國立中興大學資訊科學與工程學系

資訊專題競賽報告

一個整合記憶體快取和SSD快取的管理機制

(An Integrated Memory and SSD Caches Management Scheme)

專題題目說明、價值與貢獻自評（限100字內）：

本專題提出了新的系統架構，將快取緩衝器(page cache)和固態硬碟快取(SSD cache)的管理機制整合為一，解決了傳統 SSD based disk cache 會產生的問題，並且讓兩者互相合作，進一步提升了多層級儲存裝置的效能。

專題隊員：

姓名	E-mail	負責項目	貢獻度(%)
陳宇軒	ca768210@smail.nchu.edu.tw	全項共同分工	50
吳哲宇	jack42611@gmail.com	全項共同分工	50

指導教授簡述及簡評：

作業系統都會利用 memory caching 的技術將當下程式可能會使用到的資料暫存於快取緩衝器(page cache)中。此外，隨著固態硬碟的問世，有許多研究與產品利用固態硬碟作為機械式硬碟的快取(SSD cache)。但是，目前 page cache 和 SSD cache 的管理機制是各自獨立的，系統效能因此受到限制。有鑑於此，本專題重新定義作業系統 I/O 子系統的系統架構，將 page cache 和 SSD cache 的管理機制整合為一，讓彼此能夠互相合作，以充分發揮 page cache 與 SSD cache 的效能。

指導教授簽名：_____

中　　華　　民　　國　　年　　月　　日

目錄

一、 摘要

長久以來,機械式硬碟一直是電腦系統內部的最重要儲存裝置,雖然其容量持續向上提升,但是，其存取速度卻一直遠遠落後於 CPU 的運算速度，並且其差距隨著時間的流逝而越來越大。這是因為傳統硬碟內部機械裝置的運作限制,使其無法如同一般半導體元件可以遵循摩爾定律的速度增長。因此，在可預見的未來，機械式硬碟一直會是電腦系統效能上的主要瓶頸。

為了提升機械式硬碟的存取效能，一個直覺且常用的方法即是採用 caching，例如：page cache 利用 memory caching 的技術將目前程式會使用到的資料暫存於記憶體，以減少存取硬碟的次數。此外,近幾年來,出現了以快閃記憶體為組成的固態硬碟(Solid State Disk: SSD)。快閃記憶體為非揮發性記憶體，即使電腦關機，其上的資料也不會遺失；並且其體積輕巧，方便攜帶；最後，快閃記憶體不像傳統硬碟需要仰賴機械裝置存取資料，因此，比傳統機械式硬碟更省電，也提供更高的存取速度。但是，與機械式硬碟相比，固態硬碟每單位容量的成本遠大於機械式硬碟。為了在效能與成本間取得平衡，許多研究開始探討結合固態硬碟和機械式硬碟，其中一種常見的結合方式為利用固態硬碟作為機械式硬碟的快取，也就是將固態硬碟視為作業系統內部的快取緩衝器(page cache)之延伸，page cache 為機械式硬碟的第一層快取，固態硬碟則為第二層的快取。

二、 專題研究動機與目的

在 SSD-based Disk Cache 的架構下，目前的電腦系統存有兩層的快取(caches)。當快取的空間滿了，我們必須清出空間來存放新存取的資料，將舊資料移出以容納新資料的動作稱為替換(replacement)。Page Cache 的替換演算法是一個很經典的問題，在過去已有許多的研究者提出各式各樣的方法，較為有名的快取快取演算法有 Least Recently Used(LRU)、Clock…等。基本上，因為 page cache 被存取的頻率很高，記憶體的存取速度也很快，因此，page cache 的替換演算法在設計上會比較簡單，也就是其替換演算法不會維護有太多的 metadata，也不會有太多計算上的負擔(overhead)。

SSD cache 替換演算法的特色與 page cache 替換演算法大不相同。因為 page cache miss 時才會存取固態硬碟，所以固態硬碟被存取的頻率較低，並且其存取速度亦遠慢於記憶體的存取速度。因此，固態硬碟的替換演算法可以比較複雜，可能維護較多的 metadata，承受較高計算上的負擔(overhead)；此外，因為固態硬碟的獨有特色，其替換演算法不只著重於提高命中率(hit ratio)，而是會考慮傳統硬碟和固態硬碟彼此不同的存取性質。例如：對固態硬碟來說，讀取(read)和寫入(write)存取的速度不同；而對傳統硬碟來說，循序存取(sequential accesses)與隨機存取(random access)的速度也不同。並且固態硬碟有壽命問題，過多的寫入存取可能會導致固態硬碟的壽命縮短。

由上所述，以往關於 page cache 或是 SSD cache 的資料管理與快取策略都是各自獨立設計。但是，這樣的設計架構可能會有以下問題：

1. Metadata duplication：

當快取空間滿了，我們必須挑選適當的資料逐出。為了選擇適當的犧牲者(victim)，許多快取演算法會維護對應的 metadata，紀錄每一個 block 的存取資訊。因此，某些 block 的 metadata 可能會同時被 page cache 和 SSD cache 的替換演算法所維護，造成重複維護的現象。

2. Double Caching：

因為 page cache 和 SSD cache 各自獨立運作，在某些情形下，資料可能同時被快取於

page cache 以及 SSD cache。因為快取空間有限，重複被快取將會浪費寶貴的快取空間。

3. Inaccurate/delayed Information：

如上述，許多 SSD cache 的管理模組會維護 block 的 metadata，但是，其所取得的存取資訊既不即時也不精準。例如：許多 SSD cache 的管理模組會記錄資料被存取的次數，在現在的架構下，SSD cache 的管理模組只能紀錄資料在 page cache miss 下的存取資訊。假設某一資料在 page cache 已被存取三次，SSD cache 的管理模組無法得知此一資訊，換言之，其所取得的存取次數並不能反映資料真正的存取情形(inaccurate information)。並且，只有當某一 block 被逐出 page cache 後，SSD cache 管理模組才會開始對此一 block 維護資訊，因此，其資訊取得的時機是被延遲的(delayed information)。

4. Blind page cache management：

如圖一所示，在現行的架構下，page cache 內部可能會同時快取自機械式硬碟以及固態硬碟的資料。但是，因為 page cache 和 SSD cache 各自獨立，page cache 無法得知哪些資料是來自於機械式硬碟，哪些資料是來自於固態硬碟。換言之，page cache 無法針對來自不同儲存裝置的資料採用不同的管理策略。

為了解決上述的問題，本篇研究目的為提出一個新的架構，整合 page cache 以及 SSD cache 的資料管理與替換策略，使兩者互相合作以提升兩者的潛力與效能。

三、 專題重要貢獻

近幾年來，固態硬碟快速的發展，對儲存媒介掀起了一場革命，然而，快閃記憶體有寫入次數的限制，因此其壽命有限，其次，基於成本的考量，固態硬碟的儲存空間尚落後機械式硬碟相當大的一段距離；所以，結合機械式硬碟和固態硬碟組成的多層級儲存架構是一個極吸引人的架構，使用者可以以較少的價格獲得較大的效能，被認為是深具潛力的儲存裝置解決方案。

但是，同樣做為機械式硬碟的快取，目前 page cache 和 SSD cache 的管理卻是各自獨立、各自為政。如上述，這樣的架構隱藏了一些缺點，也因此限制了 I/O 的存取效能。為了拉近儲存裝置和 CPU 的速度差距，如何整合並提升整體 cache 的效能是一件重要且刻不容緩的事。因此，我們將重新檢視目前 page cache 和 SSD cache 的管理架構，思考目前架構可能引發的缺點；然後，我們將設計一個整合 page cache 和 SSD cache 管理機制的系統架構；其次，基於我們提出的新架構，我們將修改目前的 page cache 和 SSD cache 的替換演算法，使得兩者都可以利用彼此間的資訊並互相合作，以充分發揮 page cache 和 SSD cache 的效能與潛力。

四、 團隊合作方式

本篇專題報告由中興大學資訊科學暨工程學系張軒彬老師指導，並由學生陳宇軒與吳哲宇合作製作完成，包括相關文件閱讀、實驗演算法與程式設計、實驗數據分析以及書面文件製作等等。

五、 設計原理、研究方法與步驟

5.1 背景知識

5.1.1 LRU(Least Recently Used)

最有名的 page cache 替換演算法。LRU 會維護一個 queue，當存取一筆新資料或者 page cache 內某筆資料被存取了，則會將之移至 queue 的 MRU(Most Recently Used)位置；當 page cache 空間不足，需要釋放空間時，會將 LRU 位置的資料丟棄。

5.1.2 LARC(Lazy Adaptive Replacement Cache)

某些 SSD cache 的管理機制可能會頻繁地替換在 SSD cache 內的資料以提高命中率，但是，頻繁地替換會對固態硬碟造成大量的寫入動作，降低固態硬碟的壽命，而 LARC 則是盡量避免頻繁的資料替換動作，以延長固態硬碟的壽命。LARC 會維護兩個 LRU 的 queue，一個為真正在 SSD 內的資料，而另一個被稱為 ghost queue，並不實際存在 SSD 內。當存取了一筆新資料，若沒有在 ghost queue 內，則會把它加進 ghost queue 內；而如果已經在 ghost queue 內，則會從 ghost queue 移除並加到 SSD queue 內。也就是只有當一筆資料在最近被存取兩次，才會被加進 SSD queue 內，更能確保只有熱門的資料才能被快取至 SSD 內，以減少對 SSD 的寫入動作。

5.2 提出方法

5.2.1 新的架構

如同上面所說，目前架構上，page cache 無法得知哪些 I/O requests 是存取固態硬碟，哪些 I/O requests 是存取機械式硬碟。所以我們將 SSD cache 管理模組上移，使得 page cache 的管理模組知道 SSD cache 的存在，則 page cache 的管理就可以支援多層級的儲存架構，並且將 page cache 的空間切割為兩個部分，分別快取來自固態硬碟(page cache_SSD)或是機械式硬碟(page cache_HDD)的資料。當一筆讀取的 request 抵達 page cache 時，如果此一資料

被快取於 page cache，則直接存取 page cache，否則，page cache 管理模組可以查詢 SSD cache 管理模組判斷此一 request 所存取的資料是否被快取於 SSD cache，如果是(SSD cache hit)，則從固態硬碟讀取此一資料，並將此一資料快取於 page cache_SSD，否則從機械式硬碟讀取並快取於 page cache_HDD；同樣的，如果是寫入的 requests，我們則會先檢查此一資料是否位於 page cache 內部，如果是，則寫入 page cache_SSD 或 page cache_HDD，如果此一資料不位於 page cache，則詢問 SSD cache 管理模組此一資料是否位於 SSD cache，如果是，則寫入 page cache_SSD，如果 SSD cache miss，則寫入 page cache_HDD。

在這樣的架構下，首先，我們解決了上面所提到的 Blind page cache management 的問題，再來，page cache_SSD 和 page cache_HDD 這兩個空間的大小可以動態調整，也可以更聰明的選擇 victim block。

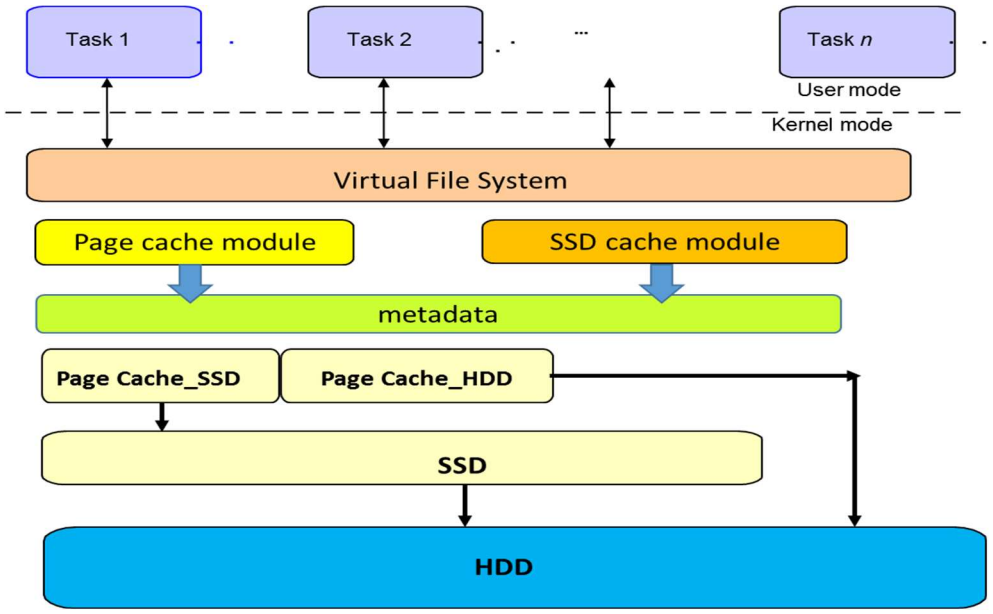

圖 5-1：新的系統架構

5.2.2 整合 Metadata

Metadata duplication:

如前述，某些 block 的 metadata 可能會同時被 page cache 和 SSD cache 的管理模組重複維護。為了解決此問題，我們設計整合的資料結構，將 page cache 需要的 metadata 和 SSD

cache 需要的 metadata 統一管理，如圖 5-2 所示，我們紀錄過去一段時間存取過的 blocks 的 metadata，其中深色部分代表 page cache 需要的 metadata、淺色部分為 SSD cache 所維護的 metadata，而灰色部分則同時被 page cache 和 SSD cache 所維護。這些灰色部分就是目前架構下造成 metadata duplication 的原因，但是，經由共享的資料結構，一來我們可以避免 metadata duplication 的缺點，二來我們可以讓 SSD cache 管理模組存取所有的 metadata。

Inaccurate/delayed Information：

如上述，許多 SSD cache 的替換演算法會維護 block 的 metadata，但是，只有當某一 block 被逐出 page cache 時，其 metadata 才會有機會被 SSD cache 管理模組取得並維護。藉由圖 5-2 的架構，我們將 page cache 和 SSD cache 需要維護的 blocks 的 metadata 統一管理，換言之，page cache 和 SSD cache 的管理模組都可以存取此一資料結構。如此可讓 SSD cache 管理模組更精準並即時的存取所需要的 metadata 資訊。例如：有些 SSD cache 的管理方法會計算資料被存取的次數，只有當資料被存取一定次數時才會被快取至固態硬碟。在此我們假設為三次，則一筆資料必須經歷三次的 SSD cache miss 才會被快取置於固態硬碟，但是，在我們新的架構下，此一資料在 page cache 被存取的次數即可約略用來判斷此一資料的熱門程度，換言之，當此一資料被逐出 page cache 時，我們即可判斷是否需要將此資料快取至固態硬碟，以避免三次 SSD cache miss 的代價。

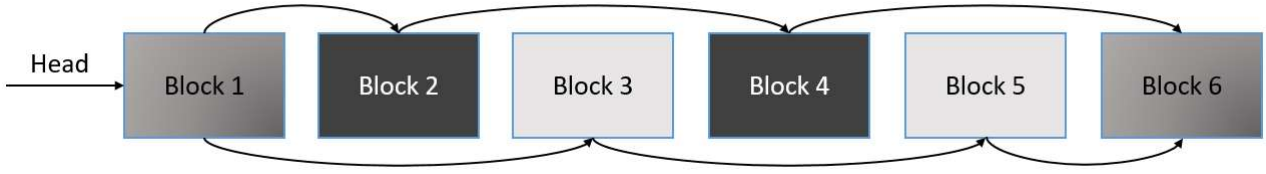

圖 5-2： 整合的資料結構

5.2.3 減少 Double Caching

如圖 5-1 所示，因為 page cache 也會快取來在 SSD cache 的資料， double caching 是不可避免的。但是，我們希望盡量降低此一現象。因此，當存取 SSD cache 發生 miss 時，我

們必須從機械式硬碟讀取此一資料，然後將之快取於 page cache，傳統的 SSD cache 管理方法可能會順便將此一資料快取於固態硬碟，此時便會發生 double caching 的現象。為了解決此一問題，我們將不此刻快取此一資料到固態硬碟，而是等待此一資料從 page cache 被逐出以後再快取於固態硬碟。而這樣的作法必須修改 page cache 的管理策略。在目前的架構下，從 page cache 逐出的資料如果是 clean，page cache 會直接丟棄，這是因為硬碟內已有對應的資料。在新的架構下，我們必須修改 page cache 的管理模組，被逐出的資料如果是 clean，也必須向下遞交到儲存裝置，如此，此一筆資料才有機會被快取於固態硬碟。

5.2.4 Page cache_SSD 與 page cache_HDD 的動態分割

如上面所提到，因為 page cache 可以知道哪些資料是來自於機械式硬碟，哪些資料是來自於固態硬碟，因此，page cache 可以適當分割其空間，分別快取來自於機械式硬碟和固態硬碟的資料。我們必須適當分配 page cache_SSD 和 page cache_HDD 的空間，以達到最大效能，也就是讓固態硬碟和機械式硬碟的負荷平均(load balancing)。我們週期性調整 page cache_SSD 和 page cache_HDD 的空間，在每一個週期，我們會計算固態硬碟和機械式硬碟的平均反應時間，使得固態硬碟的平均反應時間(Tssd)和機械式硬碟的平均反應時間(Thdd)符合下式：

$$\textbf{if } T_{HDD} <= T_{SSD} \text{ , then } T_{HDD} <= T_{SSD} <= T_{HDD} + t_{hdd}$$
$$\textbf{if } T_{SSD} <= T_{HDD} \text{ , then } T_{SSD} <= T_{HDD} <= T_{SSD} + t_{ssd} \quad (1)$$

其中，t_{ssd} 和 t_{hdd} 分別代表存取一筆固態硬碟和機械式硬碟資料的時間。由上式，我們希望固態硬碟和機械式硬碟的平均反應時間是相近的，也就是達到 load balance 的目標。因此，在每一個週期，如果我們量測出來的固態硬碟的平均反應時間和機械式硬碟的平均反應時間不符合上述式子，則我們必須調整 page cache_SSD 和 page cache_HDD 的空間。如果 $T_{HDD} < T_{SSD}$，代表從固態硬碟逐出的 requests 數目太多，也就是 page cache_SSD 的 hit ratio 太小，在 caching algorithm 假設為常數的情形下，代表 page cache_SSD 的空間太小。同樣的，

如果 $T_{SSD} < T_{HDD}$，則代表 page cache_HDD 的空間太小。

調整兩者的空間：

首先我們需要預測下一周期 page cache_SSD 和 page cache HDD 的 dirty 比率、被快取至 SSD 的比率以及 request 數，而以上所提，皆使用相同公式進行預測，預測公式如下：

$$\boldsymbol{P_n = (1-\alpha) * P_{n-1} + \alpha * C_n} \quad \textbf{(2)}$$

- P_n為這次所預測之下一周期的值
- P_{n-1}為上次所預測之值
- C_n為這一周期所記錄的值
- α為調整紀錄值與預測值之權重

而我們會將 cache 分成數個區塊，然後紀錄每個區塊在此週期中的 access 數量，並且以區塊為單位調整 page cache_SSD 與 page cache_HDD 的大小，再預測下一週期的 hit 數。以圖 5-3 為例：假設 page cache_SSD 太小，而我們將他增大一個區塊，則預測下一週期 page cache_SSD 的 hit count 為 24，而 page cache_HDD 的 hit count 為 20。

預測了下一週期的 hit 數後，我們可以搭配式(2)所計算出的 request 數計算出下一週期的 hit ratio，進而計算出下一週期的 miss 數。

$$\text{miss 數} = \text{request 數} * (1 - \frac{hit數}{request數})$$

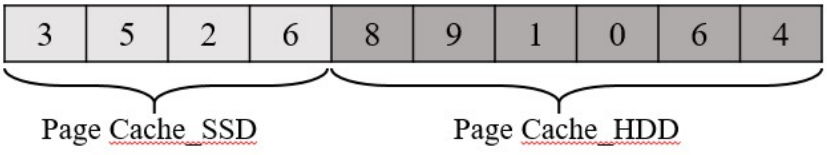

圖 5-3 紀錄每個區塊的 access 數量

由式(2)所預測出的 dirty 比率、被快取至 SSD 的比率以及 miss 數，可藉由式(3)、式(4)

算出 page cache_SSD 與 page cache_HDD miss 時所花費的時間(由於記憶體速度很快，所以我們在此忽略存取記憶體所需的時間)

- P_{pcs_read} 和 P_{pcs_write} 分別代表 page cache_SSD 的 read miss 和 write miss 個數
- P_{pch_read} 和 P_{pch_write} 分別代表 page cache_HDD 的 read miss 和 write miss 個數
- t_{read_ssd} 和 t_{write_ssd} 分別為在 SSD read, write 一個 page 所需的時間
- t_{hdd} 代表存取 disk 的一筆資料所需的時間(不分 read, write)
- D_{pch}，D_{pcs} 和 D_{sc} 分別代表 page cache_HDD，page cache_SSD 和 SSD cache 的 dirty 比率
- γ 代表從 page cache_HDD 剔除的 block 快取至 SSD cache 的比率

Page cache_SSD miss 時所花費之時間：

$$P_{pcs\_read} * \left(t_{read\_ssd} + D_{pcs} * t_{write\_ssd}\right) + P_{pcs\_write} * \left(D_{pcs} * t_{write\_ssd}\right) \quad (3)$$

Page cache_HDD miss 時所花費之時間：

$$P_{pch\_read} * \left[t_{hdd} + \gamma * \left(t_{write\_ssd} + D_{sc} * t_{hdd}\right) + (1-\gamma) * \left(D_{pch} * t_{hd}\ \right)\right] + P_{pch\_write} * \left[\gamma * \left(t_{write\_ssd} + D_{sc} * t_{hdd}\right) + (1-\gamma) * \left(D_{pch} * t_{hdd}\right)\right] \quad (4)$$

可推導出 SSD 的反應時間(T_{SSD})與 HDD 的反應時間(T_{HDD})

$$T_{SSD} = P_{pcs\_read} * \left(t_{read\_ssd} + D_{pcs} * t_{write\_ssd}\right) + P_{pcs\_write} * \left(D_{pcs} * t_{write\_ssd}\right) + P_{pch\_read} * \left(\gamma * t_{write\_ssd}\right) + P_{pch\_write} * \left(\gamma * t_{write\_ssd}\right) \quad (5)$$

$$T_{HDD} = P_{pch\_read} * \left[t_{hdd} + \gamma * (D_{sc} * t_{hdd}) + (1-\gamma) * \left(D_{pch} * t_{hdd}\right)\right] + P_{pc\ \_write} * \left[\gamma * (D_{sc} * t_{hdd}) + (1-\gamma) * \left(D_{pc}\ \ * t_{hdd}\right)\right] \quad (6)$$

在一個周期完成後，會算出T_{SSD}與T_{HDD}，若不符合式子(1)的條件，則會調整 page cache_SSD 與 page cache_HDD 的大小後再次預測下一周期的 miss request 數量，直到符合式

子(1)。

由上所述，我們將改變目前多層級儲存裝置的系統架構，透過整合 page cache 和 SSD cache 為一個 unified cache，讓 page cache 管理模組和 SSD cache 管理模組可以透過共同的資料結構來互相搭配合作，讓彼此的管理機制更為聰明，以進一步提升多層級儲存裝置的效能。

六、 系統實現與實驗

6.1 運作流程

本實驗會將舊的架構與新的架構以不同的 workload 比較兩者的 hit ratio 與反應時間(response time)。

在舊的架構下，第一層的快取為 page cache，是以 LRU 進行管理，第二層的快取為 SSD cache，以 LARC 進行管理，如圖 6-1-1。若一個 request 到達，而 page cache 與 SSD cache 皆 miss，若曾經存取過(在 ghost queue 內)，則會從 ghost queue 內移除，並加至 SSD cache 與 page cache；否則加入 ghost queue 與 page cache，如圖 6-1-2。

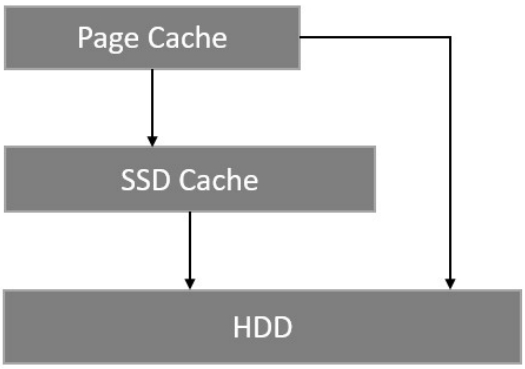

圖 6-1-1 舊的快取架構

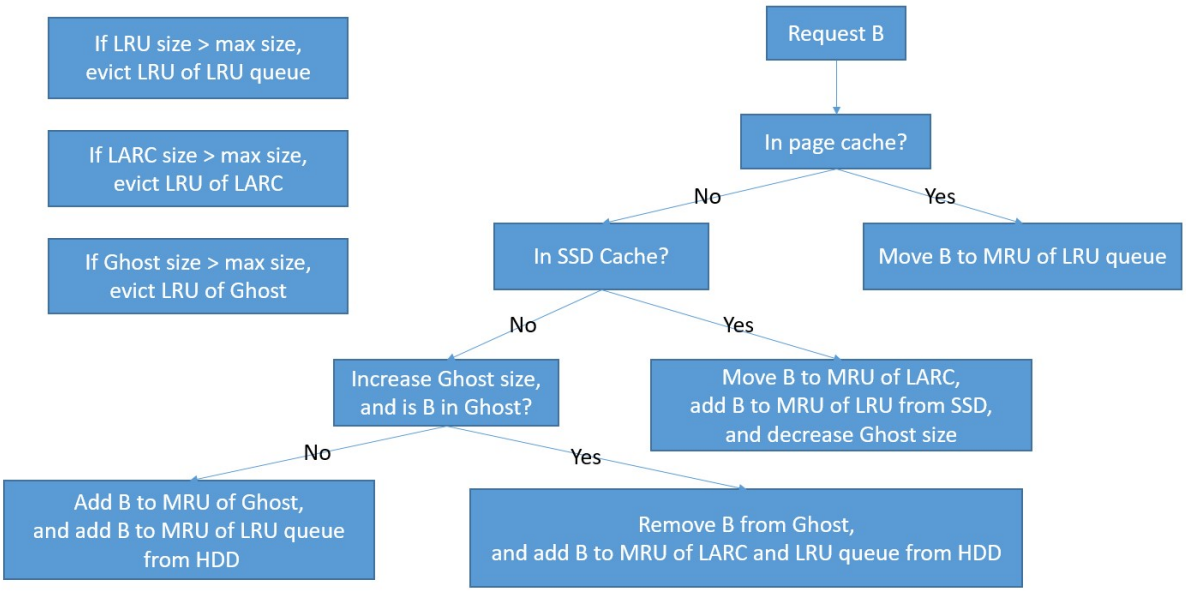

圖 6-1-2 舊的程式架構

而本文提出的新的架構則將 page cache 切割為 page cache_SSD 與 page cache_HDD，兩者仍舊採用 LRU 管理，如圖 6-1-3。

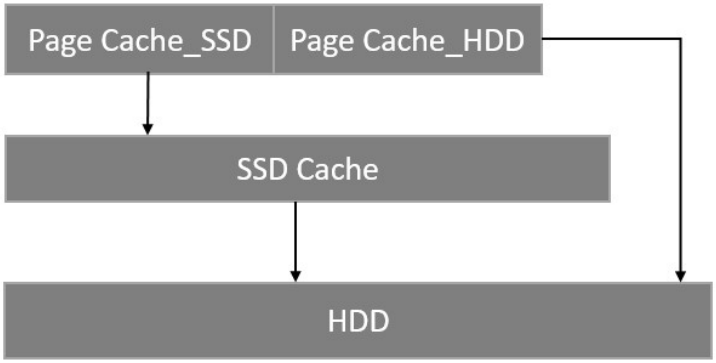

圖 6-1-3 新的快取架構

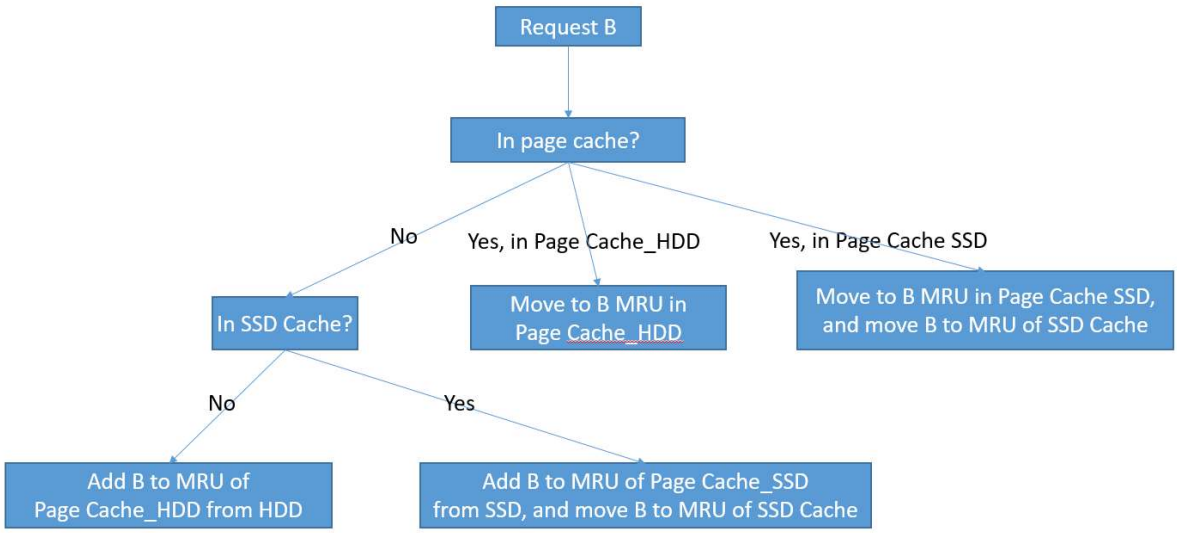

圖 6-1-4 新的程式架構

如圖 6-1-5，新的架構下，SSD cache 的管理方法改為維護一個 LRU 的 queue，而「當一個 block 被 page cache_HDD 剔除，並且它被存取的次數超過 threshold 時」，該 block 就會被加至 SSD cache 內。

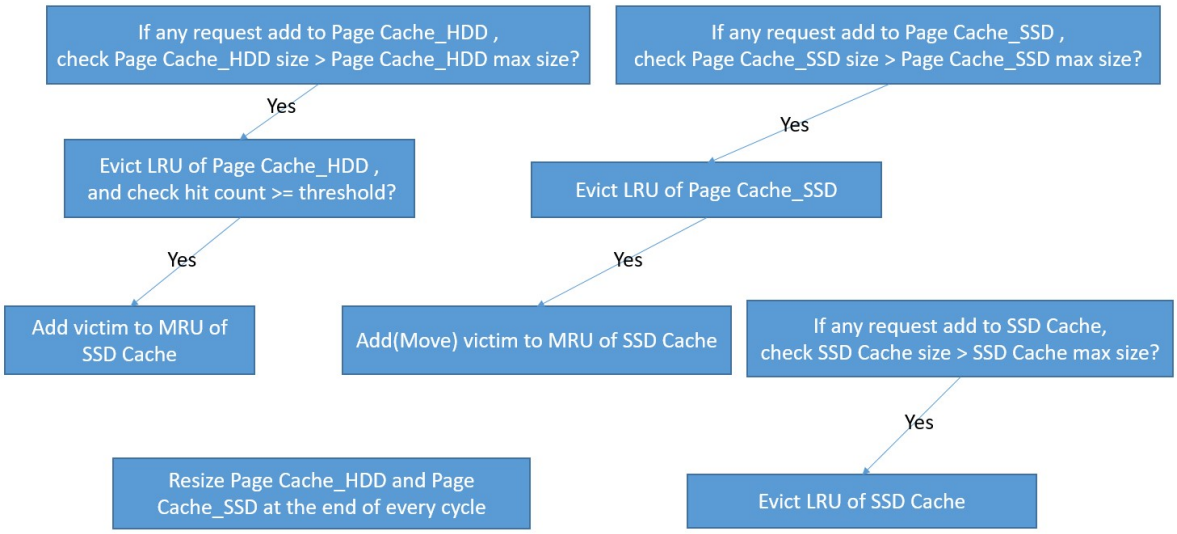

圖 6-1-5 新的架構下逐出 block 的流程

6.2 實驗環境

6.2.1 數據介紹

本實驗將依據以下三個 workload 進行實驗：

1. gcc workload

2. cscope workload

3. multi1 workload：為 gcc 和 cscope 綜合使用的 workload。

而對於以上三個 workload，我們會分析：

1. Number of reference：request 的數量。
2. Data size：所有 request 的總和大小。
3. Working set size：不重複的 request 的總和大小。
4. Read ratio：此 workload 的 read request 的比率。
5. Average inter-arrival time：request 的平均到達時間。

6.2.2 分配 Page Cache 與 SSD Cache

由於 cache 分配太小會導致效能不佳，而太大則會造成資源浪費，為了確保能夠有效進行實驗，我們將依據事先統計好的各個 workload 的 working set size 分配 Page Cache 與 SSD Cache 的大小。

6.2.3 Response Time

Response time 為在一個 workload 中，所有存取裝置的時間加總(包括存取 SSD 與存取 HDD)。如 6.1 的流程介紹，在 page cache miss 或需要釋放快取空間時，即會需要存取硬碟裝置。

七、 效能評估與成果

7.1 gcc Workload

Number of Reference	Data Size	Working Set Size	Read Ratio	Average Inter-arrival Time
158590	158590*4 KB	785*4 KB	0.879639	0.004112 s

表 7-1 gcc workload 分析的結果

圖 7-1-1 與 7-1-2 為新的架構和舊的架構在 gcc workload 下，以 page cache size 為 working set size 的 20%~90%做實驗的結果，其中 SSD cache size 為 page cache size 的 2 倍。

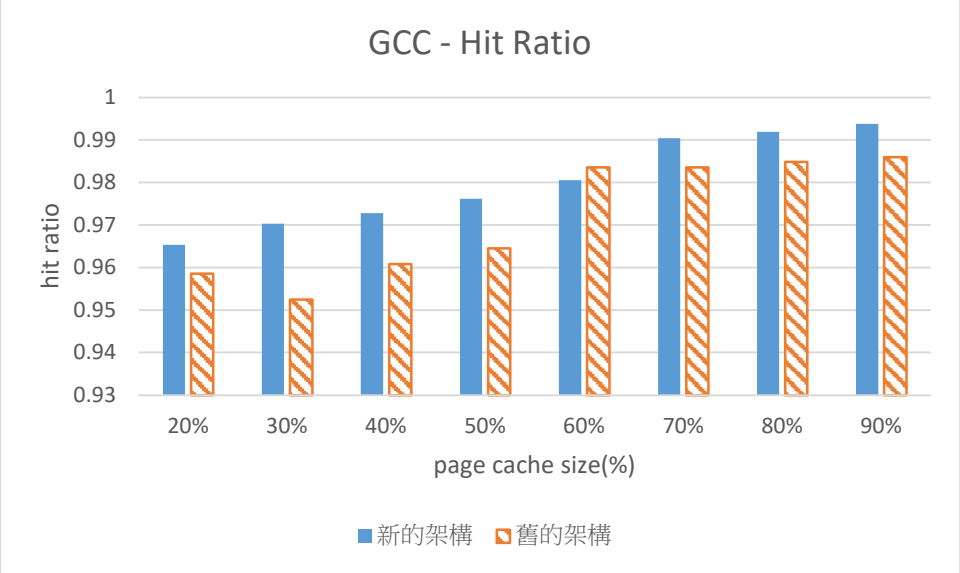

圖 7-1-1 hit ratio of gcc workload

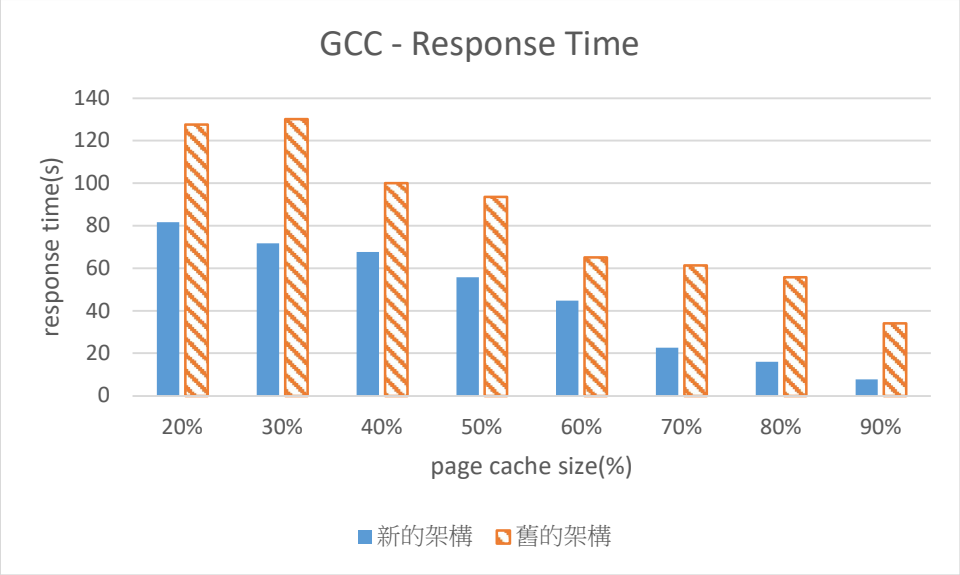

圖 7-1-2 response time of gcc workload

由以上的實驗結果可以發現，在 gcc workload 下，本文提出的方法，能有效提高 hit ratio 並降低 response time。

7.2 cscope Workload

Number of Reference	Data Size	Working Set Size	Read Ratio	Average Inter-arrival Time
1119161	1119161*4 KB	19698*4 KB	0. 984295	0. 002120 s

表 7-2 cscope workload 分析的結果

圖 7-2-1 與 7-2-2 為新的架構和舊的架構在 cscope workload 下，以 page cache size 為 working set size 的 20%~90%做實驗的結果，其中 SSD cache size 為 page cache size 的 2 倍。

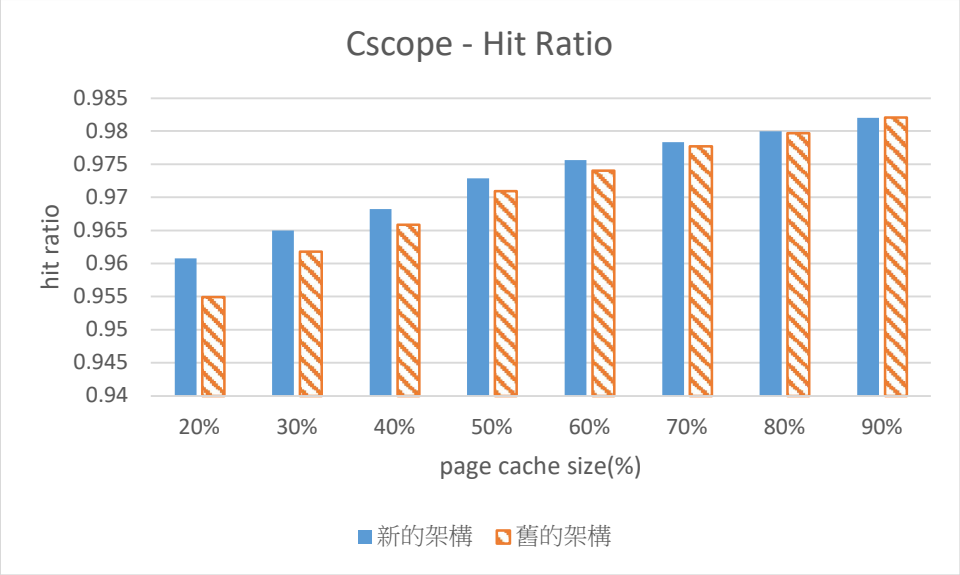

圖 7-2-1 hit ratio of cscope workload

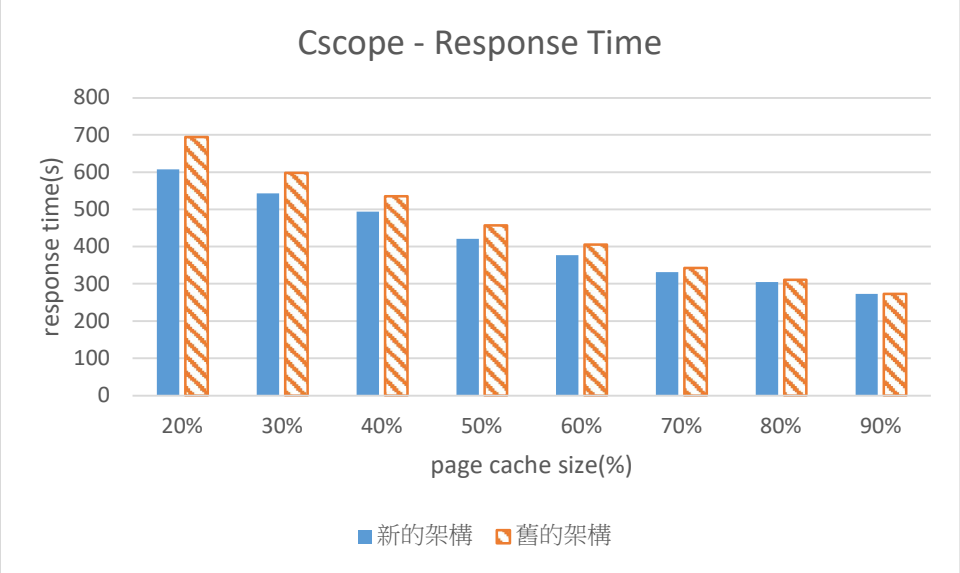

圖 7-2-2 response time of cscope workload

由以上的實驗結果可以發現，在 cscope workload 下，本文提出的方法，能有效提高 hit ratio 並降低 response time，並且隨著 page cache size 變大，改善越不明顯。

7.3 multi1 Workload

Number of Reference	Data Size	Working Set Size	Read Ratio	Average Inter-arrival Time
1264590	1264590*4 KB	19646*4 KB	0. 971337	0. 002554 s

表 7-3 multi1 workload 分析的結果

圖 7-3-1 與 7-3-2 為新的架構和舊的架構在 multi1 workload 下，以 page cache size 為 working set size 的 20%~90%做實驗的結果，其中 SSD cache size 為 page cache size 的 2 倍。

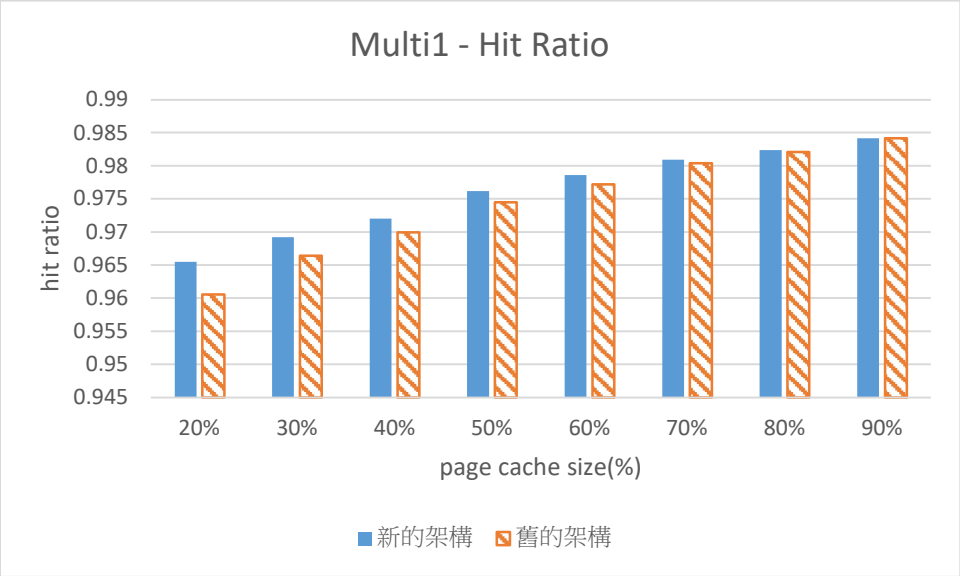

圖 7-3-1 hit ratio of multi1 workload

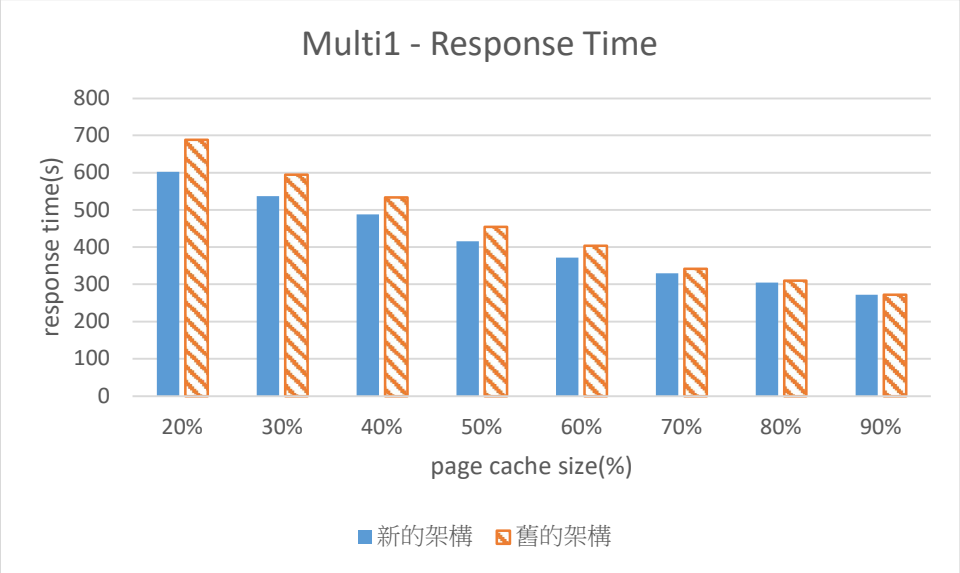

圖 7-3-2 response time of multi1 workload

由以上的實驗結果可以發現，在 multi1 workload 下，本文提出的方法，能有效提高 hit ratio 並降低 response time，並且隨著 page cache size 變大，改善越不明顯。

八、 結論

本文提出的新的架構，透過整合 metadata、動態調整等方式，改善了原先架構上會產生的問題，讓 page cache 管理模組和 SSD cache 管理模組可以透過共同的資料結構來互相搭配合作，使得彼此的管理機制更為聰明，進一步提升了多層級儲存裝置的效能。然而我們目前使用的快取管理方法皆為最簡易的方法，若能找到更為適當的快取演算法，相信可以獲得更佳的表現。

九、 參考資料

[1] S. Huang, Q. Wei, J. Chen, C. Chen, and D. Feng, “Improving Flash-based Disk Cache with Lazy Adaptive Replacement,” In IEEE 29th Symposium on Mass Storage Systems and Technologies (MSST), pp. 1–10, 2013.

[2] Young-Jin Kim and Jihong Kim, “ARC-H: Adaptive replacement cache management for heterogeneous storage devices,” Journal of Systems Architecture, No. 58, pp. 86-97, 2012.

[3] Yunpeng Chai, Zhihui Du,Member, Xiao Qin, and David A. Bader, “WEC: Improving Durability of SSD Cache Drives by Caching Write-Efficient Data,” IEEE Transactions on Computers, Vol. 64, No. 11, November 2015.

[4] Luo, Jhih-Cheng , “A Load-Balancing Data Caching Scheme for Hybrid Storage”, http://hdl.handle.net/11455/19971