國立交通大學

資訊科學與工程研究所

碩 士 論 文

固 態 硬 碟 之 自 適 性 資 源 管 理 方 法

Mapping vs. Buffering: Adaptive Resource Allocation in Solid-State Disks

研 究 生：黃偉杰

指導教授：張立平　教授

中 華 民 國　九 十 九 年 七 月

固態硬碟之自適性資源管理方法

Mapping vs. Buffering: Adaptive Resource Allocation in Solid-State Disks

研 究 生：黃偉杰　　Student：Wei-Chieh 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

July 2010

Hsinchu, Taiwan, Republic of China

中華民國九十九年七月

固態硬碟之自適性資源管理方法

學生：黃偉杰　　指導教授：張立平

國立交通大學資訊科學與工程研究所

摘要

固態硬碟的硬體資源的使用情形經常受到存取行為變化的干擾，導致在不同環境下 Buffer 與 FTL 的效能現表現可能差異極大，因此當其中一個元件需要使用較多的硬體資源時，無法將另一方多餘的資源提供給需要的元件，除了造成硬體資源的浪費之外，也容易使得系統效能低落。本篇論文提出一個自適性資源管理之架構，透過可調整資源需求之 FTL 與 Buffer 達到資源共用之目的，並以二階段式的資源分配演算法，監測元件效能表現來分配資源分配硬體資源給需要的部份，藉此來取得不同存取行為下各元件資源分配之平衡點。實驗在四種不同行為環境下進行模擬，結果顯示在不同存取行為下我們的方法都能自我適應，在一般使用者行為的環境下勝過其他 Buffer 與 FTL 之組合至少 15%以上，而在循序寫入及隨機寫入的環境下表現更是勝過其他方法至少 80%以上。

關鍵字：NAND 快閃記憶體(NAND Flash Memory)，寫入緩衝(Write Buffer)，
快閃記憶體轉換層(Flash Translation Layer)，
資源配置(Resource Allocation)

誌　　謝

轉眼間，兩年的研究所生活已接近尾聲了，對於這段時間在交大所經歷過的一切事物，內心充滿著感激。

首先要感謝我的指導教授張立平老師，老師除了在課業與專業領域上的細心教導之外，在研究方面總是能適時與學生們溝通討論，給予協助並指點正確的方向，也讓我學習到更多解決困難的技巧與能力。

再來，我要感謝同在 ESSLab 的電鍋、義勛、莉君、阿誠、玟蕙、Uma 以及小節，因為有你們的陪伴，與我一起分享及品嚐這兩年的時光，我的碩士生活才會如此精采而難忘。

也感謝家人在背後默默地支持與關懷，幫我加油打氣，讓我一路走來安心而堅定。感謝大學同學與朋友們的關心與祝福，陪我打遊戲渡過每個寂寞難耐的夜晚，讓我可以充滿活力去面對明天的挑戰。

正因為有了你們，我的人生才是彩色的。

Contents

一、Introduction. 1

二、Background. 4

2.1 Flash Geometry 4

2.2 Flash Translation Layer 4

2.3 Buffer 6

2.4 Problem Definition 8

2.4 Related Work 8

三、Adaptive Resource Allocation 11

3.1 Overview 11

3.2 FTL Design 12

3.3 Buffer Design 14

3.4 Resource Allocation Algorithm 16

3.5 Putting things together 18

四、Implementation issues 20

4.1 ARA Algorithm Adjustment 20

4.2 Component-Specific Optimizations 20

4.3 Implementation Overheads 21

五、Performance Evaluation 22

5.1 Experiment Setting 22

5.2 Performance Evaluation of buffer 24

5.3 Performance of ARA Algorithm 26

5.3.1 Evaluation with different trace 26

5.3.2 Overhead Analysis 27

5.3.3 Table size domination 29

5.4 RAM Size 30

5.5 Overprovision 31

六、Conclusion 34

Reference 35

List of Figures

圖表 1. 快閃記憶體之架構 4
圖表 2. PL MAPPING 與 BL MAPPING 之架構 5
圖表 3. HYBRID MAPPING SCHEME 6
圖表 4. 加入 BUFFER 之目的 7
圖表 5. 不同應用程式之存取行為 8
圖表 6. SYSTEM ARCHITECTURE 11
圖表 8. DUAL GRANULARITY FTL MAPPING SCHEME 13
圖表 9. THRESHOLD TESTING 14
圖表 10. BUFFER 之架構：LRU-PAGE 與 LRU-CLUSTER 15
圖表 11. 回收一個 BLOCK 的 RMW 成本或 GC 成本 17
圖表 12. 各個元件與架構與策略對應關係圖 18
圖表 13. ARA ALGORITHM FLOW CHART 19
圖表 14. BUFFER 在 NB 與 UBUNTU TRACE 下之效能表現 25
圖表 15. BUFFER 在 MULTIMEDIA 與 IOMETER TRACE 下之效能表現 26
圖表 16. 不同 TRACE 下 ARA 演算法的適應情形 27
圖表 17. 不同 RAM SIZE 下系統 PAGE WRITE 之分佈 28
圖表 18. TABLE SIZE 對系統效能之影響 29
圖表 19. 不同 RAM SIZE 下各種方法之 ERASE COUNT 30
圖表 20. 不同 RAM SIZE 下各種方法之 ERASE COUNT 31
圖表 21. 不同 SPARE SIZE 下各種方法之 ERASE COUNT 32
圖表 22. 不同 SPARE SIZE 下各種方法之 ERASE COUNT 33

List of Tables

表格 1. ENVIRONMENT SETUP 22
表格 2. COMPARISONS OF BUFFER REPLACEMENT POLICY 23
表格 3. COMPARISONS OF FTLS 23
表格 4. COMPARISONS OF WORKLOAD 24

一、Introduction.

NAND 快閃記憶體具有體積小、抗震、省電、隨機存取能力佳等優點，因此被廣泛地應用在各類嵌入式系統作為儲存裝置，如手機、MP3 player、PDA 等。而隨著製程技術的精進，以及快閃記憶體中每一個顆粒(cell)可以儲存的位元數增加，降低了單位容量之成本，因而衍生出以 NAND 快閃記憶體構成之固態硬碟(Solid-State Disk)，用以取代傳統硬碟。目前已普遍應用在筆記型電腦、Net book、大型資料庫等場合。

由於 NAND 快閃記憶體具特殊物理特性，為了提供用戶端方便的讀寫操作，固態硬碟會有一層 FTL(Flash Translation Layer)來協助處理存取位址的轉換。FTL 可依照其位址對應的單位分類：以小單位的 High-resolution mapping，具有優異的隨機寫入速度，缺點是儲存表格(Table)極大，需要耗費相當大的硬體成本來儲存 Table。而大對應單位的 Low-resolution mapping 僅需非常小的 Table，但隨機寫入要額外的複製動作以維持資料之一致(Read-Modify-Write)，導致效能不佳。另一類的 Hybrid mapping 則是原始資料以 Low-resolution mapping 存放，寫入時再將寫入資料以 High-resolution mapping 的形式處理，兼具優秀的隨機寫入效能以及可接受的 Table 成本。Hybrid mapping 在設計上同時考量到硬體成本與效能，是目前比較常見的主流 Mapping scheme。

為了提升固態硬碟的效能，目前有效且普偏被應用來提升效能的方法是加入寫入緩衝(Write Buffer)。透過 Write Buffer 吸收具有時間區域性(temporal locality)的熱資料，減少寫入的動作。還可將具有空間區域性(spatial locality)的資料收集起來，盡量以循序寫入的方式寫回 FTL，可以有效減少寫入或空間回收之負擔。

Table 與 Buffer 在使用上都需要花費 RAM 的硬體成本，但是在不同的情況下，系統對 Mapping 架構與 Buffer 的依賴性也不同。例如當存取行為包含許多高區域性(High locality)的資料時，例如檔案系統的 metadata 需要時常進行更新動作，這些資料容易對 FTL 造成不必要的寫入與搬移動作。但透過 Buffer 的協助，可以將熱資料轉為冷資料，減少寫入的動作，還可將資料轉換成無殺傷力的循序寫入。反之，當存取行為偏向低區域性(Low locality)時，例如存放在硬碟的碎裂空間的檔案，檔案會分佈散亂，此時 Buffer 吸收區域性的效果不彰，重點在於如何保有優異隨機寫入速度，並且妥善存放這些低區域性資料使得寫入負擔降到最低，而 FTL 若含有較多的 High-resolution 成分，雖然需要耗費較多的硬體成本來存放 Table，但具有優異的隨機寫入速度與最低的寫入負擔，在處理低空間區域性資料表現較為理想。

實際上的存取行為勢必混合著 冷/熱 與 循序/散亂 各種組合，固定的 Buffer 大小設定與固定的對應單位顯然無法滿足各種存取情形。現有的設計方法多在分別討論固定大小之 Buffer 與 FTL 之最佳化，然而當存取行為發生變化時卻無法有效將硬體資源分配給需要的部份，對類型不同的存取行為在表現上也極具差異也不夠彈性。因此在有限的硬體成本考量之下，我們提出一個可依不同存取行為進行自適性資源分配之演算法 ARA (Adaptive Resource Allocation Algorithm)。以及一套對應的 FTL 架構：一個可變換對應單位之 Dual-Granularity FTL。

Dual-Granularity FTL 架構於 Low-resolution mapping 之上，基本概念是各個 Low-resolution 單位可視需求配置記憶體空間來切換為 High-resolution，或是將回收記憶體空間將 High-resolution mapping 轉換成 Low-resolution，藉此控制 Table 的大小，並變換對應單位，將性質不同的資料以不同方式存放，調整 FTL 對資源的需求。上層 Buffer 除了吸收時間區域性之外，還會扮演配合 FTL 之角色，將高空間區域性資料以盡量以循序寫入方式交由 Low-resolution 存放，而低空間區域性的散亂資料則以 High-resolution 的方式儲存，協助 FTL 使其負擔減至最低。

而自適性資源分配演算法主要概念為：以自我回饋(Feedback)方式進行調整，若前一次調整使得 RMW (Read-Modify-Write)成本與 GC (Garbage Collection)成本之比例有所改善，則維持前一次的調整方式，反之則回收另一方之資源。自我回饋的調整方法的概念在於進行效能最佳化的趨近，以及存取行為改變時，可以做出即時微調。但是其中由於 GC 平均成本為長期觀測指標，因此在 Cold start 階段必須另做處理。初始階段的調整方法目的在於等待該指標趨於穩定並進行大方向之調整。而當寫入成本與空間回收成本趨近時，再切換至穩定階段。

在有限的硬體資源下，Buffer 與 Table 會出現相互矛盾或是競爭的情形，這使得動態調整的議題更加困難與混亂。例如當 Buffer 無法有效收集空間區域性時，加大 Buffer 空間可以使 Buffer 處理區域性的能力提升，理應加大 Buffer，但實際上較大的可能是系統面臨低區域性的存取行為，才造成 Buffer 效果不彰，若壓縮 Table 空間去加大 Buffer，只會使得整體效能更差，因此演算法的設計上必須能偵測並排除矛盾造成的錯誤判斷，並且在 Buffer 與 Table size 發生變化時，Buffer 內部的 Page Buffer 以及 Block Buffer 要能因資源數目的變化而自動進行調整來配合 FTL，而 FTL 也要配合 Buffer Size 並適應新的資源數量，進行對應單位的轉換，在各個元件與策略之間要能找出分割點並取得平衡，才不會牽一髮而動全身，對系統造成反效果。

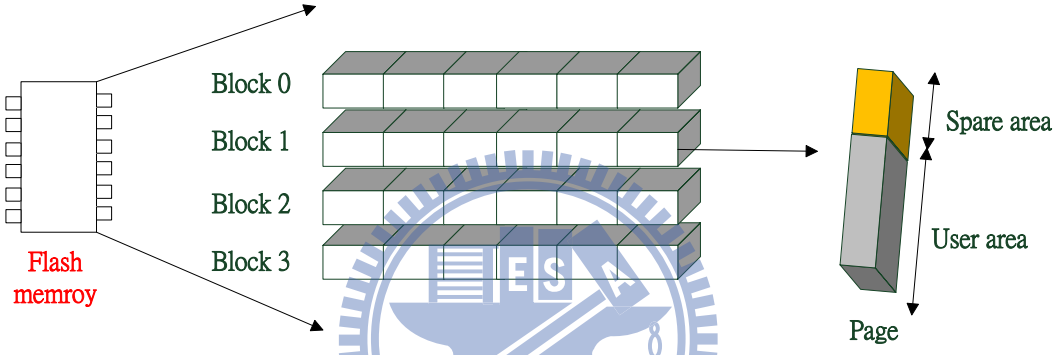
我們透過自適性資源分配演算法的分析，以及 Dual-Granularity FTL 有彈性的調整之下，可以有效地將硬體資源分配給 Buffer 與 FTL 使用，取得資源使用之平衡點，進而達到最高效益的整體效能。

接下來的章節之中，第二章會介紹 Flash Memory 特性，並說明各類 FTL 與 Buffer 演算法其設計概念與優缺點；第三章介紹我們提出的 Dual-Granularity FTL 以及與其對應的 Dual Buffer 之架構和概念；第四章會說明 Dynamic resource allocation 實作細節；第五章在不同的 workload 與設定下來進行實驗分析與比較。驗證我們的方法可以在各種不同的 workload 下進行自適性的動態資源分配來達到更加的效能；第六章為本篇論文之結論。

二、Background.

2.1 Flash Geometry

NAND Flash memory 的組成架構如圖 1，一個 NAND Flash memory 晶片是由數個區塊(Block)組成，而每個區塊是由數個頁(Page)組成，一個頁包含了 User area 以及 Spare area。User area 負責儲存寫入資料，而 Spare area 則是存放 Mapping information 以及 Error Correcting Code(ECC)等訊息。快閃記憶體的寫入基本單位是一個頁，而抹除(Erase)動作的單位則是一個區塊。由於 NAND Flash memory 的每個位元僅能從“1”變成“0”，要重複寫入同一位址前必須將整個區塊進行 Erase，將區塊內所有位元都歸為“1”之後，才能再次寫入。



圖表 1. 快閃記憶體之架構

快閃記憶體每個區塊可以容忍的 Erase 次數有限，以 SLC [1] [2] (Single Level Cell)為基礎架構的 NAND Flash memory 上限約為 100,000 次，而 MLC [3] (Multi Level Cell)架構的 NAND Flash memory 約為 10,000 次。另外，經過長久使用 NAND Flash memory 容易發生電子遺失的問題，造成位元判斷錯誤，因此需要 Error Correcting Code 來發現並校正這些錯誤資料。

2.2 Flash Translation Layer

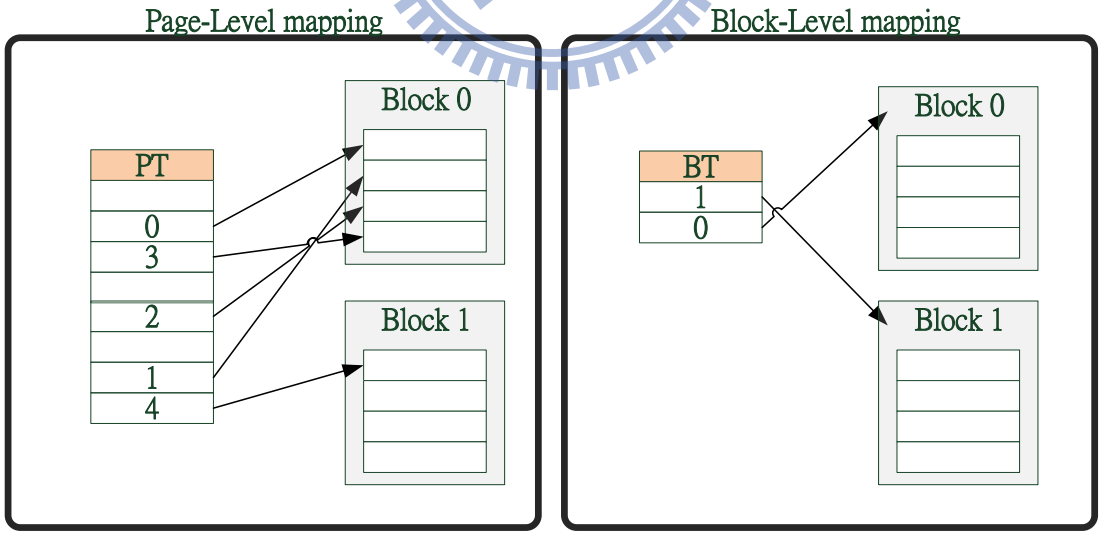
固態硬碟與傳統硬碟在架構與物理特性有著明顯的差異，以往常使用的 File system 如 FAT、NTFS、EXT2、EXT3 等等皆是建立在 Secondary storage 為傳統硬碟的特性之上，無法直接套用在 NAND Flash memory 上，為了讓固態硬碟可以適用於這些 File system，因此在固態硬碟的上層會加入一層 Flash Translation Layer(FTL)，透過 FTL 將固態硬碟模擬成傳統硬碟。由於 NAND Flash memory 無法在同一位址進行 In-Place update，因此 FTL 會將重複更新的資料寫入至其他位址存放，並且使用 Mapping table 來將更新後的位址記錄起來。當所有 Block 都被寫過，沒有額外空間可以寫入時，FTL 會呼叫 Garbage Collection (GC)動作

來清除 Invalid page，回收空間以供其他 Data 寫入。另外 FTL 還會提供平均磨損處理，延長快閃記憶體的使用生命週期，以及進行資料的 Error Correction，來驗證寫入資料的正確性。

FTL 可依其 Mapping 方式分為：High-resolution mapping、Low-resolution mapping、Hybrid Mapping 三類：

High-resolution mapping 是採用較小的對應單位來做位址的轉換，常見的如 Page-Level(PL) mapping [4] 即為具代表性的 High-resolution 架構，如圖 2 的 PL mapping 在寫入資料時，僅需將對應位址記錄至表格(Table)中即可，沒有額外的寫入負擔，因此具有優異的隨機寫入速度。但由於 Table 中必須記錄著每個 Page 的對應位址，Table size 非常龐大，因此需要耗費較高的硬體成本來存放 Table。PL mapping 在容量小的快閃記憶體儲存裝置上較為常見，因為總容量不大，Table 需要的硬體資源也較少，採用 PL mapping 做為 FTL 是可接受的做法。

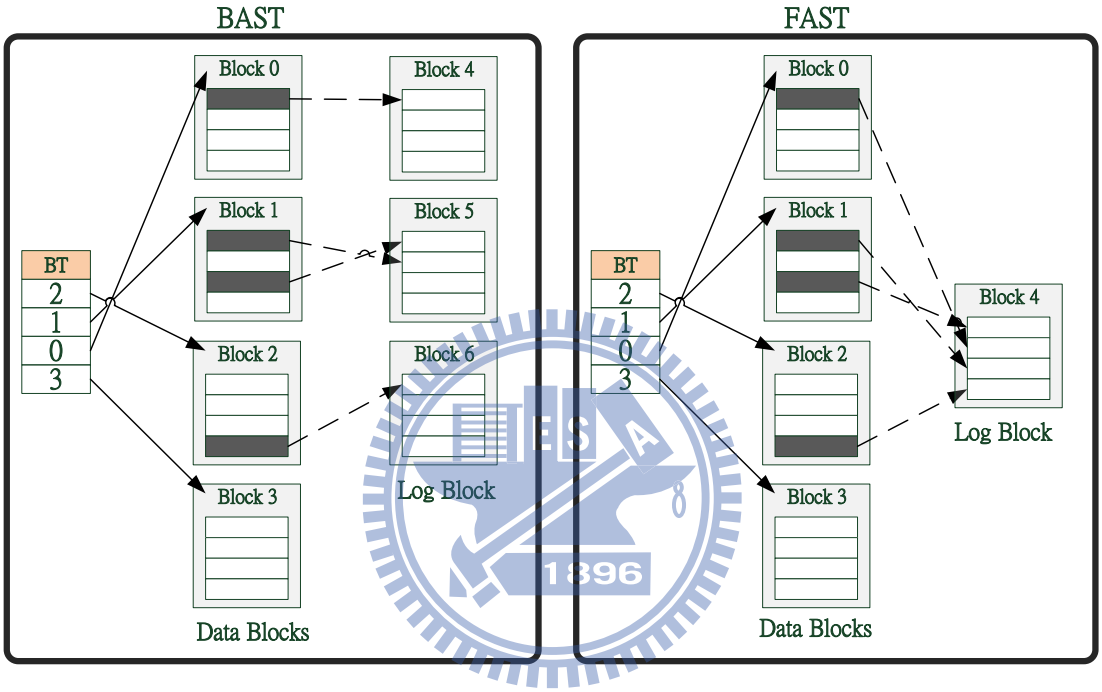
Low-resolution mapping 以較大的對應單位來 Mapping，例如以 Block 為單位的 Block-Level(BL) mapping [5] ，如圖 2 之 BL mapping，其 Table 非常小，額外的硬體成本極低，但寫入資料若未滿一個 Block，需要 Read-Modify-Write(RMW) 來將寫入資料補滿至一個 Block。因此當 BL mapping 進行小量資料寫入時，需要許多額外的複製動作，所以 BL mapping 在隨機存取的表現上極差。而 BL mapping 架構適合用在循序存取較多的多媒體儲存裝置上。



圖表 2. PL mapping 與 BL mapping 之架構

Hybrid mapping 則是混合了 High-resolution 與 Low-resolution 的概念，常見的 Hybrid mapping 架構如圖 3 的 BAST [6] 以及的 FAST [7] ，做法都是將原始資料以 BL mapping 方式存放在 Data block，新寫入的資料以 PL mapping 的方式

寫入至 Log block。在 Hybrid mapping 中只有 Log block 部份需要 Page table，因此耗費的硬體成本介於 PL 與 BL 之間，卻也可以保有優異的隨機寫入效能，同時具備 High resolution 與 Low resolution 的優點，使得 Hybrid mapping 成為目前設計 FTL 之主流。Hybrid mapping 的缺點在於，當 Log Block 被消耗完之後，FTL 會啟動 GC 來回收空間，而 Hybrid mapping 必須將 Log Block 中的有效資料以及其所屬同一邏輯區塊的有效資料，都複製至新的 Block 並整理成 Data Block 的型式存放，這個過程我們稱之為合併(Merge)。Merge 動作需要一連串的複製以及抹除動作，造成額外的負擔，使得固態硬碟效能下降。

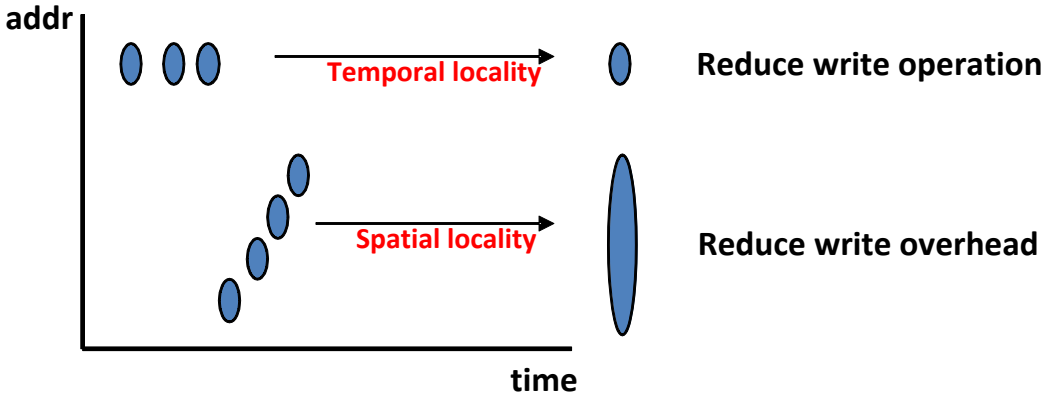


圖表 3. Hybrid mapping scheme

而 Hybrid mapping 中的 BAST 架構是每個 Data Block 可獨立對應一個 Log Block，進行 GC 只需要和同一組的 Data Block 進行 Merge，GC 額外成本較低。然而當系統中的 Log Block 數量太少，或是存取行為過於散亂，會造成各個 Data Block 都要競爭 Log Block 來使用，進而頻繁呼叫 GC 來互相搶奪 Log Block 之現象(Log Block thrashing)，使得整體效能低落。而 FAST 架構讓所有的 Data Block 共用相同 Log Block，因此不會有 Log Block thrashing 的問題，但 Log Block 中的各個有效頁可能來自不同 Data Block，使得 GC 時需要與許多 Data Block 進行 Merge，進而伴隨大量的複製以及抹除動作，付出巨大的額外回收成本，也可能使得用戶端寫入逾時。

2.3 Buffer

目前市面上的硬碟大多都會加入 Embedded RAM 做為 Write Buffer 來提升固態硬碟的寫入效能，如圖 4 所示，加入 Buffer 的主要目的有：(1)、將重複寫入、具有時間區域性的熱資料吸收掉，將熱資料轉為冷資料，並減少寫入硬碟的次數。(2)、將存取位址相近的寫入動作收集起來，並以循序寫入的方式寫回，盡量減少 FTL 寫入以及空間回收的成本。

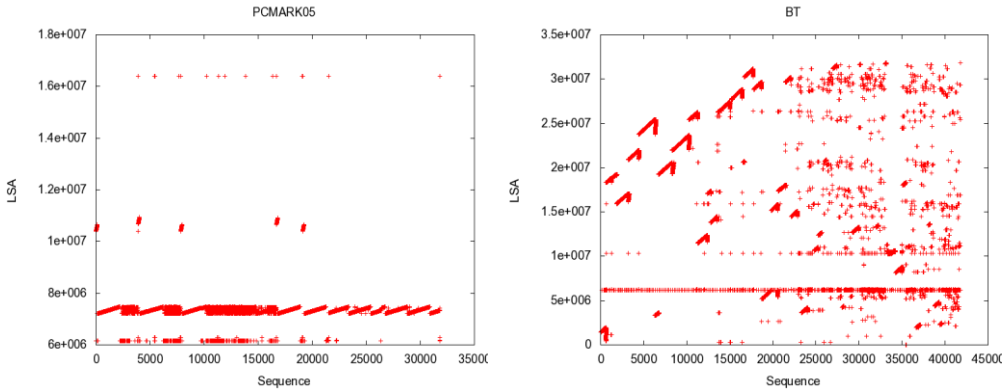


圖表 4. 加入 Buffer 之目的

當增加 Buffer Size 時，Buffer 中可以容納更多寫入資料，相對地也能吸吸收更廣泛時間區域性的資料，對空間區域性的收集能力也較佳，整體效能也會提升，因此 Buffer Size 愈大效能也會愈好。但實際上當 Buffer Size 大到足夠處理存取行為中的大多的區域性之後，整體效能提升幅度也就會逐漸趨於飽和，此時付出更多的硬體成本將 Buffer size 加大，對效能的提升也不大。

另外，在不同的存取行為下，使用 Buffer 的效果也不盡相同。當存取行為具有許多時間區域性的特性時，如檔案的 metadata，這些熱資料可以透過 Buffer 的吸收，有效減少寫入動作。反之，若出現較多冷資料，例如大量多媒體資料的搬移，Buffer 吸收不到任何時間區域性，此時 Buffer 形同虛設，僅能交由 FTL 自行處理。若以空間區域性的角度來看，當存取行為偏向隨機存取時，如存放在碎裂硬碟空間的檔案，存取位址散亂，透過 Buffer 可以收集空間區域性。反之，當出現大量循序存取時，Buffer 收集空間區域性的效果就相當有限。

如圖 5(a) 的 PCMARK 存取行為中具有較高的時間區域性，相同資料重複寫非常多次，這類型的 Hot Data 非常適合利用 Buffer 來吸收，藉此減少 FTL 的寫入次數。而圖 5(b) 為 BT 之存取行為中則是摻雜各類型的資料，重複寫入的 Hot Data 較少，Buffer 的功用較小。其中程式初期具有較多的循序寫入動作，適合以 Low Resolution 處理，後期則是大量的隨機寫入，此時除了可以利用 Buffer 收集空間區域性之外，更適合 FTL 以 High Resolution 形式來儲存，將寫入的負擔降到最低。



(a) (b)

圖表 5. 不同應用程式之存取行為

(a). PCMARK (b). BT

由圖 5 我們可以發現在不同存取行為下，適合 Buffer 與 FTL 的情況也不盡相同，若使用固定的 Buffer size 或是固定的 Resolution 都顯得不具彈性，也容易發生資源浪費的情形。因此我們希望設計一個可變 Resolution、以及可調整大小之 Buffer 與 FTL 架構，透過資源調整來適應各種環境。

2.4 Problem Definition

Buffer 以及 FTL 的 Mapping Table 都會使用 RAM，也同樣都會耗費硬體成本，然而在不同情況下，不同的 Buffer 替換策略(Replacement Policy)以及不同架構的 FTL 的效果既不相同也不對等。我們提出的方法是建構在 Buffer 與 Mapping Table 的調整之上，將硬體資源適時地分配給有需要的元件，使該元件發揮最大的功效。目前的方法都是在固定 Buffer 與 Table 大小之下，各自進行最佳化。因此我們的設計考量： (1).FTL 必須可以有彈性地調整 Resolution，除了可以控制 Table 大小，還可依照不同資料特性用不同 Resolution 儲存。 (2).Buffer 必須配合 FTL 設計，FTL 中的 High resolution 以僅需吸收時間區域性的 Buffer，而 Low resolution 需要可以處理空間與時間區域性的 Buffer，不同的 Resolution 採用不同架構的 Buffer。 (3).在 Buffer 的設計上，以時間區域性的角度看，要能留住年輕的資料來吸收熱資料；以空間區域性的角度來看，盡量將散亂的資料收集起來至同一區塊，以循序寫入的方式寫回 FTL 減少 FTL 負擔。 (4).資源配置的演算法必須監督整體系統的運作情形，調整方向以降低整體負擔為優先，並即時視系統變化調整元件資源，藉此來趨近資源使用的平衡點。

2.4 Related Work

目前已有許多改良固態硬碟效能的方法被提出，但都是分別獨立改善

FTL 架構或是 Buffer。針對 FTL 改善的研究如 N-K FTL [8] 架構與 BAST、FAST 類似，讓 N 個 Data Block 共用 K 個 Log Block，希望藉此能限制 Merge 的成本，並且舒緩 Log Block thrashing 現象。KAST [9] 則是基於 N-K 為基礎，使得 Log Block 內部的 Data Block 可不連續，改良不連續存取造成的效能低落以及 Log Block thrashing 問題。LAST [10] 將寫入依 Request 大小分別寫入至底層 BAST 與 FAST 結合之架構，將大小不同的寫入行為依適合的架構存放。

Super Block [11] FTL 將 N 個邏輯 Block 以 PL 方或對應到 M 個實體 Block，限制位址的對應範圍，藉此使用多階層的 Mapping Table 來做位址的轉換。而 DFTL [12] 與 Super Block 都是為了提升隨機存取之效能，本質上採用的仍為 PL mapping，需要大量的硬體成本來存放 Table，而 Super Block 而且限制住 Block 的對應，仍然會有 Log Block thrashing 的問題。實際上 FTL 在設計上潛藏許多看不到的問題與困難點，若能適時地使用 Buffer 可以將寫入動作簡化，對 FTL 設計或是效能都有極大幫助。

而針對改善固態硬碟上 Write Buffer 的管理方法如 BPLRU [14] ，採用 LRU-Cluster 的方式，以 Block 為單位來管理 LRU，將最久沒寫入資料的 Block 替換掉，並寫回硬碟，較注重空間區域性。另外 BPLRU 使用 Padding 機制，當 Block 要寫回 FTL 時，若資料含量大於門檻值，會將 Block 補滿之後再以循序方式寫回，藉此減少 FTL 空間回收之成本。BPLRU 的缺點是當同一 Block 中同時擁有冷、熱資料，由於熱資料持續 update，使得 Block 中的冷資料無法被寫回 FTL 而累積在 Buffer 中。而資料含量較少的 Block 也沒有特別等待，容易以隨機寫入的形式寫回造成 FTL 之負擔。

而 FAB [13] 的基本想法是優先將 Buffer 中資料含量最多的 Block 替換掉，盡量將資料含量較小的 Block 留住來收集多一點空間區域性，但 FAB 方法忽略了時間區域性，造成資料含量小的 Block 很少有機會被替換掉，導致最後 Buffer 會被許多資料含量小的 Block 佔據住。

CLC [16] 則是將 Buffer 分為二個區段，前段以 LRU-Cluster 來處理，注重的是時間區域性，當資料從 Cluster-LRU 被替換出來之後，再依其資料含量放至注重空間區域性的後段 LC 區存放。當 Buffer 空間滿了之後，會從 LC 區將資料含量最高的 Block 寫回 FTL。但 CLC 必須考量 LRU-Cluster 及 LC 段該怎麼分配大小，在不同的存取行為下對空間區域性以及時間區域性的依賴性也不同，當存取行為發生變化時，固定大小的區段配置可能無法有效處理。雖然 CLC 有另外設計 oFTL 與之搭配，但是在 Table 與 Buffer 之間沒有資源分配的概念，當其中一方的資源過多，或是效果不佳時，無法將資源配置給另一方使用。

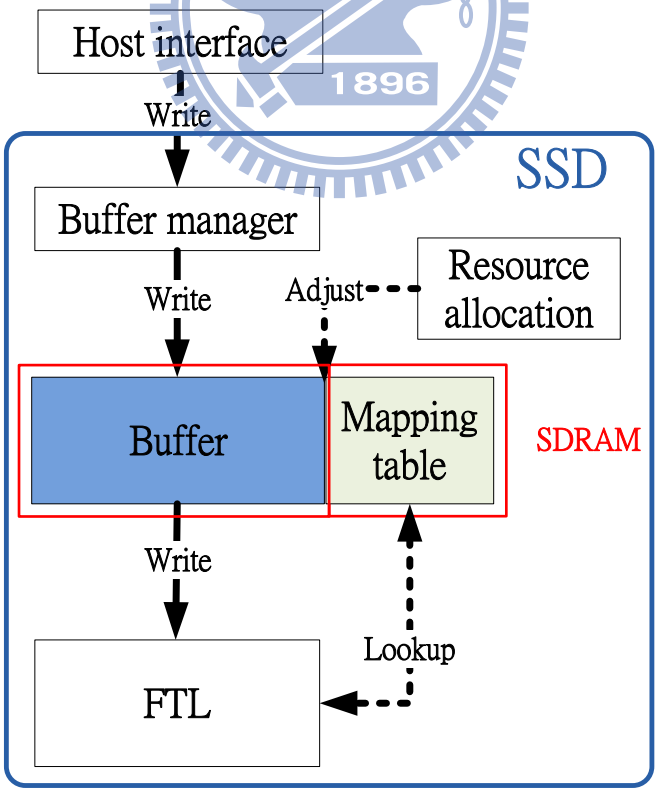
採用 Write Buffer 還需要注意系統斷電造成的資料遺失問題，而造成電腦或伺服器作業系統之 File System 損壞。目前普遍解決方式是加入電容，延緩固態硬碟停止時間，將 Buffer 中的資料寫回 NAND Flash Memory 中。另一類最近被提出的方法則是採用 Non-Volatile RAM (NVRAM)，目前已有部份方法 [16] [17] 採用 NVRAM 來做為 Buffer，採用如 PRAM [18] (Phase-Change Memory)做為 Buffer 使用，可以避免斷電造成的資料遺失問題。

就以上的討論而言，現有固態硬碟效能改進方法都侷限在部份的最佳化，而部份最佳化的設定也無法適用到各種不同的存取行為，不同的方法也都有本質上與設計上的弱點。然而實際上 Buffer 與 FTL 應扮演互相協助的角色，並視為一體來設計，利 Buffer 吸收熱資料並將散亂的資料收集起來，減少 FTL 寫入以及空間回收的負擔，而 FTL 也要妥善處理好 Buffer 無法處理的存取，兩者相輔相成才能取得平衡。

三、Adaptive Resource Allocation

3.1 Overview

Buffer 與 FTL 的 Mapping Table 同樣都是硬體資源的消耗者，兩者都需要使用記憶體空間，然而在不同 trace 下，Buffer 與 FTL 能發揮的效果也不盡相同，有時需要提升 Buffer Size 加強處理區域性的能力，有時則需要較多的 High-Resolution mapping 存放隨機寫入的資料。因此我們提出一個可自適性調整資源需求的管理演算法(ARA)，ARA 會視當前系統需求，動態分配記憶體資源給 Buffer 與 FTL 使用，來應付各種不同的存取行為。圖 6 是本篇論文的基本架構圖，其中 Mapping Table 與 Buffer 是共用同一塊 SDRAM。當 Host 端底層的 File System 來對硬碟進行讀寫操作，寫入硬碟的資料會先暫存在 Buffer，而從 Buffer 替換掉的資料會交由 FTL 寫入快閃記憶體晶片裡，過程中 Resource allocation 演算法會監測 Buffer 與 FTL 的運作情況，將硬體資源妥善地分配給需要的元件。為了達到動態調整資源的目的，因此我們必須重新設計各元件來完成此架構。此架構下主要的元件有：一個可以動態改變 Table size 的 FTL、可配合 FTL 之 Buffer、以及一個調整資源分配的演算法，之後各小節我們將針對各元件的設計做詳細的說明與分析。



圖表 6. System Architecture

當中 write buffer 與 mapping table 都需要記憶體空間，會有排擠的問題

為了達到動態資源分配的目的，我們必須重新設計 FTL 架構，使得 Mapping Table 具有調整彈性。而 Buffer 的設計必須能配合 FTL，盡可能協助 FTL 使其負擔最低。在具備可調整資源需求的元件之後，再由 ARA 演算法來取得 Buffer size 與 Table size 之平衡點。

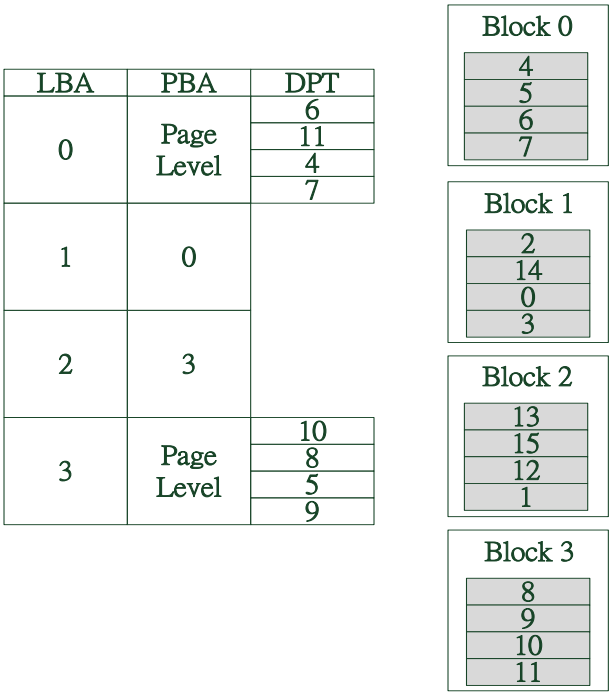
3.2 FTL Design

為了使 FTL 的 Table size 可彈性調整，基本上必須具備不同的 Resolution，並且可在不同的 Resolution 之間變換。Dual Granularity FTL 整體架構是建立在 BL 的基礎上，因此會有一個全域的 Block Table，而每個 Block 可依需求切換為 PL mapping，或是由 PL mapping 轉換回 BL mapping。

如圖 7，LBA 1、2 為 BL mapping，可直接透過 Block Table 分別對應到 PBA 0、3。而 LBA 0、3 為 PL mapping，如 LBA 0 的 Page 0 對應到的位址是 6，也就是資料存放在 PBA 1 的 Page 2 的位址，而 LBA 0 的 Page 1 對應到的位址是 11，表示資料是存放在 PBA 2 的 Page 3，在 PL mapping 中，各個 Page 對應情形會記錄在 Demand Page Table 中。

當資料由 Buffer 寫回 FTL 時，若寫入資料過於散亂，則會進行 BL 轉 PL 的動作，來減少 RMW 造成的負擔，我們將 Mapping 單位由大的 Block 換成小的 Page(L2S)，並另外配置記憶體空間提供 DPT 使用，將 Block 中每個 Page 的對應位址逐一記錄至 DPT 中，因此 L2S 的動作是屬於消耗資源的行為。

而當硬體資源不足時，若系統決定回收 Table 空間，則會進行 PL 轉 BL(S2L) 來將 DPT 回收，將硬體資源歸還給系統分配。在回收前必須將同 LBA、散亂在各處的有效資料都複製到一塊新的 Block，以 BL 方式存放，並將新的位址更新至 Block Table 中，才能將 DPT 回收，因此 S2L 需要額外的複製動作。所以進行 S2L 的時機與次數必須謹慎控制，否則過量的 S2L 負擔會降低整體效能。而 S2L 被呼叫的時機，我們將在 3.4 節討論動態資源分配時會另外詳細說明。



圖表 7. Dual Granularity FTL mapping scheme

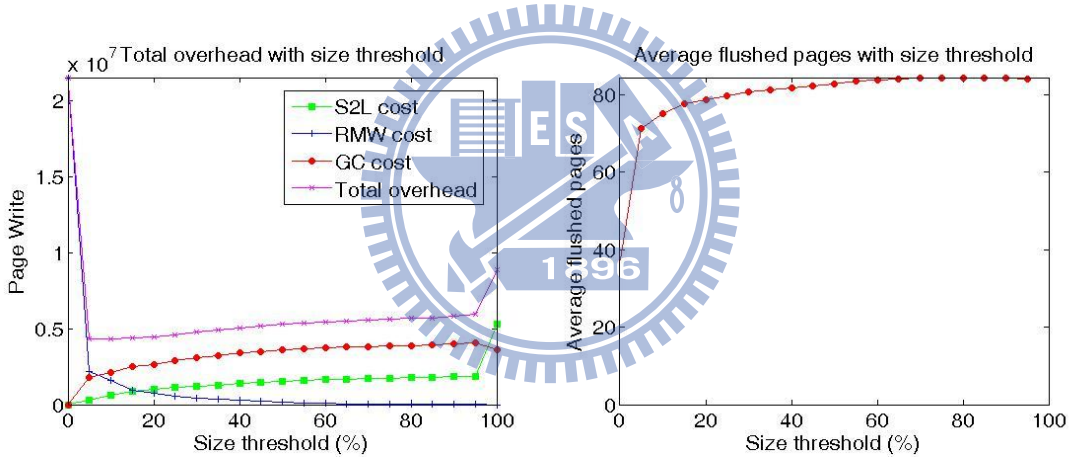
至於何時需要使用 BL mapping，何時又需要 PL mapping？以空間區域性的角度來看，循序寫入的資料較不會增加 BL mapping 的 RMW 成本，以 BL mapping 存放可以節省 Mapping table 的空間，而隨機寫入的資料較適合以沒有寫入負擔 PL mapping 存放，避免過多的 RMW 造成寫入效能不佳。因此在資料從 Buffer 寫入 FTL 時會決定以何種 Mapping 模式儲存。資料經由 Buffer 處理時間與空間區域性之後，寫出的資料多為冷資料，我們將這些冷資料分為兩類：資料量較大的寫入動作以 BL mapping 存放，資料量較小的寫入以 PL 對應，如此可以平衡效能與硬體資源消耗的關係。而資料的含量的大小判定需要一個門檻值(Size Threshold)，當寫入 FTL 的資料量大於門檻值時，就使用 BL mapping；反之就以 PL 來做 Mapping，而門檻值要能有效分類出 BL 與 PL mapping 適合的資料量，在設定上並不容易。

因此，以下我們將對門檻值的設定進行討論與分析，在此我們可以將系統所有的負擔分為三個： (1).BL mapping 中，寫入資料量未滿一個 Block，RMW 額外複製的成本。 (2).呼叫 GC 時，PL mapping 將 Victim Block 中有效資料搬移到新的 Block 存放的搬移成本。 (3).S2L 動作，需要額外複製有效資料的成本。其中(1)的成本是由 BL mapping 所引起的，而(2)、(3)的成本則是 PL mapping 造成。

我們在不同 Size Threshold 下進行實驗分析，圖 8(a) 是不同 Size Threshold 造成的額外寫入負擔比例，實驗環境是在正常使用者的 Trace 下進行，X 軸左至右為 0%到 100%，分別代表寫入資料量小於 0%就以 PL mapping (換句話說，就

是全 BL mapping)，到寫入資料量小於 100%就用 PL 做對應 (也就是全部都是 PL mapping)，可以發現 10%到 90%的 Size Threshold 整體負擔的變化不大。逐漸增加 Size Threshold，雖然系統中的 BL 的成份逐漸減少，因此(1)當中的 RMW 成本也會下降，但相對的，系統中 PL 所占比例會漸漸增加，造成(2)當中的 GC 以及(3)的 S2L 成本上升，導致 Size Threshold 只要在非極端的調整之下，對系統效能造成的影響非常小。

圖 8(b) 是在不同的 Size Threshold 下，分離 BL 與 PL mapping 對 Buffer 收集空間區域性的影響。在 0%純 BL mapping，未將 PL 分離出來情況下，此時 Buffer 每次寫回 FTL 的資料含量平均為 40 個 Page。若將 Size Threshold 設定為 10%左右，空間區域性較差的資料分離出來以 PL 方式 Mapping，剩餘的資料具有較好的空間區域性，可以讓 Buffer 平均寫回的資料含量可以提升到 70 個 Page。若 Size Threshold 提高，Buffer 寫回的資料含量的提升有限，表示經由 Buffer 處理後，寫回 FTL 的資料含量兩極化，而 Size Threshold 設定為 10%，已足夠將大部份的空間區域差的資料分離出來。



(a) (b)

圖表 8. Threshold testing

(a).不同 threshold 之 overhead (b).不同 threshold 下之 utilization

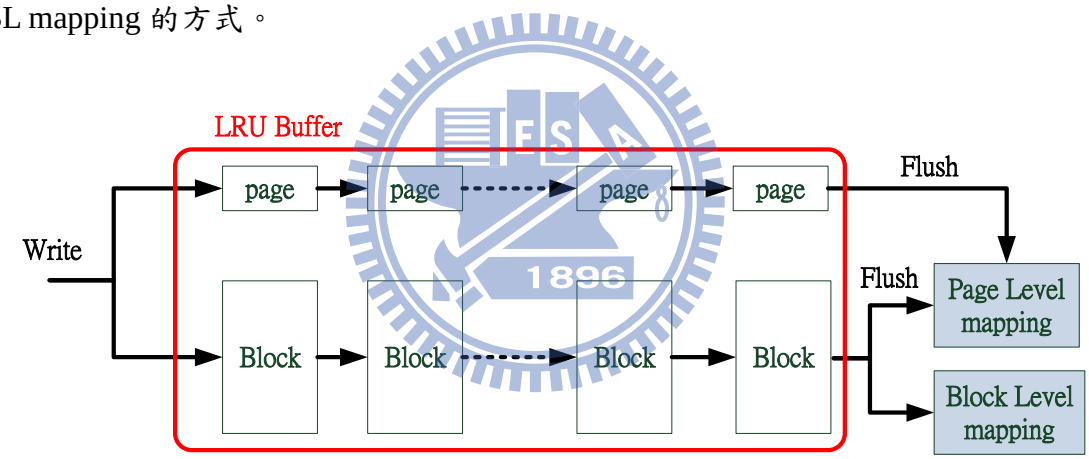
由於加大 Size Threshold，對系統效能的影響也不明顯，分離空間區域性的效果也有限，然而卻讓 PL 數量增加，增加硬體成本的消耗。因此我們選擇低標的 10%為 Size Threshold，做為系統 L2S 門檻值之設定。

3.3 Buffer Design

Buffer 在系統中除了做為寫入的緩衝之外，更重要是任務是要能協助 FTL。由於 Dual Granularity FTL 同時存在 PL 與 BL mapping，Buffer 對這兩種不同的 mapping 方式必須用不同方式處理。對 PL mapping 而言，需要 Buffer 協助將冷

熱資料分離開來，透過 Buffer 將重複寫入的熱資料轉為冷資料再寫回 FTL，可以有效減少寫入次數以及 GC 的成本，因此 PL mapping 需要一個可以吸收時間區域性的 Buffer。而對 BL mapping 而言，除了要將熱資料吸收之外，還要收集空間區域性，盡可能將同一 Block 的資料收集起來，以循序寫入方式寫回 FTL，來減少 RMW 的成本，因此 BL mapping 需要具有空間與時間區域性處理能力的 Buffer。

因此我們將 Buffer 分為二個區塊，如圖 9，第一個區塊為 LRU Page Buffer，以 Page 為管理單位，只注重時間區域性，專門提供給 PL mapping 的資料使用。第二區塊為 LRU Block Buffer，專門存放屬於 BL mapping 的資料，Block Buffer 除了利用 LRU，將最久沒被更新到的資料替換掉，除了藉此吸收資料的時間區域性之外，更希望能透過 Block 為管理單位，將同一 Block 的資料收集在一起，如此一來寫回 FTL 時，可以減少 BL mapping 寫入造成的 RMW 成本。而當資料從 Block Buffer 被移出時，當資料含量小於 Size Threshold，將會進行 L2S 轉換，將這些 Size 小的資料轉為以 PL 方式寫回 FTL，若資料含量大過門檻值，則維持 BL mapping 的方式。



圖表 9. Buffer 之架構：LRU-Page 與 LRU-Cluster

由於 Block Buffer 與 Page Buffer 的配合對象及用途都不相同，因此兩者之間大小比例的調整也必須視系統整體變化而變化。我們可以從系統中 PL mapping 的數量觀察到存取行為的空間區域性。若系統中 PL mapping 的數量愈少，表示寫回 FTL 的資料多以 BL mapping 的方式，表示此時存取行為具有較高的空間區域性， Block Buffer 必須肩負起收集空間區域性的責任，Block Buffer 對空間的需求也會比較大，因此系統會盡量提供資源讓 Block Buffer 使用。如果系統中 PL mapping 的比例較高，代表資料的空間區域性很低，系統中大多的資料為 PL mapping，因此 Page Buffer 在使用上的頻率與效果都較 Block Buffer 大，此時 Page Buffer 對資源的需求大於 Block Buffer，因此系統會盡量提供資源來協助 Page Buffer。所以我們可以藉由系統中 PL mapping 數量來觀察並調整 Block Buffer 與 Page Buffer 大小比例，並加以推導出等式(1)來做為調整之依據。

$$Size_{PB} = (TotalSize - Size_{table}) \times (\frac{Size_{table}}{TotalSize}) \quad (1)$$

我們定義了 $Size_{PB}$ 為 Page Buffer 可分配到的空間，$TotalSize$ 為系統中所有可用記憶體空間，$Size_{table}$ 為 PL mapping 中，Table 所使用的空間。當 PL mapping 數量愈多，Page Buffer 在 Buffer 中所占比例就會愈高。舉例來說，若系統擁有 32MB 的記憶體空間，而 PL mapping 消耗 8MB 的 Table 空間，因此剩餘 24MB 的 Buffer 空間中，有 8/32 的比例是配置給 Page Buffer。Page Buffer 最後可分配到 6MB 的空間，Block Buffer 則是擁有 18MB 的空間。

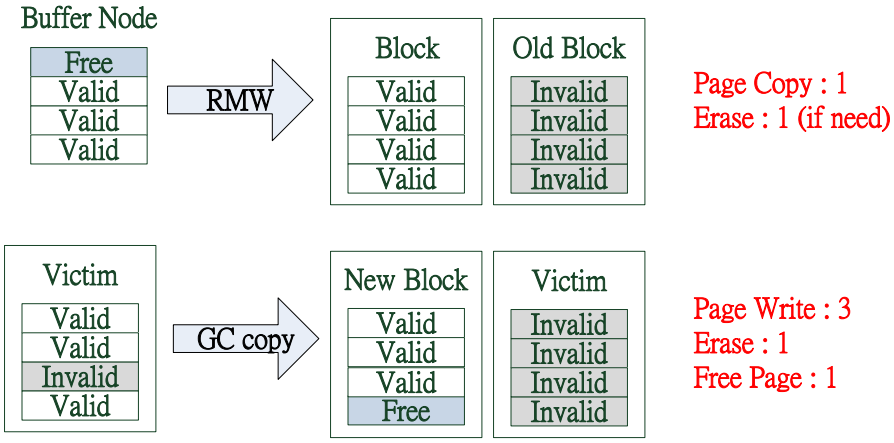
因此 FTL 與 Buffer 之間的關係如下：資料經由 Buffer 處理時間與空間區域性之後，將 Page Buffer 寫出的資料寫回 PL mapping，而 Block Buffer 寫出的資料會透過 Size Threshold 的分析，將寫回資料分為兩類，資料量較小的隨機寫入以 PL 做 mapping，而資料量大的循序寫入則是使用 BL mapping 儲存。所以當存取行為含有較多的隨機寫入時，FTL 會傾向 PL mapping，Table size 較大，相對地會擠壓使得 Buffer size 較小，Page Buffer 的比例比較高。若存取行為空間區域性較佳，循序寫入的情況較多，FTL 會偏向 BL mapping，需要的 Table size 較小，Buffer 能使用的空間較大，Block Buffer 也較 Page Buffer 來得重要。

3.4 Resource Allocation Algorithm

FTL 與 Buffer 使用資源的行為上有著競爭關係，當系統中所有資源都被消耗完時，必須決定回收 Buffer 空間，或是 FTL 的 Mapping Table 空間。若決定回收 Buffer 空間，則直接將資料由 Buffer 中移出，寫回 FTL。若是回收 Table 空間，則會進行 S2L，將最久沒被更新的 PL mapping 轉為 BL mapping 存放。空間回收的對象由 ARA 演算法決定。演算法策略的基本想法是，系統主要的負擔來源有 RMW 成本以及 GC 成本。其中 RMW 是 BL mapping 造成的負擔，而 GC 成本則是由 PL mapping 所引起，因此我們可以由這兩個指標來觀察出當下系統適合使用的 Mapping 方式，藉由控制 PL mapping 的數量，間接調整 Table 與 Buffer 的空間。

如圖 10 之比較，我們考慮系統的空間回收率，圖示上層為 BL mapping，Buffer Node 的寫回使用率(Flush Utilization)很高，有 3 筆有效資料，若寫回 BL mapping 需要 RMW 將 Block 補滿，因此需要 1 次額外的複製動作，補滿寫回的 Block 在下次空間回收時，可以直接將存放舊資料的 Block 直接抹除，就不會有額外有資料搬移的成本。而圖示下層屬於 PL mapping，進行 GC 時會以 Greedy 策略選出 Victim Block，此時 Victim 中含有 3 筆有效資料，因此需要將這 3 筆資料搬移到新的 Block，再將 Victim Block 抹除。所以可以明顯觀察出，當 Buffer Flush

Utilization 較高時，RMW 成本非常低，採用 BL mapping 的空間回收率顯然較為划算；反之，若 GC 平均成本較低的話，則採用 PL mapping 的空間回收率較佳。因此我們可以由 Buffer Flush Utilization 以及 Average GC copy 這兩個指標來決定 PL mapping 的數量，透過控制 Table size，間接調整 Buffer size。



圖表 10. 回收一個 Block 的 RMW 成本或 GC 成本

因此我們的演算法採用自我回饋的方式進行調整，將前一次的回收行為記錄下來，若前一次的回收使得整體效能有所改善，則維持前一調整步驟。若前一次調整造成效能下降，則以相反之步驟來調整。主要目的是希望將 U_{flush}/A_{GC} 的比值最大化，U_{flush} 為平均 Flush Utilization，而 A_{GC} 為平均 GC 需要搬移的有效資料數，並且可以立即對存取行為的變化做出即時性的對應調整方法。理想情況下是 U_{flush} 愈大愈好，而 A_{GC} 則是愈小愈好，此時額外付出的 RMW 成本以及 GC 搬移成本都最低。實際調整策略如式子(2)，若目前系統反應出來的 U_{flush}/A_{GC} 比值大於上一次的比值，表示前一次調整使得整體效能有所提升，因此這次調整仍會維持前一次調整方法。相反地，若比值小於前一次調整則會進行與前一次相反之調整。

$$\left(\frac{U_{flush}}{A_{GC}}\right)_{cur} - \left(\frac{U_{flush}}{A_{GC}}\right)_{pre} \geq 0 \qquad (2)$$

然而 Average GC copy 為一長期觀測指標，當系統處於初始階段(Cold start)，此時尚無法反應出當前存取行為對系統造成的 GC 成本多寡，因此我們在 Cold start 階段會選擇成本較低的部份進行調整。Cold start 階段的調整方法概念在於：若 Average GC copy 較高，系統應傾向 PL mapping，而資料經由 Page Buffer 後無法有效分離資料的時間區域性，使得平均的 GC 成本增加，我們應當減少 PL mapping 的數量，減少 GC 的次數，並盡量以 BL mapping 來減少平均的 GC 成本。如果 Flush Utilization 太低，造成 RMW 的平均成本太高，表示存取行為的空間區域性微弱，則應該少用 BL mapping，多使用 PL mapping 減少做 RMW 的次數，

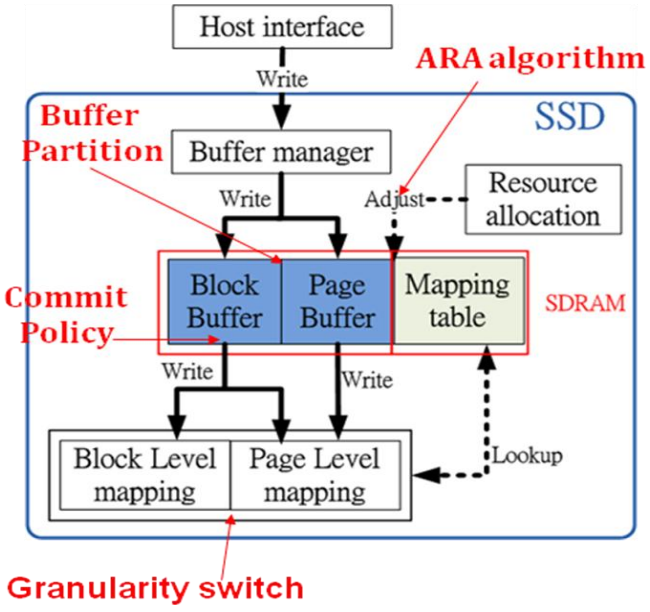
並利用成本較低的 PL mapping 空間回收方法。具體方法如式子(3)，N_{pb} 為一個 Block 的 Page 數，因此(N_{pb}-U_{flush})為 RMW 之成本。若 RMW 成本大於 GC 成本比值則會大於 1，此時則回收 Buffer 空間。反之，若比值小於 1，表示 GC 成本高於 RMW 成本，因此會回收 Table 來減少 GC 負擔。

$$\frac{N_{pb} - U_{flush}}{A_{GC}} \geq 1 \qquad (3)$$

Cold start 階段的調整方法為大方向的調整，主要目的是為了避免自我回饋方法造成的錯誤判斷，具有防呆效果，並且會將 GC 成本與 RMW 成本拉近，當兩者在空間回收的成本相近，再以自我回饋的策略進行即時性與最佳化的調整。

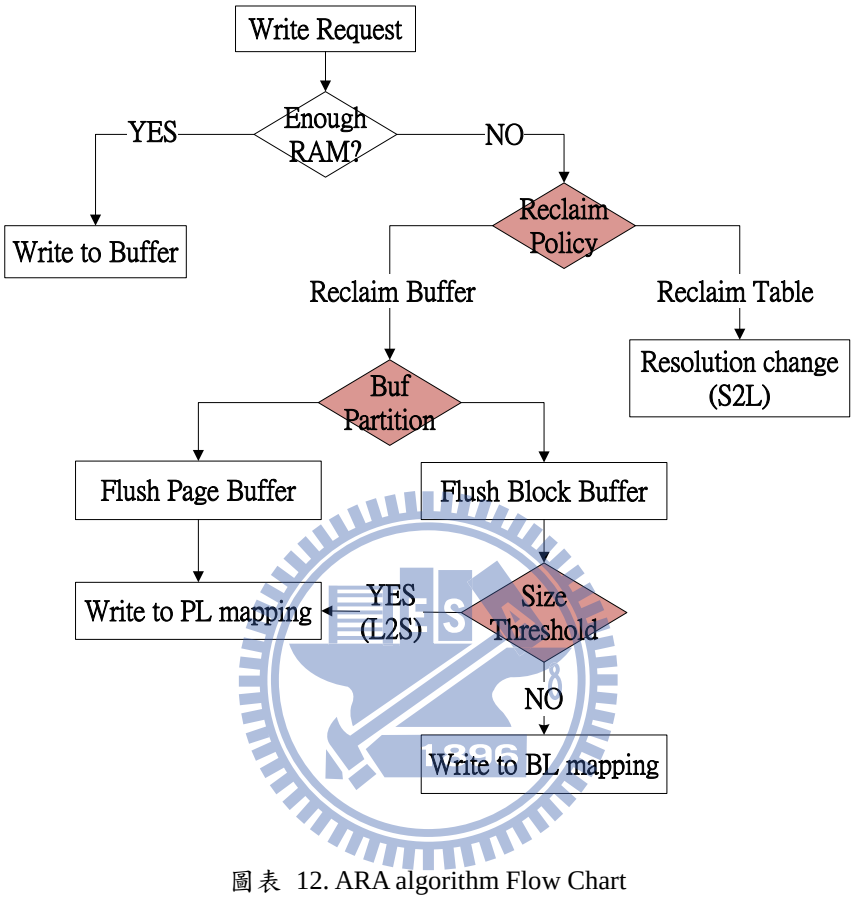
3.5 Putting things together

在完成 Dual Granularity FTL、Page Buffer、Block Buffer 以及 ARA 演算法的設計之後，實際的系統架構圖如圖 11 所示。資料寫入 Buffer 時依其 Mapping 方法存放到對應的 Buffer 中，從 Block Buffer 寫出的資料透過 Commit Policy 來決定維持 BL mapping 或是進行 Granularity Switch 轉為 PL mapping，Page Buffer 寫出的資料則是直接寫回 PL mapping。而系統中 PL mapping 所占數量會決定 Table 的大小，也會影響 Buffer Partition 決定 Page Buffer 的大小比例，PL mapping 愈多則 Page Buffer 愈大，反之亦然。而當資源不足時，ARA 演算法會依 Buffer Flush Utilization 以及 Average GC copy 來決定進行 Granularity Switch，將 PL mapping 轉為 BL mapping 減少 Table 空間，或是將 Buffer 中的資料寫回 FTL，將記憶體空間回收，並提供給需要的元件。



圖表 11. 各個元件與架構與策略對應關係圖

ARA 演算法在初始階段的基本概念是比較 RMW 成本與 GC 成本，決定回收資源的對象，當 GC 負擔的指標穩定後，再以自我回饋方式決定資源回收對象，整體流程如圖 12 所示。



圖表 12. ARA algorithm Flow Chart

資料寫入時先觀察是否具有足夠硬體資源，若資源不足則先依照策略來決定回收對象，若決定回收 Table 則以 S2L 將 PL mapping 占用的空間回收掉，若決定回收 Buffer，會以兩種 Buffer 所占的比例來決定寫回 Page Buffer 或是 Block Buffer 中的資料。若回收 Page Buffer，則將資料以 PL mapping 的方式寫回 FTL。若回收 Block Buffer，則資料先以 Size Threshold 判斷要不要以 PL mapping 方式處理，再各自寫回對應的 Mapping 模式。

四、Implementation issues

4.1 ARA Algorithm Adjustment

ARA 演算法實際上還必須考慮其他因素而有微調，由於系統中所有負擔包含了 RMW 成本、GC 成本以及 S2L 成本，其中 S2L 轉換的成本是由空間回收造成，成本上間接歸屬於 PL mapping，所以考量到 S2L 成本，實際上 PL mapping 的成本會比單純 GC 成本來得多。因此在初始階段調整資源時，我們會加上一偏壓值(約 1.1)，提高 GC 成本的比重再進行調整。而當平均 GC 成本與平均 RMW 成本差距趨近後(兩者小於 10%)，再以自我回饋的方式進行即時性的微調。

而系統會預留一個 High Resolution 所需要的記憶體空間，也就是一個 Demand Page Table 的空間。當空間不足時，若 ARA 演算法決定回收 Block Buffer，而 Block Buffer 中的資料又恰好需要資源來做為 PL mapping，此時可以將預留的 Table 空間拿來使用。以 Page Size 4KB，Block Size 512KB 的例子來看，若一個 Table Entry Size 為 4 byte，則系統需事先預留的 Demand Page Table 空間，為 128*4 byte，一個 Demand Page Table 的空間不會佔用太多資源。

4.2 Component-Specific Optimizations

而 Buffer 與 Dual-Granularity FTL 在設計上除了要相互配合之外，我們還有針對不同元件進行部份優化。其中 Buffer 部份我們採用 LRU 做為替換的策略，因此當有大量冷且循序資料被寫入 Buffer 時，Buffer 中原本存放具有時間區域性的資料會被沖刷出去，造成 Buffer 被這些循序的冷資料佔據，使得 Buffer 的效能下降，這種現象稱為 Scrubbing Buffer。為了避免 Buffer 在循序寫入後效能下降，因此若 Block Buffer 中的資料寫滿一個 Block 單位，會優先將該 Block 寫回 FTL，如此一來便可有效處理 Scrubbing Buffer 問題。而 Page Buffer 是與 PL mapping 搭配，會採用 PL mapping 的資料較為散亂，因此實際上寫入 Page Buffer 的資料較不會有 Scrubbing Buffer 的問題。

而 FTL 需要特別優化的部分在於 PL mapping 造成的 GC overhead，若能將 GC 時需要的資料搬移量壓到最低，對整體效能有明顯幫助。然而隨著 Buffer Size 的增加，寫入資料的區域性都被 Buffer 吸收，使得寫入 FTL 的資料為不具區域性的冷資料。因此在較大的 Buffer Size 下，寫回 FTL 的資料冷熱特性不夠明顯，冷熱分離的方法會愈來愈困難，冷熱分離後對整體效能影響也愈來愈小。而在 Buffer Size 較小的條件下，冷熱分離的議題就顯得較為重要。PL mapping 設計方法本身最大的挑戰在於如何有效地將冷熱資料分離開，因此我們在 GC 時，將需要搬移的資料視為冷資料，並且依照該區塊最後被更新的時間，將其分開放置在

二個不同的 GC Block，將更新時間較近的資料放在一起，而更新時間較遠的資料放一起，除了與寫入的資料區隔，避免寫入資料與搬移資料摻雜造成冷熱混合，還可以將 GC 搬移出來的資料再進行一次冷熱分離，也可以讓 S2L 的動作盡量都發生在同一 Block 中，藉此幫助整體效能的提升。

4.3 Implementation Overheads

我們的位址轉換是採用 Hash function $h(x)=x$ 做為基礎，因此在進行位址轉換的時間複雜度僅需 $O(1)$。而 FTL 是架構在 BL mapping 之上，因此需要一個全域的 Block Table，以 Page size 4KB、Block size 512KB、總容量 64GB 的 SSD 來推算，Block Table 包含 128K 筆 Entry，若一個 Table Entry 以 4 byte 計算，則全域的 Block Table 為 512KB，並視需求來配置 Page Table 的空間，每個 Page Table 需要耗費 512 byte 的空間，若採用純 PL mapping 方式，則最多需要 64 MB 的記憶體空間。

在 GC 部份，系統中每個 Block 會以有效資料含量分別插入不同的 List 中，因此一共有 0~128 共 129 條 List，GC 時僅需由資料含量最少的 List 選出 Victim 即可，選出 Victim 的時間複雜度為 $O(1)$，空間成本上 Block 中每個 Pointer 以 4 byte 計算，所有 List 總和為 512 KB。

最後我們還要額外幾個變數來分別記錄 Block Flush 的平均使用率和次數、GC 的平均搬移量和次數、自我回饋動作、High Resolution 的個數、平均被更新的時間、S2L 的次數，若一個變數以 4 byte 來計算，提供 64 byte 的記憶體空間供變數使已相當足夠。利用這些變數，ARA 演算法可以在 $O(1)$ 的時間內計算調整指標的數值，決定空間回收的對象以及 Buffer 內部的調整策略。

五、Performance Evaluation

5.1 Experiment Setting

接下來的實驗中，我們使用的 Total RAM size 以及 NAND Flash Memory 規格如表 1，其中 NAND Flash Memory 的 Block Size、Page Size 和抹除及讀寫時間我們參考 [3] ，所有的實驗如果沒有特別說明皆在這個設定下完成。並且比較不同 RAM size、Overprovision 以及 Geometry 下各種設定的表現。

表格 1. Environment setup

Total RAM size (MB)	12, 14, 16, 20, 24, 28, 32, 36, 40, 48, 56, 64, 80, 96, 112, 128
Sector size	512 Bytes
Page size	4 KB
Block size	512 KB
Pages per Block	128
Total physical size of SSD	65 GB
Total logical size of SSD	64 GB
Spare size of SSD	1 GB
Time of Page Read	60 us
Time of Page Write	800 us
Time of Block Erase	1500 us

我們的目標是鎖定在使用 16MB~64MB 的 SDRAM 硬體成本下，ARA 演算法能在不同存取行為中自我調整，適應各種情況。我們主要的比較對象為 PL mapping [4] 以及 3 種 Buffer 策略(FAB [13] 、BPLRU [14] 、CLC [16])與 2 種 FTL (FAST [6] 、BAST [7])的組合。在接下來的實驗中，為了方便設定與比較，除了 ARA 方法之外，其餘比較對象 FTL 的 Mapping Table 成本一律忽略，RAM size 即為 Buffer size。

表 2 為三種不同 Buffer 策略之比較。Buffer 策略中，FAB 是每次挑選資料含量最多的 Block 做為 victim。而 BPLRU 則是以 Block 為單位進行 LRU 管理，並且寫出資料含量大於一半(參考 [15]) 時會將整個 Block 補滿，稱為 Padding，藉此降低寫入及空間回收之成本。CLC 則是將 10%的空間做為前段 LRU 使用，剩下 90%空間做為後段 LC 策略使用，前段以 Cluster LRU 吸收時間區域性，前段被替換的資料會放置在 LC 段，LC 段則是注重空間區域性，優先將資料含量最多的 Block 寫回 FTL。

表格 2. Comparisons of buffer replacement policy

Name	Buffer Replacement Policy	Padding	Default Setting	Reference
FAB	1. Block contains the most sectors 2. Block of least recently used	No		[13]
BPLRU	1. Full block 2. Block of least recently used	1/2 Ref to [15]		[14] [15]
CLC	1. Block contains the most sectors in LC segment 2. Block contains the most sectors in CLRU segment	No	10% size partition for LRU-C and 90% for LC	[16]

表 3 則是三種不同 FTL 之比較。FTL 中的 PL mapping 寫入負擔最小、隨機存取速度快，但需要耗費大量硬體成本供 Table 使用，PL mapping 的 GC 方法為 Greedy，選出有效資料含量最少的 Block 回收，並且使用 GC block 來協助冷熱分離。而 FAST 是讓所有的 Data Block 共用 Log Block，GC 時選出最早被寫入資料的 Log Block 進行 Merge，而 Merge 時可能會引發大量的 Erase 與資料搬移。BAST 則是每個 Data Block 都能獨立使用 Log Block，GC 以輪詢(RR)方式對 Log Block 做 Merge，因此當 Log Block 不足時，Log Block 之間會有相互搶奪的現象。

表格 3. Comparisons of FTLs

Name	Mapping scheme	Association	GC policy	Reference
PL	1. All page level mapping		Block contains least valid pages	[4]
FAST	1. BL mapping for Data Block 2. PL mapping for Log Block	All : All All data blocks share same log block chain	The oldest written log block	[6]
BAST	1. BL mapping for Data Block 2. PL mapping for Log Block	1 : 1 Each data block has its log block	Round Robin	[7]

我們使用的 Workload 一共有四個，如表 4。Notebook 為一般使用者在筆記型電腦上，以 Windows XP 為作業系統，使用 Diskmon [19] 收集為期一個月，使用者對硬碟所進行的 Read/Write 行為。而 Ubuntu 則是一般使用者在桌上型電

腦以 Ubuntu 為作業系統，並使用 Blktrace 收集一個月使用者的存取行為。而 Multimedia 則是對硬碟進行大量的多媒體資料的搬移，做為 Sequential Write 的代表 Workload。最後，我們使用 IOmeter 以 4KB request，100% random 的設定來產生出純 Random Write 的存取行為，而我們實驗將以這四種不同特性的 Workload 進行各類模擬與分析。

表格 4. Comparisons of workload

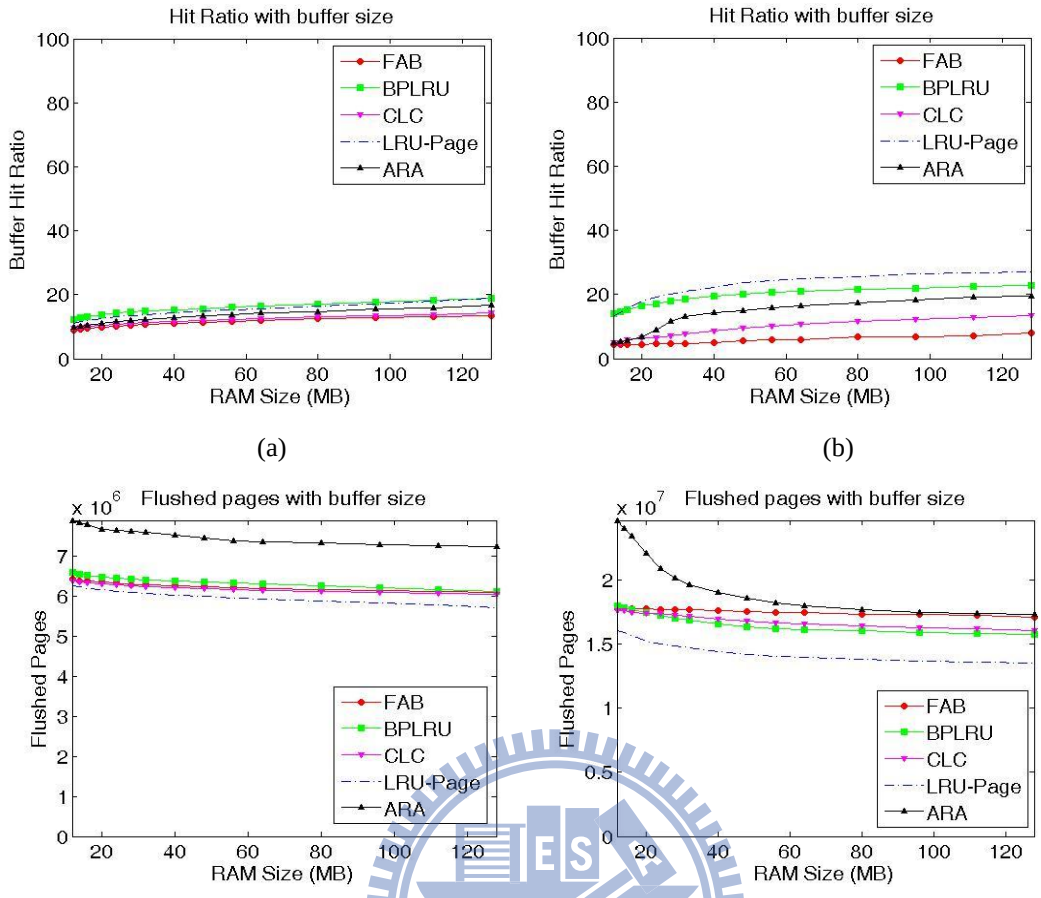
Name	Total write count	Write Ratio	Rewrite	Seq：Rand
Notebook	1,509,343	48%	61%	41%：59%
Ubuntu	3,718,295	35%	57%	29%：71%
Multimedia	352,353	100%	9%	94%：6%
IOmeter	5,102,399	100%	2%	1% ：99%

5.2 Performance Evaluation of buffer

在本章節我們將分析各種不同的 Buffer Policy 在四個 Workload 下的效果與表現，而進一步討論寫出的資料之特性。圖 13 為五種 Buffer Policy 在不同 Workload 下，增加 RAM size 對 Buffer Hit Ratio 與 Total Flushed Page 之影響。

圖 13 (a)為 NB Workload 下的 Buffer Hit Ratio，而圖 13 (b)則是 Ubuntu trace 的 Buffer Hit Ratio，在一般使用者的行為 Workload 中，具有較高的時間區域性，因此注重時間區域性的 Buffer 可以擁有較高的 Hit Ratio，也能有效地吸收重複寫入資料，如以 Page 為管理單位的 LRU-Page、BPLRU 以及我們的方法 ARA 等。而 FAB、CLC 因較注重空間區域性，因此 Hit Ratio 較低。

圖 13 (c)與圖 13 (d)為 NB 與 Ubuntu 下，各種 Buffer 的資料寫出量。ARA 與 BPLRU 雖然具有不錯的 Hit Ratio，但寫回 FTL 的資料相對地都增加，這是因為 ARA 的 RMW 以及 BPLRU 的 Padding，為了減輕寫入 FTL 與 GC 的負擔，都會將資料補滿整個 Block 再寫回，因此 Flush Page 量較多。而 FAB 因為命中率太低，無法有效吸收重複寫入資料，因此 Flush Page 也不少。

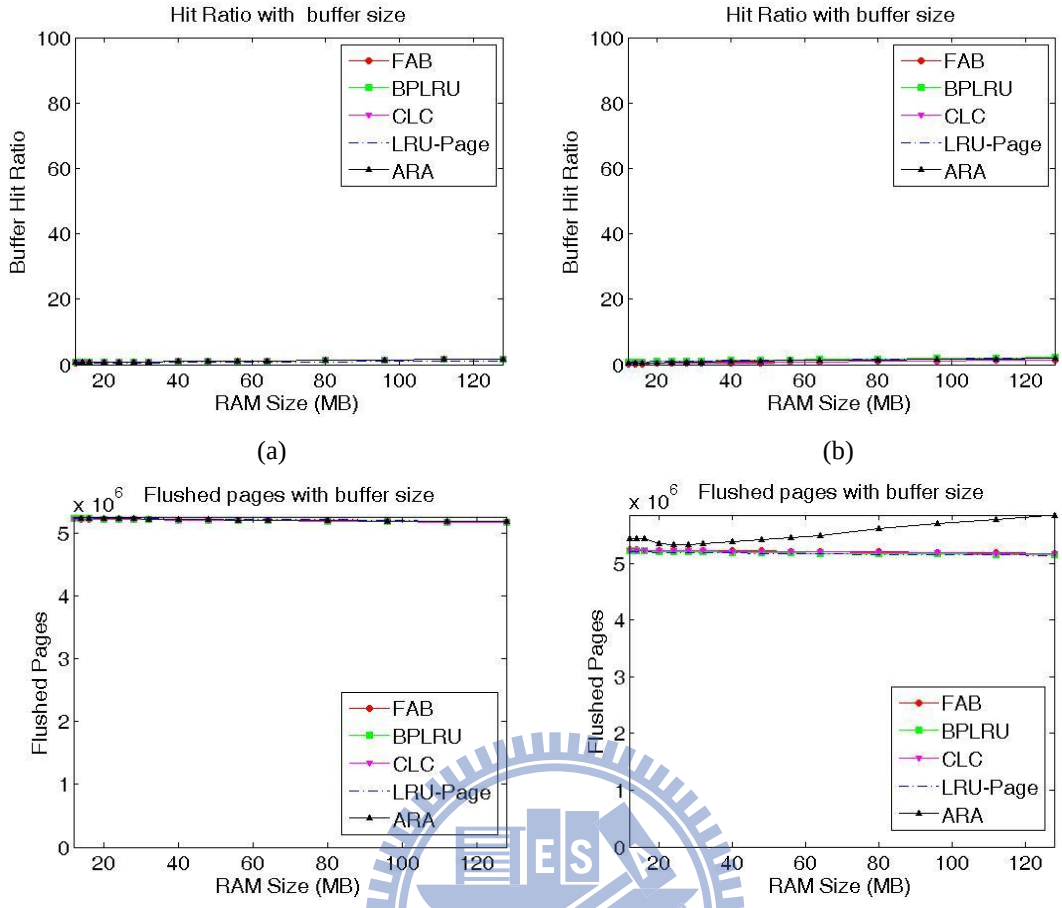


圖表 13. Buffer 在 NB 與 Ubuntu trace 下之效能表現

(a).在 NB trace 下 hit ratio (b).在 Ubuntu trace 下的 hit ratio (c).在 NB trace 下的 flushed pages (d).在 Ubuntu trace 下的 flushed pages.

站在 Buffer 的角度上來看，LRU-Page 具有最高的 Hit Ratio 與最少的 Flush Page，是表現最佳的 Buffer，然而實際上卻忽略了空間區域的問題，也沒有考慮到協助 FTL 議題，因此實際上與 FTL 搭配的表現也會相當有限。

圖 14 是在 Multimedia(MM)與 IOmeter 下各種 Buffer 的表現。這兩個 Workload 的時間區域性薄弱，如圖 14 (a)、(b)所示，由於寫入的資料不會再被 Update，因此 Buffer Hit Ratio 也不會隨著 Buffer size 而有所提升，Buffer 發揮的效果不佳。不同 Buffer 策略的表現沒有差異，如圖 14 (c)、(d)，僅有 ARA 會因 RAM size 增加，提高 RMW 的機會，因此寫入的資料量較多。



圖表 14. Buffer 在 Multimedia 與 IOmeter trace 下之效能表現

(a).在 MM trace 下 hit ratio (b).在 IOmeter trace 下的 hit ratio. (c).在 MM trace 下的 flushed pages (d). 在 IOmeter trace 下的 flushed pages.

在時間區域性差的存取行為下，Buffer 幾乎無法發揮作用，在這種情況下僅能交由 FTL 處理。因此在這兩個 Workload 下，FTL 的 Mapping Scheme 對效能的影響力極大。

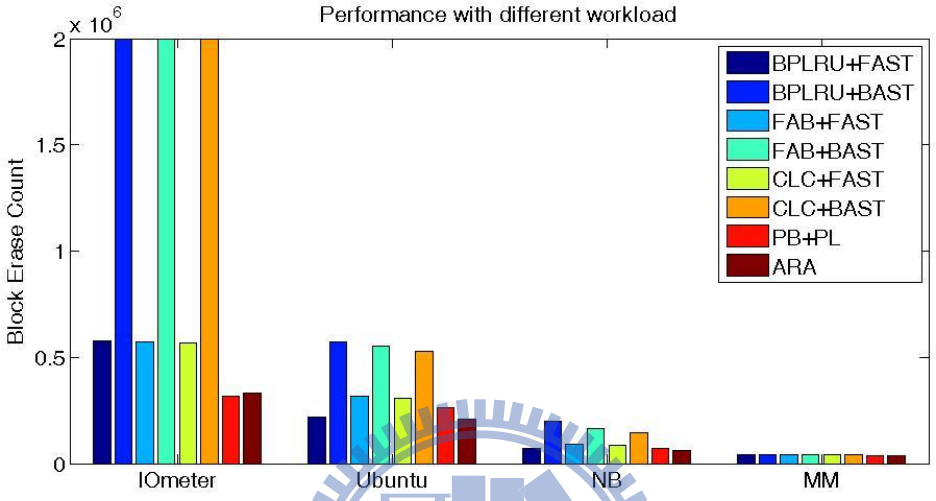
5.3 Performance of ARA Algorithm

5.3 小節我們將重點放在 ARA 本身的環境適應表現與額外負擔的討論上，先掌握住 ARA 方法的優點與限制，再進一步的分析 ARA 演算法與四種搭配的 Workload 的特性，以及 ARA 在這種 Workload 下造成的影響及表現。

5.3.1 Evaluation with different trace

圖 15 為各種 Buffer 與各種 FTL 搭配組合，在這四個 Workload 下的效能呈現，我們以 Block Erase Count 來做為整體效能評估的指標。ARA 在不同的存取

行為下，能夠自適性地調整 Buffer 與 Mapping Table 的資源分配，在 IOmeter 與 MM 這類 Buffer 效果不彰的 Workload 下，會縮減 Buffer 的空間，並調整 FTL 到趨近 PL mapping 的架構，整體效能也與 PL mapping 類似。而在一般使用者行為下的 NB 與 Ubuntu workload 中，ARA 也能視系統需求而分配 RAM size 給 Buffer 與 FTL 使用，因此在取得資源使用平衡點的情況下，ARA 的效能表現優於其他方法的組合。



圖表 15. 不同 trace 下 ARA 演算法的適應情形

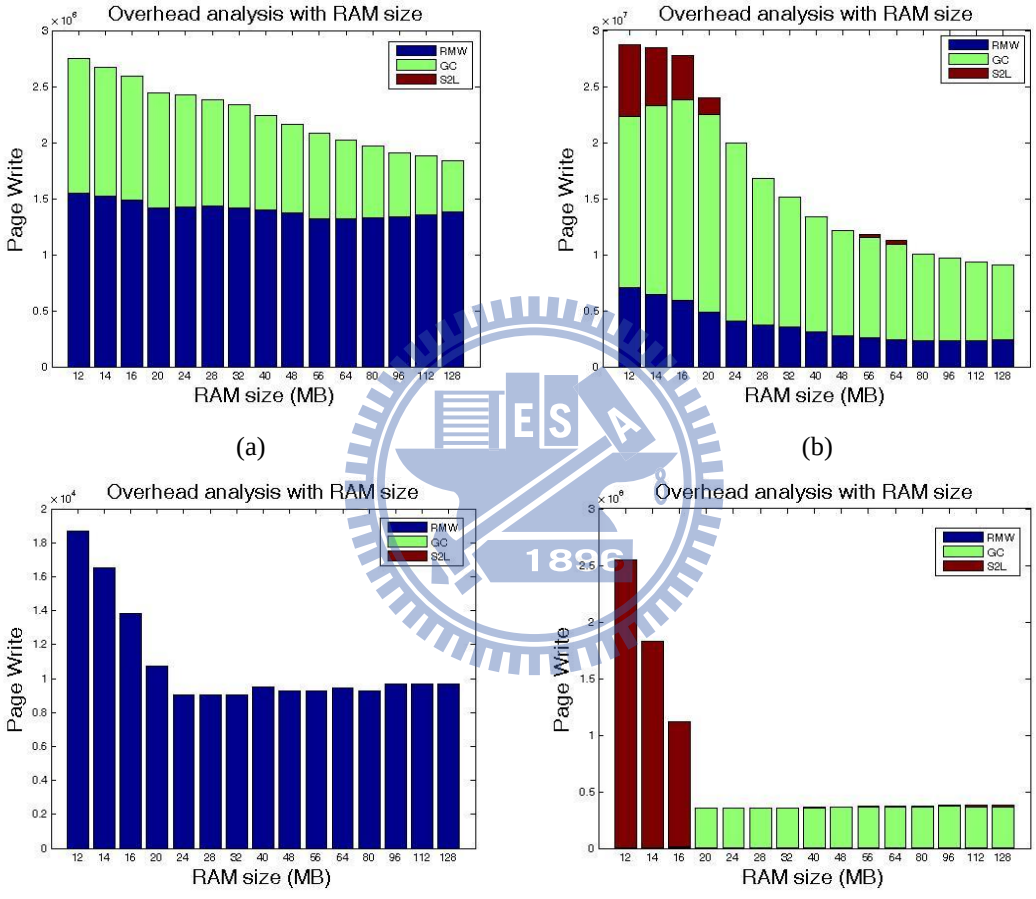
MM trace 下都是不具殺傷力的循序寫入動作，所以各種方法的表現差異不大。而 NB 與 Ubuntu trace 中，原本表現最好的 Page Buffer(LRU-Page)配上 PL mapping 反而無法有較好的表現，這是因為 Page Buffer 沒有協助 PL mapping，造成 PL mapping 的資料冷熱混合，會增加 GC 負擔。

另外在 IOmeter 的 trace 下可以明影觀察出，效能的分佈情形與 Buffer 無關，而是以 FTL 架構來分類，由於 IOmeter 是隨機存取的行為，因此 BAST 會發生嚴重的 Log block thrashing，Log Block 使用率低，因此效能極差。而 FAST 則因為共用 Log block 因此沒有 Thrashing 的問題，但 FAST 進行 GC 時，Full Merge 需要額外的 Block Erase 與 Page Copy，因此效能表現上差強人意。而 PL mapping 沒有額外的寫入負擔，回收空間也不需要如 Full Merge 額外負擔極大的動作，因此在 IOmeter 下 PL mapping 的效能表現最佳。

5.3.2 Overhead Analysis

ARA 具有動態調整資源配置的能力，相對地在 Resolution switch 時必須承擔額外的資料搬移動作，也就是 S2L 的 Overhead，圖 16 是 ARA 在不同 RAM size 下，Page Write 的來源比例，我們可以由 RMW 以及 GC 成本反應出系統中 BL

與 PL mapping 的比例。圖 16 (a)、(c)是分別在 NB 及 MM 下的結果，這兩個 Workload 含有較多的循序寫入成份，因此 RMW 佔 Total Page Write 大部份。而圖 16 (b)、(d)分別為 Ubuntu 與 IOmeter，在 RAM size 小的時候，會有較多的 S2L overhead，這是因為資源不足供 Mapping table 與 Buffer 使用，造成資源之間搶奪的情形，其中 IOmeter 最為明顯，因為 Table 空間不足，PL mapping 之間會彼此強制回收其他人的 Table，造成惡性循環。而隨著 RAM size 的增加，GC overhead 會明顯地減少，因為加大 Buffer size，可以提高 Flush Utilization，進而減少 RMW 造成的 overhead，也減少 PL mapping 的 GC 成本。



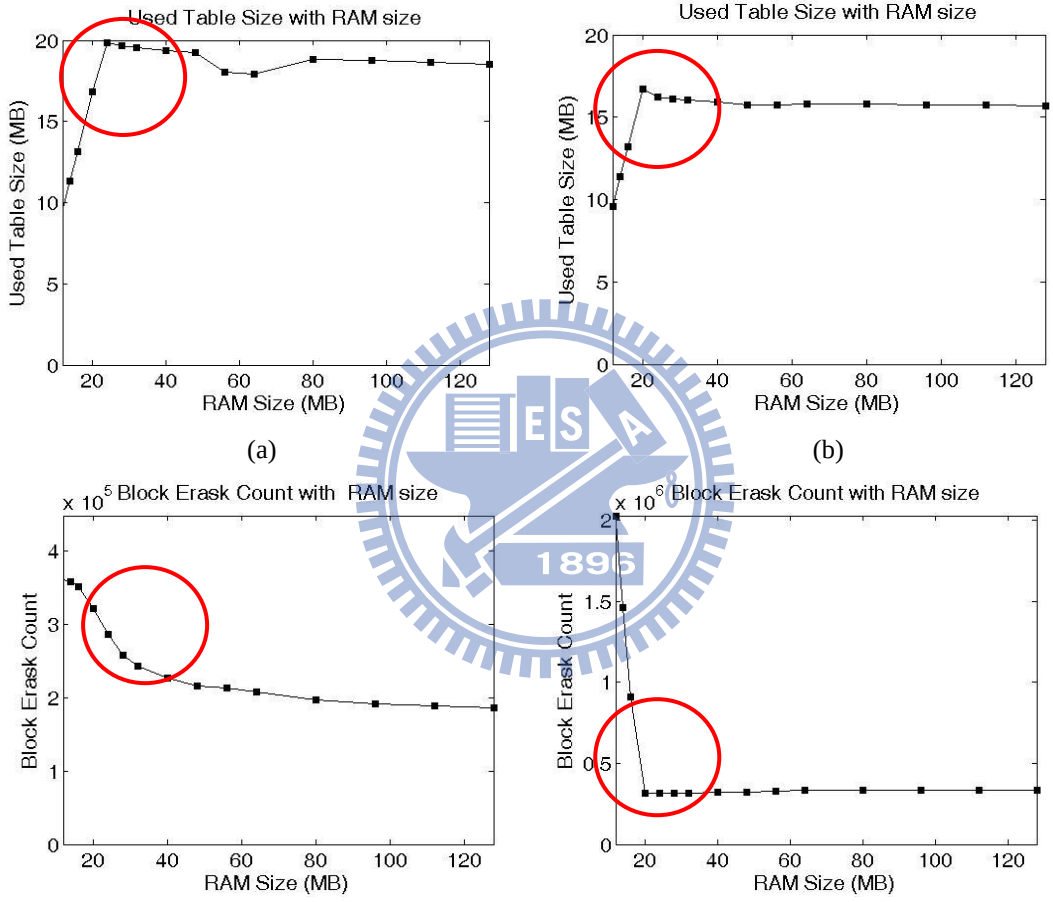
圖表 16. 不同 RAM Size 下系統 Page write 之分佈

(a).NB trace (b).Ubuntu trace (c).MM trace (d).IOmeter trace.

另外，我們也可以觀察到，當 S2L 的成本減少時，對整體系統效能的改進幅度相當劇烈。這也表示說，若系統能提供足夠資源，除了讓 S2L 的動作可以不要發生，沒有資源搶奪的情形對系統的幫助極大。

5.3.3 Table size domination

當系統提供足夠資源時，就可以有效提升整體效能。實際在系統中，對資源有明確的基本需求者就是 Mapping Table，圖 17 (a)、(b)為 Ubuntu 與 IOmeter 下，Table size 需求大小，由圖中可以發現 Mapping Table 的 size 需求會收斂在一個上限值，與之對應的圖 17 (c)、(d)為 Block Erase 的變化，如圖中紅圈處所示，兩個 Workload 的基本 RAM size 需求大約為 20 MB，當 RAM size 可以滿足 Table size 上限之後，Block Erase 會急劇下降。



圖表 17. Table size 對系統效能之影響

(a).Ubuntu 下 Mapping table 的資源使用量 (b).IOmeter 下 Mapping table 的資源使用量 (c). Ubuntu 下的系統效能趨勢 (d).IOmeter 下的系統效能趨勢.

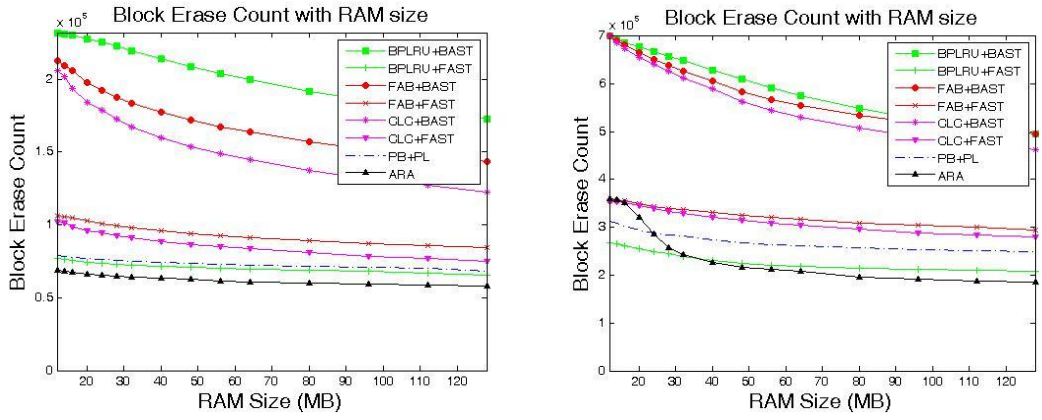
系統的整體效能門檻是由 Table size 控制，任何 Workload 下都有基本的資源需求量，意即至少要能滿足 Table size 大小，ARA 演算法才能發揮出真正的效能。否則在資源不足的條件下，資源之間會有搶奪的情形，S2L 的 overhead 增加，整體效能也會下降。

5.4 RAM Size

這一節我們以 ARA 來跟{BPLRU、FAB、CLC}+{FAST、BAST}以及 LRU-Page + PL 等方法，在不同的 RAM size 下以 Block Erase Count 做為評估效能的標準來進行比較。其中我們忽略掉 FAST、BAST、PL 三種 FTL 所需要的 Mapping table size，這裡的 RAM size 即為該三種 FTL 所搭配之 Buffer size。

圖 18 (a)、(b)為 NB、Ubuntu 下各類方法造成的 Erase Count。ARA 因為 Buffer 與 FTL 之間妥善地將資源配置給需要的元件，利用 Buffer 吸收時間區域性，並且將空間區域性佳的資料以 BL 方式儲存，空間區域性差的資料以 PL mapping 分離出來，因此在這兩種 Workload 下表現最好，僅有在 Ubuntu trace 下 RAM size 較小時，因為資源尚不足滿足 Table 需求，此時表現較差外，其餘部分都勝過其他方法之組合。

圖片中表現較差的三者分別為三種 Buffer 與 BAST 的組合，由於 BAST 會發生 Log Block thrashing，因此整體表現上與其他架構有一段差距，若使用注重空間區域性的 Buffer Policy 如 FAB、CLC 等，Buffer 可以提高 Log Block 的使用率進而減少 Thrashing 的負擔，因此對 BAST 而言 FAB 與 CLC 的幫助較大。而在 FAST 的架構下，則是 BPLRU 的幫助大於 FAB 與 CLC，因為 FAST 具有 Full Log Block 的使用率，較不易受到空間區域性的變化而對效能造成影響，因此相較之下，注重空間區域性的 BPLRU 對 FAST 的幫助就較大。至於 PB+PL 則是因為 GC Policy 無法更進一步將 Cold 與 Warm data 分離出來，使得 GC cost 太高，造成整體效能無法有效提升。

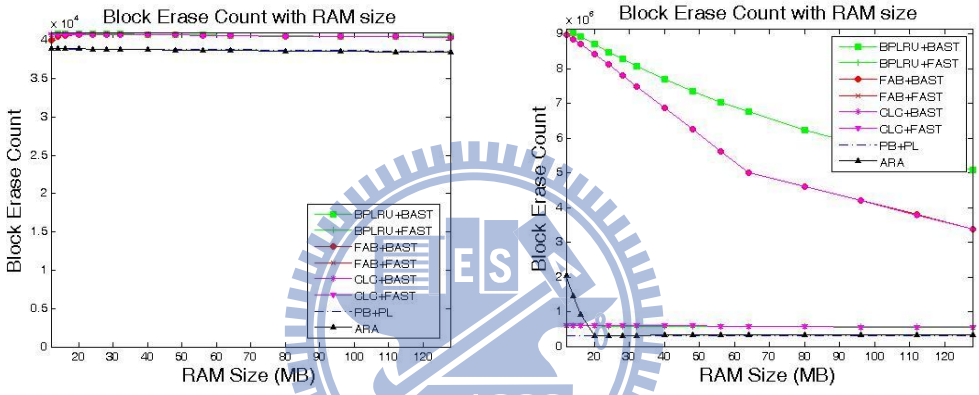


圖表 18. 不同 RAM Size 下各種方法之 Erase count

(a).NB trace (b). Ubuntu trace

圖 19 (a)、(b)分別為 MM 與 IOmeter 下，不同 RAM size 對效能的影響。由於這兩個 Workload 下 Buffer 效果不彰，因隨著 RAM size 的增加，效能也不能有所改善，只有在 IOmeter 下，BAST 可以因為 Buffer 協助收集空間區域性，提高 Log Block 的使用率並提升整體效能，較注重空間區域性的 Buffer 如 CLC 與 FAB 在這種情況下對 BAST 的幫助明顯優於 BPLRU。

在 MM trace 下，ARA 由於可以將大多的循序寫入以 BL 方式存於，而小部分的隨機寫入以 PL 方式存放，不會有額外的寫入與 GC 負擔，因此表現與 PL mapping 相同，其他的 FTL 會因為 GC 觸發的 Merge 而有額外的負擔，效能表現上略輸 PL 與 ARA。而 IOmeter 的 trace 裡，當 ARA 滿足基本 Table 需求後，效能表現會與 PL mapping 相近，表現較架構方法來得好。



(a) (b)

圖表 19. 不同 RAM Size 下各種方法之 Erase count

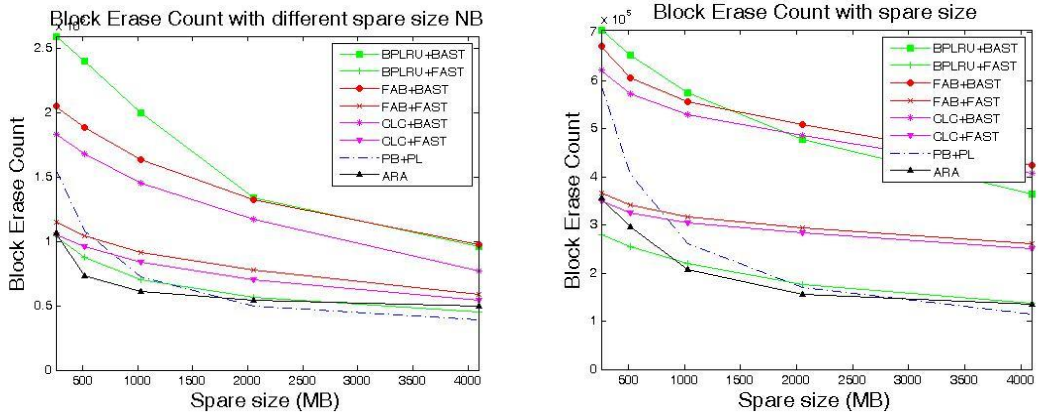
(a).MM trace (b). IOmeter trace

在相同的硬體成本花費的前提下，在 NB 與 Ubuntu trace 中，ARA 與其他方法比較，可以分別勝過至少 15%與 10%的效能，至多可勝過 250%的效能。而 MM 與 IOmeter trace 中，除了 PL mapping 架構之外，其他架構則是分別至少提升 5%與 40%。

5.5 Overprovision

在這一節中，我們將討論 Spare size 的大小對系統效能之影響。圖 20 (a)、(b)為 NB 及 Ubuntu trace 中的表現。ARA 與 FAST 架構在不同 Spare size 下的變化較小，比較不會因 Spare size 增加而有大幅度的效能變化，兩者的表現較為穩定。相較之下，當 Spare size 增加時，BAST 因為可以提供較多的 Log Block，Log Block 之間競爭使用的情形減少，可以大幅幫助效能的改善。而 PL mapping 則是因為 Spare size 增加，有利於資料的冷熱分離，可以減少 GC 時有效資料的搬移，因

此 PL mapping 在 Spare size 增加時，效能的差異也相當巨大，若當 PL mapping 可以有效將冷熱資料分離，會大大地減少 GC overhead，使得 PL 效能表現優於其他方法。



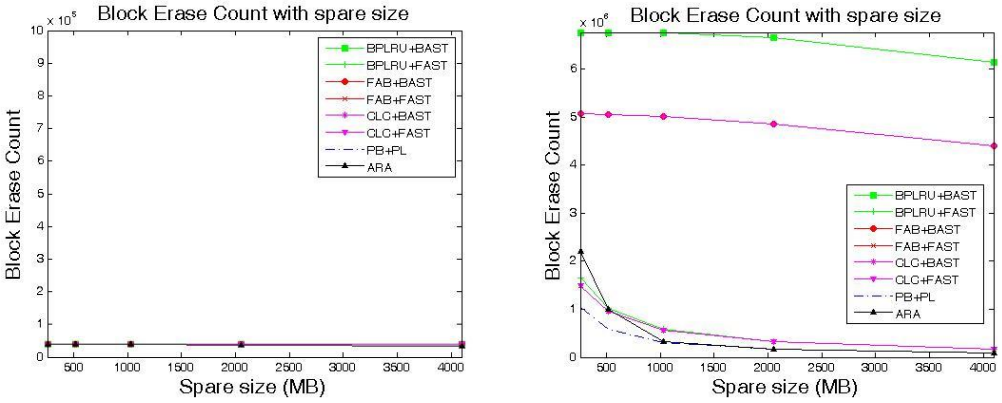
(a) (b)

圖表 20. 不同 Spare Size 下各種方法之 Erase count

(a).NB trace (b). Ubuntu trace

當 Spare size 較小時，ARA 中 PL 部份的 GC 成本較高，因此系統會漸漸偏向 BL mapping 並藉此降低 PL 造成的 GC 成本，所以在隨機存取較多的 Ubuntu trace 中，當 Spare size 小時，ARA 會因為較接近 BL mapping，所以表現較差。相對地，當 Spare size 增加時，系統中 GC 成本較低而逐漸會傾向 PL mapping，進而壓縮到 Buffer 的空間，因此在 Spare size 大時，ARA 與其他方法的差距較不明顯。

圖 21 (a)、(b)分別為 MM 與 IOmeter trace 之結果。MM trace 都是循序寫入，因此在回收空間時可將舊的 Block 直接 Erase 掉，沒有額外 GC 負擔，因此增加 Spare size 的空間對效能沒有任何影響。而 IOmeter trace 下，ARA 在 Spare size 小的時候仍然會因為 GC 成本太高而趨近 BL mapping，使得在隨機寫入的表現不佳，而且 IOmeter 的隨機寫入在透過 Buffer 處理後，會因為使得 Flush data 量太少，使得系統以 PL 方式接收這些寫出資料，但馬上又因為 GC 成本太高，被強迫轉回 BL mapping 方式儲存，造成惡性循環的現象，造成 ARA 在 Spare size 小的時候表現較差。但是隨著 Spare size 加大，ARA 會傾向 PL mapping，在 Spare size 為 1GB 時，ARA 的效能表現上就會與效能最佳的 PL mapping 結果相近，最後可以勝過 FAST 架構之效能 45%。



(a) (b)

圖表 21. 不同 Spare Size 下各種方法之 Erase count

(a).MM trace (b). IOmeter trace

六、Conclusion

隨著 Multi Thread、平行程式等多方面的軟體應用，使用者造成的存取行為愈來愈多樣化，寫入硬碟的 Workload 也愈來愈複雜。目前雖然有許多 Buffer Policy 與 FTL 架構方法被提出，但都是僅針對固定硬體資源下，各元件的最佳化，實際上不同的 Buffer Policy 對不同的 FTL 幫助也不一致，且面臨不同存取行為時，無法有效地將資源提供給需要的元件，在不同存取行為下效能的表現差異可能極大。

此篇論文提出的自適性資源管理方法包含：(1).可調整 Mapping table size 大小的 Dual Granularity FTL (2).兩個 Buffer list，依照不同 Mapping 架構特性設計，以時間/空間區域性的角度分別配合 High resolution 與 Low resolution (3).兩階段的資源調整演算法，以自我回饋方式即時性地將資源分配給需要的元件。以 Buffer 吸收時間區域性，並且將空間區域性較佳的資料以 Low resolution 儲存，散亂的隨機存取則以 High resolution 存放，各元件相互協助與分工，各類型不同的資料以不同方式處理，藉此來適應不同存取行為的變化。實驗結果在 12MB 到 128MB 的固定硬體成本之下，在一般使用者的存取行為下，ARA 演算法勝過各類 Buffer Policy 與 FAST 之組合 15%至 40%不等，更勝過各類 Buffer Policy 與 BAST 之組合 200%以上。而在 MM 與 IOmeter 的 Workload 下，ARA 的表現與額外負擔最低的 PL mapping 一樣突出，在 MM trace 約優於其他方法 5%的效能，在 IOmeter 下的效能表現更是勝過 FAST 架構約 2 倍，勝過 BAST 架構 10 倍以上。顯示出 ARA 除了在一般使用者的存取行為，即使在惡劣的存取行為環境之下，ARA 演算法也能調整架構，在不同存取行為下都能適應，並且妥善地利用資源使得效能有所提升。而 ARA 雖然有回收 Table 需要的 Resolution switch 額外資料搬移成本，但是當滿足基本的 Table 需求後，即可減少 Resolution switch 次數，使 ARA 額外負擔成本的比例降至最低。

ARA 提出各元件之間相互協調之概念，並以元件之間的配合為主要設計考量，實際上對各元件尚有改進之空間。在 Dual Granularity FTL 部份，可以加強 GC policy 來減少 GC 造成的資料搬移、或是改善 S2L 時的替換策略，盡量將 S2L 的 Victim 集中來協助 GC。而 Buffer Policy 也有再改善的空間，加強 Hit Ratio 與 Spatial Locality 的收集能力等等。未來我們將以各元件的優化為目標，藉此提升 ARA 整體效能，使 ARA 在各種情況下的表現更加優異。

Reference

1. *K9F1208U0C 64M x 8 Bits NAND Flash Memory Data Sheet.* Samsung Electronics Company, 2006.
2. *K9NBG08U5M 1G x 8 Bit / 2G x 8 Bit / 4G x 8 Bit NAND Flash Memory Data Sheet.* Samsung Electronics Company, 2005.
3. *K9GAG08U0M 1G x 8 Bit / 2G x 8 Bit NAND Flash Memory Data Sheet (Preliminary).* Samsung Electronics Company, 2006.
4. Wu, M., and Zwaenepoel, W. 1994. *eNVy: A Non-Volatile, Main Memory Storage System*. In Proceedings of the 6th International Conference on Architectural Support for Programming Languages and Operating Systems.
5. Ban,.A. 1995. *Flash file system*. United States Patent, no. 5,404,485.
6. Jesung, K., et al., *A space-efficient flash translation layer for CompactFlash systems.* Consumer Electronics, IEEE Transactions on, 2002. 48(2): p. 366-375.
7. Sang-Won, L., et al., *A log buffer-based flash translation layer using fully-associative sector translation.* ACM Trans. Embed. Comput. Syst., 2007. 6(3): p. 18.
8. Chanik Park et al.,*A Reconfigurable FTL (Flash Translation Layer) Architecture for NAND Flash-Based Applications*, ACM Transactions on Embedded Computing Systems, 2008.
9. Hyunjin Cho et al., *KAST: K-Associative Sector Translation for NAND Flash Memory in Real-Time Systems*, EDAA, 2009
10. Sungjin Lee et al., *LAST: Locality-Aware Sector Translation for NAND Flash Memory-Based Storage Systems*, ACM SIGOPS, 2008.
11. Jeong-Uk, K., et al., *A superblock-based flash translation layer for NAND flash memory*, in Proceedings of the 6th ACM & IEEE International conference on Embedded software. 2006, ACM: Seoul, Korea.
12. Aayush, G., K. Youngjae, and U. Bhuvan, *DFTL: a flash translation layer employing demand-based selective caching of page-level address mappings*, in Proceeding of the 14th international conference on Architectural support for programming languages and operating systems. 2009, ACM: Washington, DC, USA.
13. Heeseung, J., et al., *FAB: flash-aware buffer management policy for portable media players.* Consumer Electronics, IEEE Transactions on, 2006. **52**(2): p. 485-493.
14. Hyojun, K. and A. Seongjun, *BPLRU: a buffer management scheme for improving random writes in flash storage*, in Proceedings of the 6th USENIX Conference on File and Storage Technologies. 2008, USENIX Association: San Jose, California.

15. Hyojun, K., et al., *A Page Padding Method for Fragmented Flash Storage* Lecture Notes in Computer Science, 2007. 4705: p. 164-177.Fab
16. Sooyong, K., et al., *Performance Trade-Offs in Using NVRAM Write Buffer for Flash Memory-Based Storage Devices.* IEEE Trans. Comput., 2009. 58(6): p. 744-758.
17. J Park, H Bahn, K Koh. et al., *Buffer Cache Management for Combined MLC and SLC Flash Memories Using both Volatile and Nonvolatile RAMs*, Proceedings of the 2009 15th IEEE , 2009
18. Bedeschi, F., et al. *A Multi-Level-Cell Bipolar-Selected Phase-Change Memory*. in *Solid-State Circuits Conference, 2008. ISSCC 2008. Digest of Technical Papers. IEEE International*. 2008.
19. Russinovich, M., *DiskMon for Windows v2.01.* 2006.