國立台灣科技大學電子工程系

Taiwan Tech Department of Electronic and Computer Engineering

113學年度第一學期實務專題

總報告

Final Report

透過平行化設計增進固態硬碟之效能

組別：1122A2

組員：

姓名：林家慶 學號：B11002026

姓名：洪宥丞 學號：B11002030

指導老師：吳晉賢 教授

中華民國113年12月03日

題目：透過平行化設計增進固態硬碟之效能

組員姓名及學號：B11002026林家慶、B11002030洪宥丞

組別：1122A2

指導老師：吳晉賢 教授

一、 摘要：

本研究旨在針對固態硬碟（SSD）的效能優化問題，提出一種基於平行化設計的通道分配演算法，稱為SSD-keeper。隨著現代計算設備所需處理的資料量快速增長，SSD的效能成為系統性能提升的關鍵環節之一。現有的SSD系統在面對多種工作負載（Workload）時，往往無法高效管理資料存取，導致延遲時間增加並降低了系統的整體效率。為解決此問題，本研究採用SSD模擬器（SSDsim），模擬多種應用場景下的不同工作負載。每一個工作負載產生多個工作項目（Request），這些工作項目隨時間變化形成資料集（Dataset）。

在實驗過程中，我們透過對資料集的特徵分析，運用所設計的平行化處理演算法，為每個資料集動態設計專屬的通道分配模式（Channel Allocation Pattern）。該演算法依據工作負載的特性以及讀寫操作的比例，靈活地將工作項目分配至特定的記憶體通道，以優化記憶體位址的分配方式。此設計不僅能簡化SSD在資料處理過程中的複雜度，還能顯著降低總延遲時間，進一步提升系統的處理效率。

本研究的主要貢獻在於，通過動態調整通道分配策略，有效減少了工作負載在處理過程中的瓶頸，並優化了資料存取的效率。實驗結果顯示，與傳統的通道分配策略相比，SSD-keeper模組可使整體系統的延遲時間降低約2%。此成果顯示，SSD-keeper演算法具有優越的效能提升潛力，未來可廣泛應用於高效能計算、雲端伺服器等需要處理大量資料的應用場景中。

二、 簡介

隨著現代科技的飛速發展，資料的產生和存儲需求正以指數級增長，這對於儲存設備的效能提出了更高的要求。尤其是在大數據、深度學習和高效能計算等領域，記憶體的效能已成為系統整體性能的瓶頸之一。傳統上，研究者們大多集中在提升處理器的計算能力上，但在實際應用中，記憶體的存取效率和流暢度同樣至關重要。使用者在進行多任務處理時，如同時瀏覽網頁、編寫文檔、回應即時訊息等，這些不同的應用程式將生成大量的工作項目（Request），並透過儲存設備進行管理與調度。這些工作項目若未能被高效地處理，將導致存取延遲，影響使用者體驗。

固態硬碟（SSD）作為當前主流的存儲設備，憑藉其快速的讀寫速度和低延遲特性，成為提升系統效能的關鍵。然而，隨著工作負載的多樣化和資料量的不斷增長，SSD的效能優勢逐漸受到工作項目調度不佳和通道分配不合理等問題的制約。因此，如何在工作項目急劇增加

的情況下，保持SSD的高效讀寫能力，成為當前急需解決的挑戰。

為應對此挑戰，我們提出了一種名為SSD-keeper的演算法。該演算法旨在引入平行化處理的概念，根據不同的工作負載特徵，動態調整資料的通道分配策略，從而提升SSD的效能。在多任務並發的環境中，SSD-keeper將能有效優化工作項目在通道中的分配，避免單一通道過載或資源飢餓的情況。透過此方式，不僅能顯著降低系統的延遲時間，還能增強使用者在多任務操作中的流暢度，進而全面提升系統的使用體驗。

本研究的動機在於解決SSD在多任務處理環境下的性能瓶頸問題，並探討如何通過平行化設計來提高資料存取效率。我們相信，隨著資料規模的持續增長，SSD-keeper演算法將為SSD的效能提升提供一個有效的解決方案，並在未來的儲存技術發展中發揮重要作用。

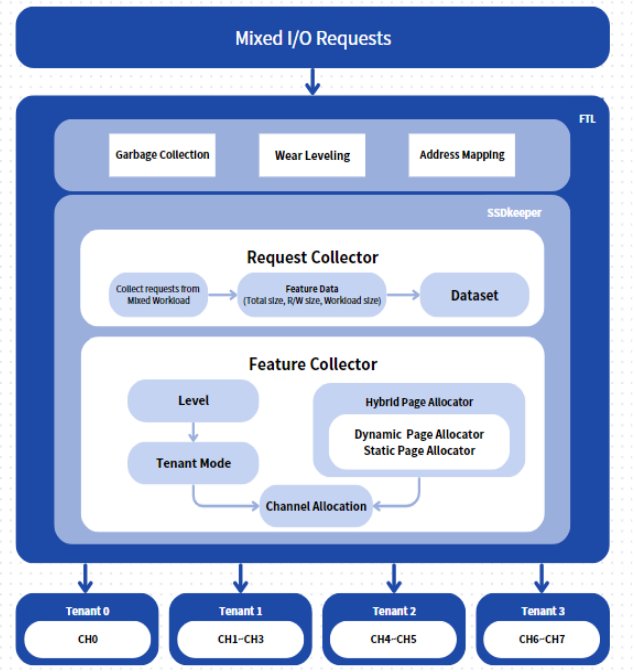


圖 一 系統架構圖

三、 方法

I. 系統設置

本研究在 VirtualBox 虛擬環境中進行實驗，選擇了 Linux 系統作為操作平台，以 Ubuntu 16.04 作為作業系統的基礎環境。該環境具備穩定的性能和廣泛的相容性，適合進行 SSD 模擬及演算法設計開發。我們選用了 SSD 模擬工具（SSDsim），該工具以其高效、精確的模擬能力，能夠對固態硬碟在不同工作負載（Workload）條件下的效能進行細緻評估。SSDsim 模擬了 SSD 內部的工作機制，允許我們對各類工作負載進行深入的效能分析和實驗測試。在開發過程中，我們採用了 C 語言作為主要開發工具，確保模擬過程中的效率和資源的有效利用。透過這種設置，我們為實驗的高效運行奠定了堅實的技術基礎。

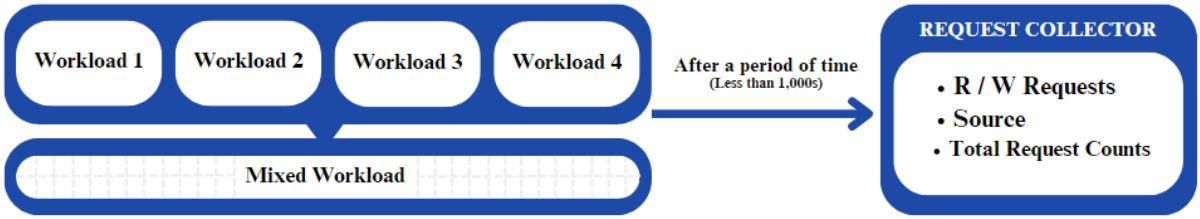
II. 數據集建構

在實驗中，我們所使用的工作負載數據集來自 MSR Cambridge Trace 中的四種不同類型的工作負載，這些工作負載反映了現實生活中常見的應用場景，如檔案服務器、網頁伺服器和數據庫系統等。為了模擬多任務環境下的 SSD 性能表現，我們將這四種工作負載混合成統一的混合型工作負載（Mixed Workload），作為實驗的主要輸入數據。這種混合型工作負載更接近於真實世界中的複雜應用環境，能夠反映出不同應用程式同時進行操作時的效能結果。

混合型工作負載經由 SSD 的 FTL（Flash Translation Layer）層進行處理。工作負載的各個工作項目（Request）會由 Request Collector 進行掃描和數據蒐集。Request Collector 在每個指定的時間段內記錄下來的資料集（Dataset），包含了工作負載的詳細統計特徵，如請求的總數量、讀寫請求的比例、各工作負載的來源等。為進一步提升數據蒐集的效率，Request Collector 設置了幾項關鍵參數：

1. 最大資料集大小（Max Size）：資料集中最多容納 10,000 筆工作項目。
2. 最大掃描時間間隔（Max Scanning Time Interval）：最多為 1000 秒，因每個請求事件發生的時間間隔為 0.1 秒。
3. 數據記錄（Data Recording）：包括讀寫工作請求總數（Read/Write Requests）、各工作負載來源的請求數量（Requests from which Workload），以及資料集中的總請求數（Total Requests）。

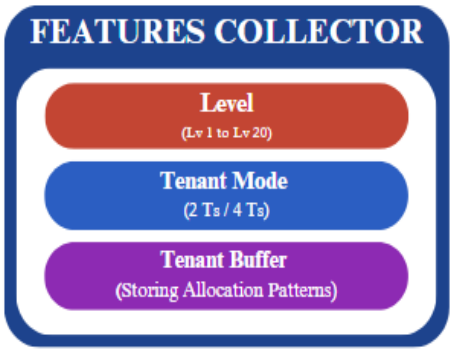
這些蒐集到的統計特徵將成為後續通道分配策略設計的基礎，為 SSD 資源分配提供數據，從而提高系統的整體效能。



圖二、Request Collector 示意圖

III. 演算法設計

本研究所提出的 SSD-keeper 演算法的核心在於基於動態通道分配的平行化處理策略。首先，我們從資料集中提取出每個時間段的統計特徵，這些特徵經由 Feature Collector 進行深入分析。Feature Collector 在分析過程中，根據資料集中的工作項目數量為其定義 Level，該 Level 範圍從 1 至 20，每 500 筆工作請求為一個 Level。這些 Level 不僅能反映資料集的規模，也為後續的通道分配模式選擇提供依據。



圖三、Features Collector 示意圖

演算法根據資料集的 Level 值來決定使用的 Tenant 模式。Tenant 模式分為「2 Tenants」和「4 Tenants」，主要依據資料集的 Level 大小來決定。當資料集的 Level 小於或等於 10 時，演算法會進入「2 Tenants」模式；反之若 Level 超過 10，則採用「4 Tenants」模式。此外，若出現 Dominant Workload，即某一工作負載的請求量佔據總資料集的 90%以上，無論 Level 為何，都將使用「2 Tenants」模式進行分配。根據選定的 Tenant 模式，SSD-keeper 演算法會動態調整通道分配策略並將策略儲存至 Tenants Buffer 中：

1. 2 Tenants：
   當資料集的 Level 較小（Level ≤ 10）又或者是出現 Dominant 情況，我們僅考慮該資料集中讀寫請求的比例。演算法將通道依照讀寫比例分配給兩個 Tenants，確保多數 Tenant 專門處理讀或寫請求。這種模式可以加快通道分配過程提高效率。此時的 Tenants Buffer 將儲存[R 比例,W 比例]。
2. 4 Tenants：
   當資料集的 Level 較大（Level > 10），通道的分配則更為複雜。演算法將資料集中的工作負載按照來源進行劃分，並且進一步根據讀寫請求的比例來動態分配通道。每一個 Workload 的讀寫請求數量與通道的比例相乘後，確定每組 Tenant 所需的通道數量，從而有效分配資源並提高工作負載的處理效率。此時的 Tenants Buffer 將儲存[Workload1*R 比例, Workload1*W 比例, Workload2*R 比例, Workload2*W 比例]。

為了避免發生資源飢餓（Starvation）現象，無論在「2 Tenants」或「4 Tenants」模式下，SSD-keeper 演算法都確保每組 Tenant 至少會分配到一條通道進行處理。例如，在「2 Tenants」模式下，即便讀寫請求的比例非常懸殊，也會強制將通道分配為 1:7，以保證讀寫請求都能被有效處理。

此外，為了驗證 SSD-keeper 演算法的效能，我們設計了兩種極端的通道分配策略作為對照。一種是 Partially Isolated 策略，也就是將所有通道完全用於處理讀或寫單一操作；另一種

是 Shared 策略，也就是將通道均勻分配給讀寫操作。這兩種極端策略是用來比較實驗結果當應對各種工作負載場景中，在同樣基準下的結果與靈活性差異。

IV. Hybrid Page Allocator 動靜態頁分配器

因應不同工作型態(讀/寫工作)的特性，記憶體在通道分配上會有著不同的考量。對於寫工作來說，因通常附帶擦除工作(Erase)，需先將塊上的資料擦除後再重新寫入。此時如果寫工作分散在不同的區塊，就會導致大量的擦除工作，進而增加「WA (Write Amplification，寫入放大)」的機會。針對這個現象，我們會需要考慮「動態頁分配」策略，它會盡可能地掃描是否有「活頁」，以減少擦除工作的產生，另一方面也能達到 WL(Wear Leveling，磨損均衡機制)的效果，以增加記憶體使用壽命。

由於記憶體執行讀工作時是不需要做擦除工作的，所以我們可以隨意地存取數據。「靜態頁分配」將能夠分配數據在「固定的」位置上，這樣做可以增加讀取效率，因為數據都被寫在特定的位置上而簡化了映射表。可以想像成每天上班的路徑都一樣，我們就無需每次上班都要看著地圖慢慢找路才走到目的地！

四、 實驗

實驗結果明確顯示，SSD-keeper 模組在提升 SSD 效能方面取得了顯著的成效。圖六對比了四種不同通道分配策略的讀取延遲、寫入延遲和擦除延遲，並展示了它們的總延遲。與原模擬器（SSDsim）的性能相比，SSD-keeper 模組將總延遲時間從 2456.7 微秒減少至 2387.6 微秒，降低了約 2%（見圖五）。這表明 SSD-keeper 在動態通道分配和多樣化工作負載處理方面具有明顯的優勢。

I. SSD-keeper 策略比較

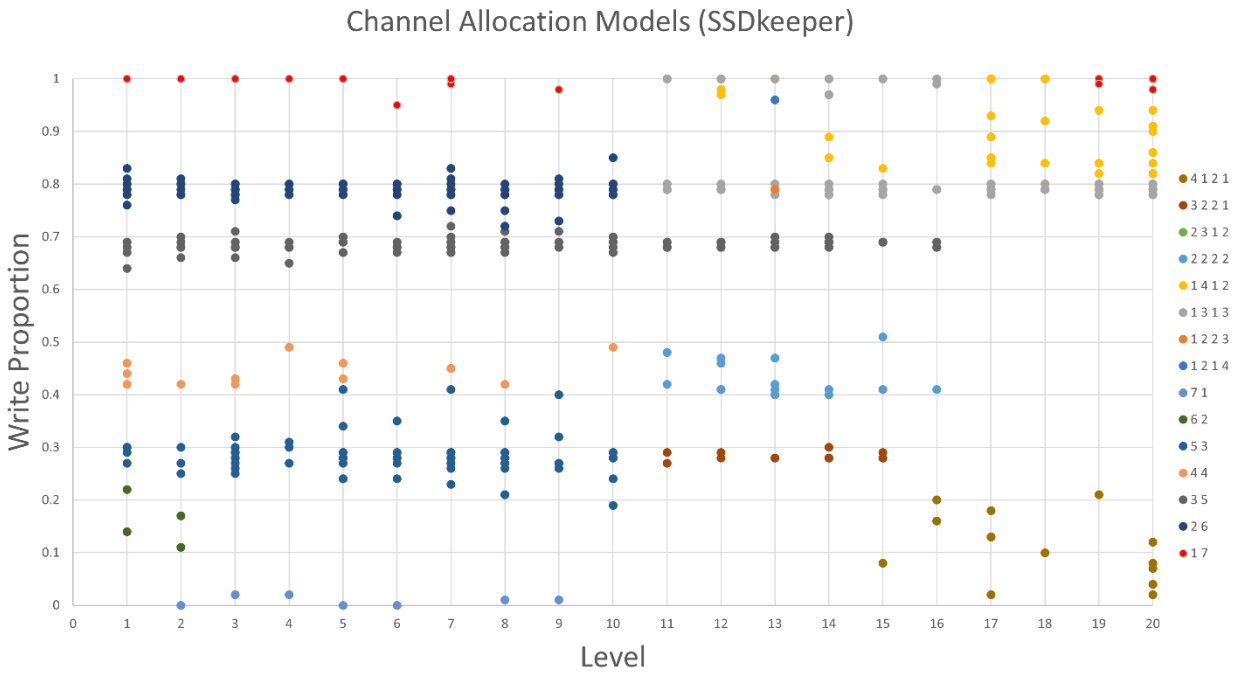
根據圖四所示，SSD-keeper 根據工作負載的 Level 動態地調整通道分配。各個 Level 的寫入比例分佈顯示出 SSD-keeper 在動態工作負載下能夠平衡資源的分配，避免了資源飢餓現象。相比之下，兩種極端策略的通道分配模型無法實現如此靈活的調整。

1. Read Latency（讀取延遲）：SSD-keeper 的讀取延遲（96.07 微秒）與 SSDsim 的 96.85 微秒相當，這表明 SSD-keeper 並未在讀取效率上做出妥協。反觀 Partially Isolated 和 Shared 策略，由於缺乏資源調整能力，它們的讀取延遲均高達 126 微秒以上（見圖五、圖六）。
2. Program Latency（寫入延遲）：SSD-keeper 在寫入延遲上顯著優於其他策略，其 2291.5 微秒的寫入延遲顯示出其在動態寫入操作中的資源優化效果，優於 SSDsim 的 2304.4 微秒（見圖五、圖六）。這顯示了 SSD-keeper 在動態寫入操作中對資源的優化能力。
3. Erase Latency(擦除延遲)：SSD-keeper 因為加入了「動靜態頁分配器(Hybrid Page Allocator)」，使工作項目更有效率地被分配到物理頁上，減少 GC（Garbage Collect）的發生，自然也就減少擦除工作的發生機率，這也成為實驗結果的關鍵成因之一。

此外 SSD-keeper 模組不僅顯著降低了總延遲，還有效地防止了資源飢餓現象。透過動態的通道分配策略，SSD-keeper 確保每個工作負載都能獲得足夠的資源，即使在高工作負載的情況下，系統資源仍能實現均衡分配（見圖四）。這進一步優化了不同 Level 下的資源利用率，從而降低了系統延遲並提升了系統吞吐量。

II. 極端策略比較

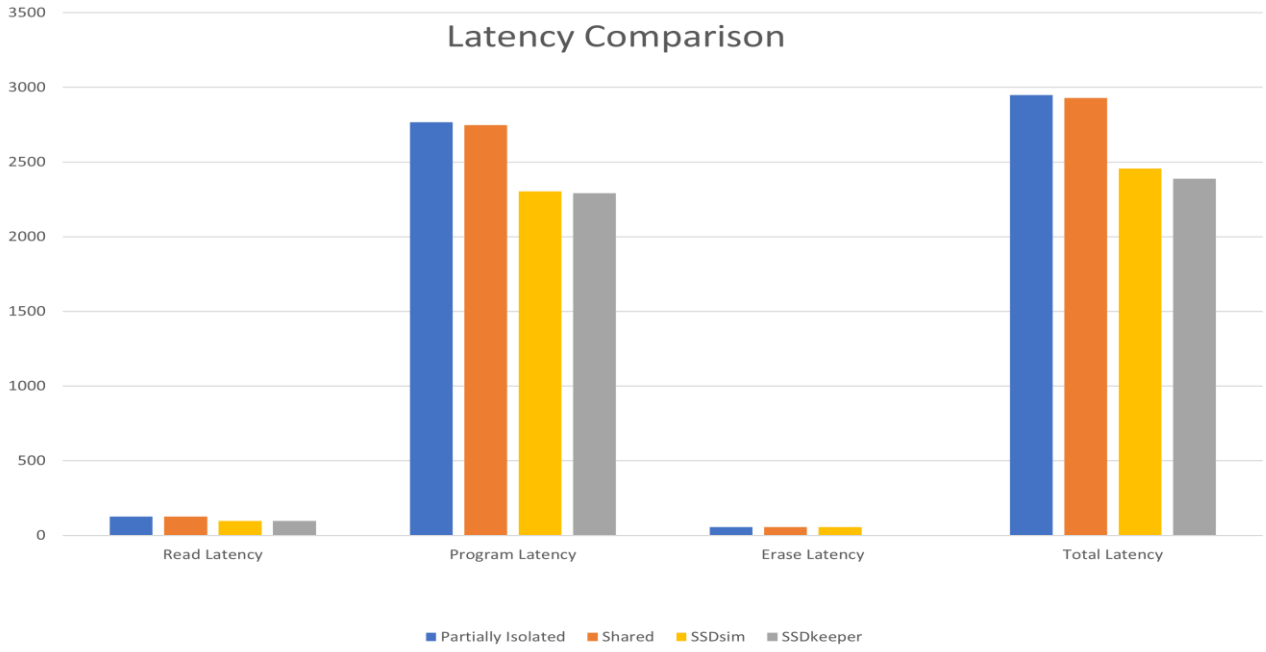
對比 Partially Isolated 和 Shared 這兩種策略（見圖五），可以明顯看出這些極端分配策略對系統效能的負面影響。這兩種策略的總延遲分別為 2948.8 微秒和 2928.9 微秒，明顯高於 SSD-keeper 和 SSDsim。Partially Isolated 策略將所有通道資源集中於單一操作（讀或寫），導致資源無法靈活利用。而 Shared 策略雖然平均分配資源，但無法應對動態工作負載變化，導致資源分配不均。因此，這兩種策略的讀寫延遲明顯增加，進一步證實了動態資源分配對提升效能的重要性。



圖四、SSD-keeper 通道策略分布圖

	Read Latency	Program Latency	Erase Latency	Total Latency
Partially Isolated	126.857	2767.3054	54.6765	2948.8389
Shared	126.257	2747.0654	55.539	2928.8614
SSDsim	96.85006	2304.3934	55.449	2456.69246
SSDkeeper	96.06928	2291.4986	0	2387.56788

圖五、延遲數據比較表



圖六、總延遲對比長條圖

五、 討論與結論

本研究通過實驗驗證了 SSD-keeper 模組在優化 SSD 效能方面的顯著作用。實驗結果表明 SSD-keeper 演算法能夠動態調整通道分配策略，根據工作負載的特性進行靈活資源管理，從而有效降低了系統延遲。與傳統模擬器（SSDsim）相比，SSD-keeper 模組將總延遲時間降低了約 2%，這主要得益於其優化的擦除延遲與寫入延遲。相比之下，Partially Isolated 和 Shared 這兩種極端分配策略因缺乏動態調整能力，導致總延遲大幅增加，進一步證明了動態資源分配策略的重要性。

SSD-keeper 演算法不僅在讀寫操作上提升了效能，還避免了資源飢餓現象，確保了系統資源在多樣化工作負載下的平衡分配。這使得 SSD-keeper 在不同 Level 的工作負載下都能實現高效運作，展示了其優異的資源管理能力與系統適應性。

基於本研究結果，SSD-keeper 在未來有廣泛的應用潛力，特別是在需要高效處理大量多樣化工作負載的應用場景中，例如深度學習資料、高效能計算系統和雲端存儲服務等等。未來研究可以進一步探索 SSD-keeper 在更大規模、更複雜工作負載下的表現，並引入更多類型的工作負載來檢驗其在實際應用環境中的適用性。此外，優化 SSD-keeper 的演算法結構，進一步提升其在多層次存儲系統中的運行效率，也是未來研究的一個重要方向。

另一個值得探索的領域是將 SSD-keeper 與其他先進技術相結合，例如機器學習與深度學習，通過智能化的負載預測資源進行調整，進而實現更精細的通道分配策略。總之 SSD-keeper 為優化 SSD 性能提供了一種有效的解決方案，其靈活的動態資源管理策略在未來存儲系統設計中具有廣泛的應用前景。

六、 參考資料

[1] Liu, R. et al. (2020) ‘SSDKeeper: Self-adapting channel allocation to improve the performance of SSD devices’, 2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS) [Preprint]. doi:10.1109/ipdps47924.2020.00103.

[2] Lim, H.J., Shin, D. and Han, T.H. (2022) ‘Parallelism-aware channel partition for read/write interference mitigation in solid-state drives’, Electronics, 11(23), p. 4048. doi:10.3390/electronics11234048.