因素，使研究更臻於完整。

6 Reference

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

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

[3] 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

[4] 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

[5] Chien-Ting Huang.” A Low-Cost LBA-Based Wear Leveling Algorithm for Solid-State Disk”, National Chiao Tung University, 2008

[6]A. Birrell, M. Isard, C. Thacker, and T. Wobber. “A Design for High-Performance Flash Disks,“ *Operating Systems Review*,41(2):88–93, 2007

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

[8] Understanding the Flash Translation Layer (FTL) Speci_cation, http://developer.intel.com/. Technical report, Intel Corporation, Dec 1998.

[9] “Flash File System. US Patent 540,448,” in *Intel Corporation*.004.

[10] “FTL Logger Exchanging Data with FTL Systems,” Intel Corporation, Tech. Rep.

[11]]Jeong-Uk, K., J. Heeseung, et al. “A superblock-based flash translation layer for NAND flash memory,” Proceedings of the 6th ACM \& IEEE International conference on Embedded software, Seoul, Korea, 2006

[12]Chanik, P., C. Wonmoon, et al. "A reconfigurable FTL (flash translation layer) architecture for NAND flash-based applications," ACM Trans. Embed. Comput. Syst. **7**(4): 1-23, Korea, 2006

[13] Sang-Won, L., P. Dong-Joo, et al. "A log buffer-based flash translation layer using fully-associative sector translation," ACM Trans. Embed. Comput. Syst. **6**(3): 18, Korea, 2007

[14] Chanik, P., C. Wonmoon, et al. "A reconfigurable FTL (flash translation layer) architecture for NAND flash-based applications," ACM Trans. Embed. Comput. Syst. **7**(4): 1-23, Korea, 2008

[15] A. Ban. “Wear leveling of static areas in _ash memory,” US Patent 6,732,221. *M-systems*, 2004

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

[17] IOmeter Project. “Iometer” ,OPEN SOURCE DEVELOPMENT LAB. 2004. www.iometer.org/.