國立交通大學

資訊學院資訊科技(IT)產業研發碩士班

碩 士 論 文

增強快閃儲存記憶體壽命的動態 sector 重新映對機制

Dynamic Sector Remapping for Flash-Storage Lifetime Enhancement

研 究 生：廖秀芬

指導教授：張立平　教授

中 華 民 國 九 十 八 年 九 月

增強快閃儲存記憶體壽命的動態 sector 重新映對機制

Dynamic Sector Remapping for Flash-Storage Lifetime Enhancement

研 究 生： 廖秀芬　　Student： Hsiu-Fen Liao

指導教授： 張立平　　Advisor： Li-Ping Chang

國 立 交 通 大 學

資訊科學資訊科技(IT)產業研發碩士班

碩 士 論 文

A Thesis

Submitted to College of Computer Science

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master

in

Industrial Technology R & D Master Program on

Computer Science and Engineering

Sept. 2009

Hsinchu, Taiwan, Republic of China

中華民國九十八年九月

增強快閃儲存記憶體壽命的動態 sector 重新映對機制

學生：廖秀芬　　指導教授： 張立平 博士

國立交通大學資訊學院產業研發碩士班

摘　要

近年在 NAND Flash 技術的急速發展，無論在容量上、穩定度，都有長足的進步，逐漸成為 Embedded System 上儲存裝置的主要原件，然而 NAND Flash 發展至今，仍存在著 wear-out 的情況，因其都有一個磨損的極限，因此，平均磨損(wear-leveling)演算法便十分的重要。為了使其壽命延長，因此發展出各種 wear-leveling 的演算法，期望透過平均磨損機制，提高 NAND Flash 的使用壽命。由於一般 wear-leveling 必須跟 disk emulation (FAL or FTL) 緊密結合，在出產時便已經內建在 block device emulation, 一般來講可能是不能修改的 driver 或者是已經是 SSD firmware。本論文使用 Windows CE 平台，在此架構下，提出一個稱之為 RLFAL 的檔案系統架構，透過本論文所提出之 RLFAL，可以在不更動原始 FAL 及廠商所提供的 NAND Flash Device Driver 的前提下，使原有系統上具備 wear-leveling 機制，並可適用於市面上各廠商所開發之 NAND flash。本論文與 RLFAL 架構中實作的平均磨損演算法(wear-leveling)包括透過 LRU、FIFO queue 機制，協助 hot、cold data 的 remap 機制。透過本論文所提出的 RLFAL 檔案系統架構不但可以不須更動 source code，又可使原有 NAND flash 檔案系統具備更好 wear-leveling 的效果。實驗證明，使用 RLFAL 的系統與原本的 FAL 相比較，其 wear-leveling 的效果遠高於原本的 FAL。

關鍵字：Wear-leveling, Windows CE, FAL, erase

Dynamic Sector Remapping for Flash-Storage Lifetime Enhancement

Student: Hsiu-Fen Liao Advisor: Dr. Li-Pin Chang

Industrial Technology R & D Master Program of
Computer Science College
National Chiao Tung University

ABSTRACT

As the flash technology dramatically develop, improving the capacity and stability of flash memory. Now flash memory is the key component of embedded system storage system. However the wear-out situation caused by the lifetime is still the major problem of flash memory. Wear-leveling algorithm is important for every flash system design. For extend the flash memory lifetime, there are many related work proposed. By these algorithms, it can improve the flash memory lifetime. WinCE is current main stream platform on the marketing. However it has higher software authorize cost is known. This thesis focus on the problem of WinCE software authorize cost. We propose a file system architecture called RLFAL. By using RLFAL make it possible to improve the wear-leveling effect on the original WinCE platform without modifying any original flash related architecture. RLFAL is consistent of three main part, LRU table, FIFO Queue and remap policy. By simulation we improve that RLFAL can reduce the develop cost, and improving the wear-leveling effect of original platform.

Keywords: wear-leveling, Windows CE, FAL, erase

誌　　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。

再來，感謝在這段期間所有 Lab 的學長、同學及學弟們的幫忙，使得我能順利走過這兩年。Fox 跟 JJJ 可以如果不要合體話說來氣我就好了，毒嘴 Ting 不要毒嘴，電鍋跟大師，不要搞 Gay，感謝明毅及宥全討論上的幫忙，當然黃義勛學弟和黃莉君學妹當然也不能忘記，你/妳們的幫忙及搞笑我銘記在心。

研究口試期間，感謝謝仁偉老師和陳雅淑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

圖目錄

表目錄

1 Introduction

由於 NAND flash 具有抗震性、非揮發性(Non-volatile)以及省電等等特性，適合使用於 PDA、GPS 以及手機等需要大量儲存空間的攜帶式設備，由於單位面積儲存容量相較於 NOR flash 大，成本也便宜許多，所以目前市面上的隨身碟、固態硬碟(SSD：Solid State Disk)等等裝置， 多採用 NAND flash 為其內部儲存元件。時至今日， 隨著 NAND flash 的低成本、高容量、省電特性，應用於 SSD 設備，使得 SSD 成為未來取代傳統硬碟的首選。

由於 NAND flash 的硬體結構特性，必須要採取 outplace-update 的方式作更新動作，要在原來的位置更新，需要先對其作 erase 動作後，才可以作寫入動作。當 flash 中充滿了不要的資料或是沒有可用的空間，此時便需要執行 garbage collection 動作，以清出更多的空間以供使用。然而 flash 元件上的儲存單元(block)使用壽命有限，有抹除次數限制，當 flash 中的 block 到達抹除壽命時，此 block 將會因過度磨損而無法再使用；因此，為了延長 flash 的使用壽命，就必須要讓 flash 中所有的 block 都能被均勻的使用，以減緩 wear-out 的情況發生，而此平均磨損技術則稱之為 wear-leveling。

在目前已被提出的 wear-leveling 演算法中，大多是以實作於 FAL 或是 driver 層，直接控制 physical sector 以使其磨損較平均。限制 sector 在寫入次數到達額定的 threshold 後，將原來要寫的位置轉存至其它的寫入次數較少之 sector，演變為 hot data 及 cold data 的手法，其作法是將 hot data 寫到 cold data 的 sector，使原先存放 hot data 的 sector 不致因一直在相同的地方寫入，造成嚴重的磨損，並利用較少被使用的 sector 分攤寫入次數。

由於市場上各家 flash 廠商在實作於 FAL 操作行為及其實際上的處理方式皆不相同，若是想要一一根據其特性對其作最佳化動化，以達到更好的 wear-leveling 效果，相當費時費力。本篇論文於 Windows CE 平台上實作 wear-leveling 機制，在原始的 FAL 加入一層 RLFAL，利用簡單的方式，不需要了解原本 FAL 的運作即可以得到較好的 wear-leveling 結果。由於 Windows CE 是一個需要支付軟體授用費的嵌入式作業系統，在 FAL 與 OS 中間這一層做 wear-leveling，不需要更動 FAL 的 source code，也不會更動

到 Windows CE 的 OS，可以節省程式碼授權費用，又可達到更好 wear-leveling 的效果。

本論文實驗部分透過程式模擬，實作本論文所提出之 RLFAL 架構，與原本之 FAL 之效用作比較。RLFAL 主要目的是幫助 FAL 之 wear-leveling，期望可以使其 erase 次數更為平均。

本篇論文的結構如下：第二章為簡述相關文獻及描述問題，第三章為說 RLFAL 的主要架構及方法的論述，第四章介紹實驗所使用的 Windows CE emulator 及相關實驗參數，第五章為實驗數據，透過 Windows CE 所提供的 emulator 作為本實驗的模擬平台，最後第六章為結論。

2 Motivation

由於有些 device 在出廠時就把所有的東西都包起來，所以，想要更改也不能動。因此希望利用 sector remaping 的方式可以直接或間接的操控 garbage collection，當需要對其內部原始碼作修改時，也可以不用去更動到原始碼即可得到更好的 wear-leveling 的效果。當系統開發廠商，希望對其產品所使用的 flash 儲存媒體，導入不同的 weal-leveling 演算法時，勢必得要更改其 flash driver 相關的內部原始碼，也因為會產生軟體授權金的成本。有鑑於此，本論文希望透過軟體架構的設計，提出一個得以在原有系統上，導入不同的 weal-leveling 演算法，且不需要對原有的原始相對修改的方法。本章同時說明 flash memory 的特性及目前已被提出的相關 wear-leveling 演算法，以及在實作上相關議題。

2.1 Overview of Flash Memory Characteristics

Flash 會劃分多個 block，而一個 block 再區分成多個 sector，如圖 2.1 所示。被寫過的 sector 必須先行 erase 後，才能再次被寫入， flash 具有以下特性：write-once、block erase、有限的 erase 次數。在更新資料時 sector 必須要以 outplace-update 方式作更新動作，想要在原來的位置更新，需要先執行 erase 動作，才可以對此作寫入動作。當 flash 中充滿無用的資料或是沒有可用的空間，此時便需要執行 garbage collection 機制，清出可用的可用空間。在 erase block 的過程中，是以 block 中的 dirty sector 數最多的 block 優先 erase。

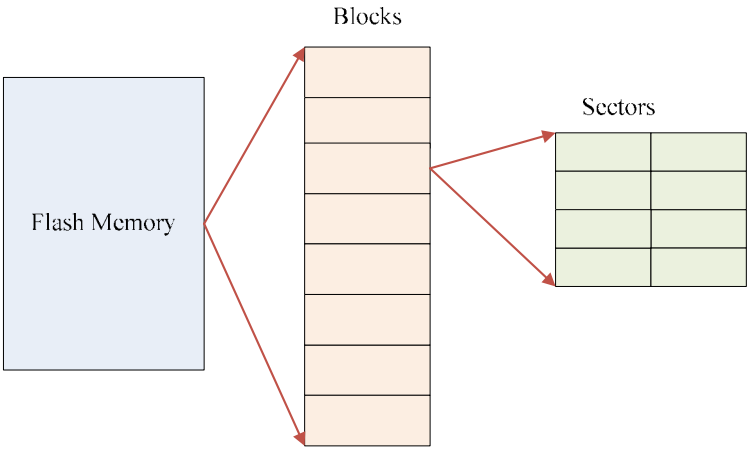


圖 2.1: Flash 內部結構示意圖

2.2 Wear-leveling

由於 flash memory 中每一個 block 都有抹除的限制，經常抹除到固定的區塊，容易使該 block 損毀無法使用，一旦可用 block 數量變少，系統內部執行 garbage collection 的頻率將會提高，導致加速其他 block 的磨損，因此藉由平均分散每一個 block 被抹除的次數，延長第一個損毀 block 的出現，進而改善 flash memory 使用壽命的方法即稱之為 wear leveling。

Block 上存放的資料可區分為兩種類型：時常更新的資料(hot data)，與不常更新的資料(cold data)。由於系統中 garbage collector(GC)執行通常以回收最大可用空間為原則(greedy)，傾向回收擁有最多 dirty sector 的 block，而存放 hot data 的 block 由於資料較常被更新，所以 dirty sector 的數量相較於其他區塊來的多，而 cold data 因為不常更動，dirty sector 數量較少，所以在執行 GC 時容易回收到 hot block，導致 hot data 將來回存放到特定的幾個 block，造成這些 block 磨損情形遠比其它 block 嚴重， wear leveling 演算法的主要目標就是要避免如此情形的發生，適時的將 cold data 存放於 hot block 以降低 hot block 被抹除的頻率，平衡整體 block 的使用情形。

目前已有許多關於 wear leveling 的研究。Wear-leveling 設計的好壞，通常以三個標準來估評：Effective、overhead 以及所耗用的 RAM space。

2.3 Problem Definition

本篇論文是以在 FAL 之上再加一層軟體層稱之為 RLFAL，以控制下層對 flash 某些 sector 不會作過多寫入，進而造成對某些 block erase 次數過多而磨損不平均的情況產生。此外，wear leveling 可實作在 translation layer 中與 garbage collector 協同運作，但這樣的作法需要修改 translation layer 的 source code。在某些作業系統中，translation layer 的 source code 是不公開的，亦或是需要授權才能作修改的，所以提高了實作 wear leveling 的技術門檻及成本，如本論文的實作平台 Windows CE 即是屬於不公開 translation layer source

code 的作業系統。在實作上面臨到的一些問題，例如：(1)無法控制在 FAL 之後的寫入之後，也就是說，在 remap 過後，交由 FAL 進行寫入，裡面所選擇的要寫入的 logical sector 實際是寫入到那一個 physical sector，在新加上的軟體層是不會去干涉的。(2)garbage collection 的動作亦無法控制，言下之意是指，它會選到那一個 block 去 erase，是無法決定的。

在這一層定義的 hot data 寫入到置放 cold data 的 sector，利用 remap 的動作，希望讓 cold sector 有機會再被使用到，但是這一層都是做 logical to logical 的方式，希望藉由在上層就做 hot sector、cold sector swapping 亦可達到相同的結果。

2.4 Related Works

目前已提出的 wear-leveling 演算法研究中，針對可不修改既有 translation layer source code 的方法，實作 wear leveling 機制，有 A Stackable Wear-Leveling Module[2]及 Unsorted Block Images [3]。其中[2]的作者於 Linux 平台中，提出一個模組化的軟體架構，透過 Linux 作業系統的模組方式，實作一個可以動態掛載的 weal-leveling 軟體模組，藉由此模組的掛載，可以協助系統上原有之 flash drive，提高對 flash 的 wear-leveling 效果，作者除了提出此模組設計外，在其 wear-leveling 演算法的部份，配合作者所提出之 Dual-pool wear-leveling[2]演算法。而在[2]中，作者提出了一個適用於 Linux 作業系統的平台，作者考量到 flash memory 在嵌入式作業系統的應用環境，提出需要針作 flash 的 partition 情況納入考量，針對開機所需要 boot loader 所在的 block，列入實作相關 flash 機制的考慮，並透過於各個 sector 中加入所設計之管理表頭(header)，以達到對各 sector 更為準確的管理及應用，然而作者提到，由於管理機制的設計，為了提高正確性，在開機時，必須完整掃描所有 sector，將會降低開機效率。

提出的方法主要目的為輔助既有磨損平均演算法，降低其執行成本。目前已有許多研究磨損平均的演算法，如在[1]中的磨損平均演算法主要是依據 Block Erase Count 將 Block 區分成 Hot Pool 與 Cold Pool 的集合，並動態的交換 Hot Block 與 Cold Block，避免經常抹除 Hot Block。

在[4]中所提的方法為利用一個 Bitmap 記錄區塊是否被抹除與使用一個計數器紀錄整體區塊被抹除的總數，一但區塊被抹除即標記對應的 Bit 為 1，反之為 0，若計數器除以 Bitmap 中 Bit 為 1 的總數大於 Threshold 則開始執行區塊交換。

本論文有別於上述兩作者所提出的方法，提出一個 Windows CE 平台上適用的機制，在 OS 與 translation layer 間新增 wear leveling 模組，藉此達到不修改 Translation Layer 的前提下，透過 re-mapping 的技巧，使 wear leveling 整合至系統中。

3 Design Implementation of RLFAL

3.1 Overview

第三章詳細介紹本論文所提出的 Remap Layer FAL(RLFAL)架構的設計概念。RLFAL 主要由三個主要模組所組成，一、LRU table，二、FIFO queue，三，Remap policy。首先透過使用一個 LRU table 來紀錄目前於 flash 中的 hot data 所置放的 sector，利用 FIFO queue 協助 LRU table 過濾短時間內大量循序寫入的 sector 及防止 LRU table 發生 table thrashing 現象，利用 queue 先進先出的特性，辨別那些 sector 為短時間內二度被使用到的 sector，其為 hot sector 的機率很高，所以該 sector 將會被存入 LRU table 中，亦利用此方式來避免 sequence sector 進入 table 之中。為了不讓 hot data 一直對同一塊 sector 寫入，利用 remap policy 來置換一個 sector 來替代，而此 sector 是利用 bitmap 來選擇，以期選出的 sector 是較少被使用到的 sector。

3.2 Hot Sector Identification

由於上層資料寫入的關係，有可能會造成某些 data 一直寫入到相同的 sector 上，因此，對於某些 sector 一直被寫入，造成該 block 過度磨損而發生資料遺失，而這些 sector 即稱之為 hot sector，而實作 wear-leveling 演算法的主要目的就是想要處理這些 sector。因為最主要的問題是出在那些 hot sector 被過度使用，因此如何正確判斷出那些是 hot sector，並對它做處理以避免是相當重要的一個問題。

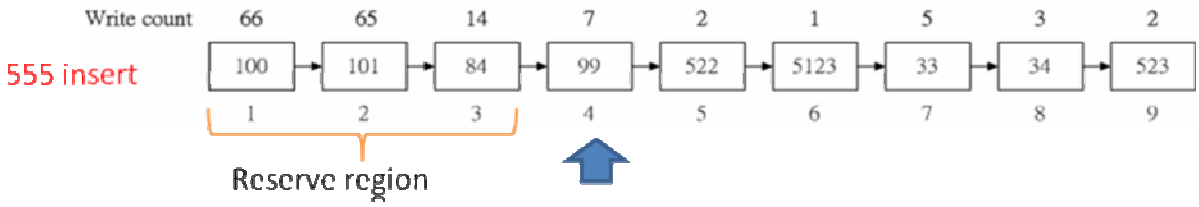
為了要區分出那些 sector 為 hot sector，所以利用了以下的機制來完成。使用一個 LRU table 來紀錄 hot sector 以便對它作處理，而分辨是否為 hot sector，則是利用 FIFO queue 來做過濾的動作，若是 sector 已存在 queue 中，而它再度進入了 queue 即認定其為 hot sector，便把它存入 LRU table 中。而 bitmap 則是紀錄了所有被使用過的 sector，當 sector

被寫入多次後，便透過 bitmap 找一個未被使用過的 sector 來替換。

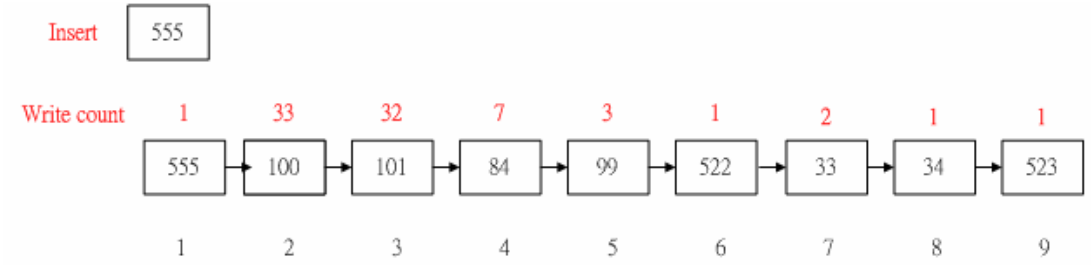
3.3 LRU table

透過採用 Least Recently Used(LRU)演算法作為 table 的管理方式，讓常被使用的 sector 被保留在 table 中，不常使用的 sector 被置換出 table 中。其目的是使 hot sector 被保留在 table，並且讓 hot sector 去置換 cold sector 以達到 wear-leveling 的功效，使 cold sector 也參與磨損，以達到平均磨損的效果。在 table 中 sector 是使用紀錄 count 的值的方式來計算，以便知道何時該去找一個 cold sector 來置換。

由於 table 無法紀錄目前系統中所有的 hot sector，因此需要有一個 policy 去決定如何去選擇需被紀錄於 table 中的 sector。因此當 table full 時，就要選擇一個 sector 剔除，此 sector 被稱之為 victim sector，選擇被使用次數最少，且最靠近 table 尾端的 sector 為 victim。因此，採用最常見的 LRU，並且利用圖 3.1(a)來說明除了使用 LRU 的特性也加了一些限制來控制此 table 的其它行為。目前被用到 sector 必定存放於 table 的最前端。利用所紀錄的 count 數，被多次用到的 sector 得以在 table 中延長其停留時間，即使最近並未使用到。同時為了避免老舊的 sector 長期佔用 table 空間，利用 aging 的機制來確保 table 使用的公平性，避免 count 值較大的 sector 長時間佔據 table 空間，因其有雖然有較大的 count 值，但在短時時間內，並不會再次被使用。此外為了避免剛進入 table 的 sector 因其 count 值過小，將會立即被選為 victim sector，定義 table 的後 2/3 為可選擇剔除 table 的範圍。保護 table 前三分之一的 sectors，不會被選作 victim，此區域稱之為 Reserve Region。另外，在選擇替換的 sector 的時候，是以被 write 次數最少的 sector 做為選擇的對象，如此設計的原則是由於被 write 次數較少的 sector，可能是 cold data，所以 cold data 愈早被換出去愈好，降低其停留在 table 中的時間。由於某些 sector 被使用多次，使得 count 數很大，所以，即使在可被剔除的區域的選擇中，也很可能由於 count 數很大，而不會去選擇它。因此，使用 aging 機制讓先前是 hot sector，後續卻未再度被使用的 sector 可以被置換出 table，將其 count 數以二次方遞減。啟動 aging 的時機是在當 table 滿時，需要去刪除一個 table 中的 entry(sector)才發生的，其做法請參考圖 3.1(b)。



(a) LRU table



(b) Insert and Aging

圖 3.1: LRU table

3.4 FIFO queue

利用一個 queue 來紀錄及確認，最近被使用到且不在 table 中的 sector 是否要將此 sector 加入至 table 中，此 queue 會紀錄 sector ID。在短時間內使用到兩次的 sector，便認為此為 sector 為 hot sector，並將此 sector 存入 LRU table 中。以上這些行為，我們利用圖 3.2 來說明。透過 second chance 的方式提高 hot sector 的正確性。若是此 sector 已在這個 queue 中，但在 table 並無紀錄，便刪除舊的那筆 sector，並加入 sector 放至 queue 的 header，以免造成 queue 中有兩筆相同的 sector 的情況發生，最後，將此 sector id 紀錄到 table 中。其目的是令非 hot data 的 sector 儘可能的不要進入 table 中，避免 table thrashing 情況發生。

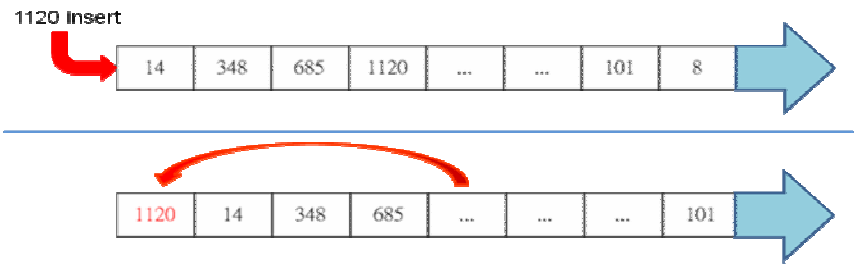


圖 3.2: FIFO queue

此外，queue size 不宜太大，以免過多的 sector 被紀錄於 queue 中，造成不是 hot sector 也增加進入 table 的機會，以致於 table 被 replace 的機會增加。過多的 sequence sector 進入 table 中，將會使 hot sector 被 flush 到可被選為替換出 LRU 的區域中，進而使得被置換的機率增加。若是被錯置換出 table 的 sector 為 hot data，將會降低 wear-leveling 的效果。

3.5 A Sector Remap Policy

現有的 mapping 方式有 block mapping 以及 page mapping，在此是使用 page mapping。因 hot data，基本上應該是一次的 request 應是小於 8 個 sector，其為 hot data 的可能性極高。若是以 request 做單位，對 FAL 的操作行為會相對的變複雜，諸如要對 fragments 的問題做處理、尋找整塊可用空間、maintain table 的設計…等都相對的麻煩。因此採用了 Page Mapping 的方法，降低實作複雜低。

為了要挑選 cold data 在選擇可用的 sector 是使用一個大小為 8K 的 bitmap 結構，來取 cold sector，以一個 32Mb 的 flash 為例，每個 block 由 8 個 sector 組成，每個 sector 為 512byte，所以需要使用 65536 bits 大小的 bitmap 結構作對映。總共有 32 個 entry，每個 entry 為 4byte。直接在透過 Bitmap 取出未被使用過的 sector 來做為可替代的 cold sector，以置換被過度寫入的 sector。在 bitmap 中，被使用過的 sector 設為 1，而沒被使用過的 sector 設為 0。其挑選的方式是從頭循序尋找被標記為 0 的 sector，選為 cold sector 來替換 hot sector 置換寫入，如圖 3.3 所示。會加上這個 bitmap 的原因是由於 LRU table 只能紀錄最近被使用到的 sector，沒有被紀錄到的 sector 將會被視為 cold sector；如此之下造成的誤判太多，使得一些被使用過的 sector，還是再一次的被選取做為 cold sector，這樣的 remap 效果只比原來沒有加上方法的 FAL 好一點，加上這個 policy 的誘因不令人認同，為了讓選取更為正確，我們使用了 bitmap 來代替去對映所有 sector 的任務，bitmap 雖然花了 8K，但是全部的方法所額外使用的空間，還是遠低於使用 full map 所佔去 ram memory 的空間，與 full map 相比約只有 3%。

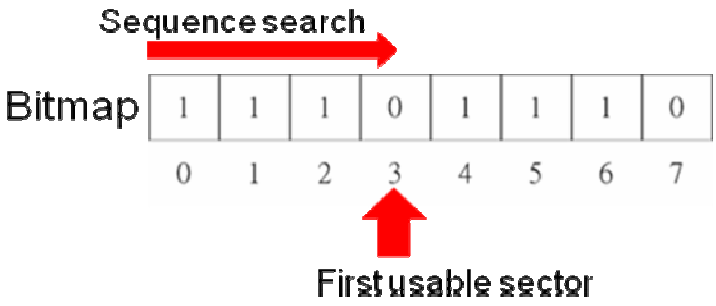
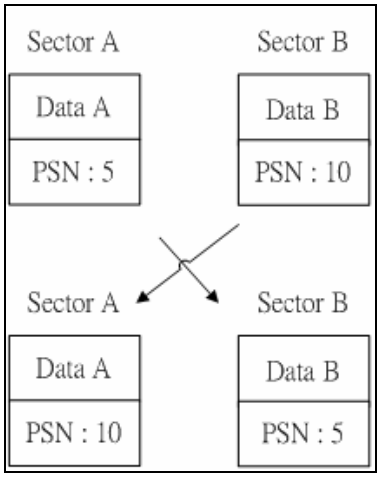


圖 3.3: Bitmap 示意圖

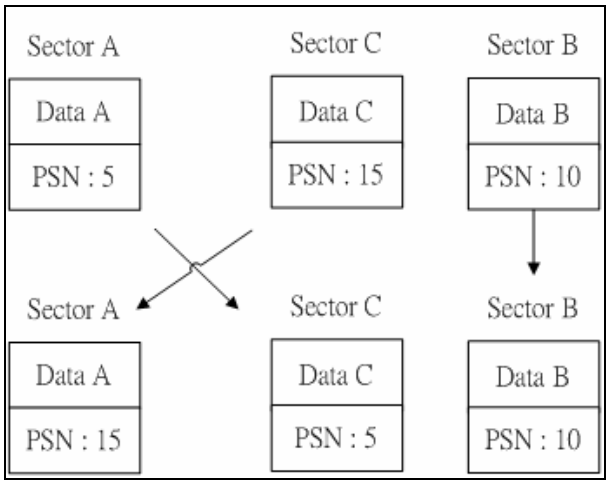
在 table 中主要的動作是去紀錄 hot data 在那個 sector，在 sector 被使用到所設定的 write 次數時，便去找一個可用的 sector 來替換。這裡所記載的 sector 都是 logical sector。以圖 3.4(a)為例：當第一次 count 數達到所設定的 threshold 時，去找一個可用的 sector 來替換，把被替換的 sector 的 data copy 至替換的 sector 上。

第二次之後達到 count 數時以 3.4(b)為例，就先將之前被替換的 sector release，置換回原先的位置，再換上選擇新的 sector。把將被拿來替換的 sector 的 data copy 至使其替換的 sector 上。

當 table 達到預設的 table 長度，當被選擇的 sector 的會寫回原來的位置；若是此 Sector 之前有取替換的 Sector，那麼被替換的 Sector 亦寫回原來的位置，以確保不被紀錄的資料的正確性。



(a): hot data 及 cold data 交換示意圖



(b): hot data 及 cold data 交換示意圖

圖 3.4: hot data 及 cold data 交換示意圖

3.6 Hot-Cold Mixture Avoidance

所謂的 hot、cold sector 的交雜的情況，是指由於 block 中含有 hot data 及 cold data，因此，block 會呈現圖 3.5 的情況，這會造成 garbage collection 在選擇要去 erase 的 block，會去選擇 block 中 dirty sector 數最多的 block 來 erase。而 dirty sector 是指 sector 中裝的是 invalid data；invalid data 往往是由於新的資料被更新至新的 sector 上，所以那個 sector 就會被視為 invalid。請注意 hot data 之所以被稱為 hot，就是它常常會去更新，因此，在 block 中若是 hot data 多，它的另一個表示就是，它極有可能也是代表 invalid data 多。而 cold data 由於更新的機會不多，因此在這裡也可變相的視為 valid data 的表示，但這不表示 cold data 就是指 valid data，當然，當 cold data 被更新時，原本寫入 cold data 的 sector 一樣會變成 invalid sector (dirty sector)，在這裡只是它的行為及型態會造成之後的情況，所以在此說明這兩者的不同性質。

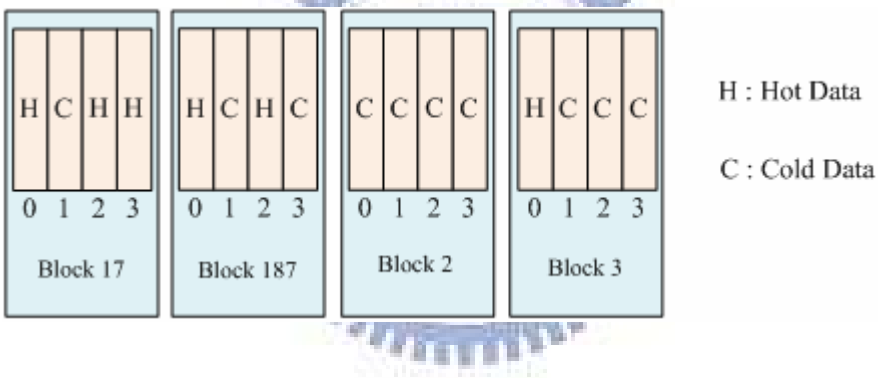


圖 3.5: hot-cold mixture

當 hot、cold data 在 block 中交雜情況太嚴重時，會造成 FAL 在做 Garbage Collection 時的效率不彰。這情形之下，其所選取到的 block 中的 sector 是 dirty 的情況有可能只有一半。意指，在 block 中的有效 sector 有一半或一半以上，如此在做 GC 時，就必須要先搬移有效資料再 erase block。此外，erase 後的可用空間也會相對的較少。因此，必須要再多選幾個 block 來 erase 才可以得到所需的可用空間。在這種情況下，不儘是 erase count 增加，而且 GC 的效率也不好。

在多次的實驗之下可得知，Windows CE 所提供的 FAL policy 是屬於 hot data 愈多愈好的情況愈好，但是在一般而言，在 Embedded System 的 device 中，多半是 hot、cold data

交錯的情況，另外，在 Embedded System 中 Device 所使用來儲存的資料，也多半是 cold data。只有少部分的是拿來存一些行事曆之類常更新的 hot data。因此在處理此類 hot、cold data 的情況是效果是不太好的。為此，提供一個可用的 policy 來使其 wear-leveling 的情況更好。

依現在的情況，若是加上了一次寫入多筆 sector 的方法，希望收集多筆 cold sector 再一次寫入，期望可以減少 hot、cold sector 在 block 中混雜的情況。但在實驗後發現，由於原本 garbage collection 的關係，所得到的 block 不一定都是空的，也會由於之前所 erase block 但是裡面卻有 valid sector，因此會需要先去把 valid sector copy 到空的 block 上再去 erase。而先成下例情形：

在選擇要 erase block 是 dirty count 多的 block 來 erase，如圖 3.6，會先 block 17 的去 erase 再取 block187 的去 erase，而從 free list 中取出的 free sector，也不能知道 sector 是否位於 block 的第一個位置，因此，在 erase 時會把 valid sector 的資料放到 free sector 上，再寫入其它正在更新的資料，情況會是當發生 block17 被 erase 時會先將 valid data 複製到 block 199 的數字為 3 的 sector 上。然而，它若是再 erase block 187 時，.也是要將 valid sector 的資料先複製到 free sector 上再 erase，所以，當下次要 erase block 199 時，它的可用空間只有二個，下次放到新的 sector 上時，又會佔掉了一些 block 的空間。而 hot、cold data 交雜情況，造成 block 需 erase 兩個才可以取得 4 個 free sector 的空間。但是以上的情形會造成 Invalid sector 過於分散，以致於所有的 block 的 dirty count 都差不多，因此 garbage collection 就必須要做多次才可以得到足夠的可用 sector。

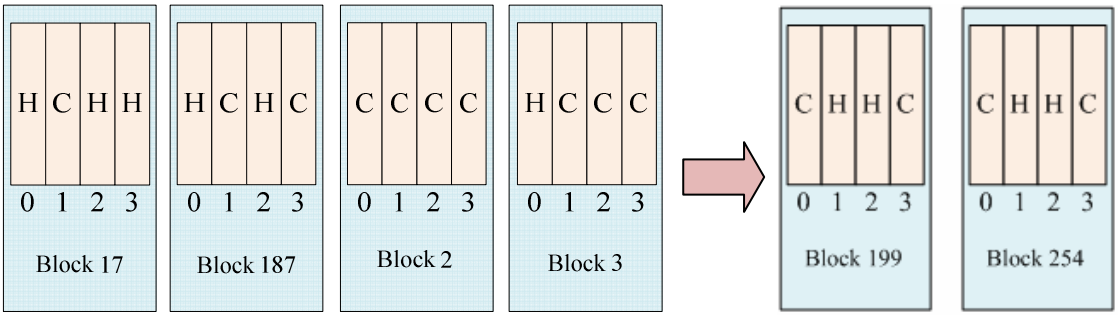


圖 3.6: 交雜影響 wear-leveling 之情況

4 Implementation Issues

本篇論文將新增 Remap Layer on FAL (RLFAL)與 Windows CE FAL 協同運作，以幫助 wear-leveling。(1)輔助 wear-leveling：將 hot、cold data 的儲存 sector 交換以平衡被寫的次數。(2)平均磨損：藉由互換 hot、cold sector 以 Trigger 一直存在 cold sector 的資料可以搬移到 hot sector 上。(3)簡單的作法加速 wear-leveling 的效果：可以在不修改 FAL 亦或去了解 FAL 就可以達到很好的效果，並且仍可使用在 Windows CE 的架構中。

在本章節是說明，如何在 Windows CE 建置一個 Device，並在其加上一個 RLFAL，讓它可以透過新加的 RLFAL，寫到原本的 Device 裡去。而 RLFAL 裡所有的 IO-Control 皆是透過 Windows CE 裡面原本的 FAL 寫入，未做任何更改。以原本的 FAL 與 RLFAL 模擬進行比較。

4.1 Architecture of Windows CE

圖 4.1 為現行 Windows CE 檔案系統架構。Flash Media Driver (FMD)可以視為是 Flash Device Driver，FMD 必須實作 FAL 所需要對 flash 的存取動作例如：初始化 Flash Device、對 block erase、對 sector 的 Read、Write…等動作。而 Flash Abstraction Layer (FAL)以 lib 的檔案格式存在於系統中，就是用來處理 File System 對 Flash 的操作的 Layer。

而在 Windows CE 中是如何執行 IO-Control 的，跟其系統的設計不太一樣的地方在於，除了初始化之外，其它的 Read、Write、GetInfo、Open……等，皆是由 DSK_IOControl 根據傳進來的 control code 再去做相應的檢查；例如對 Flash 取得其 Information(GetInfo)，要確認傳進來的 Buffer 以及 Size 是否正確，無誤之後再去做相應的動作。

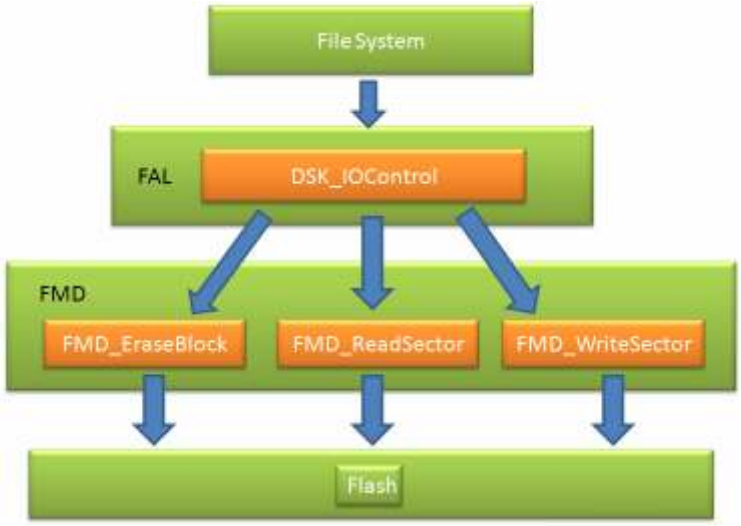


圖 4.1: WinCE 的檔案系統架構

本實驗 Windows CE 的架構：與新加一層 RLFAL 的架構如圖 4.2 所示，其它完全不改變。RLFAL 完全與 FAL 的對外呼叫一致，只有內部去修改 Mapping 的 sector，實際的 Read、Write、GC 完全是由原本的 FAL 去做，並未多加干涉。

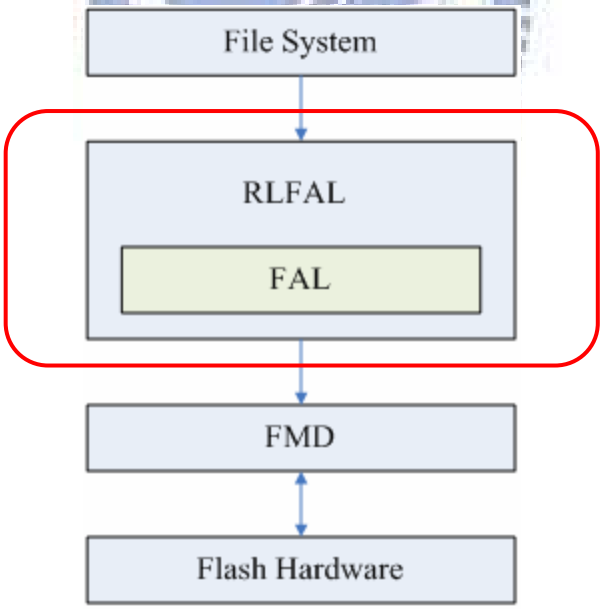


圖 4.2: WinCE 的實驗架構

4.2 RLFAL

在原本的 FMD 是 Link 到 FAL.lib 所以，要修改使其 Link 到 RLFAL，再讓 RLFAL

去 Link FAL 的 Library，讓所有的動作皆是原本的 FAL 完成。

在 Windows CE 去 Mount 一個 Device，是透過 Platform.reg(registry)讓 OS 知道有一個 Device 要 Register，再來透過規定的 Interface 來讓 OS 去使用這個 Device。FAL 及 FMD 皆是對面相同的 Public Interface；如此，OS 在對 FMD 下 Read/Write 時皆會透過 FAL 來進行寫入。本實驗所修改的部分只有 FMD Link Library，將其原本 Link FAL.lib 改為 Link RLFAL.lib，而 RLFAL 再去 Link FAL.lib，以便執行 IO 的動作。

圖 4.3 所示為在程式中所呈現的樣子而它實際上的執行動作及順序，以及 OS 所看到的情況。

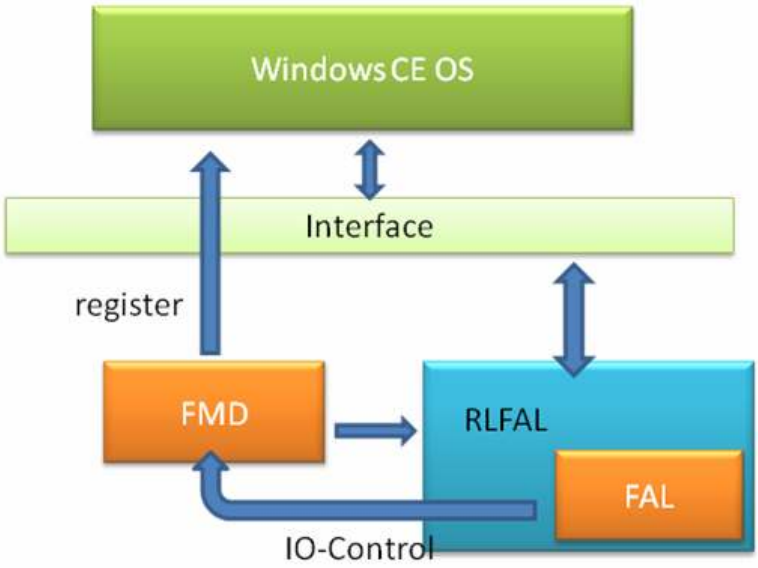


圖 4.3: 程式中呈現的架構

4.3 RLFAL Operation

在此處，OS 在對 FMD 下 Read/Write 時皆會透過 RLFAL 來進行在這一層 OS 下的 sector 皆在這一層去做 remap。此外，利用一個 L2L(logical to logical)的小型 table 去對映被 remap 過的 sector，使其可以找到對映的 sector，其它未多加處理，直接將傳進來的 sector 找到在 RLFAL 中相對映的 sector 傳給 FAL 處理。

圖 4.4 為本論文提出的 RLFAL 的架構圖，將其原本對 FAL 的寫入動作拆解成 sector Mapping，以利重新 Remap 的行為。當 Request 進來時會先去查詢是否在 table 中，此時會有兩種情況(1)若是在 table 中，就表示是 hot data，此時會先去 Check 寫入的 sector 是否已超過了設定的 Threshold，若有，便去找一個 cold sector 來替換寫入此 hot data。(2)若不在 table 中，便去 FIFO queue 找，若是在 queue 有找到，便代表它有可能是 hot data，因此將此 sector 紀錄到 table 中。(3)若兩者皆無，則將此 sector 放入 FIFO queue 中。

所有對 Flash 的動作，此層並不會直接去做讀寫，此層只對 sector 的 Mapping 做更動及對映，所有的 IO 還是去執行原本的 FAL 去做 Request 執行 IO 的動作。

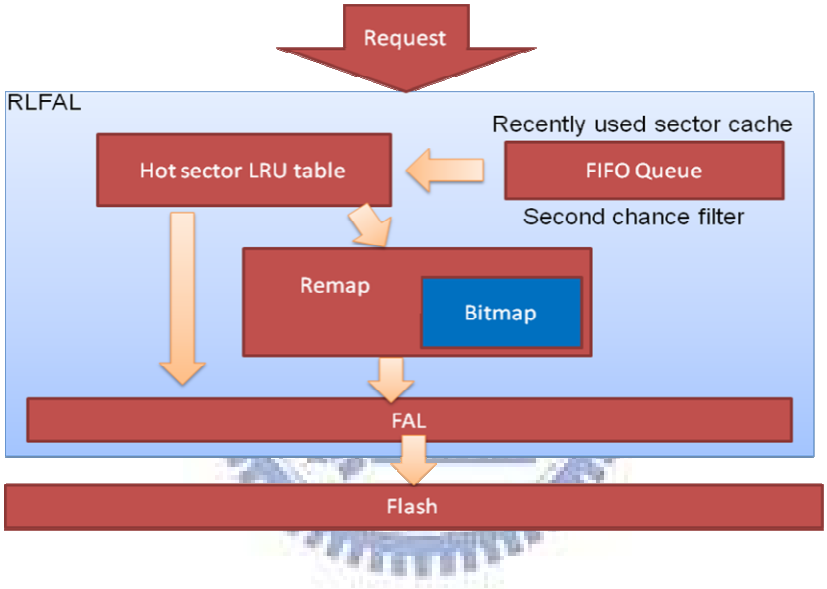


圖 4.4: RLFAL 的內部結構示意圖

在這裡用圖 4.6 來說明原先的 Mapping 的行為，之後再說明其在 remap 後它對底層的影響，和底層運動的情形

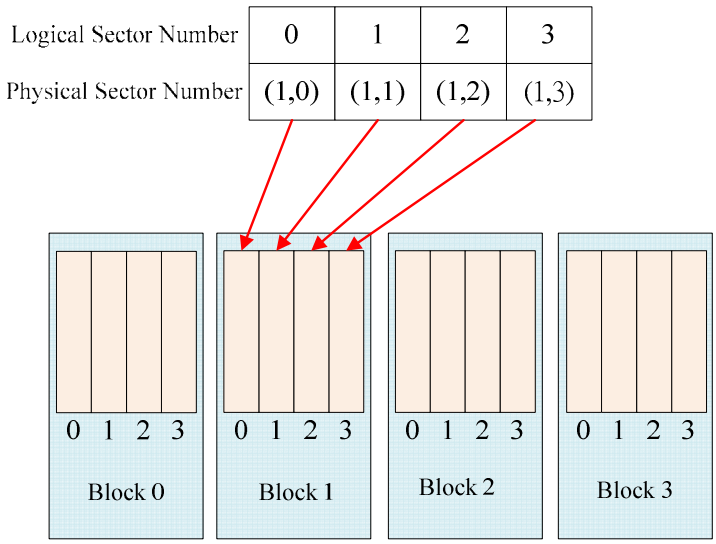


圖 4.5: 原先 block 中的 sector 對映情況

若 logical number 3 為 hot data，而 logical number 0,1,2 皆為 cold data，在一連串的 hot data、cold data 交換 policy 之下，logical number 為 1 跟 logical number 為 3 的交換寫入。

Sector 中的資料代表被寫入的資料，由於是利用將 hot data 寫入到 cold sector 的動作，這表示，在當被寫入的 sector 到了一個 Threshold 做 Remap 時，block 會有一個 cold data 寫入，且 hot data 亦會更新，因此造成了兩次寫入；另外，這裡也顯示 block 會因為這個動作，會有一筆 cold data 紀錄儲存在此，而此 cold data 會因為它的特性而很少更新。其內部資料以 hot data 的 sector Number 代表。情況如下：

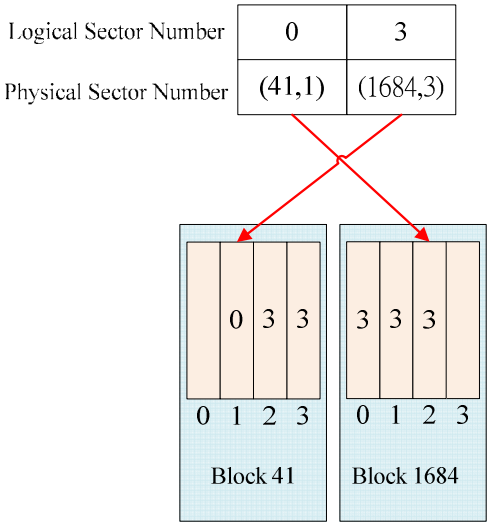


圖 4.6: FAL 交換後對 FMD 的影響

在寫入數個 sector 之後，找置放 cold data 的 sector 來替換，在換之前，亦要將原來的 sector 被拿去換的 data 換回去，因此在 Physical sector 的表示如下：

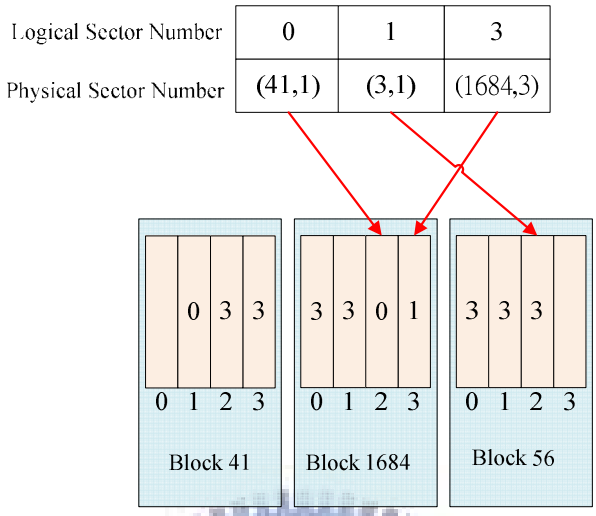


圖 4.7: 對更換的 sector 之替換

4.4 Implementation Overhead

為了讓此動作可以順利的執行，免不了要多使用一些 memory，此方法使用了 Bitmap 來紀錄了那些 sector 被使用，以避免被過度使用，另外利用 queue 來區分何為 hot sector，最後再使用 table 來紀錄最近常被使用的 sector 為那些。

而這些方法的 structure 說明如下：table 是使用 link list，因此會與要紀錄的 table 大小有關，若是需紀錄的 hot sector 愈多，設定的 table size 愈大，皆會使得此 table 所花費的 memory 愈多，每個 table entry 中紀錄的是 sector information address，而 sector information 紀錄了一些 remap 的資訊，比如說：此 sector id 目前是對映到那個 sector 上，而被替換的 sector id 是那一個，對映到的 sector 目前又被使用了多少次。而 FIFO queue 就只紀錄了 sector id。Bitmap 則是使用了 unsigned char 來對映 flash 中所有的 sector。

5 Experimentation Result

本篇論文提出在 Windows CE 的 FAL 加上一層新 Implement 的 FAL 來幫助對 Flash 的 Wear-leveling，並利用讓 hot、cold data 的特性，讓所有的 sector 都得以平均使用，不致使某些 sector 被快速的磨損造成毀壞。

在不更動 Windows CE 所以架構的環境下，若是期望可以不用更改 Device Driver 來達到 Wear-leveling 的手段，就可以在 Windows CE 的 FAL 的上一層加上新 Implement 的 FAL 來幫助對 Flash 的 Wear-leveling。

5.1 Experimental Setup

參考[5][6]的特性，對映 windows ce 的實驗環境，所以我們實驗對象，就是每個 sector 為 512bytes，每個 block 有 65536 個 sector 總大小為 32Mbytes 的 flash 實驗部分為修改 FMD 的 Link Library，使其 Link RLFAL，由 RLFAL 去連結 Windows CE 原本的 FAL；且上層的 OS 只會透過新加的 RLFAL 來對 Device 做 Read、Write 的 IO-Control。而模擬環境是由 Platform Builder 6.0 裡提供的 Emulator 來模擬實驗。

實驗的目的為測量在 Remap 之後是否會給原來 Device 的 block 增加過多的 erase 次數以及 wear-leveling 的效果。在實驗中我們所使用的 Workload 爲實際使用者在 WinCE 6.0 嵌入式系統上操作 IE 程式，並將 IE 快取檔案存放至我們模擬的 Nand Flash Memory Device 藉以紀錄系統對 Nand Flash Memory Device Read/Write 的操作。而在紀錄期間先存放數個音樂檔(MP3)於 Nand Flash Memory Device 中當作 cold data，而這 cold data 的大小約為實驗中設定可用空間的一半，以實驗 hot、cold data 對 Wear Leveling 的影響。而我們分析 Workload 的資訊，表一為對實體快閃記憶體寫入的總數。表 5.1 表示所有 sector 被寫入的次數為 14,316,837。

Sector Size	512 Bytes
Sectors Per Block	8
Total Physical Blocks	8192
Flash Size	32M Bytes

表 5.1: Test Flash

Trace Line	7,094,700
Total Write	14,316,837

表 5.2: Trace :

5.2 Effects of wear-leveling

在此節中我們將分別測試不同 threshold、table size、FIFO queue size 對 wear-leveling 所造成的影響，以及何種設定將可以得到最佳解，另外，在 RLFAL 做 remap 的行為會增加一些 write 的成本。另外，這樣的行為是否也會造成底層增加 erase 的成本，將會在此做說明。

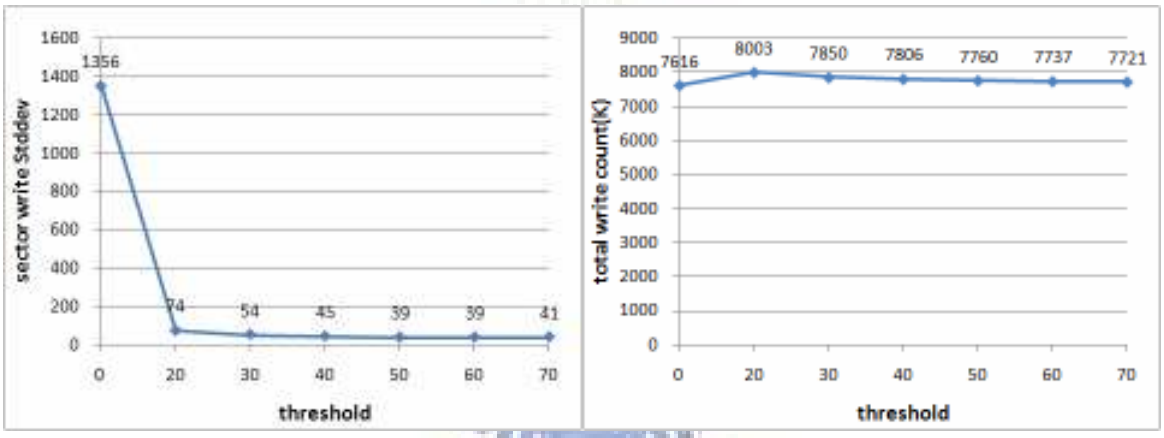
以下將會分別設定 Threshold、table size、FIFO queue size。Threshold 越小將越快速的執行 sector 間的置換，使各 block 被抹除的次數越平均，但將花費較多 erase 的成本。

首先，想要確認的是，這個方法是否有效，因此就要先看它的標準差，而標準差(stdev)則是在主要是用以衡量觀測資料與其平均數之間的差異量數，衡量值與平均值之間的離散程度。以下的數據在 Threshold、table size、FIFO queue size 為 0 的部分皆是代表沒有加上本論文的方法，只有 FAL 本身 wear-leveling 的情形。

給定一個較大的 table size 及 FIFO size 設為 100，再比較各個 threshold 的情形，說明圖 5.1(a)在上層，也就是在 RLFAL 的行為下，對 FAL 造成的影響，利用觀察在各個 threshold 情況下，比較其是否會有太多的 remap 行為，以及 wear-leveling 的效果。可以看到的是，在 threshold 各個情況下，它的標準差都不會差太多，但是它的 remap 所造成的 write 須多出的寫入，需找到一個平衡點。圖 5.2(b)說明，在下層，也就是 device 的部分，針對它 erase 的行為做比較，看上層的行為是否會對於 block 的 wear-leveling 的效果

是否有成效，且是否會造成多餘的 erase 成本。可以看到的是，在此情況下，平均的 erase count 是有下降的趨勢。這裡可以說明，在上層使用了只紀錄部分的 hot sector，並且做了 remap 的成本，在 device 的部分並不會造成實際的過多成本，也成功了平均各 block 被磨損的效果。threshold 愈小，其 wear-leveling 的效果愈好，但是卻要額外付出一些成本，表示其上層寫到的次數愈小就要找新的 cold sector 去交換，其交換時便會多一次 cold data 的寫入，因此，亦會造成在底層多一次的寫入。這個寫入便是一個成本。

實驗會在全部的 logical sector 皆從 0 找到最後一個 sector 的情況後，也就是說，令所有的 logical sector 最少皆被使用過一次，去看使用過的情況，在 physical layer 的情況也是好的，這裡先展示在 logical layer 的情況，在 wear-leveling 的情況下，可以看得出它的效果也是比沒有加上 policy 的效果好上許多。

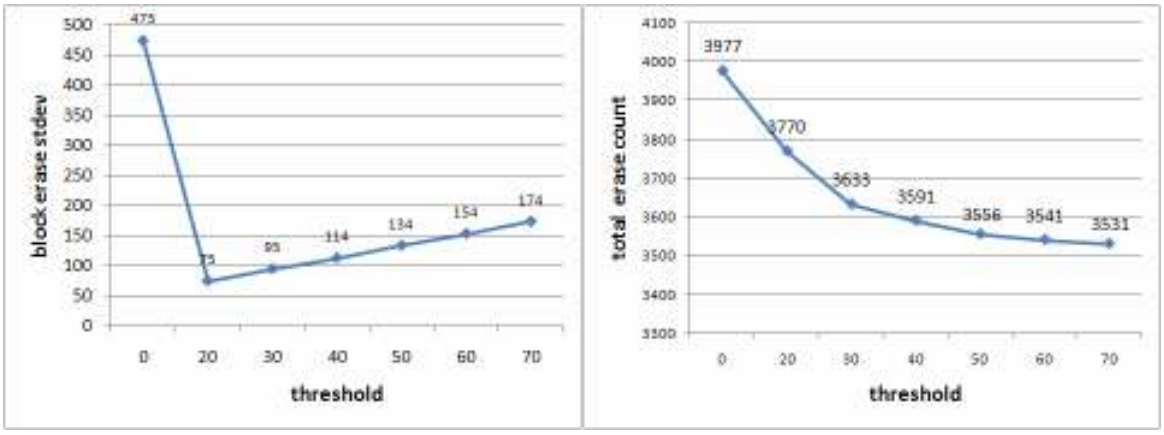


(a): Standard deviation of sector write (b): Total write count

圖 5.1: Wear-leveling 在 FAL 所造成的影響

從圖 5.2 可以看出，在 threshold 為 50 的情況下，得到的 erase count 以及標準差是可以接受的範圍，因此，下面再實驗時，便以 threshold 為 50 的設定去做各個實驗。在 FAL 做 remap 的效果，也就是變更 logical sector 對映變動，是否可以達到讓 physical layer 亦可達到相同的效果。在這個情況下，可以看到其標準差是大幅度的變小。這也代表說，wear-leveling 是有效果的。當 table size 愈大，表示它可以掌握的 hot sector 愈準確，所以也可以確保各個 block 磨損愈平均。圖 5.4(b)說明在各個各 table size 下，在 table 為 200 即有明顯的改善，再加大一點 table size 為 500 時即可達到一個很好的效果，若是讓 table size 加大到 fullmap，對照圖 5.3(a) (b)以 ram 的空間來換取 erase count 的減低(也就是

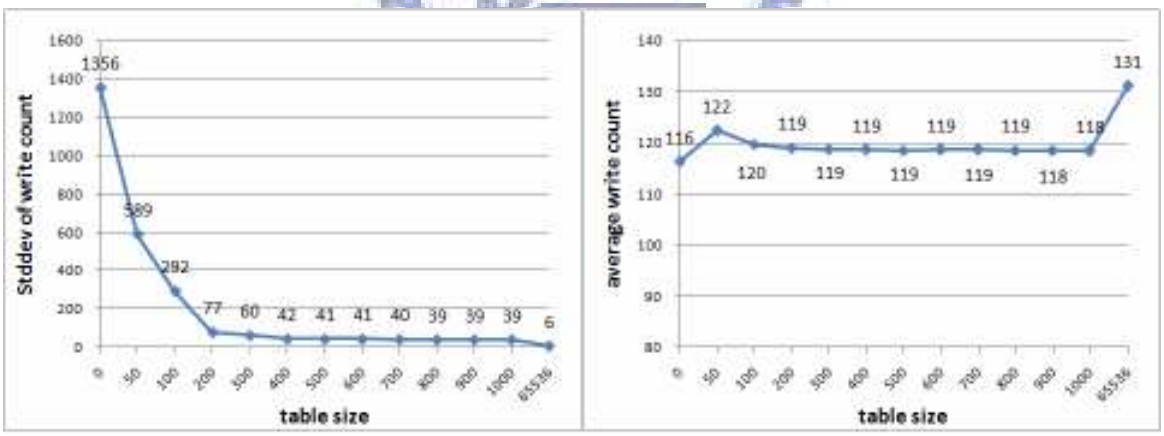
garbage collection 的效果)，仍應考慮它寫入的成本。



(a): Standard deviation of block erase (b): Total erase count

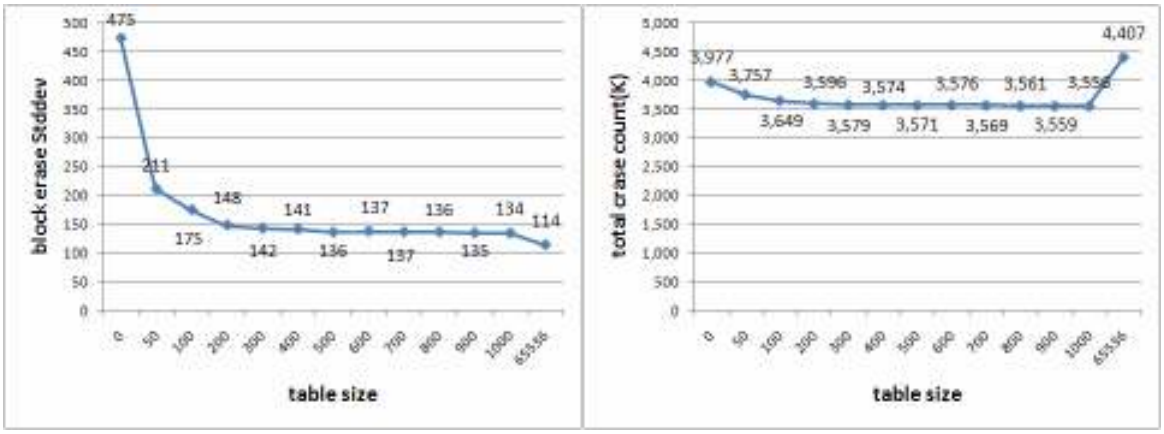
圖 5.2: Wear-leveling 在 FMD 所造成的影響

除此之外，也加上了 full-map 來做比較，full-map 也是做在 RLFAL，並且跟本論文所提出的方法相比。在 full-map 的可以很精準的控制所有的 sector，誰使用到了限定的 threshold，誰可以拿來當 cold sector 都可是被紀錄的，但是它的缺點是，多了許多的 write，雖說如此，但是它的標準差十分的好，erase count 跟其它的相較起來也較小，但是它所付出的 ram 空間，以及 write 成本也應該考慮。



(a): Standard deviation of write count (b): Average write count

圖 5.3: 各 table size 的 write count 比較



(a): Standard deviation of erase count　　(b): Average erase count

圖 5.4: 各 table size 的 erase count 之比較

以 erase count 的比較的部分，觀察 5.5 在 hot、cold data 置換寫入下，提供所有的 block erase 的情況，比較兩者在 Total erase 的情況下，在選取 block dirty count 的部分，是以各個 table 的情況下，在 block 取得其 dirty count，可以看到是 dirty count 在 block 中在 table 上的表現，由於 block 所取得的 dirty count 愈高，代表它可以少 erase 幾個 block，就可以得到所需的可用空間。所以 erase count 較低，所以這部分是說明讓 GC 更有效率的原因。

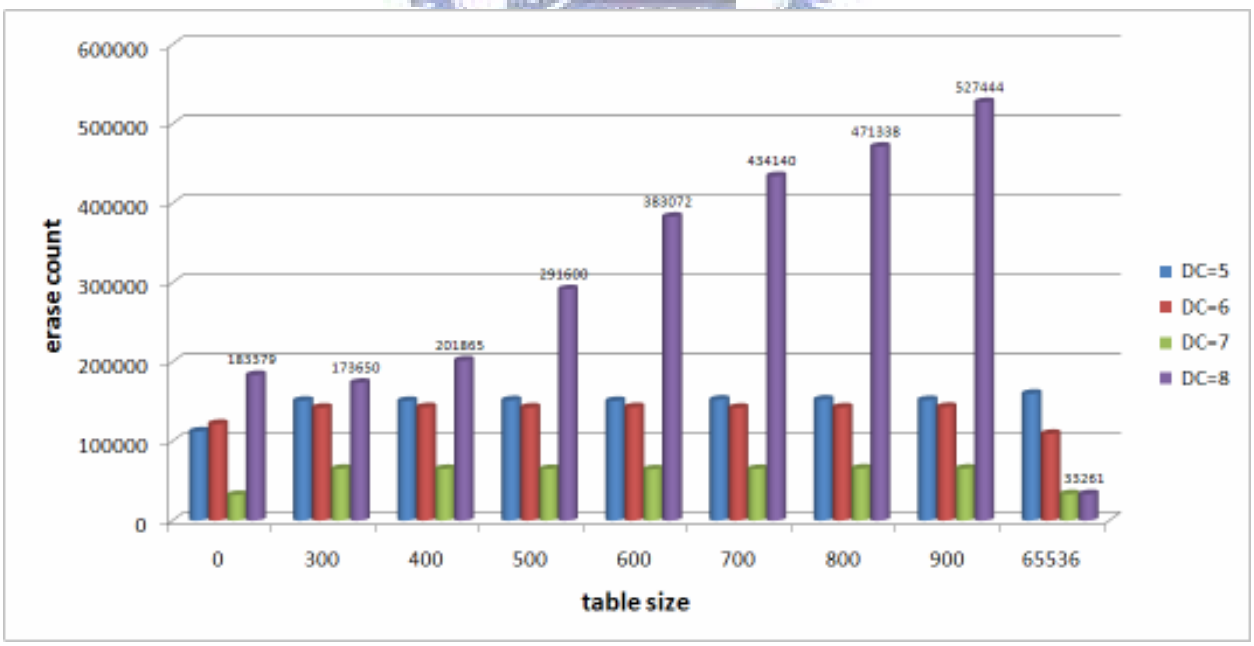
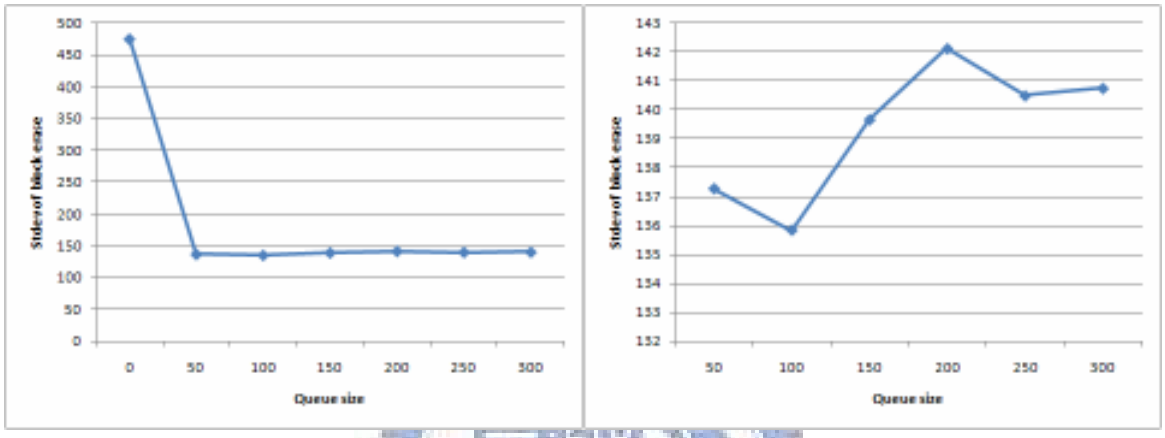


圖 5.5: 各個 table size 上，block 的 sector dirty count 之比較

另外，參考圖 5.6 的數據，利用各個 queue size，可以少 hot sector 誤判的機會，以

期達到最佳的效果。可以看到的是，在 table size 為 500, threshold 為 50 的情況下，這幾種 FIFO queue size 的效果差不多，但在 queue 為 100 的情況下，已經是最好的情形。而 queue size 為 0 是沒有加方法，FAL 原本 wear-leveling 的效果。可以看到的是有使用 FIFO queue 與沒有使用的差別，其標準差很大，那麼要取那個值做會比較合用的值，因此將其放大比較，可以看到當 FIFO queue 為 50 的情況下，其標準只差了一點點，但是 queue size 為 100 是此 case 中的最低點。



(a):各 FIFO queue size 的標準差之比較　　(b): 各 FIFO queue size 的標準差之比較

圖 5.6: FIFO queue size 之比較

6 Conclusions

在不更改原本的 FAL 或是 Device Driver 前提下，但又希望可以有更好的 wear-leveling，使用在 FAL 上層做 Remap 的做法，可使得 wear-leveling 的相應上原本 Windows CE 的 FAL 的效果來得好。

利用 RLFAL 的 Remap 做法，可以將 hot、cold data 做一個有效的利用，不致於使部分 block 被過度的磨損。並且從實驗結果可看出在 erase count 之次數及標準差，均少於原來 Windows CE 的 FAL 的 wear-leveling 的效果，因此可證明 RLFAL 是一個簡單且有效應用於 wear-leveling 的方法。可有效減少 Garbage Collection 的 erase count，以及平均各個 block 磨損的程度。未來如何讓 hot、cold sector 分開，不要使 block 中的 sector 被 hot、cold sector 是一個改進的目標。

從實驗結果可以看出，不論是 erase count 或是平均各 block 磨損效果，都比原本的 FAL 好，因此證明 RLFAL 是一個可替代的做法，以增加 wear-leveling 的效果，又不用增加過多的 Overhead。

參考文獻

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

[2] Li-Pin Chang , “ A Stackable Wear-Leveling Module For Linux-Based Flash File Systems , ” Taiwan-Korea Data Storage Symposium, 2007

[3] Thomas Gleixner, Frank Haverkamp, and Artem Bityutskiy, “ UBI - Unsorted Block Images ,” Copyright © 2006 International Business Machines Corp

[4] 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

[5] Samsung Electronics Company, “K9F8G08UXM 1Gb * 8 Bit NAND Flash Memory Data Sheet".

[6] Samsung Electronics Company, “K9XXG08UXM 2Gb * 8 Bit /4Gb * 8 Bit/6Gb * 8 Bit/8Gb * 8 Bit NAND Flash Memory Data Sheet (Preliminary)"