國立中興大學資訊科學與工程學系

碩士學位論文

利用機器學習輔助寫入緩衝管理方案

A Machine Learning-Assisted Write Buffer Management Scheme

研究生：黃聖穎 Sheng-Ying Huang

指導教授：張軒彬 博士 Hsung-Pin Chang

中華民國一百一十一年七月

國立中興大學 資訊科學與工程學系

碩士學位論文

題目：利用機器學習輔助寫入緩衝管理方案

A Machine Learning-Assisted Write Buffer Management Scheme

姓名：黃聖穎 學號：7109056252

經 口 試 通 過 特 此 證 明

論文指導教授 張軒彬

論文考試委員 張軒彬 謝仁文 張大緯

中華民國 111 年 7 月 1 日

摘要

為了改善 SSD 效能與壽命，SSD 使用了一塊 RAM-based write buffer 來減少 flash memory 的寫入次數。然而，由於 write buffer 容量有限，在面對隨機且大量的使用者 requests 時，hit ratio 仍然不甚理想，因此，本篇論文使用 machine learning 的方式，來提升 write buffer 的 hit ratio。

由於使用者所送去的 request 具備時序性，也就是說，前面的 requests 與後面的 requests 有一定的關係。要預測具備這樣性質的 requests，一般神經網路無法做到，因為一般的神經網路沒辦法得知上一個時間點的輸入，因此這篇論文使用 RNN 神經網路 RNN 與一般神經網路不同的地方在於能夠記住上一個時間點的資訊，相較於其他神經網路更適合解決關於使用者 I/O 的相關問題。

但是 RNN 並不能做到完美預測，因此我們在 online 也做了優化，為了避免 model 誤判使不必要的資料持續被放在 write buffer，我們針對很久沒被存取的資料做 Demoting，讓那些資料能夠漸漸地從 write buffer 被踢除，盡可能減少 model 誤判所帶來的影響。與此同時，我們也參考 host 端的資訊，讓 write buffer 在挑選 victim block 時，透過 host 端的資訊，也就是預測會被踢到 SSD 的 page number，可以更準確地找出踢除的對象。

Abstract

SSD uses a RAM-based write buffer to reduce flush memory write count and improve lifetime and performance. However, due to the limit of write buffer capacity, the hit ratio is not good enough when facing random and large user requests.

This paper uses RNN(recurrent neural network) to improve the write buffer hit ratio. Unlike common neural networks, RNN has internal memory that stores previous state information and is more suitable for solving user I/O-related problems.

But RNN model is not perfect, so we use online demoting for misjudging data, which will move data to the next queue(hot->mean, mean->cold) if data has not been used for a long time. Online demoting can flush unnecessary data from the write buffer and lower the influence caused by model misjudging.

At the same time, we use the host information, which records dirty page numbers that flush to the write buffer. With host information, we can select victim block more accurately.

目錄

圖目錄

表目錄

第一章　緒論

1.1　簡介

NAND flash memory 組成的固態硬碟(SSD)，比起傳統的 HDD 具備速度更快、抗震性佳、體積小等特性。近年來，SSD 價格逐漸下降，且容量變大，漸漸取代傳統硬碟，現今手機、平板等許多儲存裝置都使用 SSD。

但是SSD仍然存在些許缺點，像是，NAND flash memory無法in-place-update，這讓 SSD 在更新資料變得較為麻煩，首先要知道在 SSD 讀寫最小單位是 page，但擦除的單位卻是 block，若想原地覆寫一個 page，必須先擦除那個 page 所在的 block，然後才能寫入。此外，NAND flash memory 的每一個 block 有被擦除的次數上限，當一個 block 被擦除次數超過上限，代表這個 block 無法被存取，為了能盡可能延長 SSD 的壽命，通常會在 SSD 上層加入一個 RAM-Based 的 buffer，也就是 write buffer，短時間內頻繁被寫入的資料，可以先在 write buffer 內 in-place-update，這樣的做法可以大量減少 flash memory 被寫入的次數，進而提升 SSD 的壽命，而 write buffer 要產生作用，勢必需要具有一定的 hit ratio，因此 victim block 就很重要。

1.2　研究動機

目前，page cache 與 write buffer 都是各自獨立運作，因此 write buffer 在選擇替換 victim 時，可能會做出不好的選擇，導致過早剔除，為了能夠更精確地挑選替換的 victim，我們使用 Host-aware Write buffer Management [1]中從 Host 端所預測的資訊，來幫助我們選擇 victim，提升 write buffer 的 hit ratio，減少 Flash memory 的寫入次數，提升 SSD 的壽命。

此外，由於我們無法得知未來的資訊，對於使用者隨機的 requests，在挑選 victim 時，容易做出錯誤的判斷，為了解決這樣的問題，我們使用 machine learning 來預測，希望 machine learning 藉由訓練能夠掌握到我們無法得知的【未來】資訊，進而做出更精準的判斷。

1.3 貢獻

本篇論文利用 AI 來選擇合適的 victim block 踢除，根據資料的特性，AI model 會判斷出資料屬於 hot data or cold data，並依照 AI model 預測的結果決定踢除資料的順序，並在 Online 使用 Demoting 降低 AI model 誤判所造成的影響。

此外，為了更進一步提升 hit ratio，我們參考了主機端的資訊(Hint queue) [1]，結合現有的 AI model，讓 write buffer 不但能夠得知當前資料的特性(hot data/code data)，也能知道未來有哪些資料可能被寫入。根據這些資訊，來決定該優先將那些 block 挑選為 victim。

1.4 論文架構

第二章介紹一些相關的背景知識以及 write buffer 的相關論文， 第三章描述本論文的架構與實作方法，第四章為實驗數據與分析。最後，第五章為結論、未來工作。

第二章　背景知識與相關研究

本章先介紹 SSD 相關背景知識，接著介紹 Linux flash 方式、快閃記憶體的特性、固態硬碟的管理方法，最後介紹 Neural Network 的運作和 LSTM

2.1　SSD

SSD 是由 NAND Flash memory 所組成，因為是非揮發性記憶體，因此能在斷電的情況下仍然保存資料，此外還有一些吸引人的特性，例如，體積小、抗震，但最吸引人的部分在於它的速度，比起 HDD 快上非常多，因此漸漸被個人電腦、手機、平板當作主要的儲存裝置。

表 2-1 快閃記憶體的基本操作單位

Operation	Unit	Access time
Read	Page	25μs
Write	Page	200μs
Erase	Block	2ms

如表 2-1，基本操作可分成 read、write、erase，但由於 NAND Flash memory 的物理特性，SSD 存在幾個缺點：

1. 從表 2-1 得知，read/write 速度不對稱
2. NAND Flash memory out-of-place-update 的特性，當 NAND Flash memory 在 overwrite 時，沒辦法像 HDD 一樣 in-place-update，必須找另一個空的 block 才能夠寫入，這讓 NAND Flash memory 在 overwrite 時，變得很麻煩，首先，每次 overwrite 都找新的 block 寫入，很耗費 NAND Flash memory 的空間與時間，因此會需要在空間不足時，藉由 erase 來釋出更多空間，而 erase 的最小單位又與 read/write 不同，是以 block 為單位。
3. 每個 block 有 erase 的次數限制，如果過於頻繁的 erase，當某個 block 達到 erase 次數上限，這時候所有 block 都無法再被存取

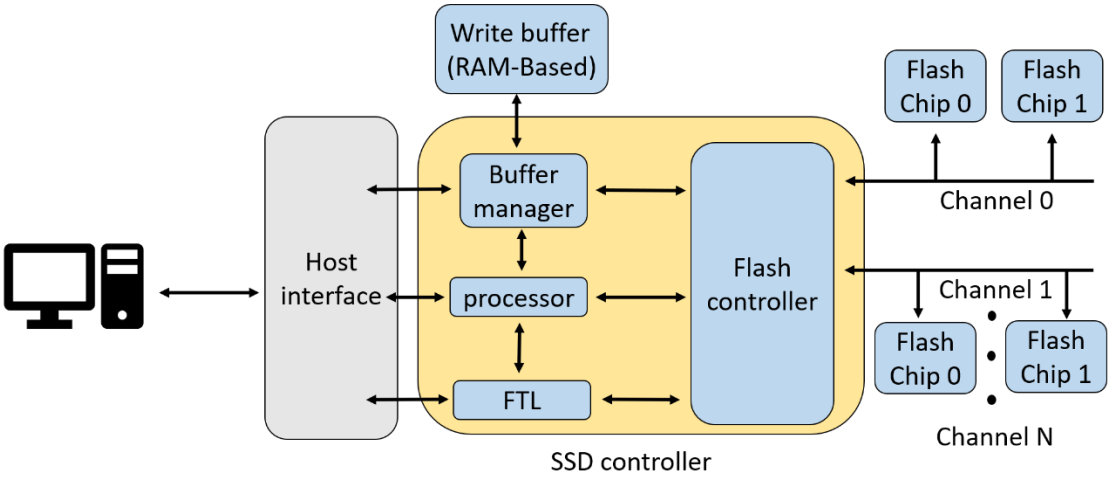

圖 2-1 SSD 架構

接下來介紹SSD架構圖，如圖2-1可看到SSD內部由process、buffer manager、flash controller所組成，SSD controller負責處理內部元件運作以及與host interface之間的溝通，而SSD controller內部有FTL做為控制中樞，此外，為了減少寫入次數& response time，SSD使用一塊RAM-Based write buffer放在SSD的上層吸收來自host端的write request，只有在write back，或是write buffer滿的時候，才需要寫入NAND Flash memory。

2.2 Write buffer 管理方式

為了延長NAND Flash memory的壽命，目前常見的方法是將一塊RAM-based write buffer放到NAND Flash memory的上層，當page cache將資料從host端踢下來時，資料會先在write buffer做寫入，若之後寫入同樣資料，就會直接在write buffer hit。等到write buffer滿了，或是之後做寫回的時候，才會需要寫入NAND flash memory，以這樣的方式，可以大量減少NADN flash memory的寫入，進而達到延長SSD壽命的目的。

然而，write buffer能夠延長SSD壽命，是建立在【hit ratio夠高】這個前提下，因此，該踢掉那些資料、該保留那些資料，成為一個不可忽視的重點，接下來介紹幾個常見的write buffer管理策略。

2.2.1 FAB(Flash-aware-buffer management policy)

在 NAND flash memory 中，Erase 是以 block 為單位，因此，一次將越多 page 寫下去，之後在做 GC 所花的時間成本就會越少。

FAB 的設計主要就是以這個概念為核心，如圖 2-2，對 FAB 來說，不會先去選擇 LRU 端的 block，而是會先選擇擁有最多 page 的 block。如果有多個 block 的 page 數目相同時，如圖 2-2 中，LB 0、LB 6、LB 3 都有 3 個 page，也就是說他們都是 victim block 的候選人，那這時候，會去挑選位於 victim block 候選人當中 LRU 端的 block，也就是 LB 0。

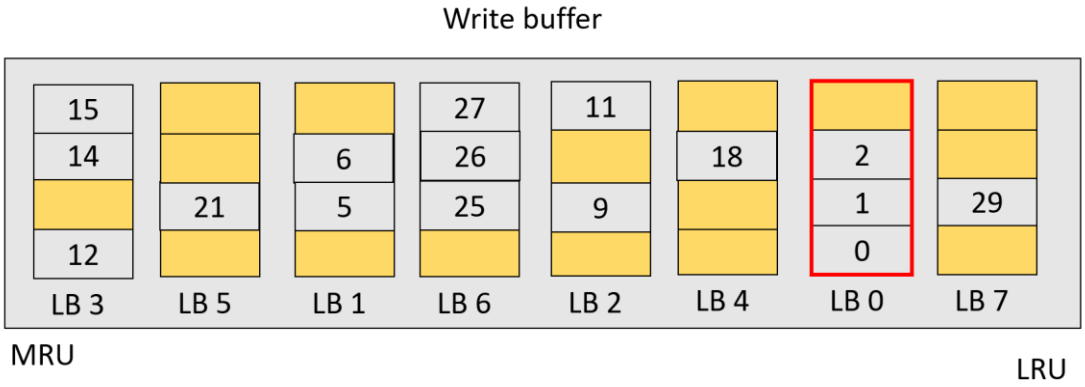

圖 2-2 FAB 架構

2.2.2 BPLRU(Block Padding Least Recently Used)

BPLRU [2]同時考慮了 temporal locality 和 spatial locality，一方面，將同個 logical block 的 page 放在一起，另一方面，只要存取到 logical block 的任何一個 page，就會將整個 block 移到 MRU 端(如圖 2-3)。

當 write buffer 滿了，會選擇 LRU 端的 LB7 當成踢除的對象，此外 BLPRU [2]為了減少 GC 的時間成本，會執行【Page padding】，如圖 2-4，先將 victim block 沒有的資料(page 5、page 7)從 Data block 讀到 write buffer，再寫入 NAND flash memory，讓原先的 block 成為 invalid block，可以節省之後 GC 所花的時間成本。

最後，BPLRU [2]還有一項機制叫做【LRU compensation】，當 write buffer 中某個 block 被循序寫滿時，就將它擺放到 LRU 端，讓它能夠優先被踢掉。

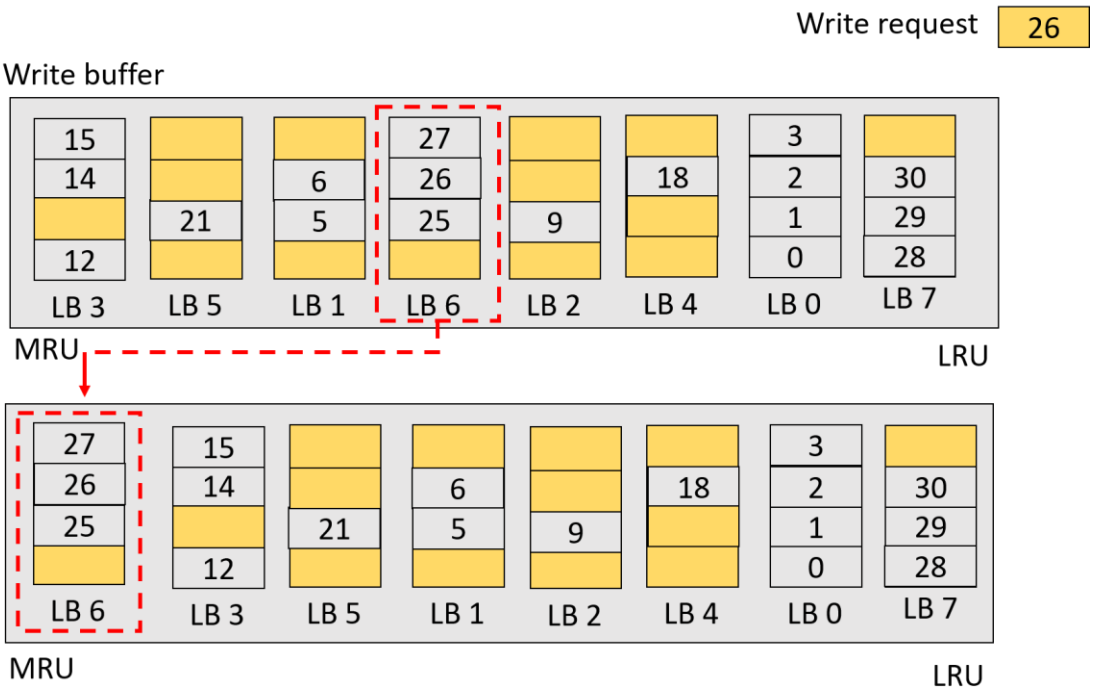

圖 2-3 BPLRU 架構

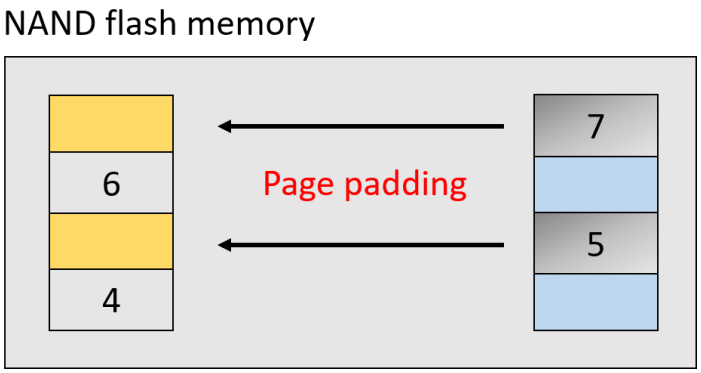

圖 2-4 Page Padding

2.2.3 Host-aware write buffer management

Host-aware Write buffer Management [1]為了能更準確的挑選 victim block，write buffer 在挑選 victim block 時，會參考 Hint queue 資訊。

Page cache 會在三種狀況下將資料踢到 SSD：

1. Replacement：當 Page cache 滿了，會將資料踢到 SSD，此時會優先選擇 LRU 端的 dirty page。

2. Dirty amount：當 dirty page ratio 達到所制定的 threshold 時，會將 LRU 端的 dirty page 踢到 SSD
3. Dirty time：:為了避免 MRU 端的 dirty page 一直沒被寫回，Page cache 會定期監測每個 dirty page 待在 Page cache 的時間，超過某個 threshold，就會將 dirty page 給踢下去。

根據這三種 Flush 方式，Host-aware Write buffer Management [1]會週期性預測即將被寫入到 SSD 的 dirty page，然後將 dirty page number 存在 SSD 內部的 Hint queue 給 write buffer 當作挑選 victim block 的參考，如圖 2-5，如此一來，就能夠避免挑選【即將被寫入】的 block 為 victim block。

此外，由於 write buffer 在選擇 victim block 時，為了減少 GC 所花的時間，會傾向於將 large block 給踢下去，但是 write buffer 通常是以 logical block 的方式擺放，這時候會產生一個問題，如圖 2-6，logical block 4 位於 LRU 端，同時也是 large block，理論上踢下去可以節省不少 GC 的時間成本，但實際上 logical block 4 在 Flush memory 的擺放位置並不如預期，在 Flush memory，Logical block 4 被分散寫入到不同的 physical block，如此一來，仍然會增加 GC 所花的時間。因此 Host-aware Write buffer Management [1]將資料以 physical block 的方式放入 write buffer，在 write buffer 就直接得知資料實際的擺放位置，避免挑選到錯誤的 victim block。

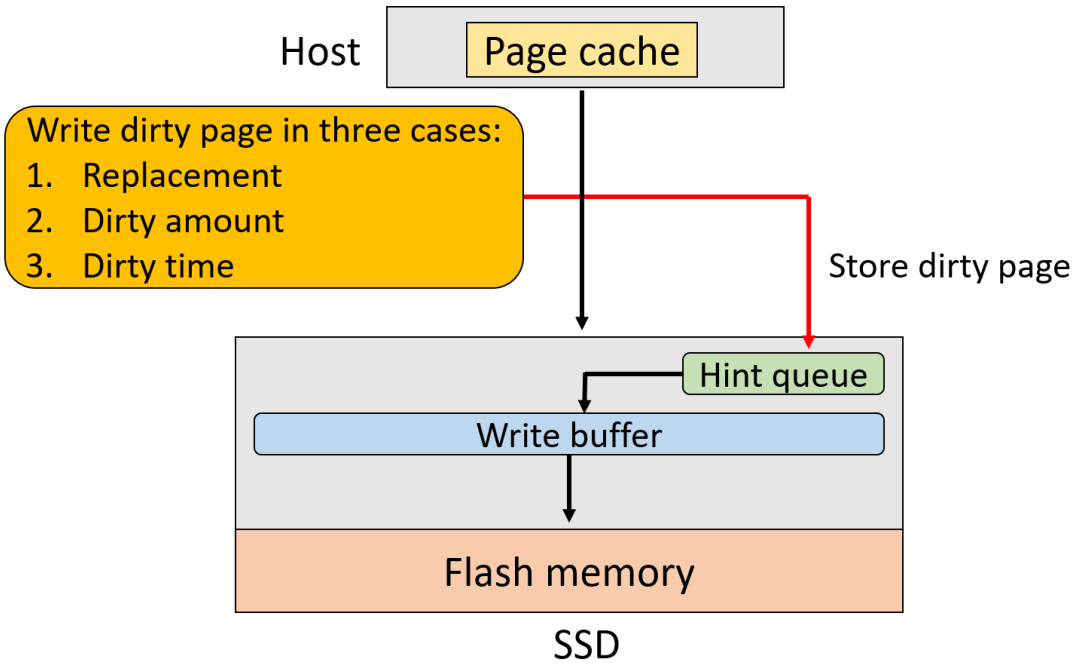

圖 2-5 Host-aware Write buffer management

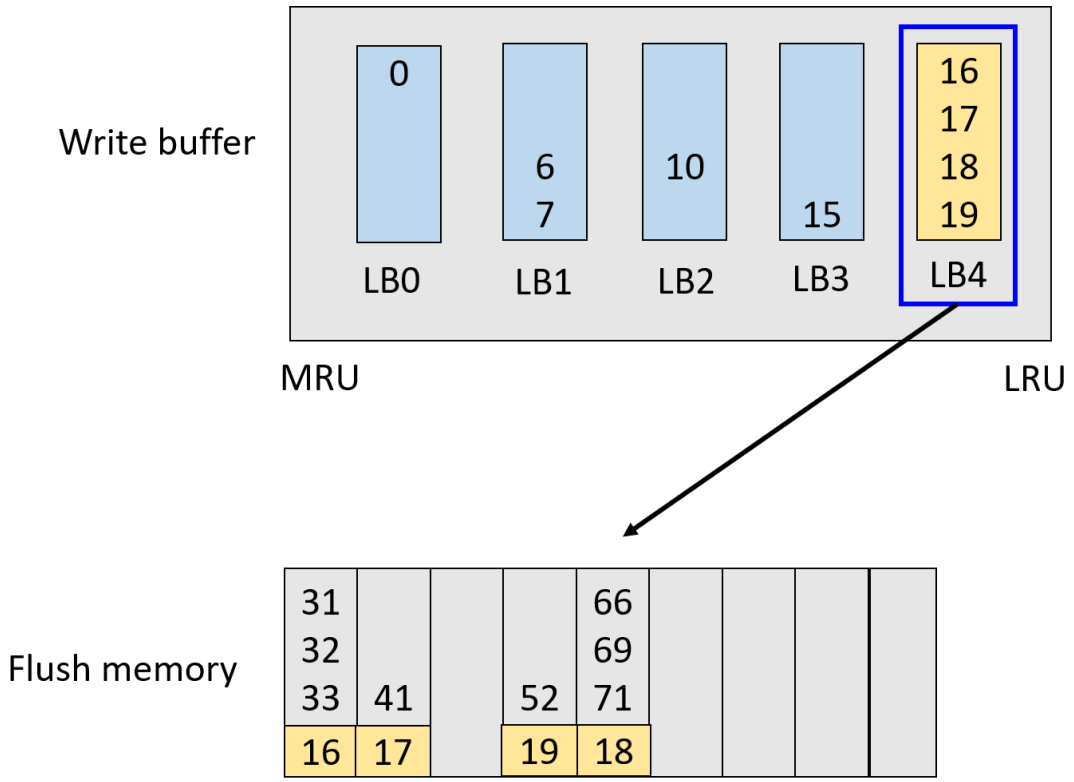

圖 2-6 Logical block-based Data management scheme

2.3 Neural network

Neural network，顧名思義就是神經網路，神經網路的目的其實只是在【尋找 function】，給它一個 input，它要給你一個 output，只是這個 output 必須符合當下任務的答案，也就是說，如果目的是預測股票售價，那這個 output 就必須是一個價格。但要找出這樣的 function 以人來說是很難做到的，因此才需要讓機器幫我們學出來。

以圖 2-7 為例，我們給的 input 是 x1, x2…xn，假設我們設定為 fully connected，也就是每個 input feature 會對應到每一個 Neural。一開始，每個 input feature 要送進任何一個 Neural 都會有一個對應的權重，以一個 Neural 來說，它會算出所有送進來的 input feature 乘上對應的權重總和再加上 bias，也就是($\sum_{1}^{n}$ xi*wi)+bias，算出的結果就是 yi，bias 的作用是為了讓相乘相加的結果為 0 時，至少有數字可以輸出。但是因為輸出的 yi 是一個線性函數，線性函數的輸出有所限制，也就是，輸出的值一定要是同一條線上的點，逼近目標函數的能力也因此有所限制，因此我們必須讓 Neural Network 裡面存在非線性函數，如此才能夠逼近目標函數。

而讓 model 從線性轉換成非線性的關鍵，就在於圖 2-7 的後半段【activation function】，經過 activation function 這個非線性函數，可以讓輸出不

受限制，加強 Neural Network 的表達力，也因此讓這個 Neural Network 可以透過學習逐步逼近目標函數。

通過 activation function 後，根據目的不同，輸出的形式也會有所不同，以圖 2-7 來說，目的是分類，因此輸出就是三維。在輸出後，會根據正確答案與預測的答案，透過 Loss function 來算出預測值與正確答案之間的差距，再透過 Backpropagation 將這資訊回傳回去，讓 Neural Network 調整權重，逼近正確答案。

而我們平常在講的 model，其實就是在講【訓練過後找出的目標函數】，訓練的目的單純是為了盡可能逼近目標函數。

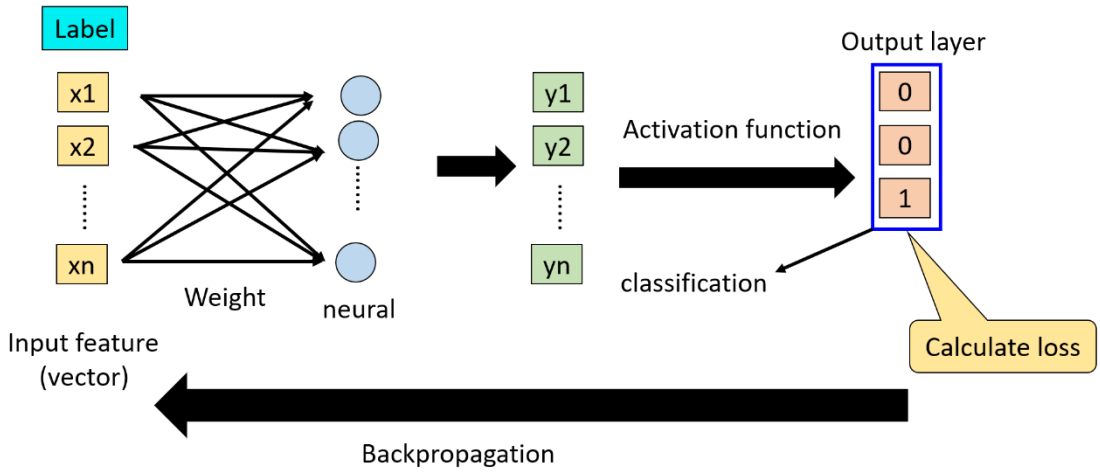

圖 2-7 Neural Network

2.4 RNN

RNN(recurrent neural network)主要用於負責需要時間的任務，它相較於一般神經網路的優勢在於能夠記憶，因為一般神經網路不論多麼複雜，都只能接收當下的 input，無法透過過去的資訊預測未來的結果。而 RNN 則是能夠處理【在一定的週期內行為規律能被預測的任務】，利用上次的資訊，去預測下次的資訊，利用下次的資訊，去預測下下次的結果。

如圖 2-8，RNN 簡單講其實就是能夠記憶的 Neural Network，而 RNN 又分為 simple RNN 以及 LSTM，這篇論文中，我們主要使用 LSTM(Long short-term memory)，因此接下來主要針對 LSTM 作介紹。

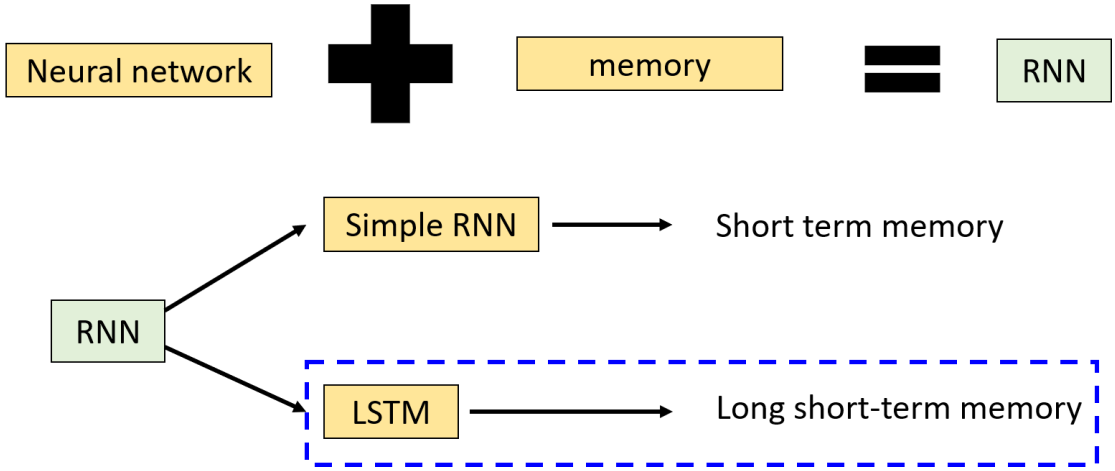

圖 2-8 Recurrent Neural Network

2.4.1 LSTM

說到 LSTM(Long short-term memory)，其實就是在講 RNN(recurrent neural network)，一般而言，RNN 分成兩種，一種叫做 simple RNN，另一種就是在講 LSTM，兩者差別在於，simple RNN 只能夠處理需要短期記憶的任務，而 LSTM 能夠處理長期記憶的任務，因此才會被稱為 Long short-term memory。

LSTM 本身分成 3 個 gate，分別是【forget gate】【input gate】【output gate】，這三組 gate 會各自決定【是否要讓當下資訊往前傳遞】。如下圖 2-9，g 是希望能夠被放到 memory cell(C)的資訊，而 Zi 則是進入 activation function 的 input，由於 LSTM 主要是以 gate 形式去運作，因此這裡的 activation function 是使用 sigmoid，主要是利用 sigmoid output 為 0~1 的這個特性，能夠以 0 來表示關閉 gate，1 表示將 gate 開啟。

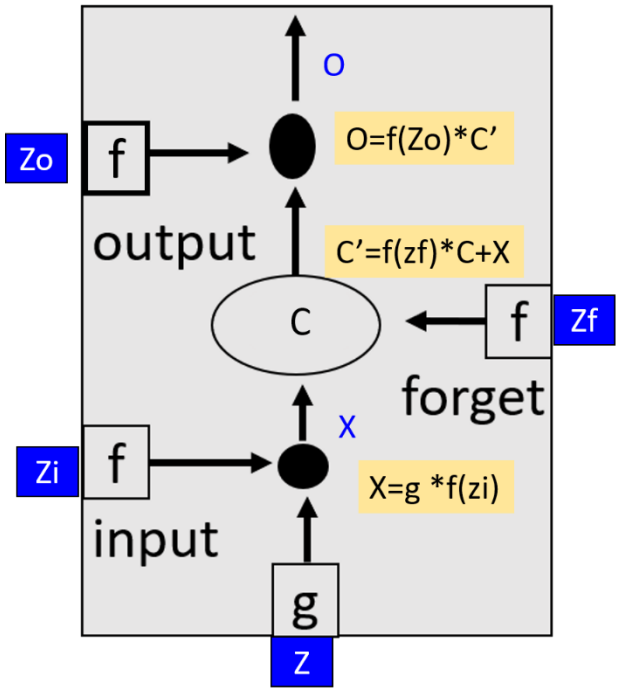

圖 2-9 LSTM

當 input 的資訊(g)進入 memory 以前，會先與 f(zi)相乘，由 f(zi)決定是否要將資訊往前送，若 f(zi)=1，這時候資訊，也就是 X，才會進入 memory cell(C)，在進入 memory cell 以前，會先依照第二個 gate，也就是 forget gate 來決定是否要記住【當下 memory cell 的資訊(C)】，換句話說，也就是上次存入 memory 的資訊(C)，而這個也是由 activation function 所決定，也就是 f(zf)。

所以當 X 實際被放入 memory cell 以前，會先加上 f(zf)*【上次 memory cell 的資訊(C)】，假如 model 認為上次的資訊無助於學習，甚至有害，那 f(zf)的 output 就會是 0，反之，則會 output 1，最後會將相加後的結果(C')覆蓋掉 memory cell 現有的資訊。

通過 memory cell 之後，仍然不一定能夠成功 output，因為接著會經過 output gate，要將目前的資訊(C')*f(zo)，最終由 f(zo)決定是否要 output，若 f(zo)判定為可以輸出，那才會真的將資訊(O)輸出。到目前為止，只是說明 1 個 LSTM unit 在 1 個時間點的 output，然而，LSTM 同一時間點通常是多個 unit，且通常會去看多個時間點，因此，接下來會介紹進入每個 gate 的 input 到底是如何產生的。

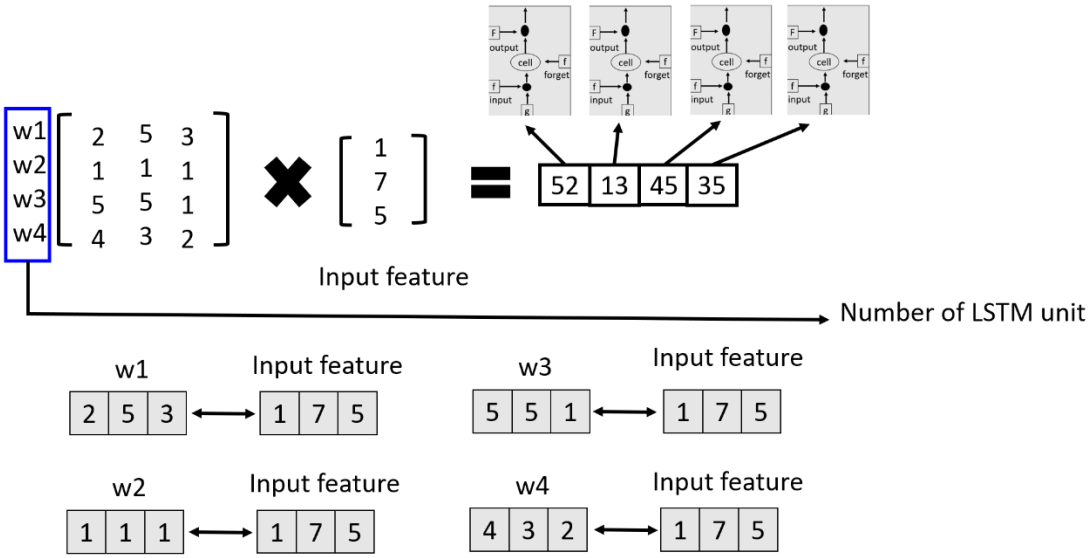

圖 2-10 LSTM input

當 input feature 進入 LSTM 後，會乘上 4 個不同的權重矩陣，目的是為了操控 input gate、output gate、forget gate、input，會多出一個 input 是因為神經網路 input 本身原本就是有對應的權重，因此 input 也會有對應的權重矩陣。

如圖 2-10，假設當前是要產生 input gate 的 input，首先會乘上一個權重矩陣，而這個權重矩陣會包含 n 組權重，以圖 2-10 來說，就是 4 組，這就是 input gate 權重的個數，而這個個數，也會決定同一時間，會有多少個 LSTM 單元。因為每一組權重會和 input feature 做矩陣乘法，而做完矩陣乘法會得到一個純量，圖 2-10 有四組權重，那就會是 4 個純量，這 4 個純量就是圖 2-10 上方的【52、13、45、35】，而這 4 個純量，會各自去操控一個 LSTM unit 的 input gate，前面圖 2-9，Zi 就是 4 個純量中的其中一個，然後 Zo, Zf, Z 也是以相同方式算出來的。

由於每個時間點輸入的資訊不同，所以每個時間點也都有各自 LSTM 的 output，有時候為了特殊目的，會抓取特定時間點的 output，但在本篇論文中，所抓取的是最後一個 output，比如說 model 看 16 個時間點，那就會是拿 time stamp 16 的 output 當作 output。最終，會將各個 LSTM unit output，合併為一個向量，以圖 2-11 來說，input feature X 乘上 4 個不同的權重矩陣後，會得到 4 個不同的 vector Zi, Zo, Zf, Z，分別用來控制 LSTM 的 4 個 gate，每個時間點這四個 vector 會同時控制多個 LSTM unit，每經過一個 time stamp，memory cell 的資訊就會被下一個時間點的資訊所取代，有可能會包含上一個時間點的 memory cell 資訊，也有可能不包含，如圖 2-12，經過了多個時間點，最終每個 LSTM unit 會 output 一個純量，以圖 2-10 為例，就是 4 個純量，然後將這四個純量合併為一個向量，如圖 2-11。

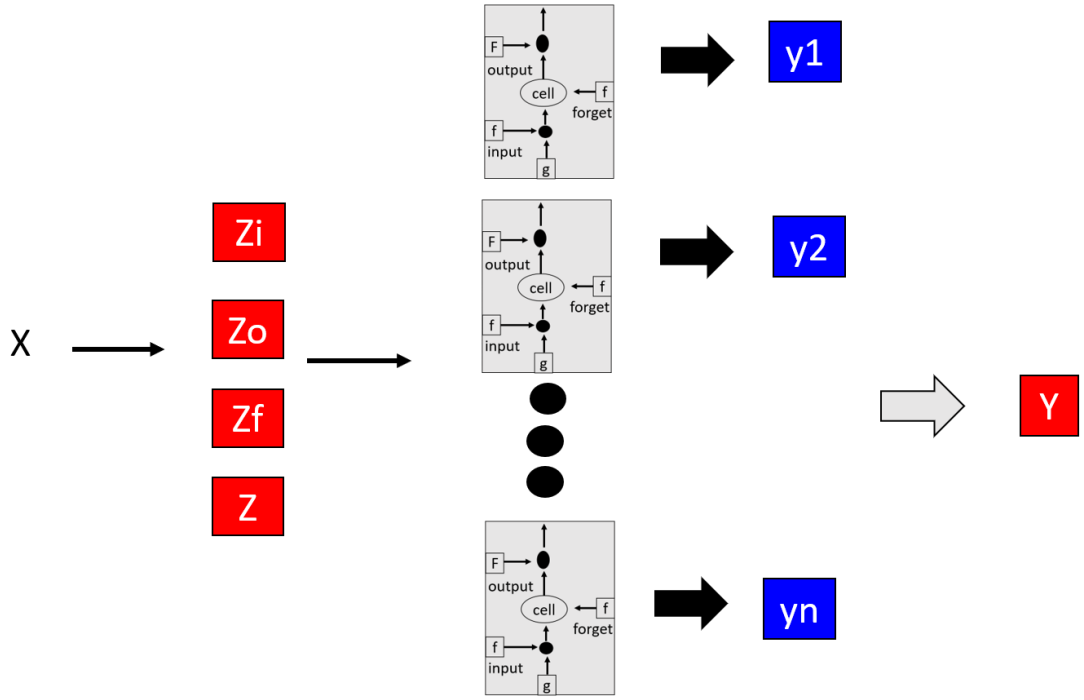

圖 2-11 LSTM Architecture (多個 LSTM 單元，在單一時間點內的狀況)

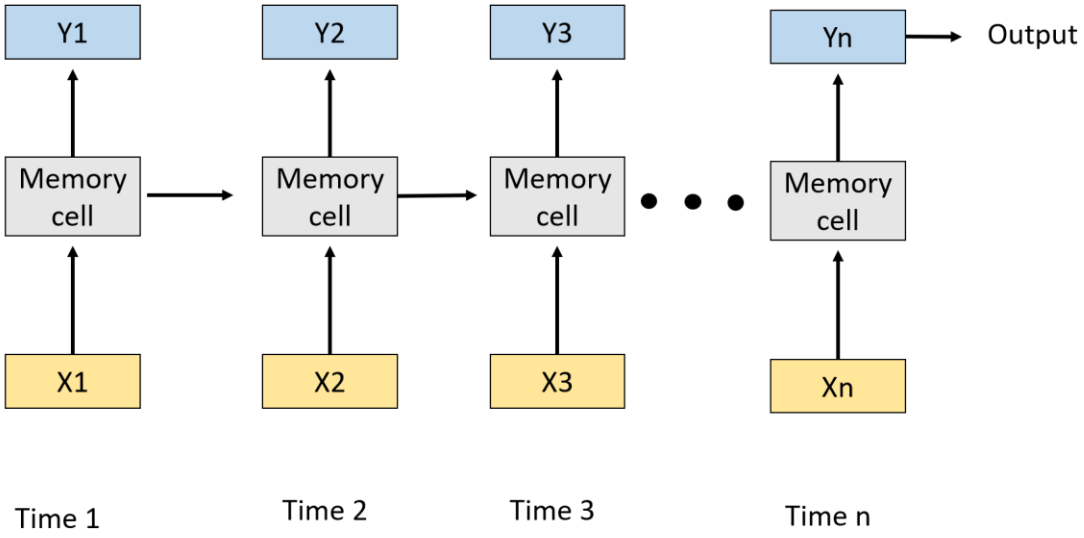

圖 2-12 LSTM unfold Architecture(一個 LSTM 單元，多個時間點的狀況)

第三章　系統架構與實作方法

3.1　系統架構

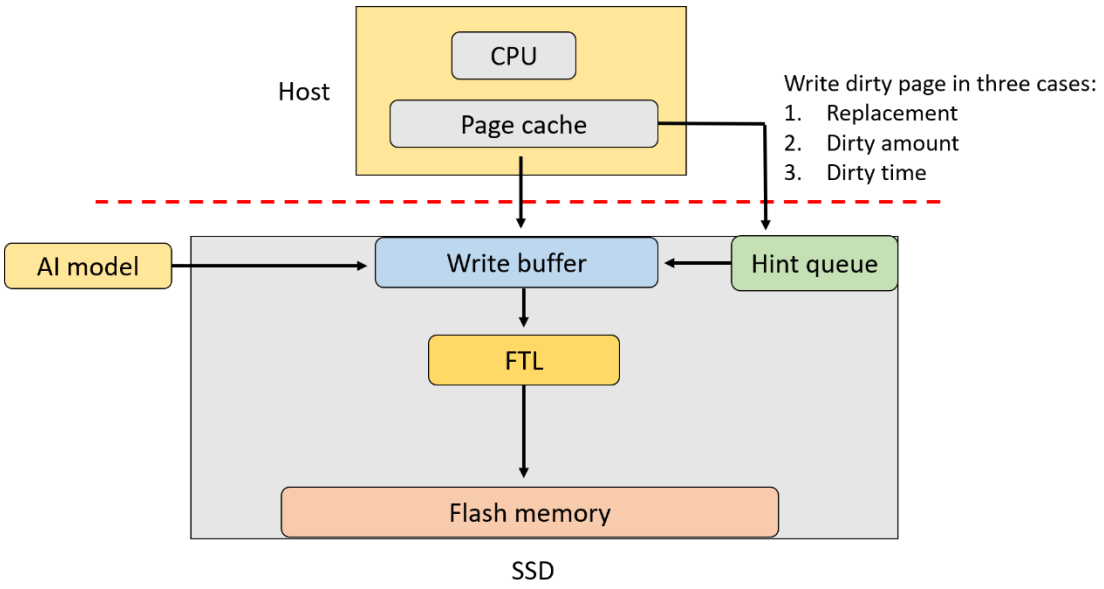

圖 3-1 系統架構圖

圖 3-1 是我們的系統架構，上層 Host 端分成 CPU 與 Page Cache，下層為 SSD，SSD 內部有 Hint queue 與 RAM-based Write Buffer，以及 FTL 和 Flash memory。

Page Cache 在三種情況會將 dirty page 寫入 write buffer，【Page Cache 已滿】【Dirty page ratio 在 Page Cache 待超過一定時間】【Dirty page ratio 超過某個 threshold】，藉由週期性預測 Page Cache 的 Dirty Page，將預測會被寫入的 Dirty Page number 寫進 Hint queue，然後送到 SSD 中，透過這樣的方式，讓 write buffer 能夠獲取 host 端的資訊，達到 host(hint queue)與 SSD(AI model)相互合作效果。

另一方面，使用 AI model 預測的結果和 Hint queue 的資訊，選擇合適的 victim block 踢除。

3.2 AI 運作流程

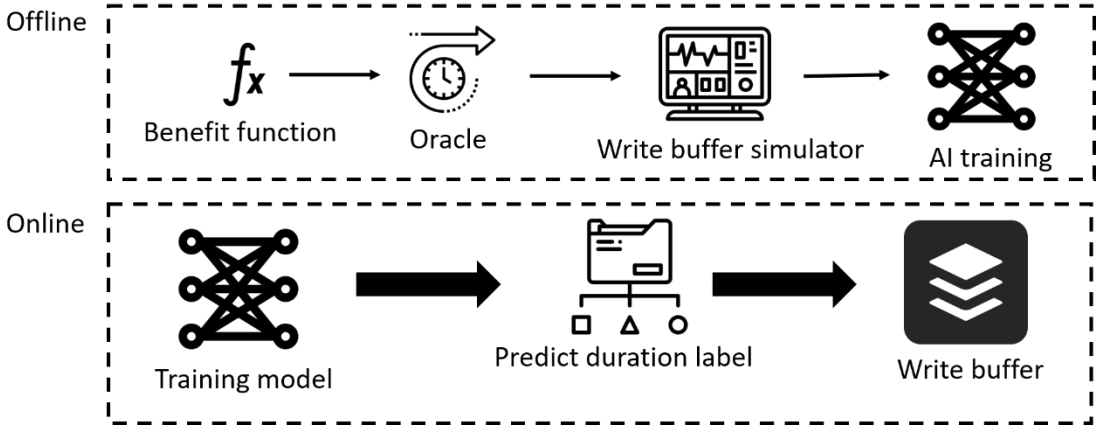

圖 3-2 AI 架構圖

AI 會分成 online testing、offline training 兩階段，如圖 3-2，在 offline 部分，會先將整個 trace 執行一次，獲得每個 block 的 benefit value，接著進入 write buffer simulator，產生出 duration value，先將 duration value 轉成 duration label 形式，再將 duration label 送進 AI model 當成比對用的 label 做訓練。

Online 部分，會利用訓練好的 model 預測當下進入 write buffer 的 requests，區分為 Cold, Mean, Hot 三種 duration label，之後 write buffer 就可以依照預測出來的 duration label 來選擇 victim block。由於考慮到也許會發生預測錯誤的問題，例如明明不常被存取，卻被放置於很常被存取的 queue，這時候就會藉由 Demoting，漸漸將那些不常被存取的 block 從 write buffer 踢掉。

3.2.1 Offline

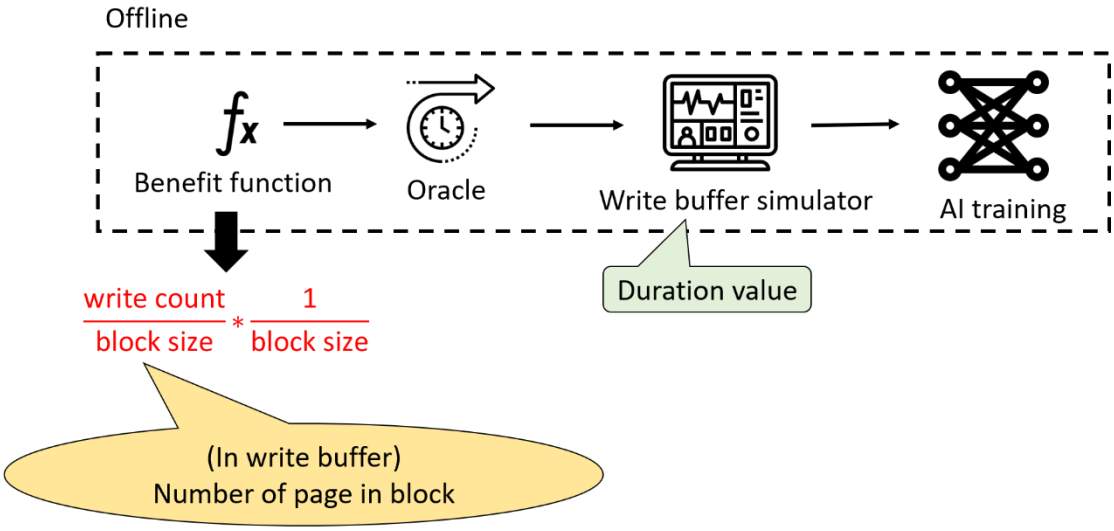

圖 3-3 Offline 架構

3.2.1.1 Benefit value

Write buffer 在踢資料的時候，基本概念就是少用的資料優先踢掉，以此為原則，我們計算每一個 block 的 benefit value，用來當作判斷是否要將資料留在 write buffer 的標準。

$$\text{Benefit value:}\frac{\text{block write count}}{\text{(write buffer) block size}} * \frac{1}{\text{(write buffer) block size}}$$

Benefit value 是以 block 為單位做計算，因為 write buffer 在踢資料時，也是以 block 為單位。以下介紹 benefit value 中每個名詞的意義：【block write count】指的是每個 block 被寫入的次數總和，因為 block 是由多個 page 組成，且寫入的最小單位也是 page，因此 block write count 其實就是該 block 每個 page write count 的總和；【block size】，計算目標包含 Write buffer 內的所有 block，而 block size 是指當下所計算的 block 中，目前有多少個 page。

會這樣設計 benefit value 的用意在於，一方面我們希望頻繁被寫入的資料能保留在 write buffer，與此同時，又希望 victim block 能夠是 large block，因此公式的前半部會去計算單位 page 被寫入的次數，後半部則是針對 block size，讓 large block 獲得較小的 benefit value，如此一來，在選擇 victim block 時，若存在多個 write count 相同的 block，則 large block 更容易被選為 victim block 從 write buffer 踢掉。

3.2.1.2 Collecting Data

我們除了要知道每個 block 的 benefit value 以外，我們還必須知道每個 block 實際上在 write buffer 待多久，為了能夠區分出每筆資料的類型(code data/hot data)。要做到這件事，我們會先建立一個 write buffer simulator，模擬 request 從 write buffer 進入以及離開的狀況，以此來掌握每個 block 的資訊。

首先，如圖 3-3 當中的 oracle algorithm，我們為了掌握每個 block 的大小以及被寫入的次數，我們會先跑過整個 trace 一次，將相關資訊都存起來，以建構出每個 block 的 benefit value，接著，才會進入到 write buffer simulator 的部分，在獲得每個 block 的 benefit value(也就是未來的資訊)之後，我們會在 write buffer simulator 中跑一次，觀測每個 block 實際待在 write buffer 多久，這個資訊，我們以 duration 來稱呼它，duration 的單位是 request，也就是【該 block 進入 write buffer 到它被選為 victim block 踢掉】這段期間，總共有多少個 request 進入 write buffer，因為 request 單位是【Page】，因此，只要有 request 進入 write buffer，現存在於 write buffer 的每一個 block 的 duration value 都會被累加。而在 write buffer simulator 中，我們踢掉資料的依據就是【benefit value】，前面已經提過 benefit value 的公式，而 benefit value 的意義，顧名思義，其實就是每個

block 的價值，因此在選擇時，會挑小價值最小的 block 踢掉，換句話說，也就是挑選 min benefit block 作為 victim block，如圖 3-4。

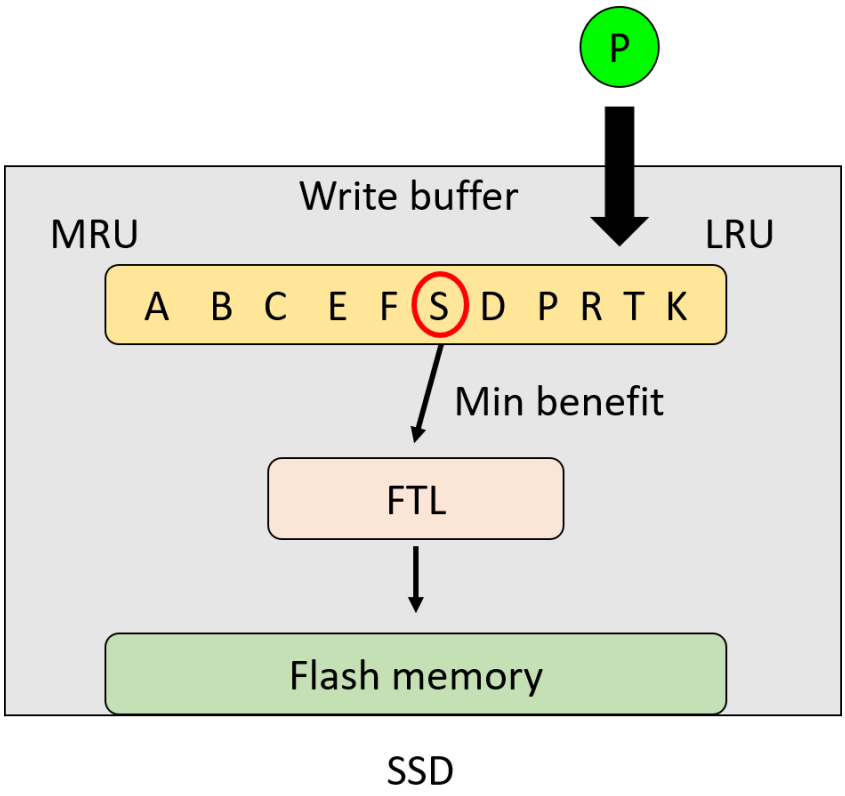

圖 3-4 Write buffer simulator

以這樣的方式，在 write buffer simulator 跑完之後，我們可得知每個 block 的【Duration value】。

3.2.1.3 Transform duration value into duration label

因為我們打算讓 model 預測 label，因此，我們需要先將 duration value 轉換成 duration label。轉換方式如圖 3-5，我們先設定一個 threshold，若 duration value<threshold，則判定為 cold(label=0)，若 duration value 介於【threshold*1~threshold*5】之間，則判定為 mean(label=1)，若【duration value>=threshold*5】，代表這個 block 很可能是 hot block，因此判定為 hot(label=2)。

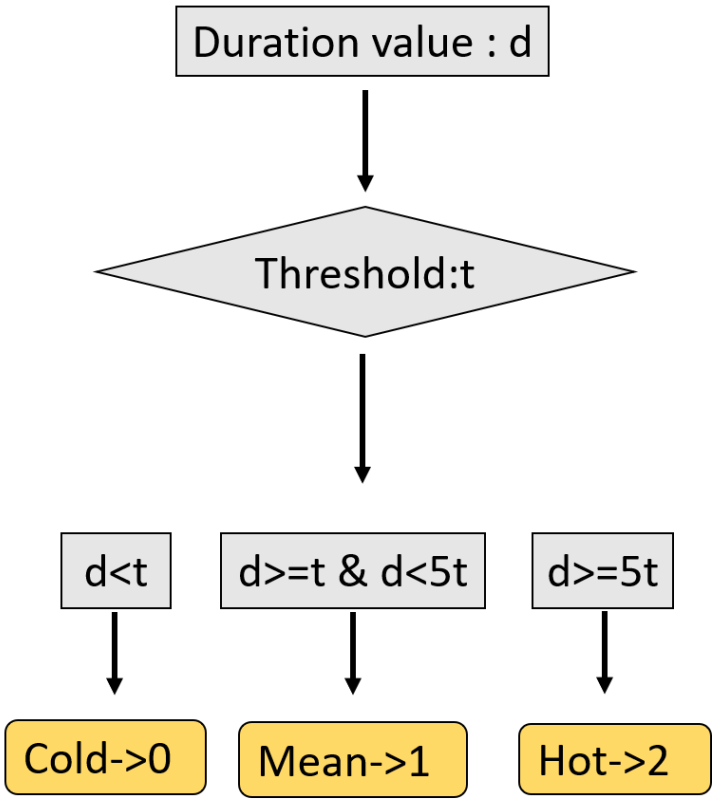

圖 3-5 Generate Duration Label

將每個 block 的 duration value 轉換成 label 之後，就可以開始進行訓練了。

3.2.1.4 Offline training

在 offline training，input feature 送進三個 LSTM 層，input feature 就是 model 會看到的資訊，會被用來做訓練，通過 LSTM 後，將輸出結果通過 softmax activation function 的 output 就是預測的結果，之後 loss function 會計算出 loss，也就是預測值與正確答案的誤差，之後 model 會根據 loss 來調整權重，以獲得更高的 accuracy。

在這篇論文，我們使用的 input feature 分別是 reuqest 的【arrive time】【read count】【write count】【block size】【current block write count】【current page write count】，以下分別介紹上述每個 feature 的用意:

- Arrive time: 為了讓 model 知道不同時間點的 request 狀況
- Read count: 因為頻繁寫入的資料通常讀取次數會比較少，因此希望藉由 read count 讓 model 區分這個 block 是 write intensive 或是 read intensive block
- Write count: 為了避免 model 單靠 read count 無法掌握 block 的 read、write 資訊，這裡多給一個相對的特徵讓 model 對照

- Block size: 由於我們的目的是希望 model 能夠踢掉 large block，因此給予 block size 讓 model 知道當下 block size 是多少
- Request 存取的 block 的 write count/read count:給予 Request 存取的 block 的 read、write count 的用意在於希望能夠讓 model 掌握 frequency 的部分，所以告訴 model 當下的 page/block 被寫入、讀取幾次。

除了特徵之外，LSTM unit 也能有效幫助我們做預測，在這篇論文當中我們使用三個 LSTM，每一層各自有 128 個 unit，越多 unit 就能夠越有效地逼近正確答案，因為如果 unit 很少，那當 model 預測不好的時候表現就會很差，但如果 unit 夠多就可以避免這個問題。因為在 LSTM 中，可以視每個 unit 為各自獨立的神經網路，當 unit 只有一個的時候，錯誤的預測就會直接影響到結果，但如果 unit 有多個，即使某些 unit 預測錯誤，但其他的 unit 仍然能夠 output 正確的結果。

實際面對複雜的任務時，多個 unit 會有更好的表現，其中原因在於，複雜的任務會讓 model 更容易給出錯誤的結果，如果存在多個 LSTM unit，則能夠有效的避免這個問題，因為在 LSTM 當中每個 unit 各自的權重都不同，即使某些 unit 的 output 是錯誤的，也不會影響到其他 unit 的 output，因此建構更多的 unit 能夠讓 LSTM 有更高的機率學習到我們所給它的資訊。

3.2.2 Online

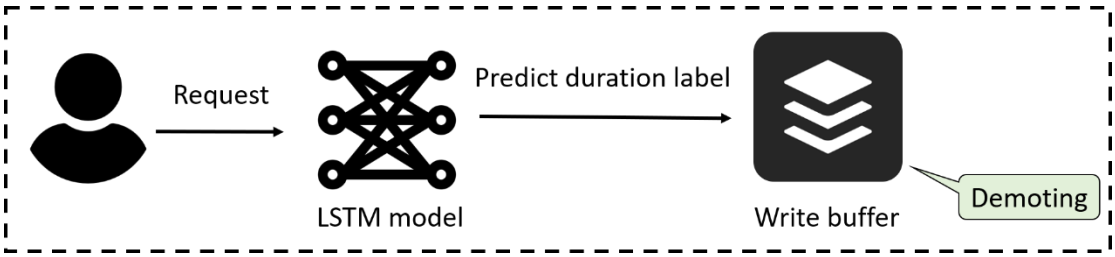

圖 3-6 Online 架構

在 online 部分，訓練好的 model 會預測每個進入 write buffer 的 block 的 duration label，此外，write buffer 會監測每個 block 來決定是否做 demoting。

3.2.2.1 Demoting

首先介紹 online 部分 write buffer 的擺放方式，有別於以往的 write buffer 只有一個 LRU queue，在本篇論文中，write buffer 共有三個 LRU queues，如圖 3-7，分別是 Cold、Mean、Hot，分成三個 queues 的用意在於，以此來區分 cold data、hot data，Hot queue 放很頻繁被存取的資料，接著是 Mean queue，最後 Cold queue 放的資料則是很少被用的資料。

因此，在踢資料時，會優先從 Cold queue 的 LRU 端開始踢，當踢光 Cold queue 的資料或是 Cold queue 是空的狀況發生，就往上到 Mean queue LRU 端找，如果 Mean queue 也是空的，最後才會去 Hot queue LRU 端找 victim block 踢掉。每次資料被存取，就會被移動到該 queue 的 MRU 端，而不常被存取的資料會位於各個 queue 的 LRU 端，但因為存在 model 誤判的狀況，所以 Mean queue 和 Hot queue LRU 端的資料，是有可能長期沒有被存取，而占用 write buffer 的空間，由於踢資料會以 Cold queue 為優先，因此的確可能發生某些資料占用 Mean queue 和 Hot queue 的空間卻完全沒被使用到的狀況。為了避免這種狀況發生，我們在 online 使用了【Demoting】來解決這樣的情形。

在 online，我們會監測每個 block 待在 write buffer 的時間，目前先以【pass_req_count】來稱呼它，單位與 duration 一樣，是以 request 為單位，差別在於，一旦該 block 被存取，pass_req_count 就會被歸零，以這樣的方式，就能夠知道在 Mean queue、Hot queue 中，有哪些 block 很久沒被存取，卻仍然在 write buffer 占空間。除了 pass_req_count 以外，我們還需要有一個 threshold 當作標準，當 pass_req_count>=threshold 時才會針對該 block 做 demoting，每次有 request 進入 write buffer 就會檢查一次 write buffer，看是否有需要做 demoting。而由於每種 trace 的行為不太相同，因此我們針對每個 trace 都設定一個 demoting 的 threshold。

如圖 3-7，當存在某些 block 的 pass_req_count>=threshold 時，這時候，只要那些 block 所在的位置不是 Cold queue，就會藉由做 Demoting 將 Hot queue 的資料移到 Mean queue、將 Mean queue 的資料移到 Cold queue，由於 pass_req_count 只會在被存取的時候歸零，因此若資料原本位於 Hot queue，在完成一次 Demoting 之後(Hot->Mean)仍然沒被存取，那很快就會進行第二次 Demoting(Mean->Cold)。如果資料原本就在 Cold queue，則不用理會，因為位於 Cold queue LRU 端的資料，原本就會優先被選擇為 victim。

這樣的做法，能夠確保 write buffer 內所有 block 的 pass_req_count 不會超過 threshold，因為一旦超過，就會被 Demoting， 如果 Demoting 後仍然沒有被存取的話，很快就會從 write buffer 被踢掉，如此一來就能確保 write buffer 內的資料都是真正必要的資料。

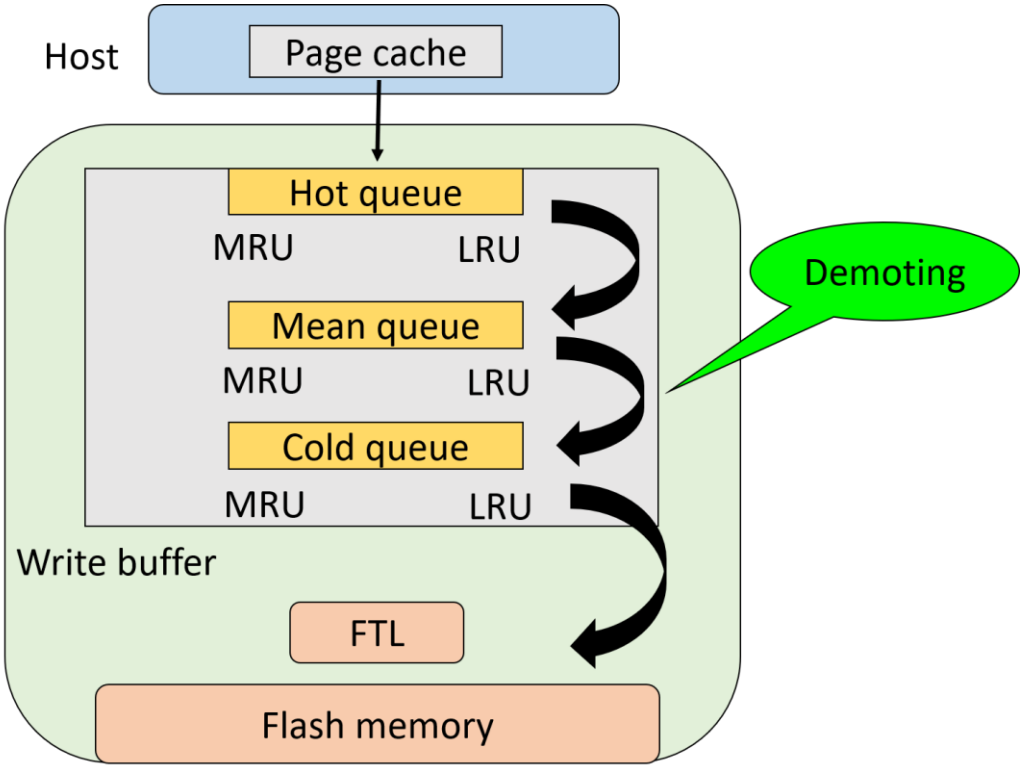

圖 3-7 Our Write Buffer Data Placement

3.3 結合 AI 與 Hint 資訊

儘管我們在 AI 做了 Online demoting 與 Offline training，但仍然可能因為 model 無法徹底掌握使用者 request 的規則，而造成表現不佳。為了改善這樣的問題，我們除了使用 AI 以外，更將 AI 與 hint 資訊 [1]做結合。Hint 資訊 [1]存放的是通過週期性預測，預測即將被踢到 SSD 的 dirty page number。透過 Hint 資訊 [1]，我們一方面使用 model 預測，另一方面使用 Hint 資訊 [1]，有時候可能 model 會預測錯誤，有時候可能 Hint 資訊 [1]會預測錯誤，但透過 model 與 Hint 資訊 [1]交叉比對，我們就能夠選出最佳的 victim block。

但如何知道應該依照 Hint [1]或者是 model 的資訊?下一章節，我們將會詳細介紹。

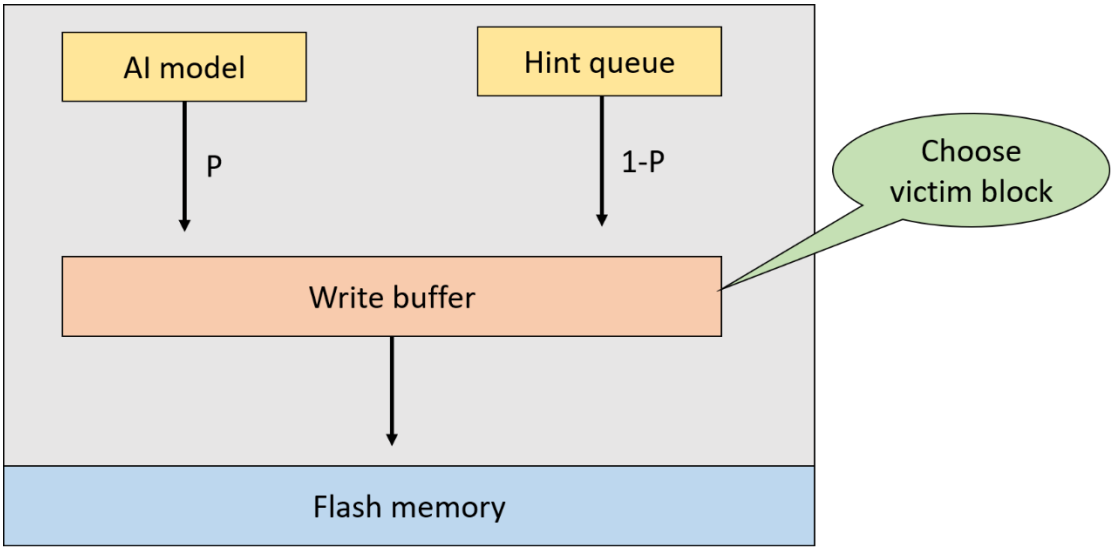

圖 3-8 AI+Hint 架構圖

3.3.1 選擇 victim block

如圖 3-8，由於我們不知道該依照 AI 的資訊或是 Hint queue 的資訊，因為兩邊都是預測的資訊，只是預測的內容不太一樣，因此，我們透過實驗來決定 AI 與 Hint quque 各自的權重。

$$\text{profit}(x) = \text{priority} * p + \text{Hint} * (1 - p)$$

首先介紹上面的公式: 【profit】，這個公式也是以 block 為單位，因為目的是為衡量 block 的重要性；【p】是一個 0~1 之間的數字，在這裡當成權重使用，目的是為了得知到底 model 預測比較能夠影響效能，還是 hint queue 的資訊；【priority】是為了將 LRU queue 數字化，簡單來說，將 Cold queue(假設 Cold queue 是有資料可以踢)LRU 端的資料設為 0.01，然後接著往 MRU 的方向，每次遇到一個新的 block 就遞增 0.01，如下圖 3-9；最後是【Hint】，也就是預測會被寫入 write buffer 的 page 中，寫進 block x 的 page 總和，這裡的寫進 block x 不單單指 overwrite 的 page 個數，也包含 new write，也就是該 page 當下不在 block x 中，但之後會寫入。所以如圖 3-9，E、B、C、A、D 之後各自會被寫入(包含 overwrite)5、3、11、24、21 次。

介紹完公式，我們仍然不知道該將哪邊給予更高的權重，因此，我們執行多次測試，為每個 trace 找出 hit ratio 最高時的 p 值，並以上述公式來決定 victim block：當需要挑選 victim block 時，挑選 profit 值最小的 block 當成 victim block。

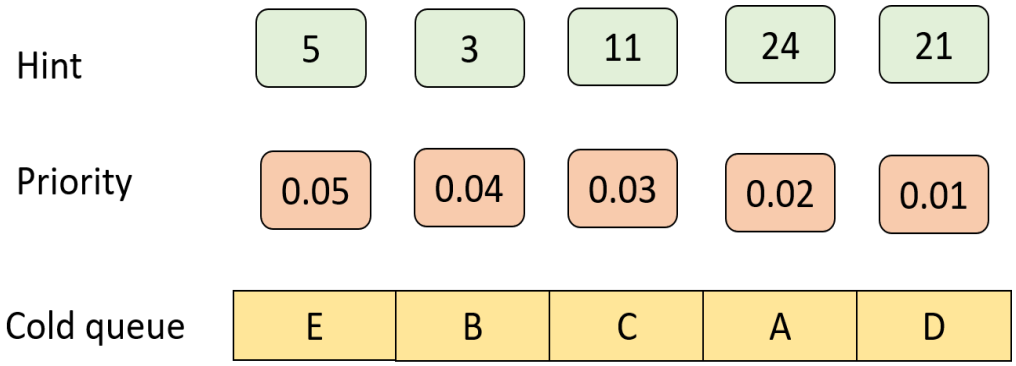

圖 3-9 AI+Hint 資訊

3.3.2 過早踢除

雖然我們看似已經能夠選出合適的 victim block，但仍然存在一個問題:【過早踢除】，因為被我們選中的 victim block 有可能位於 MRU 端，因此未來會再度被存取。

然而，雖然看起來是這樣，其實這個問題並不存在，因為 Hint queue 預測的是未來的資訊，而 AI model 預測的是當下的資訊。因此 write buffer 內目前擺放的狀況和未來的實際狀況並不相符。如圖 3-10，乍看之下我們選中了 block B 而 block B 位於 MRU 端，理論上，有很高機率會再度被存取，但是為什麼看似過早踢除仍然能夠有更高的 hit ratio?

關鍵在於 Hint queue 所存放的(預測的)是未來的資訊，代表即將寫入但目前還沒有寫入的資料，而透過 profit 這個公式得知，我們會選擇 Hint 值小的同時靠近 LRU 端的資料，因此若選擇 block B 代表 block B 未來被寫入的次數應該很少。未來不會寫入 block B，但仍然有 request 會進入 write buffer，那被寫入的就會是除了 block B 以外的 block(C、A、D、E)。所以在未來的時間點，block B 會被其他 block 擠到 LRU 端，因此 victim block 選擇 block B 是正確的。

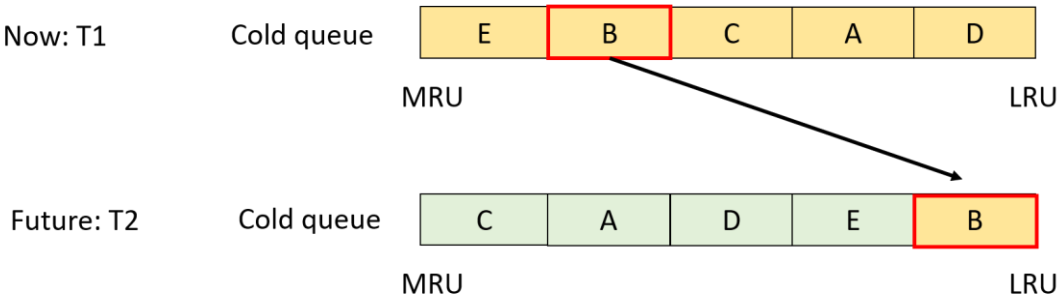

圖 3-10 誤判的狀況

第四章　實驗結果

4.1　實驗環境

本篇論文使用 disksim 4.0 來模擬 SSD 的環境，主程式執行在虛擬機上，作業系統為 Ubuntu 16.04 32bit，AI model 是在 keras library 中進行訓練，本篇論文是基於 Host-aware Write buffer Management [1]進行改善，比較對象分別為 Host-aware Write buffer Management [1]、FAB [3]、EXLRU [4]。

表 4-1 為使用的 traces，這些 traces 是從 Storage Networking Industry Association(SNIA)上取得的，其中 IOzone 是透過 IOzone benchmark 執行產生的 trace。表 4-2 介紹 Offline 訓練時使用的參數，表 4-3 介紹 Online 所使用的參數，其中 demoting threshold 代表 demoting 時的 threshold，而 P 則是 AI 與 Hint 做結合的時候，AI model 所佔的比例。

表 4-1　Trace 特性說明

Trace	**Number of request**	**Read ratio**	**Write ratio**	**Sequential read ratio**	**Sequential write ratio**
UG-fileserver	560144	0.25589	0.74411	0.456874	0.689882
Postmark	367941	0.467635	0.532365	0.708808	0.677919
IOzone	1405227	0.295923	0.704077	0.00075	0.009433

表 4-2 Offline parameter

Time stamp	16
Input shape	16*6
Activate function of output layer	softmax
Activate function of LSTM	sigmoid
Loss function	Categorical crossentropy
Optimizer	Rmsprop
Training data : Testing data	8 : 2

表 4-3 Online parameter

Trace	Demoting threshold	P (in AI+Hint formula)
Postmark	0.9	7000
UG-fileserver	0.9	8000
IOzone	0.3	5000

4.2 實驗結果

4.2.1 Hit ratio

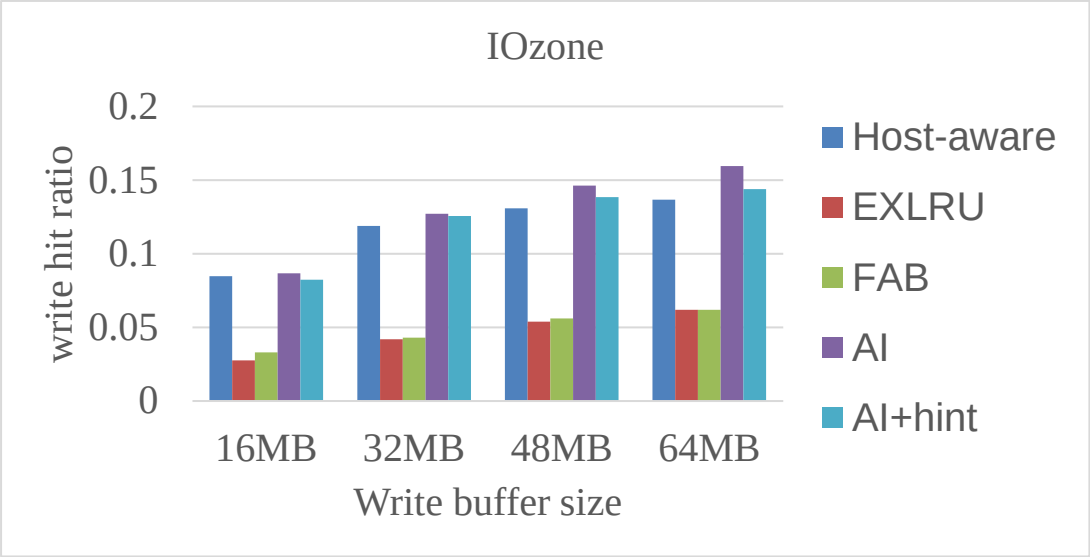

圖 4-1 IOzone Write hit raito

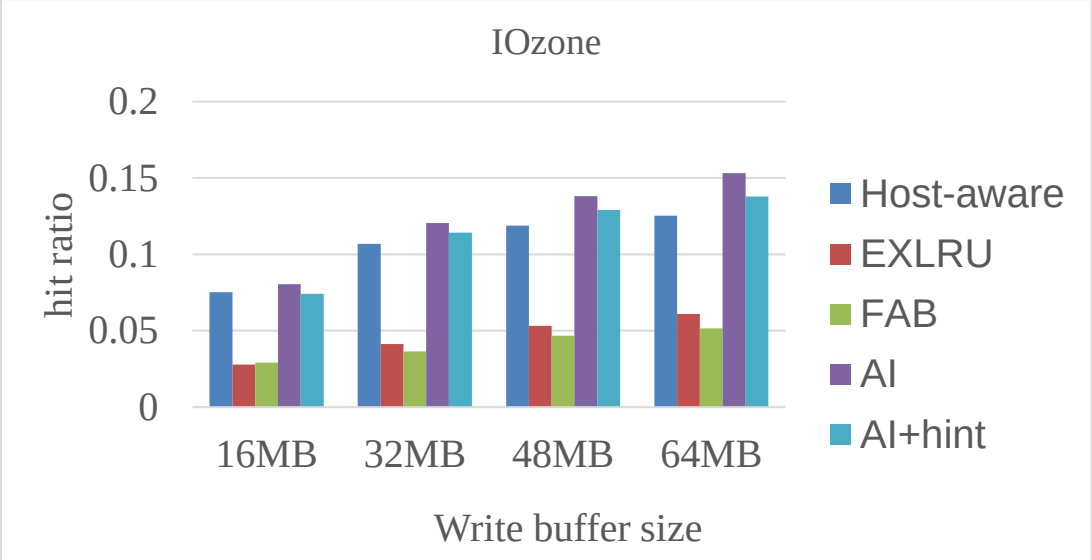

圖 4-2 IOzone Total hit ratio

雖然理論上 AI+Hint 應該要勝過 AI，但圖 4-1、4-2 卻出現意外狀況，也就是 AI 比 AI+Hint 來的更準。會有這樣的結果，原因在於實際測試時，因為想要結合 Hint 資訊 [1]，因此在設權重(p)時只會將權重在 0.1~0.9 之間做挑整，並

不會將 AI model 權重設為 1。這樣的作法會忽略掉一種可能性，也就是 AI model 非常好的狀況，而在這種狀況下，最佳的權重就是將 AI model 的權重設為 1，Hint 資訊 [1]設為 0，但因為我們所採取的權重範圍不包含 0 or 1，因此如果 AI model 表現很好就有可能發生類似圖 4-1、4-2 時的狀況。

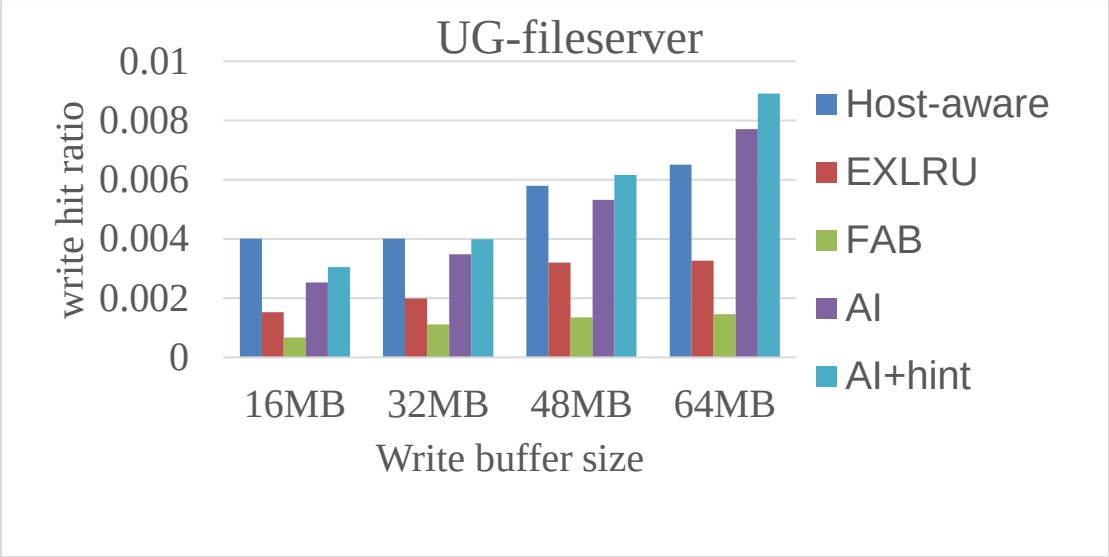

圖 4-3 UG-filserver write hit ratio

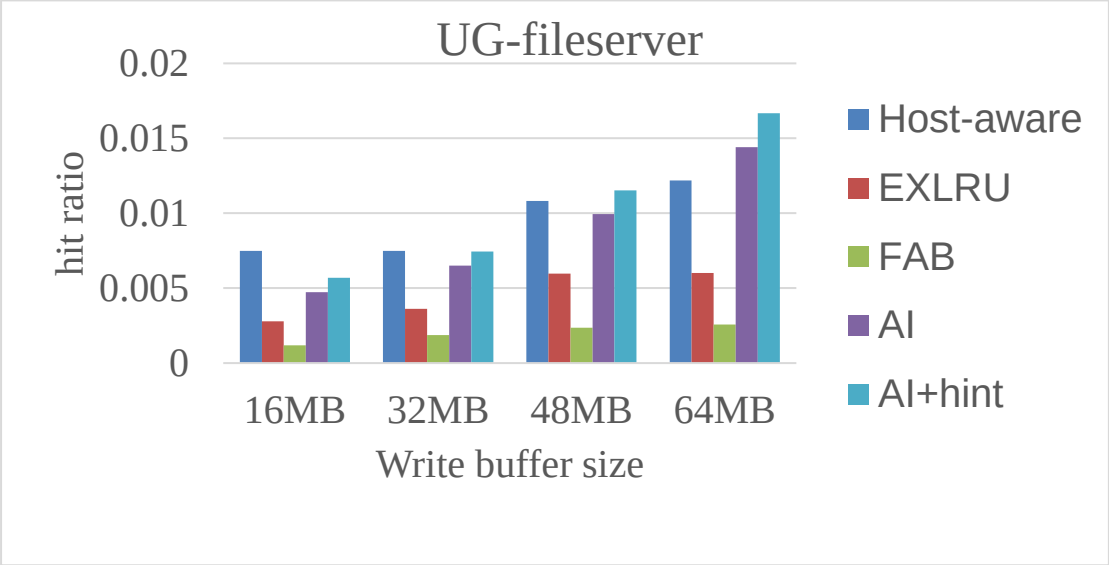

圖 4-4 UG-fileserver total hit ratio

圖 4-3、4-4 可以很明顯看出 AI 的某種特性，也就是 Write buffer size 設的越大，精準度提升的越快。一般而言，不論使用任何演算法選擇 Victim block，只要提升 Write buffer size，Hit ratio 一定也會跟著提升，但在我們所使用的 AI model 中，Write buffer size 變大時不但 Hit ratio 會提升，而且提升的速度非常的快，甚至能夠大幅度拉開與其他論文之間的差距。

會有這樣的結果在於，AI model 的預測方向主要是針對資料的特性進行預測(Hot data/Cold data)，而當 Write buffer size 不夠大時，Cold queue LRU 端的資料很可能是 Cold data、Hot data 混雜，實際被踢下去資料的有較高機率會是 Hot

data。而 Write buffer size 越大，這種誤踢的可能性就越低，因為 Hot data、Code data 會漸漸往 MRU、LRU 集中。

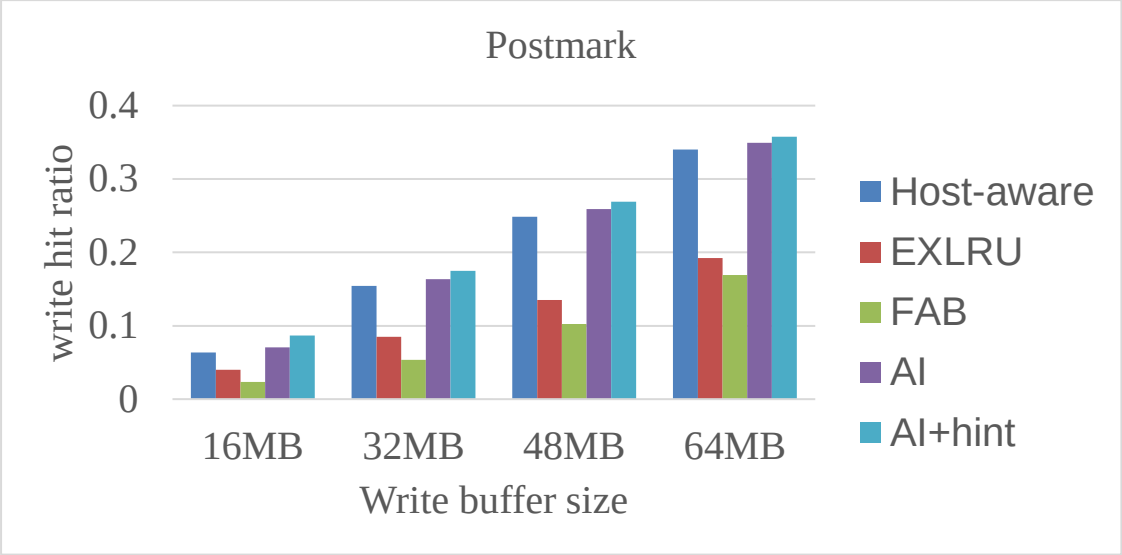

圖 4-5 Postmark Write hit ratio

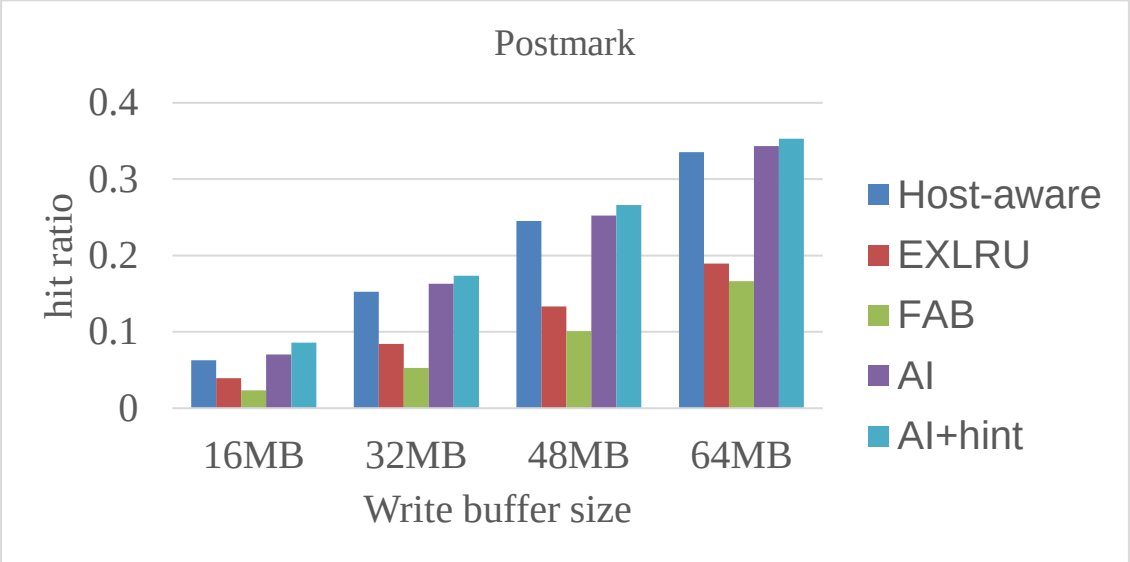

圖 4-6 Postmark Total hit ratio

圖 4-5、4-6 算是比較理想的結果，就是 AI model 相較於其他論文都有顯著的提升，與 Host-aware [1]相比也能夠做到些微的改善，而參考了 Hint 資訊 [1]之後，因為事先得知進入 Write buffer 的 Dirty page，因此在 Hit ratio 表現上能夠更進一步提升。

4.2.2 Response time

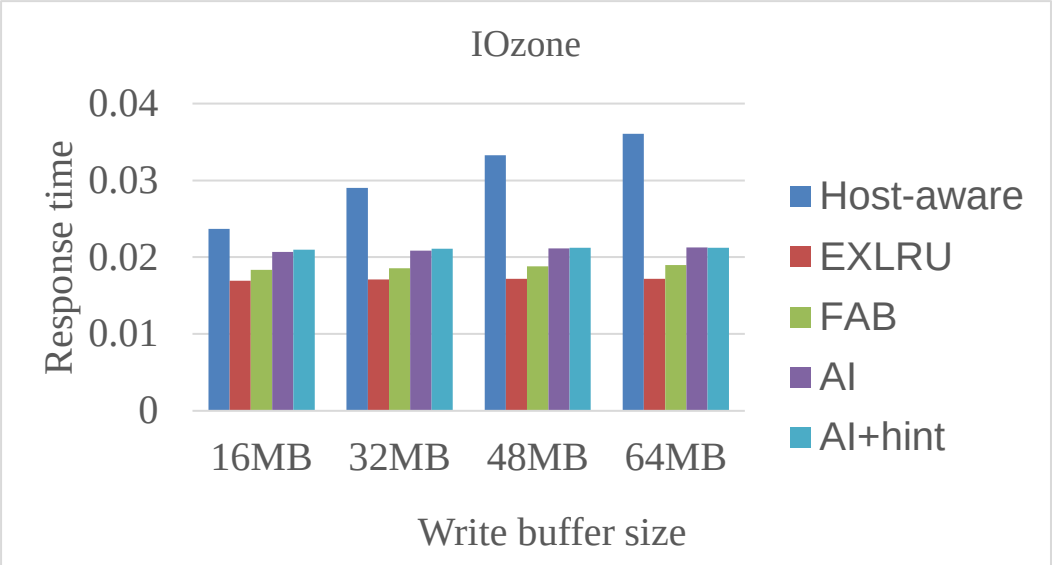

圖 4-7 IOzone Response time

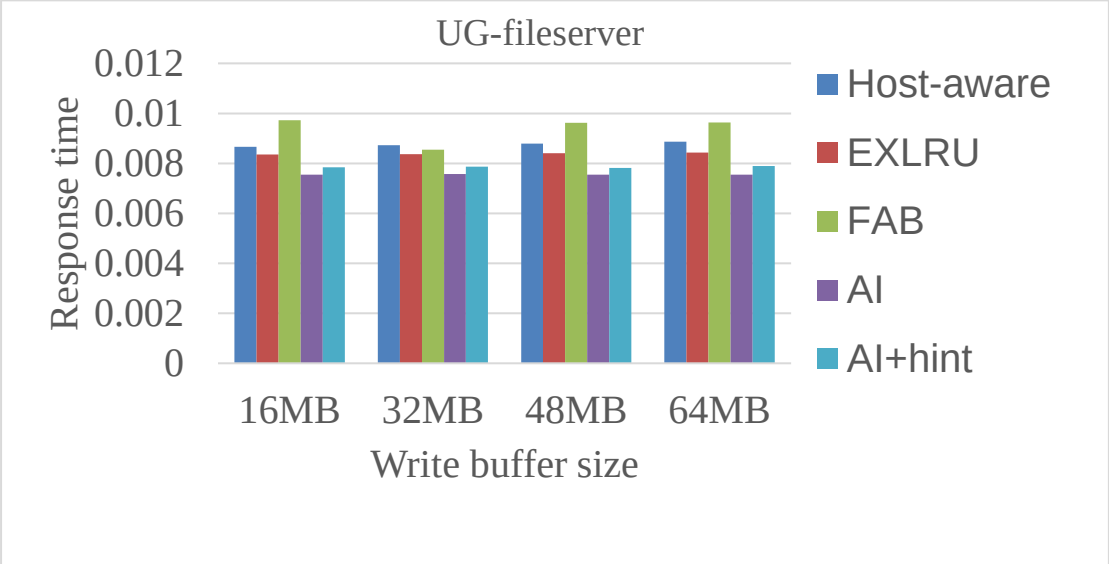

圖 4-8 UG-fileserver Response time

如圖 4-7，整體而言，AI、AI+Hint 都能夠比 Host-aware [1]表現來的更好，那是因為 Hit ratio 提升帶來的效益，但卻會輸給 EXLRU [4]、FAB [3]，這是因為在 EXLRU [4]、FAB [3]中挑選 Victim block 都有針對 GC 方面作改良，而 AI、AI+Hint 容易挑出 GC cost 較高的 Victim block，因此會導致 Response time 較高。

此外，圖 4-8 當中 AI+Hint 表現卻比 AI 來的差，那是因為 GC cost 比較高的緣故。AI 部分一半考量 GC cost 另一半考量 Write count，而 Hint queue [1]卻是只考量 Write count，因此當我們將 AI 與 Hint 結合之後，基本上 Write count 的比重會大於 GC cost，這會讓實際選擇 Victim block 時，在某些 Workload 容易出現將 small block 踢到 Flash memory 的狀況。

而 AI 相較於 AI+Hint，Write count 所佔的比重較低，因此比較不會出現將 small block 踢下去這個狀況。

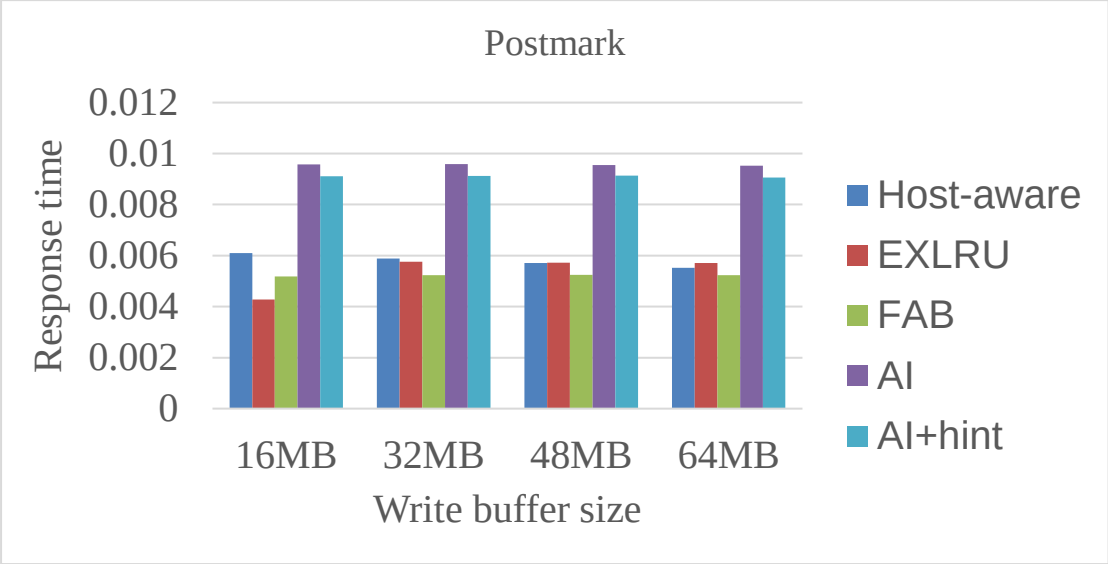

圖 4-9 Postmark Response time

如圖 4-9，在這個 Workload 中，AI、AI+Hint 的 Response time 都輸給其他的論文，那是因為這個 Workload 的 Hit ratio 成長幅度最小，前面兩個 Workload 之所以能夠表現比較好，是因為提升 Hit ratio 所帶來的效益大於 GC cost，因此整體 Response time 相較其他的論文來說會比較長。

此外觀察上述三種 Workload 發現，雖然在 Hit ratio 表現都還不錯，但是對於 GC 的部分似乎有待加強，AI、AI+Hint 在挑選 Victim block 時，容易挑出 Write count 低，但是 GC cost 高的 block。如果單論 GC cost ，AI、AI+Hint 會比其他的論文更高。

因此 AI、AI+Hint 整體 Response time 能夠改善多少，其實是取決於 AI、AI+Hint 在 Hit ratio 所提升的幅度。

4.2.3 Kick page count

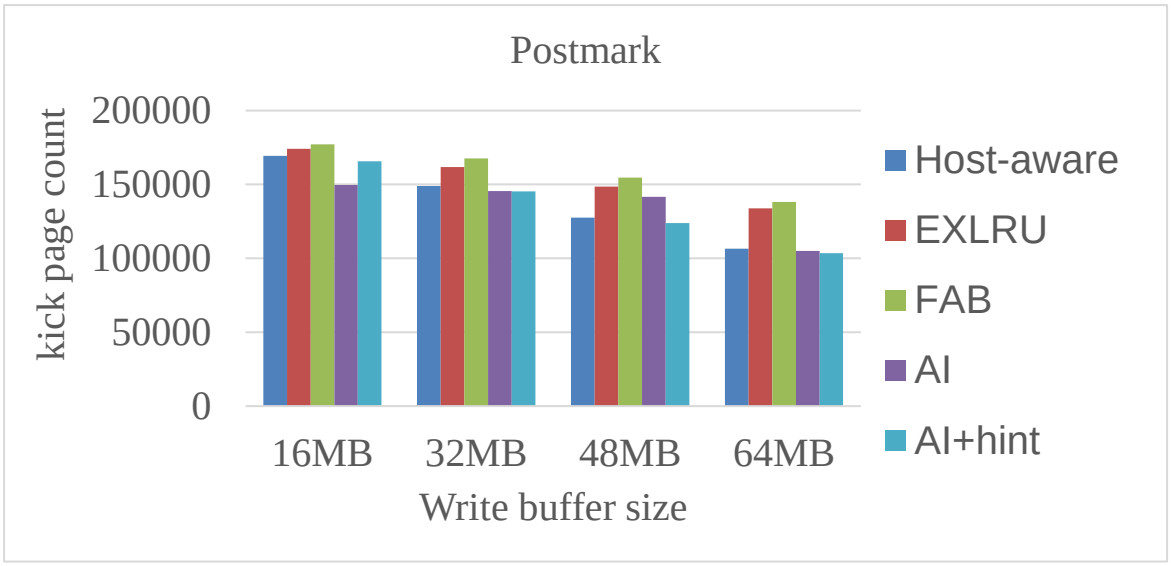

圖 4-10 Postmark Kick page count

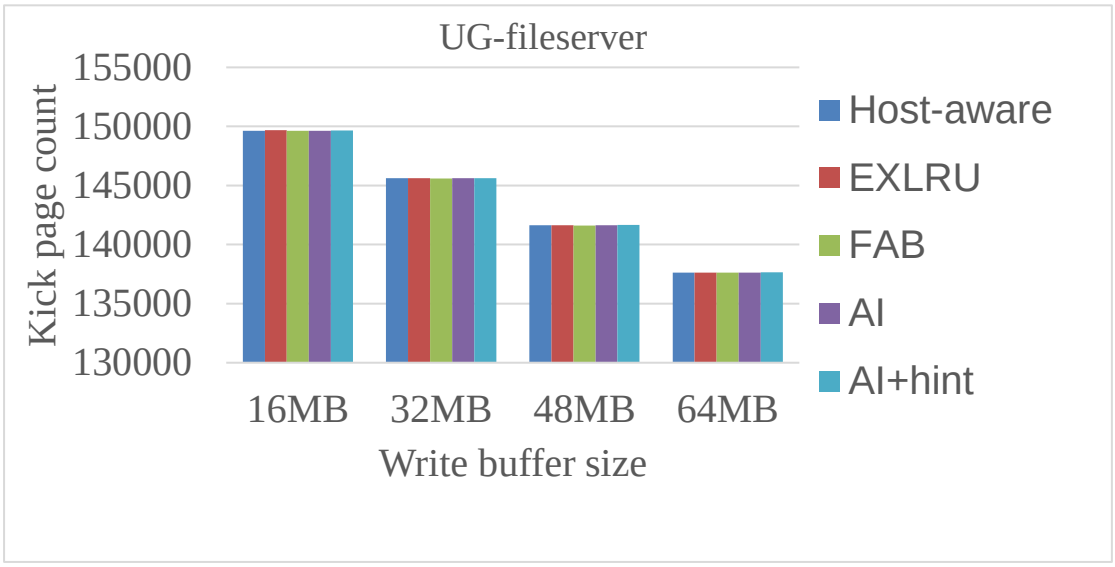

圖 4-11 UG-fileserver Kick page count

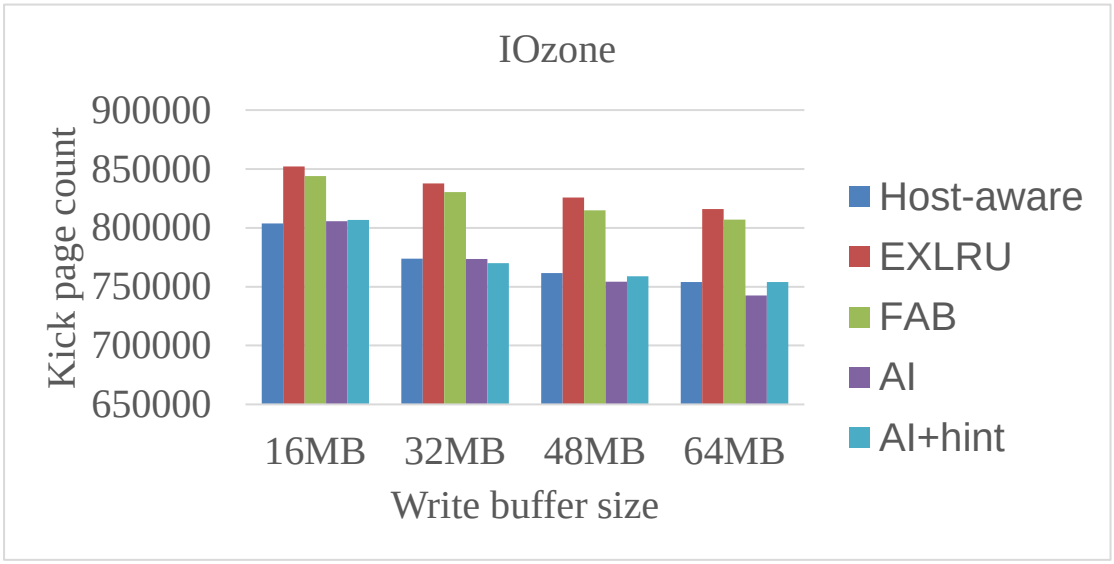

圖 4-12 IOzone Kick page count

如圖 4-10、4-12，大部分情況下 AI、AI+Hint 的 Kick page count 都比能夠贏過其他論文，那是因為在 Hit ratio 上的提升能夠反映到 Response time 和 Kick page count 上面，也因此整體而言，AI、AI+Hint 的 Kick page count 能夠比其他論文來的更進步。

此外，由圖 4-11 可看出，雖然 Hit ratio 有所提升，但 Kick page count 卻與其他論文差不多，會有這樣的表現可能是因為每一篇論文在 UG-fileserver 這個 Workload 的 Hit ratio 表現似乎都很差，因此即使 Hit ratio 有所提升，也無法很直接地從 Kick page count 看出來。

4.2.4 Improvement

由於 AI、AI+Hint 的主要目的是為了改善 Host-aware [1]，因此在這個章節說明 AI、AI+Hint 與 Host-aware [1]相比的改善幅度。

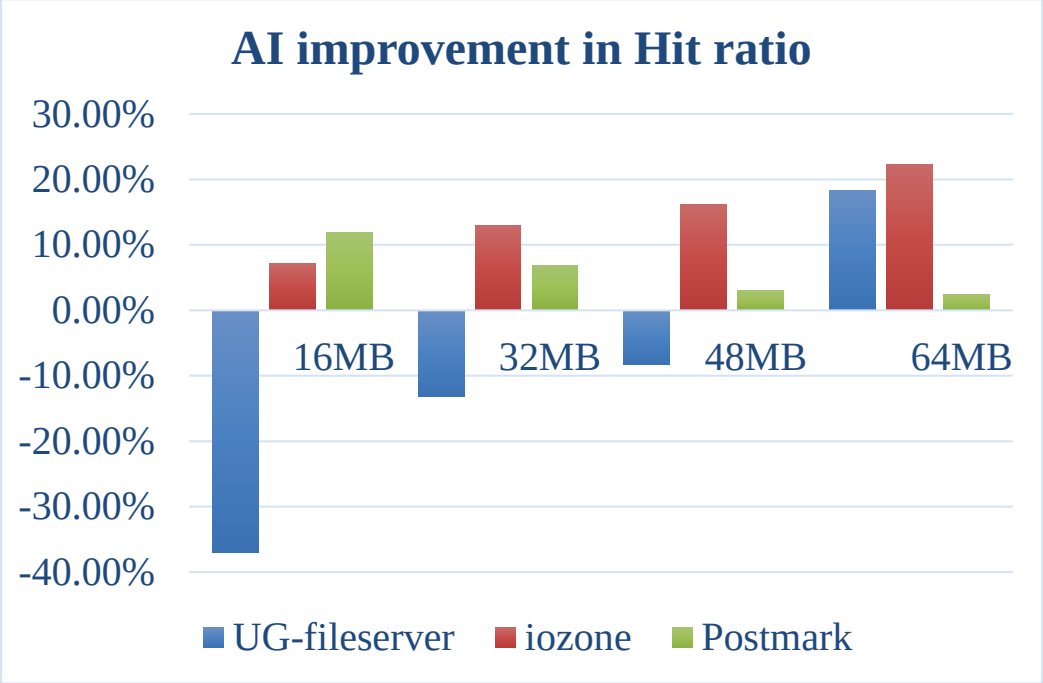

圖 4-13 AI improvement in Total hit ratio

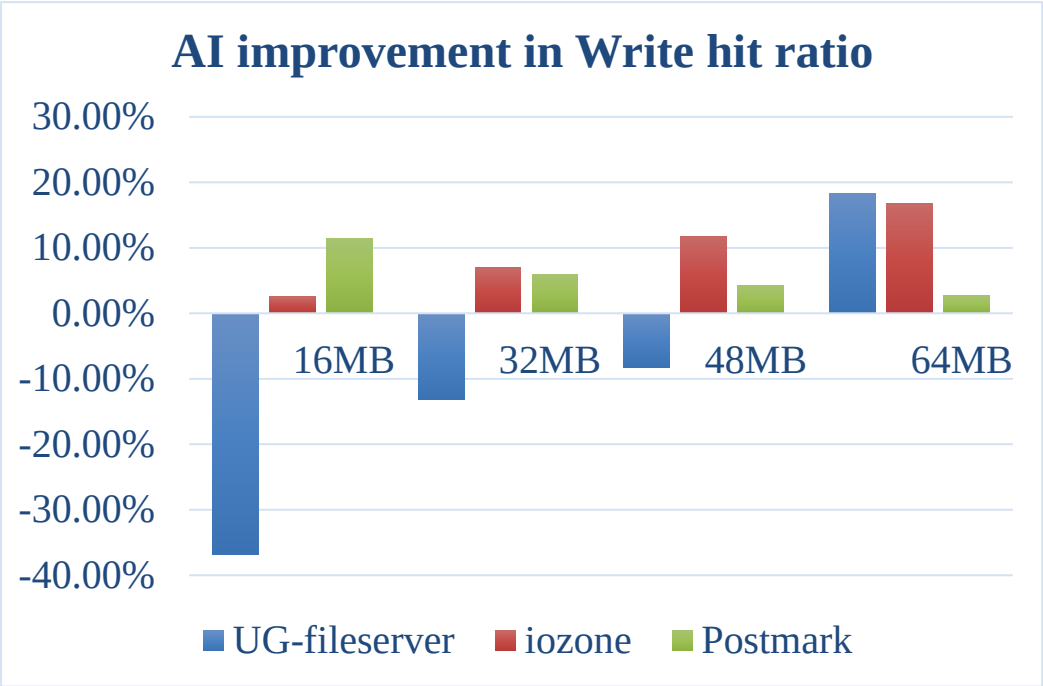

圖 4-14 AI improvement in Write hit ratio

從圖 4-13、4-14 可以看出，不同 workload 適合的 Write buffer size 不一樣，即使以同樣方式去選擇最佳的 p(權重)、Demoting threshold，但在 UG-fileserver 上，Write buffer size 越大，Hit ratio 成長的幅度也越大。但換成 Postmark 就完全相反，反而是 Write buffer 較小的時候，成長幅度比較大。

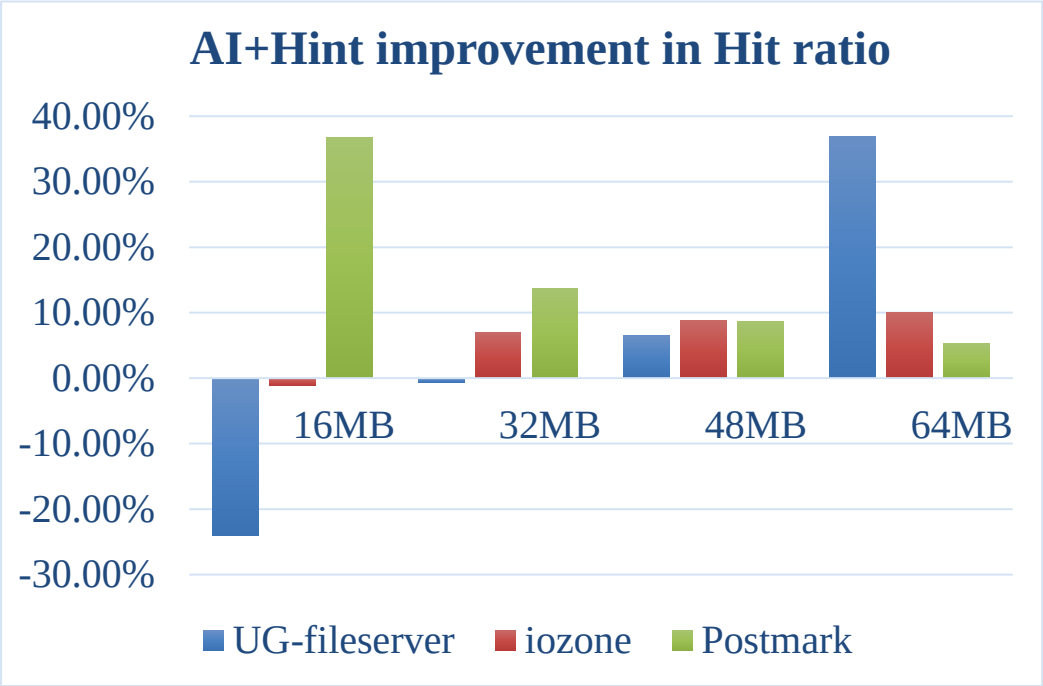

圖 4-15 AI+Hint improvement in Total hit ratio

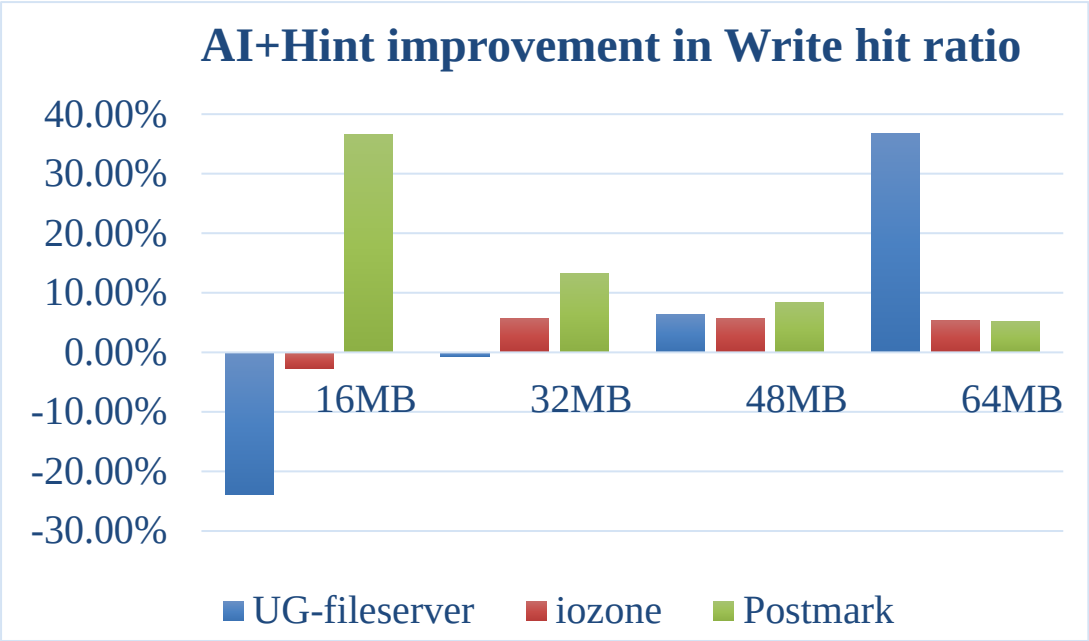

圖 4-16 AI+Hint improvement in Write hit ratio

如圖 4-15、4-16，如果去比較前面的圖 4-13、4-14，可以發現在 Hint [1]的幫助下，Hit ratio 的改善幅度都有顯著的提升。

4.2.5 Model response time

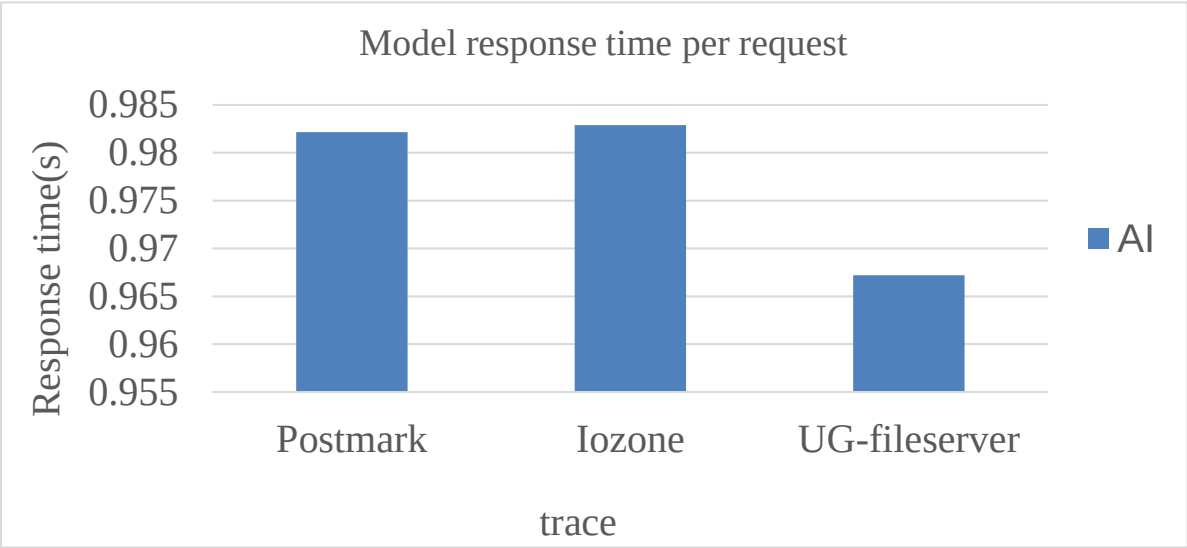

圖 4-17 Model response time per request

為了能夠知道 AI 在實際應用上的效果，如圖 4-17 我們測量了在三種 Workload 下 Model 預測每個 request 平均所花的時間。

可看出基本上每個 Model 都能在一秒內預測出對應的結果，以這個結果來說，如果放在 SSD 內似乎顯得太慢，因此實際應用時，將 Model 放在 Host 端預測比較合適。

4.2.6 Feature importance

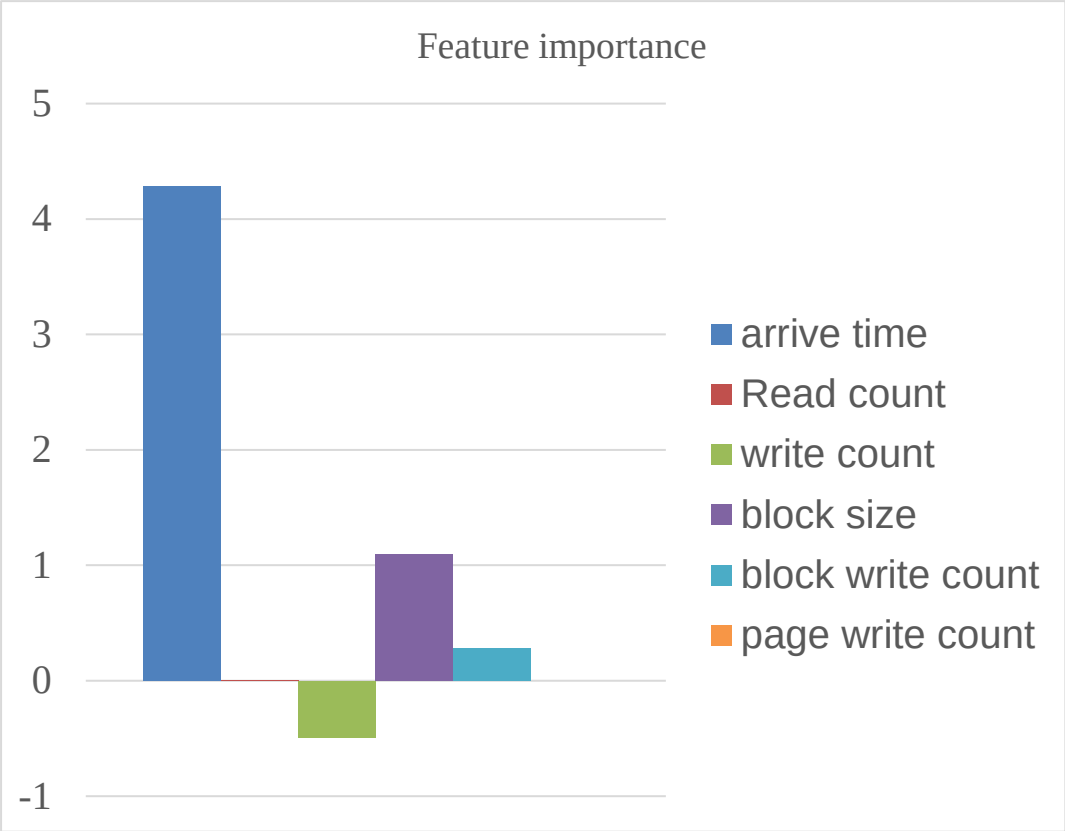

圖 4-18 Feature importance

圖 4-18 透過 Shaply value 來量測每個 input feature 的重要性，從上圖可看出，實際給予 AI model 當下 request 的時間對於 Model 來說是幫助最大的，可能是因為 LSTM 雖然可以【記住】之前的資訊，但應該沒辦法很有效的掌握多個資訊間的前後關係，因此 arrive time 對於 AI model 有很大的幫助。

除了【arrive time】以外，【block write count】、【block size】也都能對 AI model 的預測帶來正面的效益，那是因為前面提到的 benefit value 包含的 write count 與 GC 特性，各自對應到【block write count】、【block size】這兩個 feature，因此這兩個資訊讓 AI model 更容易給出正確的答案。

第五章　結論及未來工作

這篇論文透過使用 LSTM model 與 Host-aware Write buffer Management [1]中的 Hint 資訊做結合，能夠有效的提升 Write buffer 的 Hit ratio，在多數 Workload 中 Response time 也都還可以接受，比較可惜的是由於這篇論文並沒有特別針對 GC 改良，因此在少數 Workload 中會因為 GC 所造成的影響，大幅度增加 Response time。

LSTM model 雖然可說是目前很不錯的 AI 模型之一，但由於現存的 AI model 十分多樣化，有時候表現不好也許並不是參數、設計上的問題，而是挑選到不合適的 Model 訓練，因此，未來如果處理類似任務，其實可以嘗試使用 Automl(自動化模型)來處理，讓機器不只擔任預測的工作，連選擇 Model 這件事也讓機器來做。

此外，由於在 Device 做計算的 Overhead 太大，因此未來可以嘗試將 AI 預測、計算的部分移到 Host 端處理，目前其實阿里巴巴已有相關設計，將許多原本在 Device 做的事情搬到 Host 處理以減少 Overhead，未來若能夠將 AI model 搬到 Host 端處理，相信能有不錯的表現。

參考文獻

[1] Y.-M. Chen, “Exploiting Physical Device-aware and Logical Host-hinted information for Data striping and Write Buffer Management,” 2020.

[2] H. Kim and S. Ahn, “BPLRU: A buffer management scheme for improving random writes in flash storage,” *Proc. 6th USENIX Conf. File Storage Technol.,* p. 1 – 14, 2008.

[3] H. Jo, J.-U. Kang, S.-Y. Park, J.-S. Kim, and J. Lee, “FAB: Flash-aware buffer management policy for portable media players,” *IEEE Trans. Consumer Electron., vol. 52, no. 2,* p. 485 – 493, May 2006.

[4] L. Shi, J. Li, C. J. Xue, C. Yang, and X. Zhou, “EXLRU: A unified write buffer cache management for flash memory,” *Proc. ACM Int. Conf. Embed. Softw.,* pp. 339-348, 2011.

[5] “DiskSim,” [線上]. Available: http://www.pdl.cmu.edu/DiskSim/.

[6] “Understanding the Linux Kernel,3rd Edition,” [線上]. Available: http://www.johnchukwuma.com/training/UnderstandingTheLinuxKernel3rdEdition.pdf.

[7] D. W. Chang, H. H. Chen, D. J. Yang, and H. P. Chang, “BLAS: Block-Level Adaptive Striping for Solid-State Drives,” *ACM Transactions on Design Automation of Electronic Systems, Vol. 19, No2,* March 2014.

[8] Shahriar Ebrahimi, Reza Salkhordeh, Seyed Ali Osia, Ali Taheri, Hamid R. Rabiee, and Hossen Asadi, “RC-RNN: Reconfigurable Cache Architecture for,” *IEEE Transactions on Emerging Topics in Computing (2021),* p. 17, 5 11 2021.

[9] Li-Pin Chang, Sheng-Min Huang, “Exploiting Page Correlations for Write Buffering in Page-Mapping Multichannel SSDs,” *ACM Transactions on Embedded Computing Systems,* p. 25, 13 2 2016.

[10] Hung-yi-Lee, “Machine Learning(Hung-yi Lee,NTU),” [線上]. Available: https://www.youtube.com/playlist?list=PLJV_el3uVTsPy9oCRY30oBPNLCo89yu49.