國立交通大學

資訊科學與工程研究所

碩士論文

# 適用於線上調整的快閃記憶體資料區塊與位址轉換表區塊之分析

# Flash block partitioning for performance optimization of FTLs using mapping cache

研 究 生: 李文平
指導教授: 張立平 教授

中華民國 一〇一 年 七 月

# 適用於線上調整的快閃記憶體資料區塊與位址轉換表區塊之分析

# Flash block partitioning for performance optimization of FTLs using mapping cache


研 究 生: 李文平　　Student:Wen-Ping Li

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

July 2012

Hsinchu, Taiwan, Republic of China

中華民國一〇一年七月

# 適用於線上調整的快閃記憶體資料區塊與位址轉換表區塊之分析

學生: 李文平　　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

在 Page-level mapping FTL 的位址轉換表議題中, 大部分都是關於快取機制上的研究, 例如提高整體效能以及節省儲存空間等等。

此篇論文, 我們則於 flash memory, 分析資料區塊和位址轉換表區塊的最佳分配比例, 並提出可線上調整的計算方法。因爲 user data 和 mapping information 所含的資料, 其冷熱程度有所不同, 我們認爲必須將它們分開來儲存。所有儲存 user data 的區塊我們稱之爲 data blocks(D-blocks), 而所有儲存 mapping information 的區塊我們稱之爲 translation blocks(T-blocks)。由於它們有不同的資料冷熱程度, 如果不去控制 T/D blocks 分配的比例, 則 Garbage Collection(GC) 使用 greedy policy 即會造成額外的寫入成本。在每種系統環境設定下, 都會存在一個 T/D blocks 的最佳分配之比例, 其根據不同的 flash gemotry 等因素而改變。但是透過我們的方法, 可以推導出最佳 T/D blocks 比例與影響因子彼此之間的成本關係, 並且用簡單的公式描述之。因此, 我們提出一種可線上調整的策略, 透過這些成本公式, 我們可以馬上爲目前的系統設定最佳 T/D blocks 比例, 省下測量所有狀況的最佳 T/D blocks 比例的時間, 而不必使用一張巨大的查表法。

關鍵字: 快閃記憶體、資料區塊、位址轉換表區塊、線上調整、區塊分割

# Flash block partitioning for performance optimization of FTLs using mapping cache

Student:Wen-Ping Li Advisors:Li-Ping Chang

Department of Computer and Information Science
National Chiao Tung University

## Abstract

Many caching mechanisms for mapping tables are proposed to improve efficiency and to save spaces based on page-level mapping FTL.

In this study, we further explore the mapping table management in flash memory. The user data and the mapping information are separated in flash blocks, similar to the hot-cold data separation. Those blocks for storing data called data blocks (D-blocks) and those blocks for storing mapping table called translation blocks (T-blocks). Due to their different degree of hot and cold, uncontrolled allocation for T/D blocks using the greedy GC policy will cause more overhead. For a system setting, there exists a best ratio for T/D block partition. But the optimal ratio for T/D blocks will change from various flash geometry factors. However, we can solve it by our reduction of cost formulae and analysis. As a result, We propose a on-line tuning optimal ratio for T/D blocks method instead of the huge look-up table.

**Keywords:**Flash memory, data block, translation block, on-line tuning, block partition

# 誌謝

時光不知不覺地過去, 在交大待了六年的時間, 也到了要離別的時刻。首先謝謝交大資工系的栽培, 讓我可以順利地成長, 有所收穫。而在研究所期間, 要由衷地感謝我的指導教授-張立平老師, 從大三的專題帶起, 十分辛苦地教導我, 一路跌跌撞撞到碩士畢業。這期間老師付出很多的關心, 同時指導我論文的研究, 也教了我許多道理, 讓我眞的覺得十分受用。

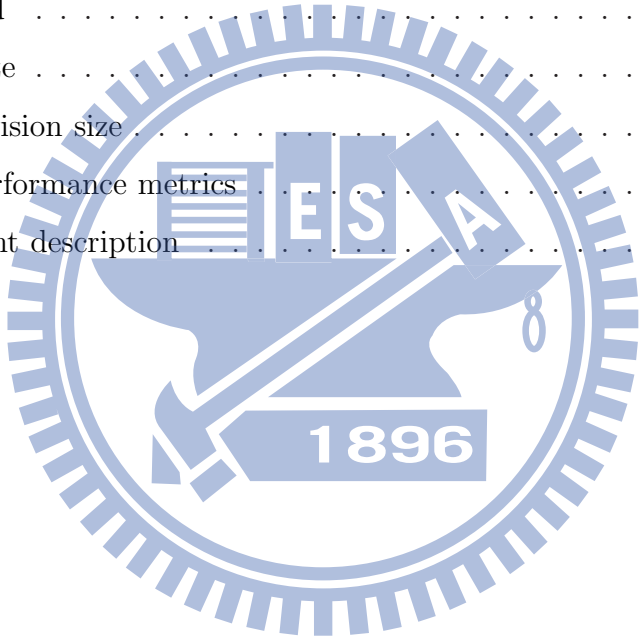
另外, 我也要特別感謝實驗室中的每一個人, 大家都很樂於助人、包容我, 例如: 李盈節學長、吳翊誠學長, 在我剛進實驗室時, 不管是課業或是實驗都會教我; 還有電鍋學長、Master 學長, 在我們碩一期間, 還會回實驗室關心; 而一起就讀的同學, 像柏翰是夜場王子, 逸康以實驗室爲家, 棟揚是很有智慧的人, 晨意是可靠的聰明人, 都幫助我很多, 十分感謝你們。也要特別謝謝學弟黃聖閔、毛超遠、黃鼎傑、洪政猷, 實驗室有你們絕對會有不錯的成長。

由於有你們, 才能讓我沒有顧慮, 順利完成學業。最後謝謝一路扶持我的父母與家人, 在學業上能讓我很自由地發展, 得以成長至今。

# 目錄

# 表目次





# 圖目次

# 第一章 Introduction

固態硬碟 (Solid-State Drive, 簡稱 SSD) 使用 NAND 快閃記憶體 (NAND flash memory) 來儲存資料。NAND flash memory 具有體積小、低耗電、耐震、快速隨機讀取等優點; 而一般機械式的 Hard Drive (HD) 則是低耐震、體積大、噪音大。因此, 固態硬碟在可攜式裝置上取代 HD 已成爲一種趨勢。

NAND flash memory based SSD 有別於一般機械式的硬碟, 裡面所採用的 firmware algorithm-快閃記憶體轉換層 (Flash Translation Layer, 簡稱 FTL) 會實際地影響 IOPS 效能與使用壽命。FTL 能將快閃記憶體模擬成 block device, 提供 host 一種方便操作的介面。因爲快閃記憶體在操作上有一定的限制, 例如它無法將資料重新寫回相同的位置, 除非先將該位置抹除才能再寫 (erase-before-write) ; 而且抹除的單位比讀寫的單位大很多, 前者是以區塊 (block) 爲單位, 後者則是以頁面 (page) 爲單位; 每一個區塊又有抹除次數的限制, 所以我們會在 FTL 裡面實作 Address translation、Garbage collection、Wear leveling 等功能, 去讓快閃記憶體在操作上更有效率、延長它的使用壽命。

FTL 演算法使用 out-of-place 的更新, 將新資料寫入一個可寫的區塊中, 通常是從預先保留的區塊 (log block) 取得。當預留區塊用完時, 要將舊資料回收的動作稱之爲 Garbage collection (GC) 。GC 的過程是要挑選一個 victim block, 搬移其中的有效資料頁 (valid page) 到別的區塊, 再抹除掉 victim block, 並重複這種動作直到能多出一個 free block 爲止。由於抹除的時間通常都比讀寫還要來得大很多, 而且要抹除的區塊中若包含很多的 valid pages, 則 GC 的成本會非常高, 所以使用好的 GC policy 可以有效地降低抹除與讀寫次數之成本、提高效能。Wear leveling (WL) 則是爲了要讓快閃記憶體每一個區塊的抹除次數平均化, 而將 cold data 搬到抹除次數較高的區塊之過程。

由於快閃記憶體的資料會因 out-of-place 更新、GC、WL 等動作, 不斷的改變實體的位置, FTL 必須有 Address translation 的機制, 實際上我們會用位址對映表 (mapping table) 記錄其邏輯位址所對應的實體位址。不同的 FTL 設計中, 可能會有不同大小的位址對映表, 常見的方法有 Page-level mapping、Block-level mapping 以及 Hybrid mapping 等三種方式。Page-level mapping 採用比較精細的對映方法 (fine-grained mapping) , 對映表內記錄 logical page address 所對應的 physical page address, 所以對映表會比較大; 而 Block-level mapping

(又可稱爲 coarse-grained mapping) 的對映表則比較小, 因爲它只記錄 logical block address 所相對應的 physical block address; Hybrid mapping 方式則是在主要的 data block 中採用 block level 之對應, 但是在那些預先保留用來更新 data block 資料的 log block 之內, 則是採用 page level 之對應, 其對映表的大小會介於 Page-level 和 Block-level 之間。

Page-level mapping FTL 利用快閃記憶體以 page 爲寫入單位的特性, 當資料一被更新只要修改對映表即可, 其優點就是較彈性。它在隨機的小寫入時不會因爲要保持 block 內的 page 之順序來排列, 而付出額外的寫入成本, 但是相對的就要付出較大的空間來儲存對映表。然而最近隨著控制器的進化, 以及 RAM 容量越來越大, 例如 [1] 指出,Intel 510 系列的SSD 採用 Marvell 的雙核心控制器以及內含128 MB 的 DDR RAM。如今, 越來越多的廠商願意使用 Page-level mapping FTL 來提升效能表現。

儘管如此, 在大容量的固態硬碟中, 此種控制器的 RAM 依然不夠大。舉例來說, 假設對映資訊的傳輸單元大小爲4 KB 的單位, 在128 GB 的 disk size, 可以將所有的對映資訊都放在 128 MB 的 DDR RAM 上面。但是若 disk size 增加到256 GB, 則所有的對映資訊儲存在 RAM 上面需要256 MB, 因此就變成放不下了。

以往傳統的 page-level mapping 管理對映表的作法是在 spare area 的 metadata 裡面記錄對映資訊。因爲 spare area 空間較小而需要將對映表分開來儲放, 在 RAM 上面則會保留常用的對映表, 其餘的等到要用時再去建立。如此一來, 每次建立對映表時需要經過多次的讀取才能完成。和以往做法不同, 我們將多個邏輯連續地對映資訊都存放在實體頁面的 data area 上, 因此只要花一個頁面的讀取即能建立一張比較大的對映表。而儲存 user data 的頁面我們稱爲 data pages (D-pages) , 儲存對映資訊的頁面我們稱爲 translation pages (T-pages) ; 所有都存放 D-pages 的區塊我們稱之爲 data blocks (D-blocks) , 而所有都存放 T-pages 的區塊我們稱之爲 translation blocks (T-blocks) 。因爲它們所包含的資料冷熱程度有所不同, 所以我們要把 T-page 和 D-page 分開來儲存。此外, GC 時我們有 T-block 和 D-block 兩種犧牲區塊可以挑選, 犧牲區塊內的有效頁面是我們要 copy 的成本。若放任 T/D-block 個數自由成長, 當 D-block 減少 T-block 增加時, 可以發現 D-block 內的 valid D-pages 會隨之上升, 而 T-block 內的 valid T-pages 則會下降, 反之亦然。所以, 有效地管理 T-blocks 與 D-blocks 的比例是很重要的議題。

我們說明三種典型的 workloads 之讀寫行爲, 首先是多媒體存取的 workload, 其大部分都是 sequential 在連續的邏輯頁面存取, 而且具有良好的空間區域性。由於它 GC 的活動並不強烈, 只需要給一些 T-block 即可; 第二個是桌機的 workload, 它有一些隨機的存取, 也包含比較強的空間區域性。其資料會有明顯冷熱的區別, 所以比起多媒體存取樣式只要再多微調 T-block 個數; 第三個是隨機存取的 workload, 它是隨機地在邏輯頁面存取, 其資料並無明顯地冷熱得區別。但因爲它的 cache 擊中率很低, GC 活動又很強烈, 所以效能會大受影響, T-blocks 個數的多寡對於效能來說非常敏感。由於各家廠商加載 mapping cache 之後, 發現隨機存取的行爲之下的效能並沒有明顯提升, 影響他們使用 mapping cache 的意願。但我們認爲隨機存取是可以分析的, 故此篇論文將重點放在隨機存取的 workload 上。

針對隨機存取 workload 我們提出一套找最佳 T/D-block 個數的方法, 能夠有效解決資料冷熱的問題, 而且最佳比例的設定也可適用於線上調整。本論文主要有四個研究議題, 第一個是影響因子, 我們發現有些因子可以互換, 而有些因子會影響最佳 T/D-blocks 比例的設定, 所以我們要先找出這些主要影響因子。第二個是成本模型的分析, 我們建立影響因子與成本之間的模型, 並且推導出影響因子與最佳 T/D-blocks 比例的成本關係。第三個是係數校正, 由於推導的成本公式會有一些常數係數, 用來涵蓋數個微小或固定的因子影響, 我們透過少數幾個量測的數據來做這些常數係數的校正。最後一個是線上調整最佳 T/D-blocks 的比例, 根據我們的公式, 可以計算出最佳 T/D-block 的比例, 而且在一些可動態調整因子的改變下, 能夠馬上爲系統設定 T/D-block 的最佳比例。因此, 在隨機存取 workload 下, 我們可以大幅提升效能, 如果使用 mapping cache 爲全存的50%, IOPS 從原本使用 greedy 的174.46提升到204.88, 原本有21.5%的 IOPS degradation, 降低到只剩下7.85%的 IOPS degradation。

接下來的篇章結構如下: 第二章 Background, 介紹對映表在快閃記憶體上的儲存管理方式, 以及需要考量的問題; 第三章 Block partitioning for T/D-block management, 介紹此篇論文的方法, 並提出可線上調整的策略; 第四章是實驗結果; 第五章是結論。

# 第二章 Background

## 2.1 Logical-to-Physical mapping cache

在這一節我們會介紹 SSD 的 mapping cache。FTL 的位址轉換方法中, 通常會使用一張 Logical address to Physical address 的表, 我們 稱之爲對映表。SSD 會把這些對映表放進 RAM 裡面當作 cache, 用來快速的查詢位址或是更新對映資訊, 我們稱這塊 RAM 的空間爲 mapping cache。

不同的 FTL 設計會有不同大小的對映表, 以下介紹常見的三種 FTLs。如圖1的 (a) Page-level mapping FTL, 它採用比較精細的對映方式, 記錄每一個 page 所對應的位址。所以其對映表會很大, 例如256 GB disk size 的固態硬碟, 其對映表需要256 MB 的空間。爲了減少對映表的大小, 可以將 logical page address 拆解爲一個 logical block number 和一個 page offset。因此對映表只記錄每一個 block 所對應的位址, 若一個 block 包含128個 page, 對映表大小就可從256 MB 減少到2 MB, 如圖1的 (b)Block-level mapping FTL。

但是 Block-level mapping 必須保持 block 裡面所有的 pages 都是最新的資料, 因此每當寫入一個 page 就要更新一整個 block, 不利隨機的小寫入動作。所以, 又有 Hybrid mapping 另一種的方法被提出來。如圖2, 它是將 block 分爲 data block 和 log block, 在 data block 內採用 block level 之對應, 但是在那些預先保留用來更新 data block 資料的 log block 中採用 page level 之對應。這樣做是爲了在 page-level mapping 和 block-level mapping 之間取得平

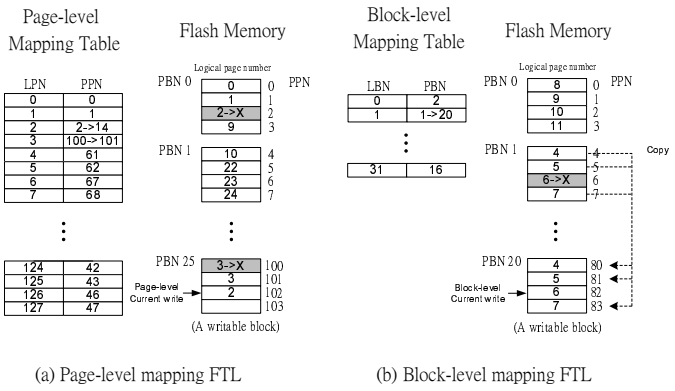



圖 1: Page-level mapping FTL and Block-level mapping FTL

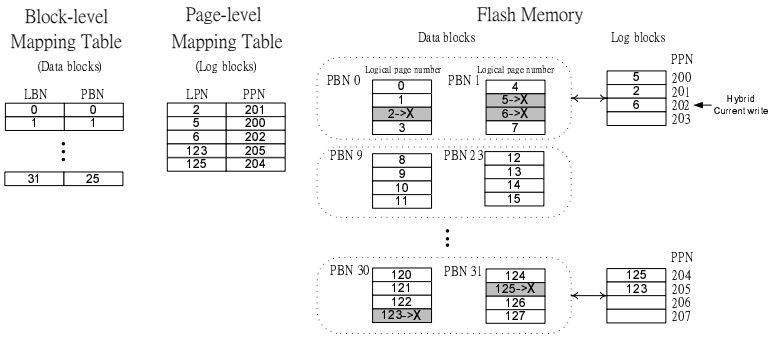



圖 2: Hybrid mapping FTL

衡, 但是若 log block 的利用度太低的話, 很容易發生 block threshing 的問題。此外, GC 時要將 log block 內的 valid pages 所關聯到的 data blocks 作 merge, 對於隨機的小寫入依然要付出許多額外成本。

Page-level mapping FTL 處理隨機小寫入的動作較佳, 不須付出額外的成本, 也不用擔心空間利用度過低的問題, 但相對的需要較大的 mapping cache size。因此, 如何妥善的管理 mapping cache 是我們主要的研究目標。

## 2.2 Mapping cache management

在這一節會介紹我們使用的 two-level mapping cache 架構, 並且定義一些在 mapping cache 中的使用項目。

首先, 我們說明 Global Translation Table。由於我們會將對映資訊分散在快閃記憶體上儲存, 且這些對映資訊的數量是固定的, 所以必須再多一層表格去記錄它們的實體位址。這張表就像是對映資料的對映表一樣, 我們稱之爲 Global Translation Table (GTT), 如圖3。若在256 GB disk size 下, 這張表的大小則只有256 KB, 因爲其成本相對地小, 所以 Global Translation Table 是可直接儲存在 RAM 上面的。圖3表示 GTT 假設有 T 個 entries, 則 GTT 的 size 就是$T \times 4\ Bytes$。我們定義一個 segment (圖3稱之爲 T-page, 之後我們會詳加定義 segment) 可以儲存的對映資訊之個數爲 N。由於 N 值是透過系統設定的, 即 $N = \frac{PageSize(Bytes)}{4\ (Bytes)}$ , 所以參數 T 的大小 (即 segment 的數量) 就是固定值, 是可以算出來的。例如256 GB disk size, 則 $T = \frac{MappingCount}{N} = \frac{256\ GB/4\ KB}{4\ KB/4\ B} = 64\ K$ 。

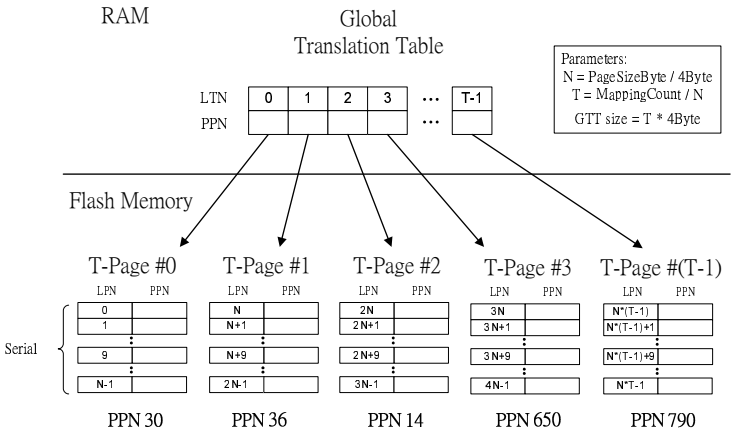



圖 3: Two-level table mapping cache : Global Translation Table

接下來我們定義對映資訊的大小。由於 host user write 或者 GC 都會更改邏輯頁面編號的對映資訊 (LPN, PPN) , 因此對映資訊 (LPN, PPN) 我們定義爲一對 mapping pair。例如圖4上 Cache Mapping Table 中, 某一欄的對映資訊 (128, 98)。另一種則是 segment, 一個 segment (可看成一個 T-page) 我們定義爲由多個邏輯上連續的 mapping pairs 所組成, 就像是圖4的一個 Translation Page (在2.3節會詳加敘述並定義) 一樣, 其裡面都是記錄連續的邏輯頁面編號 (64, 65, 66,..., 127) 所對映的實體頁面編號 (105, 116,..., 130) 。

Mapping cache 可以儲存很多對的 mapping pairs, 但因爲其容量放不下所有的對映資訊, 所以要制定它們的帶入以及替換機制。有兩種情況都會更新其對映資訊, 一個是 user write, 另一個是 GC 的 copy write。因此, 我們可將更新的對映資訊帶入快取 (稱之爲 fetch) 。而當對映資訊要帶入快取時, 如果已經滿了, 就必須將快取上的對映資訊替換下來 (稱之爲 replacement) 。

我們定義帶入的單位, 分別有兩個, 一個是 pair-wise, 另外一個是 segment-level。如圖4的 Case(a): pair-wise fetch, 它的方式是每次都只帶一對 mapping pair 到快取。例如, 我們從系統得知更新的邏輯頁面爲 $100_{LPN}$ , 於是先計算出邏輯的 T-page 編號 (即圖3的 LTN) 和 offset。假設從 GTT 上面找到此 LTN 所對映的實體頁面編號爲705, 接著利用算出來的 offset 將 (100, 201) 的 mapping pair 帶入快取。Pair-wise 的帶入會比較不占 RAM 的空間。對於沒有空間區域性的存取樣式來說, 可以降低對映資訊快取被替換的機會, 例如隨機存取 workload; 而 Case(b): segment-level fetch 的方式則是每次都帶一個 segment 的對映資訊進入快取。我們一樣算出 LTN 之後到 GTT 找, 假設其實體頁面編號706儲存著 $150_{LPN}$ 的對映資訊。之後

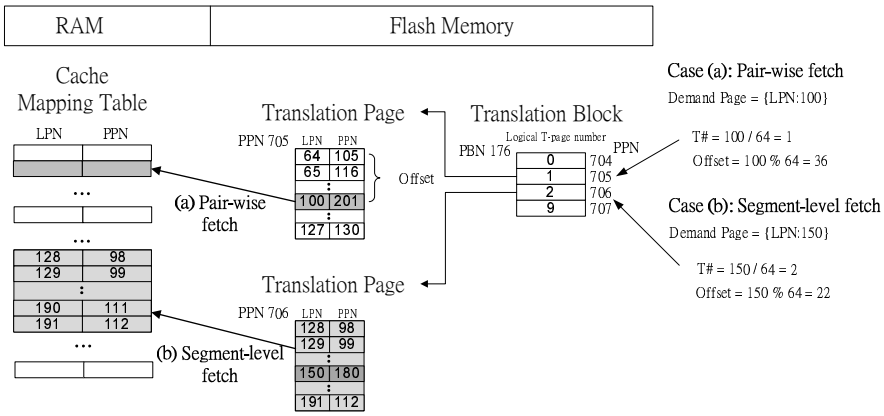



圖 4: Pair-wise fetch and Segment-level fetch

就將 (128, 129,..., 191) 整個 segment 的對映資訊都帶入快取, segment-level 的帶入對於擁有空間區域性的存取樣式會較有利。

爲了讓有時間區域性的存取樣式能夠很有效率地去配合使用, 我們的 mapping cache 替換機制採用 LRU 演算法。將最久沒有使用的對映資訊替換掉, 目的是保留較常使用的對映資訊在快取上。而替換的單位則和 Segment-level fetch 的單位一樣都是 segment (即一個 Translation page) 。

如圖5, 我們再將 fetch/replace 的過程以及 two-level table mapping cache 的架構詳細的說明一次。首先, 我們可以從系統得知需要服務的邏輯頁面位址爲 $100_{LPN}$ 。而因爲 T-pages 數量固定且裡面儲存邏輯連續的對映資訊, 所以我們可以算出 T-page 的邏輯編號 (LTN) 。假設一個 T-page 能儲存64個連續的對映資訊, 則 T-page 編號 (LTN) 算出來爲1, T-page 內的位移爲36。然後就到 GTT 找出 LTN 1的實體頁面編號, 假設爲705, 於是就將此對映資訊讀出來帶進快取。

當快取的空間不足時, 我們必須把快取上的對映資訊替換掉, 採用的替換演算法是 LRU ( Least-Recently-Use ) 。若挑選出來的是未被更新的對映資訊我們稱作是 clean 的 victim, 則可以直接捨棄掉; 若是被更新過的對映資訊我們稱作是 dirty 的 victim, 則必須將它寫回。如圖5, 假設我們挑到 LTN 10的 T-page 要寫回。因爲並非在此 T-page 內所有的對映資訊都在 cache 上, 所以要先到 GTT 找到 LTN 10的實體頁面編號 $745_{PPN}$ 。然後在 $745_{PPN}$ 的地方讀出 LTN 10原本的對映資訊, 再將最新的對映資訊都寫回 flash, 例如寫到 $760_{PPN}$ 的地方。最後我們修改 GTT 的 LTN 10對應的實體頁面編號爲760, 於是 LTN 10的對映資訊之後就變成在 $760_{PPN}$ 。因此, 替換的動作我們總共要花一次頁面讀取以及一次頁面寫入。

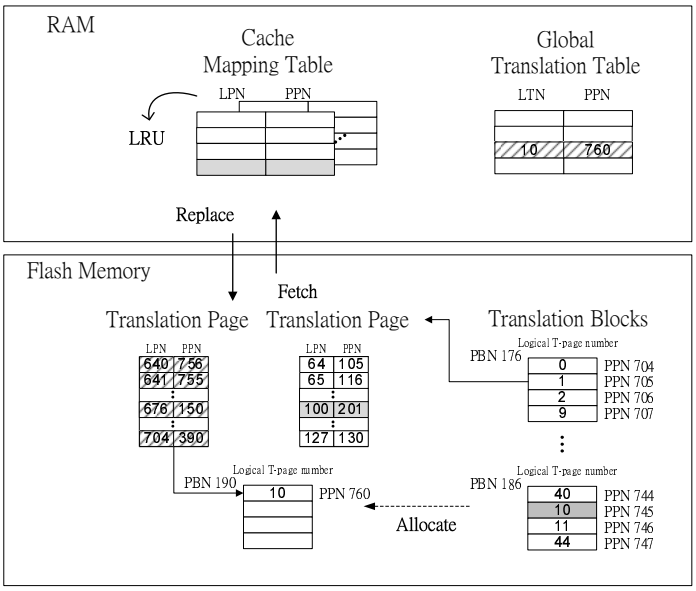



圖 5: Two-level table mapping cache scheme

總結來說, 對於我們的架構, 都會將需要用到的對映資訊放入快取, 以利有效率地去使用。在兩層 table 的管理下, 我們可以每次都只花一次的讀取頁面即能找到對映資訊, 而且因爲一個 segment 可以儲存很多對映資訊, 所以管理對映資訊的對映表 (Global Translation Table) 的成本也隨之減少很多。因此, 使用一個好的 mapping cache 管理機制是個重要議題。

## 2.3 Block allocation/partition for user data & mapping information

在 flash 上, 我們將 block 分爲兩種, 一種用來儲存資料, 另一種則用來儲存對映資訊。如圖6, 我們定義儲存對映資訊的 page 稱爲 translation pages (T-pages) , 而儲存資料的 page 則稱爲 data pages (D-pages) ; 所有儲存對映資訊的 block 我們稱爲 translation blocks (T-blocks) , 而所有儲存資料的 block 我們稱爲 data blocks (D-blocks) 。如圖6所示, 一個 translation Page 上面可儲存很多的 mapping pairs, 即我們在2.2節定義的一個 segment。由於我們認爲其兩種資料的冷熱程度有所不同, 所以將它們分開來儲存。一個 T-page 因爲包含很多個邏輯上連續地對映資訊, 很多不同的 D-pages 其對映資訊都指向同一個 T-page 上面。舉例來說, T-page 若爲4 KB, 則能夠儲存1024個對映資訊, 所以一個 T-page 等於是1024個 D-page 的縮影, T-page 很容易就被 invalid, 因此 T-block 會比 D-block 還要 hot, 如圖7。

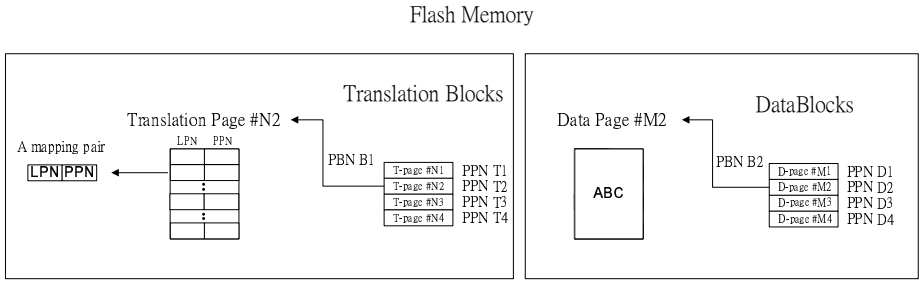



圖 6: Translation blocks and Data blocks

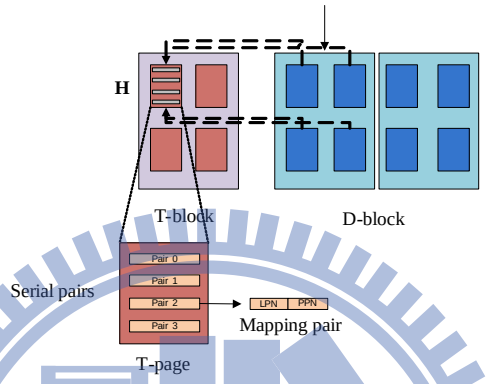



圖 7: T-blocks and D-blocks imply that hot/cold separation

D-page 和 T-page 在設定系統規格時, 就能固定其數量, 然而 D-block 和 T-block 的數量卻是會變動的, 當它們需要 allocation 的配給時就到 free block list 中取得, 因此 T-block 個數和 D-block 個數會互相消長。若放任 T/D-block 自由成長, 當 D-block 減少 T-block 增加時, 可以發現 D-block 內的 valid D-pages 會隨之上升, 而 T-block 內的 valid T-pages 則會下降, 反之亦然。一般而言, GC 爲了回收出最多的空間, 我們會用最簡單的演算法 greedy policy, 如圖8, 我們使用 GC lists 來挑選犧牲區塊, 第一層 list 爲 valid pages 數最少的那些區塊, 例如包含1個 valid page 的區塊, 而第二層 list 則是包含2個 valid pages 的那些區塊, 以此類推。我們每次都從最上層挑最少 valid page 數以及最久沒被更新的 block 來回收, 以利降低 GC 次數, 由於 GC lists 混合著 T/D-blocks, 所以 GC 時挑到的 victim block 可能有兩種, 一種是 D-block, 另一種是 T-block。使用 greedy 的 GC policy 的話則發現會常常挑中 T-block, 其原因在於 T-block 的 valid page 會較少, 因爲它比較 hot, 於是 T-block 個數會變少, 造成效能低落。所以控制好 T/D-blocks 個數的比例是很重要的議題。

因此, 我們想到在 GC 時也要控制 T/D-block 的個數, 如圖9所示, 我們將 T/D-block 做 partition, 但是並不是在 physically 上做 partition, 而是將它們做分群。假設全部的 block

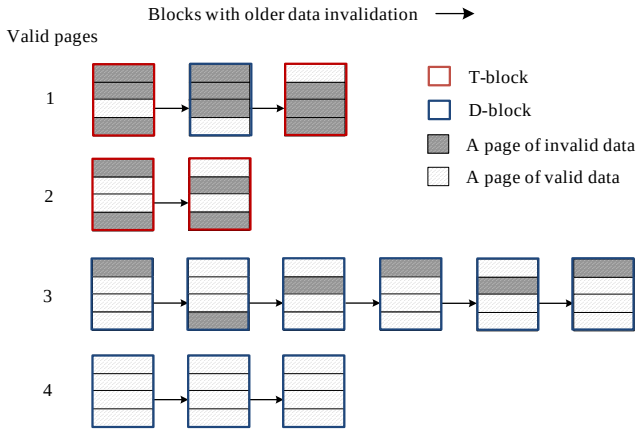



圖 8: GC lists of flash blocks using greedy policy

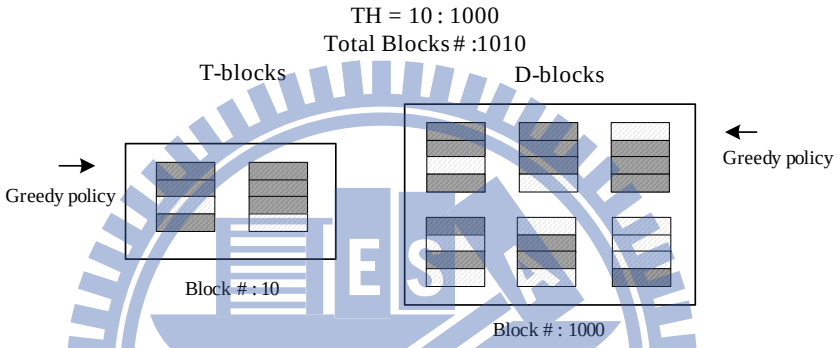



圖 9: Fixing the ratio of T/D-blocks partition

有1010個, 而 block partition 的方法將 T-block 個數限制在10個, T/D-blocks 的比例就是 10:1000, 當 free block list 不夠時, 我們會啓動 GC 演算法, 回收出一個可用的 block 給系統使用, 此時我們加入 T-block threshold 值 (TH) 的設定。當 GC 時只要 T-block 個數超過10個 (TH) 就回收 T-block; 而 D-block 個數超過1000個 ( $1010 - TH$ ) 也會回收 D-block。換句話說, 我們在 GC 加上的條件就是當 T-block 未超過 TH, 則我們挑 D-block 回收; 相反地,T-block 如果超過 TH 我們才挑 T-block。因此, TH 設定得比較小時, 會常常回收掉 T-block 而造成效果不好, 當 TH 設定得比較大時, 優點是可以避免常常回收 T-block, 但是缺點則是會造成 D-block 可能變得太少, 使得 GC 的成本大幅增加。

如圖10, 在隨機存取 workload 下, 當 T-block 的限制個數 (TH) 慢慢增加時, 會因爲避免挑中 T-block 來回收, 使得效能會有很大的改善。然而並非一直避免挑 T-block 來回收就會比較好, 如果 D-block 數量變得太少, 則 D-block 中的 valid page 數會一直增加, 直接造成 GC 負擔成本的加重, 也因此會間接造成 T-page 的更新, 而使得效能之後就一直變不好。圖10所示, 在 cache size 爲全存的50%下, 隨機存取 workload 的 IOPS 效能可以從原本使用 greedy 的

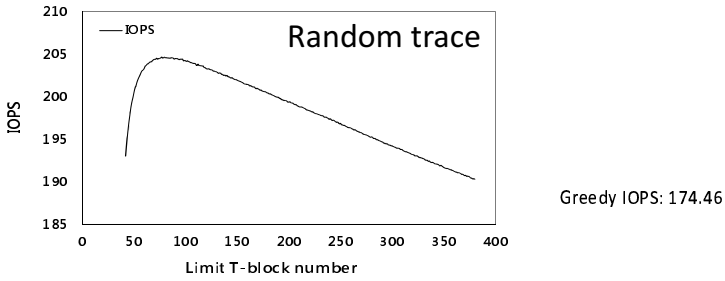



圖 10: Random's IOPS tendency with 50% cache size using fixed T:D ratio

IOPS 174.46提升到204.88, 原本有21.5%的 IOPS degradation, 可以降低到7.85%的 IOPS degradation。

因此, 這給了我們動機去研究, 如何管理 T/D-blocks 的數量, 此篇論文的首要目標即要找出最佳 T/D-block 的比例。此外, 我們需要一個適用於線上調整 T/D-block 最佳比例的策略, 當可動態改變的環境因子改變時, 能夠及時設定最佳的 T/D-block 比例, 我們在第三章會提出詳細的方法。

## 2.4 Related work

由於 Page-level mapping FTL 的對映表非常大, 如何有效率地使用 mapping cache, 其相關的文獻有 DFTL [2]、CDFTL [3]、SFTL [4] 等, 而關於對映表的使用架構則有 DFTL [2] 和 Superblock [5] 等文獻所提出。

首先, 我們討論對映表的儲存方式, 在容量大的flash memory 中, 我們可以設計成 multi-level mapping table 來管理, 這樣的優點是可以把對映表分散在 flash 上, 並且可以一次 cache 住較常使用的對映表而不太占空間; 但是缺點是若 table miss 時, 就必須再多花幾次讀取對映表的時間, 才能找到所需的對映資訊。

Superblock [5] 即是使用三層table 的架構, 因爲它要將對映表儲存在 physical page 的 spare area, 透過分層的管理, 對映表才能放得進去。舉例來說, 在32bit 的 logical address 下, 可將之分爲 (S bits, N bits, P bits) , 第一層是 block-level 的 table, 所以其中的 S bits 爲 logical block number, 當作第一層表的 index, 用來查詢第二層表之位址, 以便找到每一個 block 內的 page-level table (PT) 所在的位址, 若每一個 entry 是4 bytes, 則在256 GB 的

disk size 下, 第一層表只需要2 MB 的空間; 因爲用一個 spare area 的空間容納不下其 block 內的 PT, 所以我們還要再把它切成等份, 分別儲存在不同的 physical pages 之 spare area 上, 而第二層表就是用來記錄這些 PTs 的位址, 故我們將取 N bits 來當第二層表的 index, 用來查詢第三層表的位址, 第三層表即是每個 block 內的 PT, 所以我們最後用剩餘的 P bit 來當 index, 即可找到所需的對映資訊。

若是 SLC 的 flash, 則一個 block 有64個 pages, 故其中 $(N+P)$ bit 必須滿足 $2^{(N+P)} >= 64$ , 即 $(N+P)$ 至少大於等於6, 理論上, 節省最多空間的情況爲 $N = P$ , 但 P 的值是代表第三層的對映表大小, 若變大則能一次帶較大的對映表進 cache, 換句話說, 犧牲一點儲存空間, 有機會可以提高第三層表的 hit ratio, 而減少每次 table miss 造成的額外讀取對映表之時間。對映表儲存在 spare area 上的優點是當每次更新其 page 時, 其最新的對映資訊也能寫入到其 page 的 spare area 中, 所以不會有任何的對映表之更新的負擔; 但因爲在 spare area 空間的受限下, Superblock [5] 會受其group 大小的限制, 如果 D-block 和 U-block 的個數達到限制的話, 就必須做 GC 而造成額外的寫入成本。

DFTL [2] 則是使用兩層table 的架構, 將位址對映表儲存在 physical page 的 data area 來管理, 我們即採用此種架構。它的優點有很多個, 首先, 第一層表來記錄儲存連續對映資訊的 T-pages 之實體位址, 假設256 GB 的 disk size, 其 page 大小爲4 KB, 則第一層表的大小只有256 KB, 可以完全放在 RAM 上面。此外, 當我們要將對映資訊帶進 cache 時, 能夠一次帶一個 Segment 的對映資訊進去, 能增加 cache 擊中率。而且不必受到 D-block 加 U-block 個數的限制, 不需付出額外的 GC 成本。在 two-level mapping 的架構下, 若發生 table miss 時, 也只要花一次讀取對映表的時間即可。然而其缺點是當資料的實體位址改變時, 就會增加更新對映資訊的成本, 尤其是 random 的存取樣式, 不但 cache 擊中率差, GC 的活動又很強烈, 造成更新對映資訊的壓力, 使得效能受到影響而變得低落。

其他的文獻如 CDFTL [3] 則是採用 DFTL [2] 的架構, 它針對快取的機制來改良, 像是一次都帶比較大的對映表上來, 若T-page 在 cache 的使用率過低則讓它保存較久等等的方法。SFTL [4] 也是同樣的架構, 它的研究是在有空間區域性的workload 下, 去使用較少的空間來儲存邏輯位址, 以便減少佔用 cache 的空間, 增加 cache 的效率等方法。然而, 這些文獻都沒有針對快閃記憶體上的對映資訊和一般資料的管理方式來討論, 沒有更進一步去處理資料冷熱的問題。

# 第三章 Block partitioning for T/D-block management

## 3.1 Overview

對於 host user write 以及 GC 兩種寫入行爲, 我們會更新其對映資訊, 所以都有可能引起 T-page 的寫回。在2.3節提到過 T-blocks 比 D-blocks 還要 hot, 而且 GC 有 T-block 和 D-block 兩種犧牲區塊可以選。若不好好區分 block, 則會引起資料冷熱的問題, 使得常常挑到比較 hot 的 T-block 來回收, 讓 T-block 個數都很少效能變得低落, 尤其是隨機存取 workload 會更加明顯, T-block 個數的多寡對它的效能會非常敏感。

如圖11的 (a) 和 (b), 我們適當的區分 T-block 和 D-block, 使用固定 block partition 的方式加上 greedy 挑選犧牲的區塊來回收。圖11中 reclaim 所指向的就是挑選犧牲區塊的地方, 我們從 valid page 數最少的區塊開始挑, 而犧牲區塊內的 valid pages 就是需要 copy 的成本。若移動調整這兩種 block 的個數, 如圖11的 (b), 當分割線往右移動的時候, D-block 個數變少, 所以 D-block victims 的平均 valid page 數就會變多; 反之, 若分割線往左移動時, T-block 個數會變少, T-block victims 的平均 valid page 數就會變多。因此, 調整分割線的話不是造成 T-block victims 的平均 valid page 數上升, 就是讓 D-block victims 的平均 valid page 數上升。要精準地得知當下系統應分配多少個 T-block 很難, 但是在隨機存取 workload 下, 一個 block 內的 valid pages 減少的速度, 會隨著時間越久而變得越慢, 最後 copy valid pages 的成本會趨近穩定, 如圖11(a) 的 Invalid T/D-pages 和 Valid T/D-pages 之間的曲線箭頭, 其存取樣式是可分析的, 所以我們要用 block partition 的調整方式, 找出最佳 T/D-block 的比例。

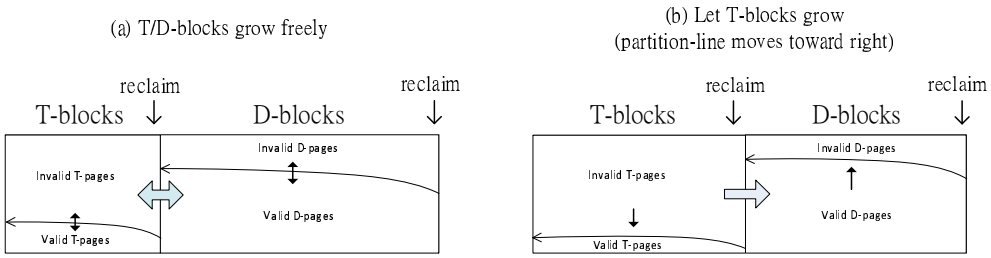



圖 11: Valid pages in T/D-blocks to reclaim

有許多系統設定因子都會影響這個最佳的比例, 但分析之後, 發現它們都可以轉換爲兩個基本因子, 分別是 overprovision 以及 cache size。首先, 我們建立這兩個影響因子與成本關係的模型, 再經由 T-block 個數與影響因子的關聯, 推導出最佳 T-block 個數與影響因子之間的成本關係。爲了避免公式太過複雜而無法實際地被應用, 我們以一些常數係數來涵蓋數個微小或者固定因子的影響。接下來是做係數的修正, 成本公式會有一些常數係數, 我們透過少數幾個量測的數據來做這些常數係數的校正。最後用公式計算所得到的最佳 T-block 個數跟實際上所測量的做比較, 分析 overprovision 和 cache size 這兩個因子影響的成本, 運算後發現其 T-block 個數約相差一個常數倍數, 所以透過簡單的運算再和實際上測量的相比, 最佳 T-block 個數的誤差都很小, 所得到的 IOPS 都十分接近。

Block partitioning 是一個適用於線上調整的方法, 當動態可變因子 write span 和 mapping cache size 調整改變時, 可以馬上爲系統設定最佳 T/D-blocks 的比例。此外, 其他因子例如 disk size、block size 等, 也可透過簡單的線性處理計算爲它們設定最佳比例。暴力查表法是要一個一個預先模擬得到最佳的 T/D-blocks 比例, 但我們的方法卻只需測量幾組數據, 比起暴力法去測量的力氣來得少很多。最後, 此方法也能應用於一套 pattern-aware 的方式, 針對典型的三種 workloads ( Sequential、PC、Random ), 提供 mapping cache 良善的機制與適當的 T-block 設定, 強化實際上應用的可行性。

## 3.2 Partition tuning strategy

我們經過三個步驟完成最佳 T-block 個數的計算, 第一步是成本公式的推導, 首先是找出影響因子與成本之間的關係, 然後推導出最佳 T-block 個數與影響因子之間的成本關係。第二步是常數係數的校正, 將成本公式的常數係數透過少數幾個測量的數據做校正。第三步是使用推導的公式去計算不同 cases 的最佳 T-block 個數。

因爲有許多因子會影響最佳 T-block 個數, 然而最後都可轉換爲兩個主要影響因子, 分別是 overprovision 以及 cache size, 所以我們對這兩個影響因子做分析。接下來章節安排如下: 3.2.1節推導 overprovision 與最佳 T-block 個數之間的成本關係, 並透過常數係數的校正, 得到 overprovision 對最佳 T-block 個數的公式函數。3.2.2節推導 cache size 與最佳 T-block 個數之間的成本關係, 透過常數係數的校正, 得到 cache size 對最佳 T-block 個數的公式函數。3.2.3

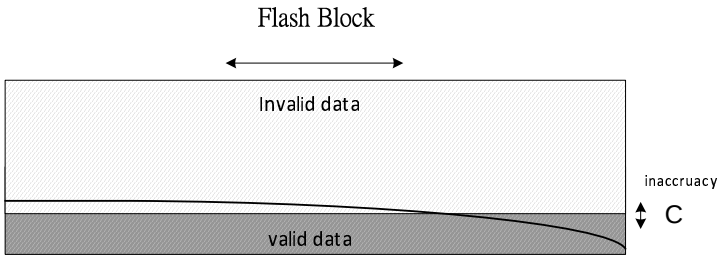



圖 12: GC cost model

節介紹如何用3.2.1節和3.2.2節所得到的兩個公式函數去計算各種不同設定的最佳 T-block 個數。

### 3.2.1 Overprovision vs. optimal T-block number

首先, 我們分析 overprovision 所影響的 FTL 寫入成本。Overprovision 爲一開始預先保留給系統, 用來做更新資料或對映資訊的 spare blocks。當全部使用完時, 不但會影響 GC 所挑選出來犧牲區塊內的有效資料數, 也會影響 GC 的速度, 於是間接影響對映資訊更新的速度, 造成最佳 T-block 個數之變動。所以我們要推導 overprovision 與 GC cost 的成本關係。

如圖12, 因爲在 random workload 下, 其有效資料頁可以假設會平均分佈在 flash block 上, 所以 GC cost 與 overprovision 可以說是反比的關係, 即 GC cost = 1 / overprovision。然而在實際的實驗中, 並非這麼理想, 會有個誤差[6]存在。假設c 爲誤差, 則因爲 block 個數若越多, 有效資料在裡面待得越久, c 就有機會越多。雖然 c 表面上跟時間有關, 但事實上是跟有多少 spare blocks 有關, 而有多少 spare blocks 是一個常數項特性, 所以我們認爲其誤差也是常數項係數。至於指數項也是非單純的一次, 假設指數項是 d 次, 所以可以推導出 overprovision 與 GC 成本的模型M1:

$$GC\ cost = c \times overprovision^{-d} \tag{M1}$$

我們拿公式M1去和實際測量的數據互相驗證, 其結果也和[6]的論述一致, overprovision 若越少, 則GC 的成本 (copy valid pages) 就越多; 反之, overprovision 越多, GC 成本就越低。

分析 block partition 中的成本, 我們可以將所有的成本分爲五種: (A) GC 時 copy valid T-pages。(B) GC 時 copy valid D-pages。(C) GC copy valid D-pages 要更新的 T-pages。(D) Mapping cache 寫回的 T-pages。(E) User write 更新的 D-pages。Page read、block erase 的成本和 page write 成本是呈正比關係的, 因此我們以 page write 爲主即可。由於 User write 的成本是一固定值, 而且在固定 cache size 的實驗下, mapping cache 寫回 T-pages 的

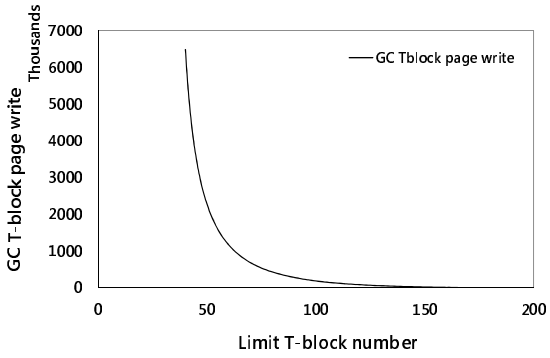



圖 13: GC T-block copy T-page cost. (The x-axis is T-block number)

次數也因而固定, 所以成本 (D) 和成本 (E) 都是不變量。然而, 其他三種成本我們都可套用 GC 成本的模型 (M1), 假設 (A)、(B)、(C) 的成本分別如下:

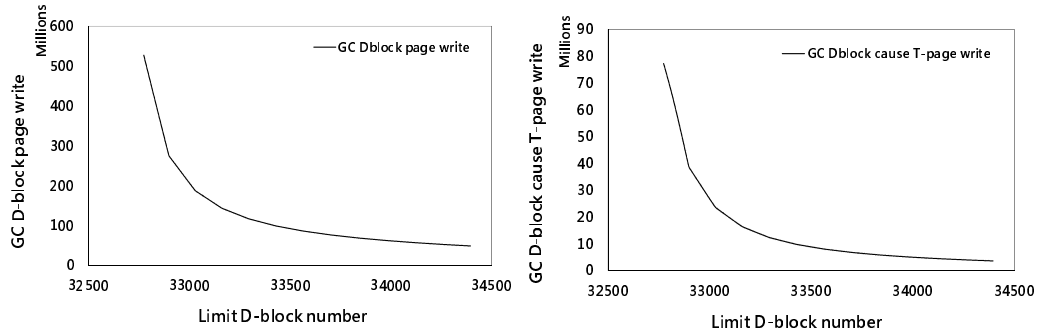
$$f_T(N_t) = c_1 \times N_t^{-c_2} \tag{A}$$

$$f_D(N_t) = c_3 \times N_d^{-c_4} = c_3 \times (N_{Ovr} - N_t)^{-c_4} \tag{B}$$

$$f_G(N_t) = c_5 \times N_d^{-c_4} = c_5 \times (N_{Ovr} - N_t)^{-c_4} \tag{C}$$

其中, 我們定義 $N_t$ 爲分給 T-block 的 spare 個數, $N_d$ 爲分給 D-block 的 spare 個數, $N_{Ovr}$ 爲給定 overprovision 的 block 個數, 且 $N_{Ovr} = N_t + N_d$。第一個函數 $f_T(N_t)$ 爲 GC 挑到 T-block 之下, 計算 copy valid T-pages 的成本函數。如圖13, T-block 個數給的太少的話, GC T-block 的成本會很高, 而當 T-block 給到一定數量程度 (例如: 150個) 的時候, 成本幾乎爲零, 即每次都會挑到沒有 valid page 的 T-block, 故式子 (A) 就像之前我們推導 overprovision 和 GC cost 的模型 (M1) 一樣; 第二個函數 $f_D(N_t)$ 則是 GC 挑到 D-block 之下, 計算 copy valid D-pages 的成本函數, 如圖14的圖 a, 在給定 $N_d$ 之下, 其成本函數與模型 (M1) 也是一致的, 而 $N_d$ 隱含著 $N_{Ovr} - N_t$ , 所以我們在這邊可替換成 $N_{Ovr} - N_t$, 如式子 (B); 第三個函數 $f_G(N_t)$ 是 GC 挑到 D-block 之下, 計算要寫回 T-pages 的成本函數, 如圖14的圖 b。因爲 cache size 固定, 其成本會和函數 $f_D$ 只差一個常數倍, 而且一樣將 $N_d$ 替換成 $N_{Ovr} - N_t$ 後, 可以得到式子 (C)。

最後, 我們將這三種成本加總起來, 如圖15的粗虛線條所示。當整體的 overhead 過了最低點之後, 只會上升不會再下降, 所以有一個整體寫入效能的最佳 T-block 個數設定值。我們定義函數 $g(N_{Ovr}, N_t)$ 爲所有成本總和的函數, 如圖15的粗虛線條, 它可計算不同的 T-block 個數




(a) (b)

圖 14: (a) GC D-block copy D-page cost. (b) GC D-block cause T-pages write cost. (The x-axis is D-block number and $N_d + N_t = N_{Ovr}$)

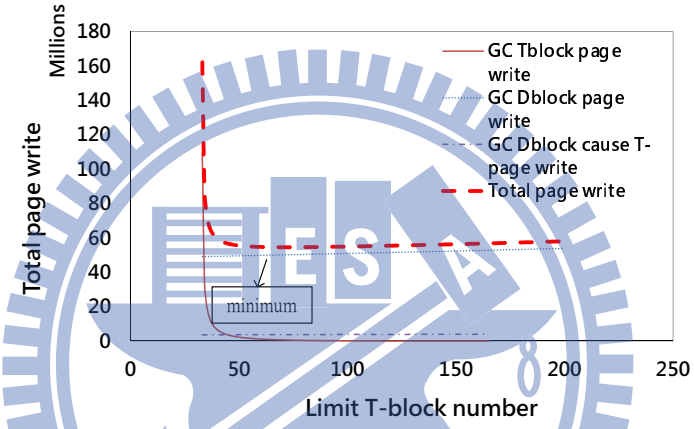



圖 15: Total costs for overprovision factor

下形成的 overhead:

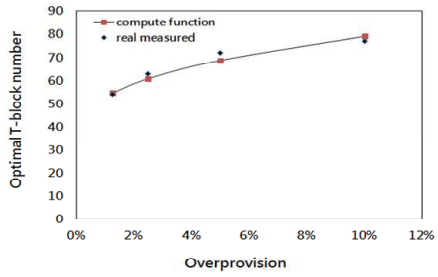
$$g(N_{Ovr}, N_t) = c_1 \times N_t^{-c_2} + c_3 \times (N_{Ovr} - N_t)^{-c_4} + c_5 \times (N_{Ovr} - N_t)^{-c_4}$$

要找出最佳 T-block 個數可十分迅速, 只要代入簡單的微分公式求斜率爲0的 $N_t$ 值:

$$\frac{d}{dN_t} g(N_{Ovr}, N_t) = 0$$

$$-c_1 c_2 \times N_t^{-c_2-1} + c_3 c_4 \times (N_{Ovr} - N_t)^{-c_4-1} + c_4 c_5 \times (N_{Ovr} - N_t)^{-c_4-1} = 0 \qquad \text{(E1)}$$

左式經由廣義二項式定理可化成一個 $C(k,r)N_{Ovr}^r N_t^{k-r}$ 無窮級數, 令這個二次項級數爲 $H(N_{Ovr}, N_t)$。若存在一根 $R$ , 表示能將 (E1) 式子因式分解爲 $(N_t - R)(H'(N_{Ovr}, N_t)) = 0$ 。經由指數律可知, $R$ 的形式應該爲 $N_{Ovr}$ 的級數, 且 $H'(N_{Ovr}, N_t)$ 中 $N_{Ovr}$ 的最高冪次爲 constant, 故 $R$ 中 $N_{Ovr}$ 的最高冪次也會是 constant。因此, 在合理的 $N_{Ovr}$ 範圍內, 我們以 $a(N_{Ovr})^b + c$




$$F_o(ovr)_{cache50\%} = 20.904x^{0.351} + 32$$

圖 16: Compute optimal T-block number for overprovision

之形式去逼近 $N_t$ 的解, 其中$a$ 、$b$ 、$c$ 都是 constant:

$$N_t = a(N_{Ovr})^b + c \tag{E2}$$

所以, 我們能推導出最佳 T-block 個數與 overprovision 之間的關係 (E2), 這些公式的常數係數涵蓋了數個微小的影響。只要再用少數的實際數據固定出 $a$ 、$b$ 、$c$ 即可。

接下來我們用少數幾組不同的 overprovision 參數去做實驗, 根據成本公式 (E2), 我們使用乘冪迴歸 (Power Regression) 來做迴歸分析。於是可以得到 overprovision 和最佳 T-block 個數之間的關係其係數校正的結果, 如表1:

| cache size \ overprovision | 1.25% | 2.5% | 5% | 10% |
|---|---|---|---|---|
| 50% | $T_{1.25\%,50\%}$ | $T_{2.5\%,50\%}$ | $T_{5\%,50\%}$ | $T_{10\%,50\%}$ |

表 1: Coefficient calibration for $F_o(ovr.)_{cache50\%}$

我們設定了 overprovision 1.25%、2.5%、5%、10%, 且 cache size 固定爲全存的50%, 其他條件皆不變; 假設 $T_{ovr,\,cache}$ 爲在 overprovision 和 cache size 設定下所得到的 T-block 個數, 利用這四個點可以固定出成本公式的係數, 於是得到一條有關不同 overprovision 與最佳 T-block 個數的函數, 如圖16, 而此函數是在 cache size 固定在50%下所計算出來的, 稱之爲:

$$F_o(ovr.)_{cache50\%} \tag{$F_o$}$$

如圖16, 我們得到函數 $F_o(ovr.)_{cache50\%} = 20.904(ovr.)^{0.351} + 32$ , 和推導的公式 (E2) 一樣。利用此公式函數, 若輸入不同的 overprovision, 即可計算出它的最佳 T-block 個數, 我們在第3.2.3節會說明如何使用此公式來計算更多其他的最佳 T-block 個數。

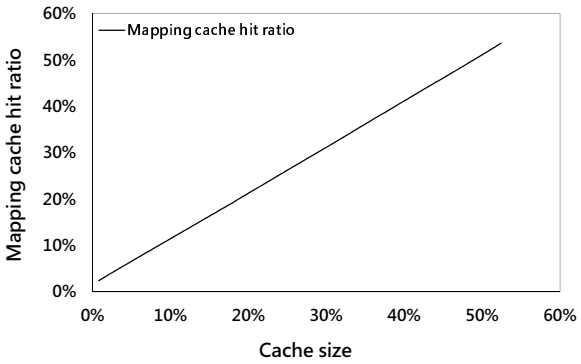



圖 17: Maping cache hit ratio for random trace

**3.2.2** Cache size vs. optimal T-block number

另一個影響因子是 cache size, 它的大小直接地影響對映資訊更新的頻率, 所以會改變最佳 T-block 個數。我們一樣將所有成本分爲五種: (A) GC 時 copy valid T-pages。(B) GC 時 copy valid D-pages。(C) GC copy valid D-pages 要更新的 T-pages。(D) Mapping cache 寫回的 T-pages。(E) User write 更新的 D-pages。其中, 成本 (E) 爲不變量, 所以只需要考慮其他四種成本, 成本 (A)、(B)、(C) 於3.2.2節有介紹過, 分別對應式子 (A)、式子 (B)、式子 (C)。而在這裡我們要再加上成本 (D) mapping cache 寫回次數的成本, 如圖17, 因爲在 random workload 下, 每個 page 被 access 的次數很平均, 所以 mapping cache 的擊中率會跟 cache size 成正比, 即使是使用 LRU 演算法碰到 random 也是會退化成 FIFO。所以我們假設 (D) mapping cache 寫回的 T-pages 的成本如下:

$$f_C(N_c) = -c_7 \times N_c + c_8 \tag{D}$$

其中, 我們定義 $N_c$ 爲 cache size, 函數 $f_C(N_c)$ 是計算 mapping cache 寫回 T-pages 的成本, 如式子D, 它與 cache size 爲線性關係。此外, 由於 overprovision 固定, $N_{Ovr}$ 可視爲一常數 (我們以 $c_6$ 表示), 而 cache size 改變下, 會影響 T-page 寫回的速率, 故成本 (A) 和成本 (C) 應該將它們乘上寫回 T-page 速率的倍數, 成本 (B) 則是不用乘上 cache 因子, 因爲它是有關寫 D-pages 的成本, 我們以 $f_C(N_c)$ 表達此 cache 因子, 將它乘上成本 (A) 和 成本 (C)[1]。最後我們將所有成本加總起來, 如圖18, 我們定義圖15粗虛線的函數爲 $g(N_c, N_t)$ , 它是所有成本總和的函數, 可計算不同的 T-block 個數之下形成的 overhead:

$$g(N_c, N_t) = (-c_7 \times N_c + c_8) \times (c_1 \times N_t^{-c_2} + c_5 \times (c_6 - N_t)^{-c_4}) \tag{E3}$$

[1]在3.2.1節沒有將成本 (A) 和成本 (C) 乘上 cache 因子, 是因爲改變 overprovision 因子時, 我們固定 cache size, 所以可以假設 cache 擊中率皆相同, 其成本差異只會在一個常數倍, 式子 (A) 的 $c_1$ 和式子 (C) 的 $c_5$ 兩個常數其實已隱含了 cache 因子。

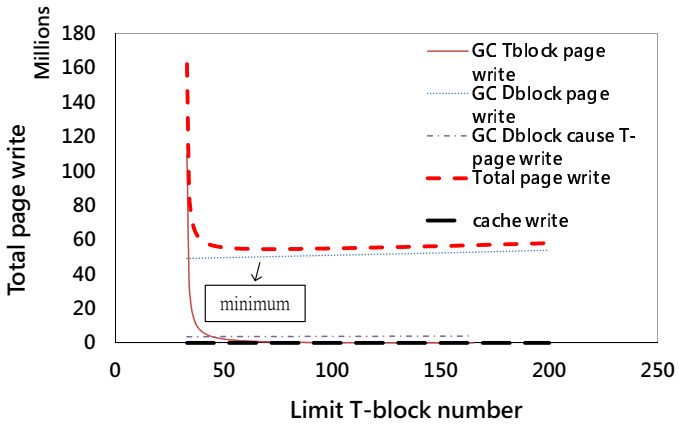



圖 18: Total costs for cache size factor

$$+c_3 \times (c_6 - N_t)^{-c_4} + (-c_7 \times N_c + c_8)$$

我們一樣代入微分公式後, 找出最佳 T-block 個數:

$$\frac{d}{dN_t} g(N_c, N_t) = 0$$

$$c_1 c_2 c_7 \times N_c \times N_t^{-c_2-1} - c_1 c_2 c_8 \times N_t^{-c_2-1} + c_3 c_4 \times (c_6 - N_t)^{-c_4-1}$$

$$-c_4 c_5 c_7 \times N_c \times (c_6 - N_t)^{-c_4-1} + c_4 c_5 c_8 \times (c_6 - N_t)^{-c_4-1} = 0 \qquad \text{(E4)}$$

左式經由廣義二項式定理可化成一個 $C(k,r)c_6^r N_t^{k-r} + C(k,r)N_c N_t^{k-r}$ 無窮級數。令這個二次項級數爲 $H(N_c, N_t)$, 則若存在一根 $R$ , 表示我們能將 (E4) 式子因式分解爲 $(N_t - R)(H'(N_c, N_t)) = 0$ 。由指數律可知, $R$ 的形式應該爲 $N_c$ 的級數, 且 $H'(N_c, N_t)$ 中 $N_c$ 的最高冪次爲 constant, 故 $R$ 中 $N_c$ 的最高冪次也會是 constant。所以, 在合理的 $N_c$ 範圍內, 我們可用 $a(N_c)^b + c$ 之形式去逼近 $N_t$ 的解, 其中$a$ 、$b$ 、$c$ 都是 constant:

$$N_t = a(N_c)^b + c \qquad \text{(E5)}$$

於是根據成本公式我們能推導出最佳 T-block 個數與 cache size 之間的關係式 (E5), 之後我們要再對這些乘數係數做校正。

接下來我們用少數幾組不同的 cache size 參數去做實驗, 根據成本公式 (E5), 我們使用乘冪迴歸 (Power Regression) 來做迴歸分析。於是我們得到 cache size 與最佳 T-block 個數之間的成本關係其係數校正的結果, 如表2:

| overprovision \ cache size | 7.64% | 17.56% | 32.5% | 45% |
|---|---|---|---|---|
| 12% | $T_{12\%,7.64\%}$ | $T_{12\%,17.56\%}$ | $T_{12\%,32.5\%}$ | $T_{12\%,45\%}$ |

表 2: Coefficient calibration for $F_c(cache)_{ovr.12\%}$

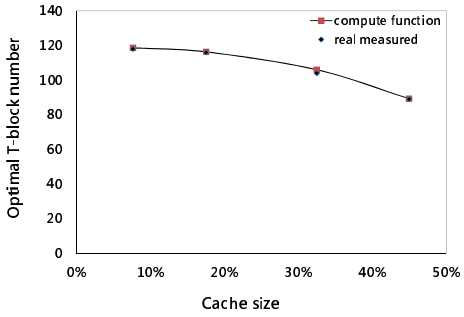



圖 19: Compute optimal T-block number for cache size

我們設定了 cache size 7.64%、17.56%、32.5%、45%, 且 overprovision 固定爲12%, 其他條件皆不變; 假設 $T_{ovr,\,cache}$ 爲在 overprovision 和 cache size 設定下所得到的 T-block 個數, 利用這四個點可以固定出成本公式 (E5) 的係數, 於是得到一條有關不同 cache size 與最佳 T-block 個數的函數。如圖19, 此函數是在 overprovision 固定在12%下所計算出來的, 我們稱爲:

$$F_c(cache)_{ovr.12\%} \qquad (F_c)$$

我們發現此函數若 overprovision 在12%, 則 $F_c(cache)_{ovr.12\%} = -0.00188 \times cache^{2.54} + 119$ , 和推導的公式 (E5) 一樣。於是, 我們利用 $F_c$ , 在輸入不同的 cache size 下, 可立即計算出最佳 T-block 個數, 在3.2.3節會說明如何使用此公式來計算其他更多的最佳 T-block 個數。

### **3.2.3** Compute the optimal T:D ratio

在3.2.1和3.2.2節我們獲得函數 $F_o$ 以及 $F_c$ , 然而這兩個公式是在特定固定的條件下所計算的, $F_o(ovr.)_{cache50\%}$ 爲 cache size 固定在50%, 而 $F_c(cache)_{ovr.12\%}$ 是 overprovision 固定在12%, 如果想要計算不同 overprovision 和不同 cache size 組合的最佳 T-block 個數, 需要找到它們和 $F_o$ 、$F_c$ 的關係。

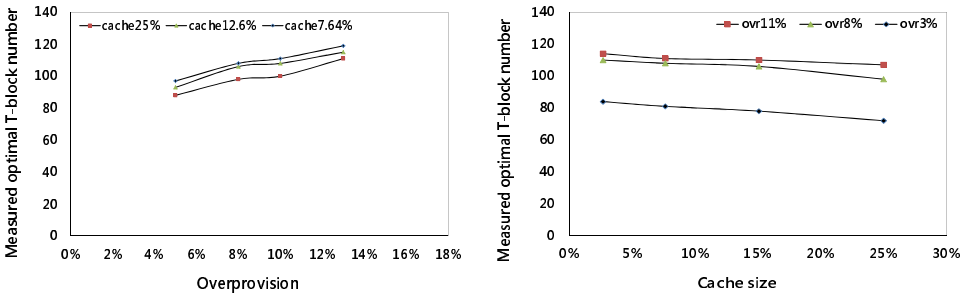



圖 20: Measured optimal T-block number for overprovision and cache size

我們定義 $F(ovr,\ cache)$ 爲輸入不同的 overprovision 以及 cache size 下, 可得到最佳 T-block 個數的函數:

$$F(ovr,\ cache) \tag{F3}$$

觀察實際的實驗結果, 可以發現 $F(ovr, 25\%)$ 的最佳 T-block 函數, 幾乎與 $F(ovr, 50\%)$ 函數乘上一個常數係數幾乎相同, 如圖20的左邊, 我們在合理的 cache size 範圍內 (全存的0.2% ∼ 25%) 取三種 cache size 分別是7.64%, 12.6%以及25%, 並且實際測量在 overprovision 變大 (分別是5%, 8%, 10%, 13%) 時, 其最佳 T-block 個數的趨勢變化, 從圖上可知, 只要是同樣程度 overprovision 的改變, 其最佳 T-block 個數的趨勢變化就會很接近 (約只差一個常數倍數), 例如在不同 cache size 固定之下, 只要 overprovision 從5%變成8%, 最佳 T-block 個數幾乎都會一樣地成長1.5倍。而 $F_t(5\%, cache)$ 的最佳 T-block 函數也會和 $F_t(11\%, cache)$ 函數乘上一個常數係數幾乎相同, 如圖20的右邊, 我們在合理的 overprovision 範圍內 (1%∼20%) 取三種 overprovision 分別是3%, 8%以及11%, 並實際測量在 cache size 變大 (分別是2.68%, 7.64%, 15.08%, 25%) 時, 其最佳 T-block 個數的趨勢變化。從圖上可知, 不同的 overprovision 固定下, 其最佳 T-block 個數也約只差一個常數倍數。

而在成本模型中我們也能得到同樣的結論。在3.2.2節提到的式子 (E3), 若將 $N_{Ovr}$ 的變數不要視爲常數 $c_6$ , 微分之後會變成

$$c_1c_2c_7 \times N_c \times N_t^{-c_2-1} - c_1c_2c_8 \times N_t^{-c_2-1} + c_3c_4c_8 \times (N_{Ovr} - N_t)^{-c_4-1}$$

$$-c_3c_4c_7 \times N_c \times (N_{Ovr} - N_t)^{-c_4-1} + c_3c_4 \times (N_{Ovr} - N_t)^{-c_4-1} = 0$$

其中會出現 $N_cN_{Ovr}N_t$ 項, 因此將它提出來做因式分解會變成 $(N_t - a(N_c)^b\ (N_{Ovr})^c + d)\ (H'(N_c, N_{Ovr}, N_t)) = 0$ , 故我們可以用 $a(N_c)^b(N_{Ovr})^c + d$ 之形式去逼近 $N_t$ 的解, 其中 $a$ 、$b$ 、$c$

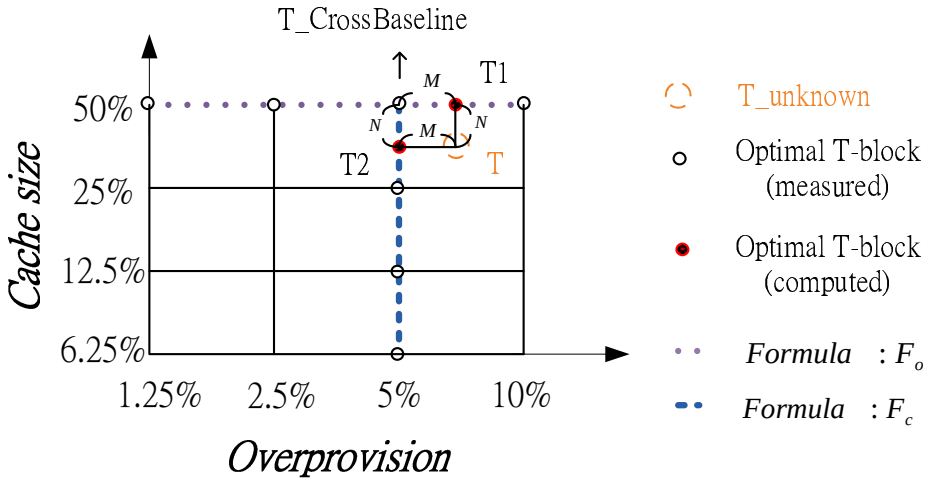



圖 21: Compute the optimal T:D ratio by M or N multiple

、$d$ 都是 constant:

$$N_t = a(N_c)^b(N_{Ovr})^c + d$$

因此, 當我們固定 overprovision, 則 $N_t$ 對於 $N_c$ 的解, 彼此都只是差常數 $(N_{Ovr})^{c_1}/(N_{Ovr})^{c_2}$ 倍而已。當固定 cache size, 則 $N_t$ 對於 $N_{Ovr}$ 的解, 彼此也都只是差常數 $(N_c)^{c_3}/(N_c)^{c_4}$ 倍

於是, 我們經由以上的推論即可去計算 runtime 的 T/D-block 最佳比例, 如圖21, 因爲 $F_o$ 和 $F_c$ 可視爲互相獨立的成本公式, 不同 cache size 固定下的許多 $F_o$ 以及不同 overprovision 固定下的許多 $F_c$ 差別都只在一個常數倍數。所以我們可以經由兩個步驟即可得到不同 overprovision 和 cache size 的最佳 T-block 個數。

假設我們要計算 overprovision 7%, cache size 30%的最佳 T-block 個數, 且已知兩條公式 $F_o(ovr.)_{cache50\%}$ 以及 $F_c(cache)_{ovr.5\%}$。第一步是先算出 $F_c(30\%)_{ovr.5\%}$ 與 $F_c(50\%)_{ovr.5\%}$ 之間的常數倍數: $N$ , 而 $N$ 代表 overprovision 同樣在5%之下, 乘上不同 cache size 所差異的常數倍數因子。我們從以上可知, overprovision 若同樣在7%之下, 則 $F_o(7\%)_{cache30\%}$ 與 $F_o(7\%)_{cache50\%}$ 也會乘上相同的常數倍數 $N$ , 於是, 第二步使用 $F_o(7\%)_{cache50\%}$ 乘上 $N$ , 即爲所求。故我們的式子可寫成如下:

$$F_o(7\%)_{cache30\%} = \frac{F_c(30\%)_{ovr.5\%}}{F_c(50\%)_{ovr.5\%}} \times F_o(7\%)_{cache50\%}$$

如圖21所示, 即爲

$$T = \frac{T_2}{T_{CrossBaseline}} \times T_1 \qquad \text{(E6)}$$

同理可得, 我們也能先計算 $F_o(7\%)_{cache50\%}$ 與 $F_o(5\%)_{cache50\%}$ 之間的常數倍數: $M$ 。其中, $M$ 代表 cache size 同樣在50%之下, 乘上不同 overprovision 所差異的常數倍數因子。從以上可知, cache size 若同樣在30%之下, 則 $F_c(30\%)_{ovr.7\%}$ 與 $F_c(30\%)_{ovr.5\%}$ 也會乘上相同的常數倍數 $M$ , 於是使用 $F_c(30\%)_{ovr.5\%}$ 乘上 $M$ , 即爲所求。故我們的式子可寫成如下:

$$F_c(30\%)_{ovr.7\%} = \frac{F_o(7\%)_{cache50\%}}{F_o(5\%)_{cache50\%}} \times F_c(30\%)_{ovr.5\%}$$

如圖21所示, 其結果與式子E6一樣:

$$T = \frac{T_1}{T_{CrossBaseline}} \times T_2 \tag{E7}$$

由式子E6和式子E7可證明兩種方式運算後可得到一樣的結果, $F_o$和$F_c$可視爲互相獨立的成本公式, 因此以上兩種計算方式都能適用, 且不必有計算順序的關係。然而, 計算出來的個數還是會和實際上的有點小差距, 經過我們的測試, 在合理的 overprovision 與 cache size 範圍內, 其最佳 T-block 個數與計算上的都不會差很多, 且跟它們的 IOPS 比較起來其實差異非常小, IOPS 都相差在0.5%以內, 故我們的方式依然適用。

## 3.3 Practical issues

由於 flash memory 在實際應用上, 還有一些其他因子要考慮, 例如可動態改變的因子以及互相轉換的因子, 因此, 我們要說明如何針對這些因子去做相應的策略, 讓我們的方法能夠適用於線上調整。此外, 在3.3.2節會說明, 實際上在不同的 workloads 下, mapping cache 不只要有好的 cache policy, 更要有好的 block patitioning 方法, 廠商可能會覺得爲什麼即使 mapping cache 加大, 也不怎麼有效, 這是因爲他們沒有考慮到資料冷熱引起的原因, 造成了廠商的盲點, 所以我們會說明如何在做完 pattern detection 之後, 利用調整 T-block 個數的策略來將效能最佳化, 使之能和 mapping cache 相輔相成。

### 3.3.1 Dynamic factors and convertiable factors

在前面的章節中, 我們提出了針對隨機存取 workload 做效能最佳化的 block partition 方法, 由於它的可分析性, 我們更進一步討論一些環境設定的因子, 包含可動態改變的因子以及可互相轉換的因子。可動態改變的因子有 write span 和 cache size 兩種, 它們都會改變最佳

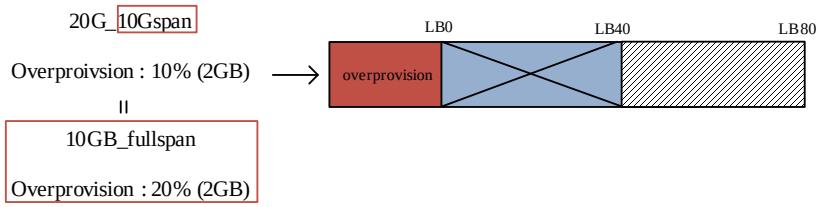



圖 22: Convertible foctor: span, overprovision

| Disk size | 16GB | 20GB | 40GB | 80GB |
|---|---|---|---|---|
| Initial T-block number | 32 | 40 | 80 | 160 |
| Optimal T-block number | 77 | 101 | 204 | 385 |
| Optimal IOPS | 210.3645 | 210.1407 | 210.0289 | 210.038 |

表 3: Linear relation of disk size and optimal T-block number

T/D-blocks 比例。Write span 定義爲在固定的 span size 之下進行的隨機寫入行爲, span 在邏輯上可不連續, 而在 span 之外的資料和其對映資料都不會去存取更新, 因此若已經知道 span size 下, 可視爲縮小範圍的隨機寫入, 其中 overprovision 和 span size 可以互換, 如圖22, 以 20 GB disk size 打10 GB span size 爲例, 其原本的 overprovision(紅色區塊) 是10%(2 GB), 而它只會在10 GB 的空間內 (斜線區塊) 存取, 等於是在10 GB disk size 打 full span 之下, overprovision 有2 GB, 但對於10 GB disk size 卻等於是20%, 所以, write span 不但可動態改變同時也是可轉換成 overprovision 的因子。

因爲 RAM 不只是用來當 mapping cache, 也可用來當作 write buffer, mapping cache size 能在不同的情況下做適當的調整給 write buffer 的空間來利用, 所以它是可動態改變的因子。Cache size 要調整方法以及公式是之前所提過的, 所以不管是 write span 或 cache size 改變時, 我們都能立即做線上調整最佳 T-block 的個數。

互相轉換的因子有 span size 和 overprovision, 以及 disk size 和 block size。Span 因子我們已介紹過, 而 disk size 和 block size 因子則是可以透過線性關係來計算最佳 T-block 個數, 我們以實際例子爲例, 在不同的 disk size 下, 使用相同密度的 random workload, 其中 overprovision 和 cache size 都設爲10%及50%, page size 皆爲4 KB, 結果如表3所示, 當 Disk size 從16 GB 變成80 GB, page size 不變, 表示 total block 個數變成五倍, 因爲最佳 T:D 比例會保持不變, 最佳 T-block 個數也變爲五倍。Block size 則也是一樣, 當 block size 從512 KB 變成1 MB, total block 個數變爲$\frac{1}{2}$倍, 則最佳T-block 個數也變爲$\frac{1}{2}$倍。

| Block size | 256KB | 512KB | 1MB |
|---|---|---|---|
| Optimal T-block number | 223 | 101 | 49 |
| Optimal IOPS | 213.3404 | 210.1407 | 208.028 |

表 4: Linear relation of block size and optimal T-block number

所以, 上述的互換以及動態因子我們都可透過簡單的線性處理以及線上調整的策略爲它們設定出最佳 T-block 個數。而系統環境的這些因子的設定組合有上千種可能, 暴力查表法則要一個一個去預先模擬得到最佳的 T/D-blocks 比例, 所以比起暴力法, 我們的方法所花的力氣可以來得少很多。

### 3.3.2 T-block number adjustment algorithm

在不同的 workloads 下, 對於 block partition 造成的效能變化也會有所不同, 所以我們也要提供適合的 T-blocks 調整的方法。典型的 workloads 有 sequential、PC 以及 random 三種。在 sequential 和 PC workload 中, 因爲它們都具有良好的空間區域性, 加上 GC 的活動並不強烈, 所以效能影響並不大。Sequential workload 只要給一些 T-block 即可, 而 PC workload 則需要再微調 T-block 個數; 在 random workload 中, 則因爲其 cache 擊中率很低,GC 活動很強烈, 所以效能會大受影響,T-block 個數造成其效能變化非常敏感, 此篇論文重點即在這方面之上, 在3.2節我們提供了計算最佳 T-block 個數的解決方法。

此外在這一小節, 我們也針對 pattern-aware 的方式, 利用 mapping cache 來使用合適的 fetch policy, 使得 mapping cache 在存取效能上更有效率。在 sequential workload 之下, 我們應該要將比較大的單位的對映表帶進 cache, 以增加 cache hit ratio, 減少讀取 mapping 資訊的機會, 我們一次帶一個 T-page 進入 cache, 使用 segment-level fetch; 而在 random workload 之下, 我們一次只將一個 mapping pair 帶進 cache, 以不增加 cache replacement 的頻率爲原則, 使用 pair-wise fetch, 這是因爲即使我們將較大單位的對映表帶進來, random 的存取特性下反而擊中的機會不會很高; PC workload 則是因爲包含比較強的空間區域性, 故我們以較大單位的對映表帶進 cache 會比較好。

在 mapping cache size 設定爲全存的50%下, 使用 LRU 的替換機制, 比較不同的 fetch policy, 如表5所示, random workload 使用 pair-wise 的 fetch policy 會比較好, 而 seqential workload 則是使用 segment-level policy 比較好。Random workload 在使用 segment-level,

| Workload \ Fetch-in unit | Segment-level | Pair-wise |
|---|---|---|
| IOmeter (random) | IOPS degrade 35% | IOPS degrade 3.7% |
| Multimedia (sequential) | IOPS degrade 0.12% | IOPS degrade 6.14% |

表 5: Performance for different fetch policy

cache size 爲全存的50%下, Miss ratio 爲50%, 因此每帶進兩個 segment 單位, 就會替換一個 segment, 故 IOPS 降低了35%(約33%); 而使用 pair-wise fetch 則可降低 replacement 發生的頻率, 因此效果會比較好, IOPS 只降低3.7%。Sequential workload 在使用 segment-level 下, 則可利用空間區域性, 因此 IOPS 只降低了0.12%, 而 pair-wise 則常發生 miss 所以 IOPS 降低了6.14%。

總結這一小節來說, 我們能用 patter-aware 的方法, 使得 mappin cahce 和 block partition 可以同時有效利用。光是 cache size 加大, 若是沒考慮到資料冷熱所引起效能的低落的話, 可能不會提升很多效能。因此, 針對典型的三種 workloads ( Sequential、PC、Random ), 我們不但提供 mapping cache 良好的機制, 也有適當的 block patition 設定, 強化了實際應用的可行性。

# 第四章 Experimental results

## 4.1 Experimental setup and performance metrics

在我們的實驗中, 撰寫了以 page-level mapping 爲基礎, 使用 mapping table cache (簡稱 table cache) 的 FTL simulator。第一個實作是使用 greedy 演算法加上 table cache 的 FTL 稱之爲 PL+greedy+tc。第二個是本論文的實作, 使用 approximate optimal block partition 加上 table cache 的 FTL 稱之爲 Aobp。最後, 我們的方法會和使用暴力法 (brute-force table generation) 測量的方式來做比較。

我們使用大容量的 NAND flash memory 的 geometry, 包含16 GB、20 GB、40 GB、80 GB 等規格, 其詳細 geometry 內容如表6:

| | |
|---|---|
| Sector size | 512 B |
| Page size | 4 KB |
| Block size | 512 KB |
| Total size | 16GB 20GB 40GB 80GB |

表 6: Flash memory geometry

實驗測試使用 RND workload, 它是我們從工業標準效能評比工具 IOmeter 蒐集而來, 其中的設定爲4 KB request size 和100% random write, 預設是 full span。在不同的 volume size 下, 爲了實驗公平起見, 我們會保持寫入的密度相同且夠散亂, 假設20 GB 的 random workload 總共有500萬筆 requests, 則40 GB 的總共要有1000萬筆的 requests, 詳細的內容如下表7:

| OS | Windows XP | | | |
|---|---|---|---|---|
| File system | NTFS | | | |
| Disk volume size | 16 GB | 20 GB | 40 GB | 80 GB |
| Total writes | 18.6 GB | 23.2 GB | 46.4 GB | 92.9 GB |

表 7: Workload

Mapping cache 實作上主要使用 LRU+pair-wise 的替換和帶入機制, 以16 GB disk size 爲例, 對映表若全存的 cache size 需要16 MB, 如果只存一半 (50%) , 因爲我們使用兩個 pointers 佔了8 bytes 來實作 LRU, 對映資訊的每個 pair 也需要8 bytes, 加上我們會用1 bit 拿來

| Cache | 5.16% | 10.12% | 15.08% | 22.52% | 50% |
|---|---|---|---|---|---|
| Total size | 3.95 MB | 7.15 MB | 10.35 MB | 15.15 MB | 32.25 MB |

表 8: Cache size

| Overprovision | 1% | 4% | 7% | 10% |
|---|---|---|---|---|
| Total flash size | 16.16 GB | 16.64 GB | 17.12 GB | 17.6 GB |

表 9: Overprovision size

標示 clean/dirty state (clean 表示沒有更新的狀態不用寫回, dirty 表示有更新的狀態需要寫回) , 與16 KB 的 Global Translation Table, 所以50%的 cache size 反而需要32.25 MB, 其他的儲存比例所需要的 cache size 如表8。Overprovision 爲 flash memory 提供 spare 空間的部分, 以16 GB disk volume size 爲例, 若 overprovision 爲10%, 則 total flash size 爲 $16 \times (1 + 10\%) = 17.6\,GB$ , 其他的比例如表9。

我們的 performance metric 以 IOPS 爲主, 它可以有效展現 garbage collection 和 address translation 的成本, 然而它並不包括 requests 在 I/O driver queues 中的 delays。IOPS 的計算方式如下, 其中的縮寫名稱對照如表10:

$$IOPS = \frac{RNs}{PRCs \times TPR(S) + PWCs \times TPW(S) + BECs \times TBE(S)}$$

| Abbreviation | Name | Value |
|---|---|---|
| RN | Request Numbers | |
| PRC | Page Read Counts | |
| PWC | Page Write Counts | |
| BEC | Block Erase Counts | |
| TPR | Time of Page Read | 60 us |
| TPW | Time of Page Write | 800 us |
| TBE | Time of Block Erase | 1500 us |

表 10: IOPS performance metrics

## 4.2 Performance evaluation and verification

針對不同的環境設定, 我們在此節比較 Aobp、暴力法和 PL+greedy+tc 的實際效能, 實

| 名稱 | 方法 |
|---|---|
| Approximate optimal block partition | 利用成本公式, 一個影響因子以四組數據作爲係數的校正, 全部使用$7 \times 168 \times 21GB$的資料測量 |
| 暴力法 (brute-force table generation) | 將所有的最佳T-block 個數逐一分別測量, 而每一個最佳 T-block 個數使用$168 \times 21GB$的資料測量 (如圖23) |
| PL+greedy+tc | 使用table cache 和 greedy 的 page-level mapping FTL |

表 11: Implement description

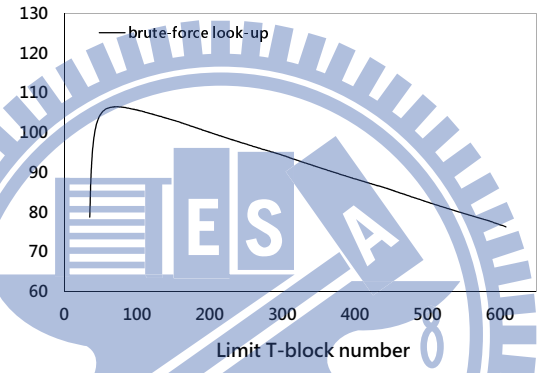



圖 23: brute-force look for optimal T-block number

驗以16 GB 的 random workload 爲主, Mapping cache 使用 pair-wise fetch 和 LRU replacement, 表11列出所有方法。

Approximate optimal block partition 爲本論文使用的方法, 由於我們推導出成本公式, 則每個影響因子只需要四組數據來做係數的校正, 所以可省下很多的資料測量。暴力法 (brute-force table generation) 則如圖23, 假設 T-block 個數從初始個數32個慢慢增加, 到200個 T-block 個數停下來可以找到最佳 T-block 個數, 用一個 workload 當作基本的測量單位的話, 我們需要 $168 \times 21GB$ 的資料測量, 估計完成要 $(\frac{168 \times 21GB}{4KB}) \times 800us = 8.6$ 天, 因此使用我們的方法可以省下力氣做這些大量資料的測量。而 PL+greedy+tc 是使用 greedy 加上 table cache 的 page-level mapping FTL, 其架構內容於2.4節的 DFTL 中介紹過。

首先,Aobp 的做法是固定出 overprovision 對於最佳 T-block 個數的公式函數和 cache size 對於最佳 T-block 個數的公式函數之係數, 其函數是分別從成本式子推導而來的。在合理

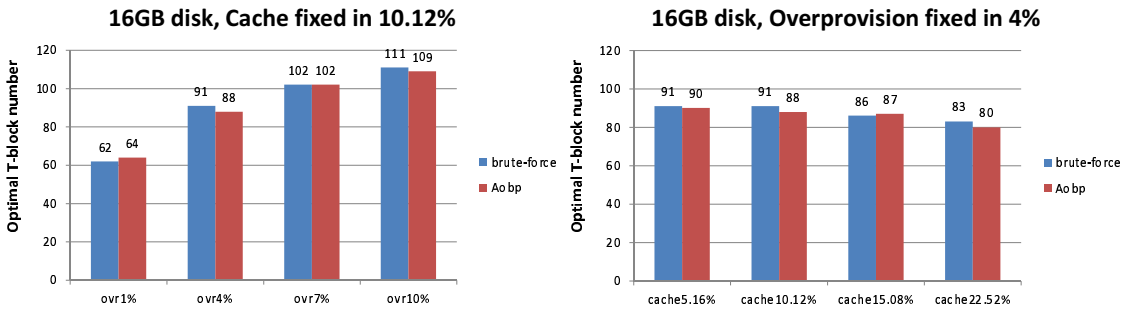



圖 24: T-block number verification

的 overprovision 範圍內, 我們選擇4個 overprovision 分別爲1%, 4%, 7%, 10%, Cache size 固定在10.12%, 我們可得到係數固定後的成本公式, 算出最佳 T-block 個數; 在 cache size 這邊, 我們選擇合理的 cache size 分別爲5.16%, 10.12%, 15.08%, 22.52%, Overprovision 固定在4%, 也能固定出成本公式的係數, 並算出最佳 T-block 個數。

如圖24, 我們利用函數公式$F_o$和$F_c$即可算出最佳 T-block 的個數, 圖24的紅色長條即爲 Aobp 算出來的最佳 T-block 個數, 其結果和暴力法 (藍色的長條) 測量的都十分接近。

最後, 我們用 Aobp 來比較暴力法和 PL+greedy+tc 的 IOPS 效能差異。暴力法在合理的 overprovision 與 cache size 範圍下, 將它們分爲十等份之後一個一個去測量, 而 Aobp 則是經由不同 overprovision 和 cache size 各四組的數據去計算。如圖25,Aobp 計算最佳 T-block 個數所得的 IOPS 和用暴力法一個一個實際測量的 IOPS 都相差不到0.1%, 效能幾乎都非常接近。此外,Aobp 和 PL+greedy+tc 兩種方法相比,Abop 的效能會有十分明顯的提升, 當 overprovision 越來越大時, 效能會差異更多, 原因在於若 overprovision 越大, 可分配的 T-block 個數就越多, 調整之後的效能也就越好。如果 cache size 改變 Aobp 效能的差異都不大, 皆可保持很好。

我們還要考量其他不同 overprovision 和 cache size 的排列組合, 但是經由實驗發現, 使用 Aobp 的公式所算出來的效能和暴力法的差異都十分接近, 舉一例子來說, 我們要算 over-provision 17%, cache size 25%的最佳 T-block 個數, 依照3.2.3節的計算方式, 我們從 $F_o(ovr.)_{cache10.12\%} = 63.968 \times ovr^{0.2341}$ 能夠算出 $T_1$ 爲124, 而從 $F_c(cache)_{ovr.4\%} = -0.0287 \times cache^2 + 0.2421 \times cache + 89.245$ 能夠算出 $T_2$ 爲77, $T_{cross}$ 爲91, 所以我們按照差距的位移計算 $T_{ovr.17\%,cache25\%} = T_2 + (T_1 - T_{cross}) = 110$ , 而實際測量到最佳 T-block 個數卻是138, 此爲我們所看到的最大差異。但是經由 Aobp 所得到的 IOPS 爲283.468202, 實際測量的

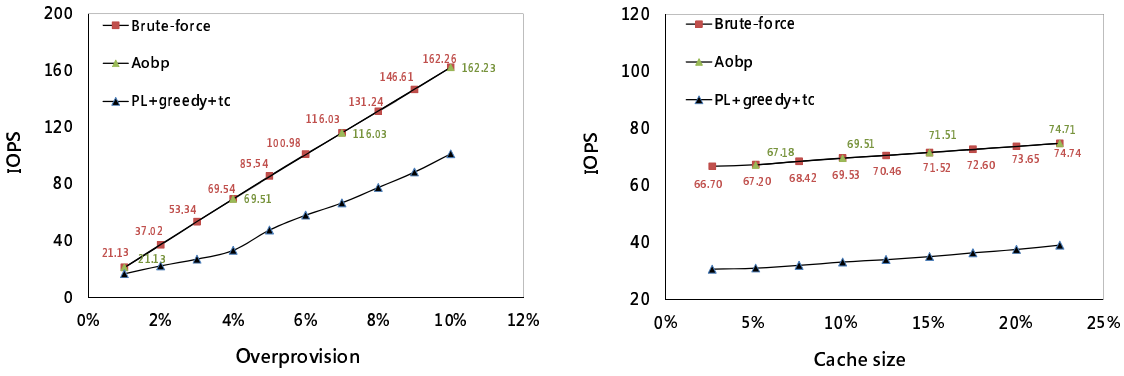



圖 25: Performance evaluation

IOPS 爲283.798425, 其誤差在0.11%, 依然非常的小, 故我們的方法在 overprovision 1% ∼ 20%, cache size 2.68% ∼ 25%的合理範圍內, 都可以適用的很好。

# 第五章 Conclusion

SSD 在採用高效能的 page-level mapping FTL, 要加載 mapping cache 下, 會有許多的問題需要考慮, 其中包括對映表的儲存管理以及資料冷熱的處理。我們使用的方式爲 two-level mapping cache, 儲存 user data 的區塊我們稱之爲 data blocks (D-blocks) , 而所有儲存對映資訊的區塊我們稱之爲 translation blocks (T-blocks) 。本研究發現這種對映表的儲存方法之下, 隱含了資料冷熱的問題, 特別是在 random workload。因爲 T-page 明顯的比 D-page 來得 Hot, 所以必須解決 GC 在處理回收熱資料所帶來不小的 overhead。

本研究針對可分析的 random workload 提出有效的解決方法:Aopb。Aobp 可以透過成本公式的分析, 在 flash memory 上找到資料區塊和位址轉換表區塊的最佳分配比例, 因此可達到 27.5% IOPS 效能的提升。此外, 本方法適用於線上調整的計算方法, 針對不同的可變因子或可轉換因子, 我們可以馬上爲系統設定最佳 T/D-blocks 的比例; 暴力查表法是一個一個去測量找到最佳 T/D-blocks 的比例, 但我們只需要測量幾組數據, 所以比起暴力法所花的力氣來得少很多。

最後, 我們的 Aobp 也能應用於一套 pattern-aware 的方法, 針對典型的三種 workloads ( Sequential、PC、Random ), 提供 mapping cache 良善的機制與適當的 T-block 設定, 強化實際上應用的可行性。Random workload 已經能透過我們的方法提升效能, 而 Sequential workload 只要給一些 T-block 個數即可。未來我們要分析 PC workload 的最佳設定, 並且更進一步研究 pattern detector 的方法來幫助所有效能的提升。

# 參考文獻


[1] W.H. Lin and L.P. Chang. Dual greedy: Adaptive garbage collection for page-mapping solid-state disks. 2012.

[2] A. Gupta, Y. Kim, and B. Urgaonkar. *DFTL: a flash translation layer employing demand-based selective caching of page-level address mappings*, volume 44. ACM, 2009.

[3] Z. Qin, Y. Wang, D. Liu, and Z. Shao. A two-level caching mechanism for demand-based page-level address mapping in nand flash memory storage systems. In *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2011 17th IEEE*, pages 157–166. IEEE, 2011.

[4] S. Jiang, L. Zhang, X.H. Yuan, H. Hu, and Y. Chen. S-ftl: An efficient address translation for flash memory by exploiting spatial locality. In *Mass Storage Systems and Technologies (MSST), 2011 IEEE 27th Symposium on*, pages 1–12. IEEE, 2011.

[5] J.U. Kang, H. Jo, J.S. Kim, and J. Lee. A superblock-based flash translation layer for nand flash memory. In *Proceedings of the 6th ACM & IEEE International conference on Embedded software*, pages 161–170. ACM, 2006.

[6] X.Y. Hu, E. Eleftheriou, R. Haas, I. Iliadis, and R. Pletka. Write amplification analysis in flash-based solid state drives. In *Proceedings of SYSTOR 2009: The Israeli Experimental Systems Conference*, page 10. ACM, 2009.