SSD 就是大U盤？聊聊閃存類存儲的轉換層

目錄

上篇 「柱面-磁頭-扇區尋址的一些舊事」 整理了一下我對磁盤類存儲設備（包括軟盤、硬盤，不包括光盤、磁帶）的一些理解， 算是爲以後討論文件系統作鋪墊；這篇整理一下我對閃存類存儲設備的理解。

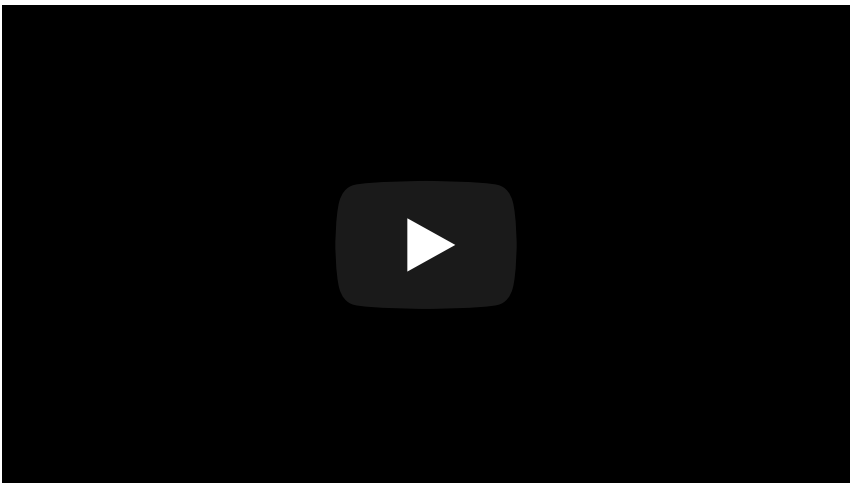
這裏想要討論的閃存類存儲是指 SSD 、SD卡、U盤、手機內置閃存等基於 NAND 又有閃存轉換層的存儲設備（下文簡稱閃存盤），但不包括裸 NAND 設備、3D Xpoint （Intel Optane）等相近物理結構但是沒有類似的閃存轉換層的存儲設備。 閃存類存儲設備這幾年發展迅猛，SD卡和U盤早就替代軟盤成爲數據交換的主流，SSD 大有替代硬盤的趨勢。 因爲發展迅速，所以其底層技術變革很快，不同於磁盤類存儲技術有很多公開資料可以獲取， 閃存類存儲的技術細節通常是廠商們的祕密，互聯網上能找到很多外圍資料， 但是關於其如何運作的細節卻很少提到。所以我想先整理一篇筆記，記下我蒐集到的資料，加上我自己的理解。 本文大部分信息

來源是 Optimizing Linux with cheap flash drives 和 A Summary on SSD & FTL，加上我的理解，文中一些配圖也來自這兩篇文章。

1 NAND Flash 原理

比 NAND Flash 更早的 EEPROM 等存儲技術 曾經用過 NOR Flash cell，用於存儲主板配置信息等少量數據已經存在 PC 中很久了。後來 NAND Flash 的微型化使得 NAND Flash 可以用於存儲大量數據，急劇降低了存儲成本，所以以 NAND Flash 爲基礎的存儲技術能得以替代硬盤等存儲設備。

Tutorial: Why NAND Flash Breaks Down



這裏不想涉及太多 NAND Flash 硬件細節，有個演講 Tutorial: Why NAND Flash Breaks Down 和 YouTube 視頻 介紹了其原理，感興趣的可以參考一下。只羅列一下視頻中提到的一些 NAND Flash 的特點：

- NAND Flash 使用 floating gate 中束縛電子來保存二進制數據，對這些 Cell 有讀取（Read）、 寫入（Programming）、擦除（Erase）的操作。擦寫次數叫 P/E cycle。
- 電子的量導致的電勢差可以區別 1 和 0 ，這是 Single Level Cell (SLC) 的存儲方式。 或者可以用不同的電勢差區分更多狀態保存更多二進制位，從而有 Multi-Level Cell (MLC)， TLC， QLC 等技術。可以對 MLC 的 Flash Cell 使用類似 SLC 的寫入模式，物理區別只是參考電壓， 只是 SLC 模式寫入下容量減半。
- 高密度設計下，一組 NAND Flash Cell 可以同時併發讀寫。所以有了讀寫頁 2KiB/4KiB 這樣的容量。 頁面越大，存儲密度越高，爲了降低成本廠商都希望提高讀寫頁的大小。
- 爲了避免添加額外導線，NAND Flash Cell 是使用基板上加負電壓的方式擦除 floating gate 中的二進制位的，所以擦除操作沒法通過地址線選擇特定 Cell 或者讀寫頁，於是整塊擦除有塊大小。
- 寫入操作對 SLC 單個 Cell 而言，就是把 1 置 0 ，而擦除操作則是把整塊置 1 。SLC 可以通過地址線單獨選擇要寫入的 Cell ，MLC 則把不同頁的二進制放入一個 Cell ，放入時有順序要求， 先寫處於高位的頁，再寫低位的。所以 MLC 中不同頁面地

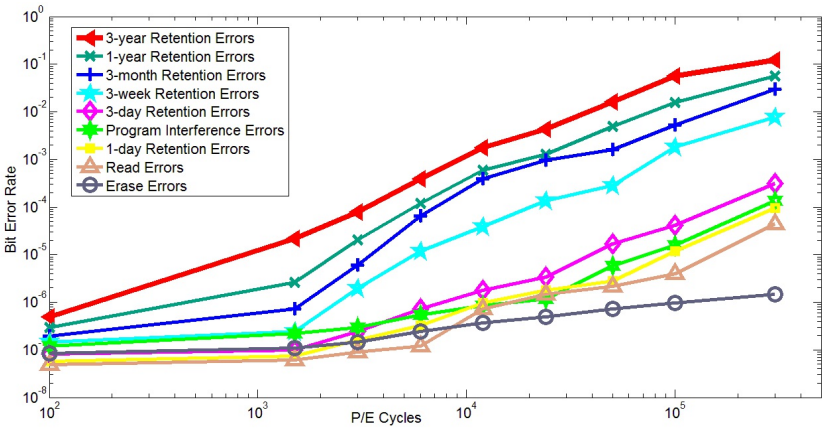
址的頁面是交錯在同一組 Cell 中的。

- SLC 其實並沒有特別要求擦除塊中的寫入順序，只是要求僅寫一次（從 1 到 0）。 MLC 則有先寫高位頁再寫低位頁的要求。廠商規格中的要求更嚴格，擦除塊中必須滿足按頁面編號順序寫入。
- 寫入和擦除操作是通過量子隧道效應把電子困在 floating gate 中的，所以是個概率事件。通過多次脈衝 可以縮小發生非預期概率事件的可能性，但是沒法完全避免，所以需要 ECC 校驗糾錯。
- 根據 ECC 強度通常有三種 ECC 算法，強度越強需要越多算力：
    - 漢民碼 可根據 n bit 探測 $2^n - n -1$ 中的 2 bit 錯誤，修正 1 bit 錯誤。
    - BCH碼 可根據 $n*m$ bit 糾錯 2^n bit 中的 m bit 錯誤。
    - LDPC 原理上類似擴展的漢民碼，能做到使用更少校驗位糾錯更多錯誤。
- 因爲 ECC 的存在，所以讀寫必須至少以 ECC 整塊爲單位，比如 256 字節或者整個頁面。
- 也因爲 ECC 的存在， $ECC(\texttt{0xFF}) \ne \texttt{0xFF}$ ，空頁（擦除後全1的頁面）必須特殊處理。所以需要區分寫了數據全 1 的頁和空頁。
- ECC校驗多次失敗的頁面可以被標記爲壞頁，出廠時就可能有一些壞頁，這些由轉換層隱藏起來。
- 斷電後，也有小概率下束縛的電子逃逸出 floating gate ，時間越長越可能發生可以探測到的位反轉。 所以基於 NAND Flash 的存儲設備應該避免

作爲存檔設備離線保存。

- 電子逃逸的概率也和溫度有關，溫度越高越容易逃逸，所以高溫使用下會有更高的校驗錯誤率。
- 讀取時，因爲用相對較高的電壓屏蔽沒有讀取的地址線，有一定概率影響到沒被讀取的頁面中存儲的數據。 控制器可能考慮週期性地刷新這些寫入後多次讀取的頁面，這可能和後文的靜態擦寫均衡一起做。
- 正在寫入或者擦除中突然斷電的話下，寫入中的一整頁數據可能並不穩定，比如短期內能正常讀取但是難以持續很長時間。

MLC 擦寫次數與錯誤率



上篇講硬盤的筆記中提到過，硬盤物理存儲也有越來越強的校驗機制，不過相比之下 NAND Flash 出現臨時性校驗失敗的可能性要高很多，需要控制器對校驗出錯誤的情況有更強的容忍能力。 廠商們製作存儲設備的時

候，有一個需要達到的錯誤率目標（比如平均 \(10^{14}\) bit 出現一次位反轉），針對這個目標和實際物理錯誤率，相應地設計糾錯強度。校驗太強會浪費存儲密度和算力， 從而提升成本，這裏會根據市場細分找折衷點。

2 封裝結構

從外部來看，一個閃存盤可能有這樣的結構：

從上往下，我們買到的一個閃存盤可能一層層分級：

1. 整個閃存盤有個控制器，其中含有一部分 RAM 。

然後是一組 NAND Flash 封裝芯片（chip）。

2. 每個封裝芯片可能還分多個 Device ，每個 Device 分多個 Die ，這中間有很多術語我無法跟上，大概和本文想討論的事情關係不大。
3. 每個 Die 分多個平面（Plane），平面之間可以並行控制，每個平面相互獨立。從而比如在一個平面內 做某個塊的擦除操作的時候，別的平面可以繼續讀寫而不受影響。
4. 每個平面分成多個段（Segment)，段是擦除操作的基本單位，一次擦除一整個段。
5. 每個段分成多個頁面（Page），頁面是讀寫操作的基本單位，一次可以讀寫一整頁。
6. 頁面內存有多個單元格（Cell），單元格是存儲二進制位的基本單元，對應 SLC/MLC/TLC/QLC 這些， 每個單元格可以存儲一個或多個二進制位。

以上這些名字可能不同廠商不同文檔的稱法都各有不同，比如可能有的文檔把擦除塊叫 page 或者叫 eraseblock 。隨着容量不斷增大，廠商們又新造出很多抽象層次，比如 chip device die 這些， 不過這些可能和本文關係不大。如果看別的文檔注意區別術語所指概念，本文中我想統一成以上術語。 重要的是有並行訪問單元的平面（Plane）、擦除單元的段（Segment）、讀寫單元的頁（Page）這些概念。 抽象地列舉概念可能沒有實感，順便說一下這些概念的數量級：

1. 每个 SSD 可以有数个封装芯片。
2. 每个芯片有多个 Die 。
3. 每个 Die 有多个平面。

4. 每个平面有幾千個段。比如 2048 個。
5. 每個段有數百個頁到幾千頁，比如 128~4096 頁，可能外加一些段內元数据。
6. 每個頁面是 2KiB~8KiB 這樣的容量，外加幾百字節的元數據比如 ECC 校驗碼。

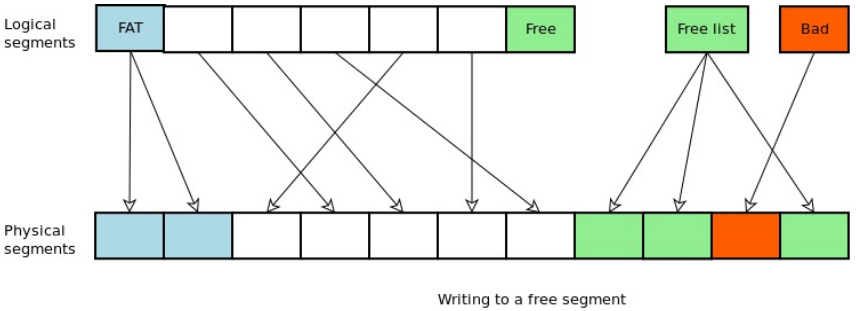
和硬盤相比，一個閃存頁面大概對應一個到數個物理扇區大小，現代硬盤也逐漸普及 4KiB 物理扇區， 文件系統也基本普及 4KiB 或者更大的邏輯塊（block）或者簇（cluster）大小，可以對應到一個閃存頁面。 每次讀寫都可以通過地址映射直接對應到某個閃存頁面，這方面沒有硬盤那樣的尋址開銷。 閃存盤的一個頁面通常配有比硬盤扇區更強的 ECC 校驗碼，因爲 NAND 單元格喪失數據的可能性比磁介質高了很多。

閃存有寫入方式的限制，每次寫入只能寫在「空」的頁面上，不能覆蓋寫入已有數據的頁面。 要重複利用已經寫過的頁面，需要對頁面所在段整個做擦除操作，每個段是大概 128KiB 到 8MiB 這樣的數量級。每個擦除段需要統計校驗失敗率或者跟蹤擦除次數，以進行擦寫均衡（Wear Leveling）。

3 擦寫均衡（Wear Leveling）和映射層（Flash Translation

Layer)

Animation: wear leveling on SSD drives



擦除段的容量大小是個折衷，更小的擦除段比如 128KiB 更適合隨機讀寫， 因爲每隨機修改一部分數據時需要垃圾回收的粒度更小；而使用更大的擦除段可以減少元數據和地址映射的開銷。 從擦除段的大小這裏，已經開始有高端閃存和低端閃存的差異，比如商用 SSD 可能比 U 盤和 SD 卡使用更小的擦除段大小。

閃存盤中維護一個邏輯段地址到物理段地址的映射層，叫閃存映射層（Flash Translation Layer ）。每次寫一個段的時候都新分配一個空段， 寫完後在映射表中記錄其物理地址。映射表用來在讀取時做地址轉換，所以映射表需要保存在閃存盤控制器的 RAM 中，同時也需要記錄在閃存內。具體記錄方式要看閃存盤控制器的實現，可能是類似日誌的方式記錄的。

「段地址映射表」的大小可以由段大小和存儲設備容量推算出來。比如對一個 64GiB 的 SD 卡，如果使用 4MiB 的段大小，那麼需要至少 16K 個表項。假設映射表中只記錄 2B 的物理段地址， 那麼需要 32KiB 的 RAM 存儲段地址映射表。對一個 512GiB 的 SSD ，如果使用 128KiB 的段大小， 那麼至少需要 4M 個表項。記錄 4B 的物理段地址的話，需要 16MiB 的 RAM 存儲地址映射，或者需要動態加載的方案只緩存一部分到 RAM 裏。控制器中的 RAM 比 NAND 要昂貴很多，這裏可以看出成本差異。

除了地址映射表，每個物理段還要根據擦除次數或者校驗錯誤率之類的統計數據，做擦寫均衡。有兩種擦寫均衡：

- 動態擦寫均衡（Dynamic Wear Leveling）：每次寫入新段時選擇擦除次數少的物理段。
- 靜態擦寫均衡（Static Wear Leveling）：空閒時，偶爾將那些許久沒有變化的邏輯段搬運到 多次擦除的物理段上。

低端閃存比如 SD 卡和 U 盤可能只有動態擦寫均衡，更高端的 SSD 可能會做靜態擦寫均衡。 靜態擦寫均衡想要解決的問題是：盤中寫入的數據可以根據寫入頻率分爲冷熱， 總有一些冷數據寫入盤上就不怎麼變化了，它們佔用着的物理段有比較低的擦除計數。 只做動態擦寫均衡的話，只有熱數據的物理段被頻繁擦寫，加速磨損， 通過靜態擦寫均衡能將冷數據所在物理段釋放

出來，讓整體擦寫更平均。 但是靜態擦寫均衡搬運數據本身也會磨損有限的擦寫次數，這需要優秀的算法來折衷。

除了擦寫均衡用的統計數據外， FTL 也要做壞塊管理。閃存盤出廠時就有一定故障率，可能有一部分壞塊。 隨着消耗擦寫週期、閒置時間、環境溫度等因素影響，也會遇到一些無法再保證寫入正確率的壞塊。 NAND Flash 上因爲量子隧道效應，偶爾會有臨時的校驗不一致，遇到這種情況，除了根據 ECC 校驗恢復數據， FTL 也負責嘗試對同一個物理段多次擦除和讀寫，考察它的可用性。排除了臨時故障後， 如果校驗不一致的情況仍然持續，那麼需要標註它爲壞塊，避免今後再寫入它。

出廠時，閃存盤配有的物理段數量就高於標稱的容量，除了出廠時的壞塊之外，剩餘的可用物理段可以用於 擦寫均衡，這種行爲稱作 Over Provisioning 。除了盤內預留的這些空間，用戶也可以主動通過分區的方式或者文件系統 TRIM 的方式預留出更多可用空間， 允許 FTL 更靈活地均衡擦寫。

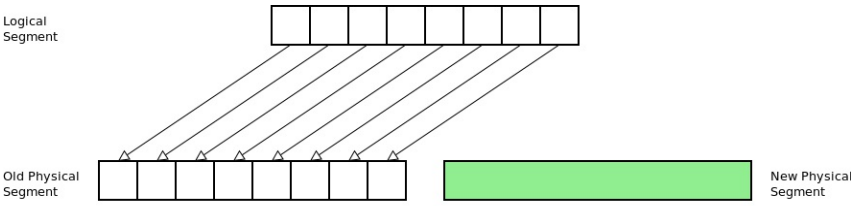
4 段內寫入順序與垃圾回收策略

段是閃存盤的擦寫單元，考慮到段是 128KiB ~ 8MiB 這樣的數量級，現實中要求每次連續寫入一整段的話，這樣的塊設備接口不像硬盤的接口，不方便普通文件系統使用。所以在段的抽象之下有了更小粒度的頁面抽象， 頁面對應到文件系統用的邏輯塊大小，是 2KiB~8KiB 這樣的數量級，每次以頁面爲單位讀寫。

寫入頁面時有段內連續寫入的限制，於是需要段內映射和垃圾回收算法，提供對外的隨機寫入接口。 寫入操作時， FTL 控制器內部先「打開（open）」一個段，等寫入完成，再執行垃圾回收「關閉(close)」一個段。寫入過程中處於打開狀態的段需要一些額外資源（RAM 等）跟蹤段內的寫入狀況，所以閃存盤同時能「打開」的段數量有限。並且根據不同的垃圾回收算法，需要的額外資源也不盡相同，在 Optimizing Linux with cheap flash drives 一文中介紹幾種可能的垃圾回收算法：

4.1 線性寫入優化

Animations: linear-access optimized



假設寫入請求大部分都是連續寫入，很少有地址跳轉，那麼可以使用線性優化算法。

- Open：當第一次打開一個段，寫入其中一頁時，分配一個新段。如果要寫入的頁不在段的開頭位置，那麼搬運寫入頁面地址之前的所有頁面到新段中。
- Write: 在 RAM 中跟蹤記錄當前寫入位置，然後按順序寫下新的頁面。
- Close: 最後搬運同段中隨後地址上的頁面，並關閉整段，調整段映射表。

如果在段內寫入了幾頁之後，又跳轉到之前的位置，那需要在跳轉時關閉當前段寫入（並完整搬運剩下的頁面）， 然後重新打開這一段，搬運調轉地址之前的頁面，從跳轉的頁面位置開始寫入。

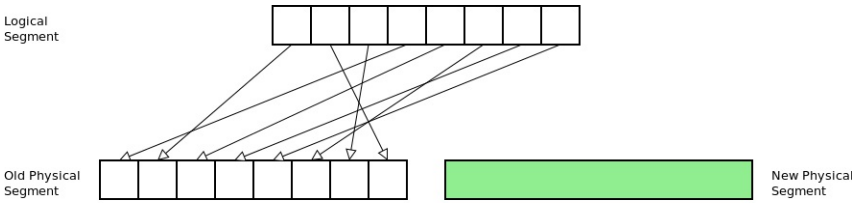
線性優化算法的好處在於：沒有複雜的頁面地址映射，段內的邏輯頁面地址就是物理頁面地址。 讀一頁的時候根據頁面偏移和當前寫入位置就能判斷讀新物理段還是老物理段。遇到突然斷電之類的情況， 即使丟失最近寫入的新物理段，老物理段的數據仍然還在，所以沒必要保存 RAM 中的地址映射到閃存元數據中。

線性優化算法的壞處是:每遇到一次亂序的寫入，都要整段執行一次搬運，造成 寫入放大（Write Amplification） 。

一些文檔中，將這種地址映射垃圾回收方式叫做「段映射（Segment Mapping）」，因爲從 FTL 全局來看只維護了擦寫段的地址映射關係。

4.2 段內地址映射

Animations: block remapping



對需要隨機亂序寫入的數據，可以使用段內地址映射。方式是額外在段外的別的閃存區域維護一張段內地址映射表， 像段地址一樣，通過查表間接訪問頁面地址。

- Open: 分配一塊新的段，同時分配一個新的段內映射表。
- Write: 每寫入一頁，在段內映射表記錄頁面的在新段中的物理地址。
- Close: 複製老段中沒有被覆蓋寫入的頁到新段，並記錄在段內映射表中，然後釋放老段和老的段內映射表。

也就是說同時維護兩塊不同大小的閃存空間，一塊是記錄段數據的，一塊是記錄段內地址映射表的， 兩塊閃存空間有不同的寫入粒度。可以在每個物理段內額外留出一些空間記錄段內地址映射表，也可以在 FTL 全局

維護一定數量的段內地址映射表。 每次讀取段內的數據時，根據映射表的內容，做地址翻譯。新段中頁面的排列順序將是寫入的順序， 而不是地址順序。

根據實現細節，段內地址映射可以允許覆蓋寫入老段中的頁面，但是可能不允許覆蓋寫入新段（正在寫入的段） 中已經寫入的頁面，遇到一次連續的寫請求中有重複寫入某一頁面的時候，就需要關閉這一段的寫入，然後重新打開。

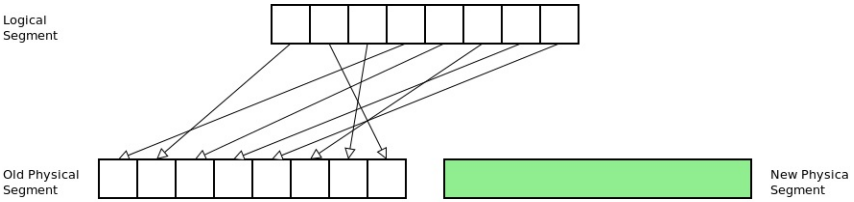
段內地址映射的優點是：支持隨機寫入，並且只要段處於打開狀態，隨機寫入不會造成寫入放大（Write Amplification）。

缺點是：首先地址映射這層抽象有性能損失。其次遇到突然斷電之類的情況， 下次上電後需要掃描所有正打開的段並完成段的關閉操作。

和「段映射」術語一樣，在一些文檔中，將這種段內地址映射的方式叫做「頁面映射（Page Mapping）」，因爲從 FTL 全局來看跳過了擦寫段這一層，直接映射了頁面的地址映射。

4.3 日誌式寫入

Animations: data logging



除了大量隨機寫入和大量連續寫入這兩種極端情況，大部分文件系統的寫入方式可能會是對某個地址空間 進行一段時間的隨機寫入，然後就長時間不再修改，這時適合日誌式的寫入方式。

日誌式的寫入方式中寫入一段採用三個物理段：老物理段，用於日誌記錄的新物理段，和垃圾回收後的段。

- Open: 分配一塊新的段。可能額外分配一個用於記錄日誌的段，或者將日誌信息記錄在數據段內。
- Write：每寫入一頁，同時記錄頁面地址到日誌。
- Close：再分配一個新段執行垃圾回收。按日誌中記錄的地址順序將數據段中（新寫入）的頁面或者老段中 沒有被覆蓋的頁面複製到垃圾回收結束的新段中。

日誌式寫入在寫入過程中像段內地址映射的方式一樣，通過日誌記錄維護頁面地址映射關係， 在寫入結束執行垃圾回收之後，則像線性寫入的方式一樣不再需要維護頁面映射。 可以說日誌式寫入某種程度上綜合了前面兩種寫入方式的優點。

日誌式寫入的優點：允許隨機順序寫入，並且在執行垃圾回收之後，不再有間接訪問的地址轉換開銷。

日誌式寫入的缺點：觸發垃圾回收的話，可能比段地址映射有更大的寫入放大（Write Amplification）。

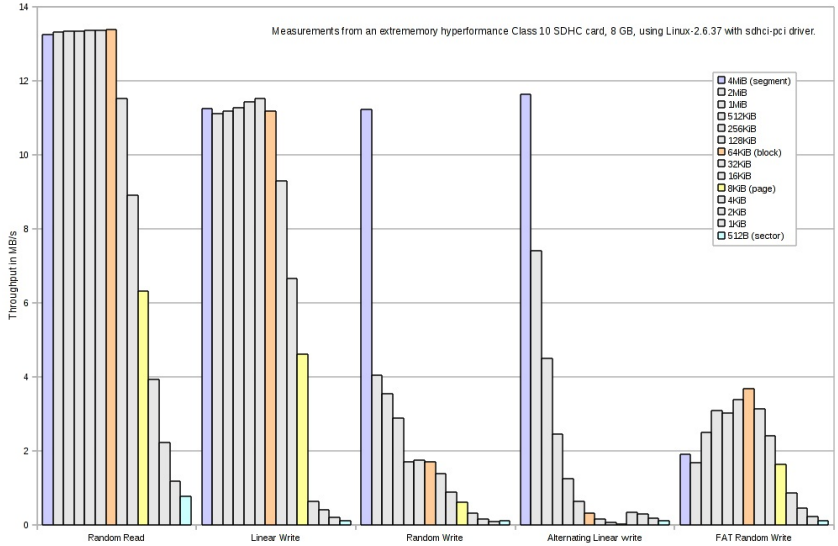
在一些文檔中，將這種日誌式寫入方式稱作「混合映射（Hybrid Mapping）」，因爲在段開啓寫入期間行爲像頁面映射， 在段關閉寫入後行爲像段映射。

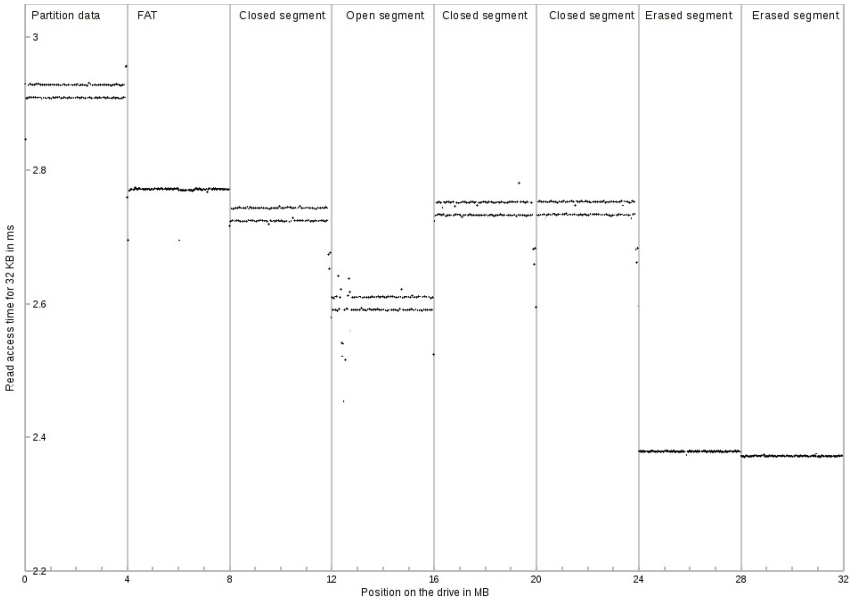
5 針對特定寫入模式的優化

上述三種地址映射和垃圾回收方式，各有不同的優缺點，根據數據塊的寫入模式可能需要挑選相應的策略。 並且「全局段地址映射表」、「段內頁面地址映射表」、「寫入頁面地址日誌」之類的元數據因爲頻繁修改， FTL 也可能需要用不同的策略來記錄這些元數據。這裏面向不同使用場景的閃存設備可能有不同的 FTL 策略，並且 FTL 可能根據邏輯地址來選擇哪種策略。

5.1 混合垃圾回收策略

Performance measurements on a class 10 SDHC card





用來記錄照片、視頻等的 SD 卡、microSD、U盤等設備可能根據數據的邏輯地址，爲特定文件系統佈局優化， 這裏特定文件系統主要是指 FAT32 和 exFAT 這兩個 FAT 系文件系統。 FAT 系文件系統的特點在於， 地址前端有一塊空間被用來放置 文件分配表(File Allocation Table) ，可以根據文件系統簇大小和設備存儲容量推算出 FAT 表佔用大小，這塊表內空間需要頻繁隨機讀寫。對 FTL 自身的元數據，和 FAT 表的邏輯地址空間，需要使用「段內地址映射」來保證高效的隨機讀寫， 而對隨後的數據空間可使用「線性寫入優化」的策略。

右側上圖有張性能曲線，測量了一個 class 10 SDHC 卡上，不同讀寫塊大小時，順序讀取、順序寫入、隨機寫入、 對 FAT 區域的寫入之類的性能差異。下圖是測量的讀取延遲。可以看出 FAT 區域的隨機寫入和其餘邏輯地址上有明顯不同的性能表現。

爲容納普通操作系統設計的 eMMC 和 SSD 難以預測文件系統的讀寫模式，可能需要使用更複雜的地址映射和垃圾回收策略。 比如一開始假定寫入會是順序寫入，採用「線性優化」方式；當發生亂序寫入時，轉變成類似「日誌式寫入」 的方式記錄寫入地址並做地址映射；關閉段時，再根據積累的統計數據判斷，可能將記錄的日誌與亂序的數據 合併（merge）成順序的數據塊，也可能保持頁面映射轉變成類似「段內地址映射」的策略。

5.2 利用 NAND Flash 物理特性的優化

再考慮 NAND Flash 的物理特性，因爲 MLC 要不斷調整參考電壓做寫入， MLC 的寫入比 SLC 慢一些，但是可以對 MLC Flash 使用 SLC 式的寫入， FTL 控制器也可能利用這一點，讓所有新的寫入處於 SLC 模式，直到關閉整段做垃圾回收時把積攢的 SLC 日誌段回收成 MLC 段用於長期保存。 一些網頁將這種寫入現象稱作「SLC 緩存」甚至稱之爲作弊，需要理解這裏並不是用單獨的 SLC Flash 芯片做 writeback 緩存，更不是用大 RAM 做緩存，處於 SLC 模式的寫入段也是持久存儲的。

5.3 同時打開段數

上述地址映射和垃圾回收策略都有分別的打開（open）、寫入（write）、關閉（close）時的操作，閃存盤通常允許同時打開多個段，所以這三種操作不是順序進行的，某一時刻可能同時有多個段處在打開的狀態， 能接受寫入。不過一個平面（Plane）通常只能進行一種操作（讀、寫、擦除），所以打開寫入段時， FTL 會儘量讓寫入分部在不同的平面上。還可能有更高層次的抽象比如 Device、 Chip 、 Die 等等，可能對應閃存盤內部的 RAID 層級。

閃存盤能同時打開的段不光受平面之類的存儲結構限制，還受控制器可用內存（RAM）限制之類的。 爲 FAT 和順序寫入優化的 FTL ，可能除了 FAT 區域之外，只允許少量（2~8）個併發寫入段， 超過了段數之後就會對已經打開的段觸發關閉操作（close），執行垃圾回收調整地址映射，進而接受新的寫入。 更高端的 SSD 的 FTL 如果採用日誌式記錄地址的話，同時打開的段數可能不再侷限於可用內存限制， 連續的隨機寫入下按需動態加載段內地址映射到內存中，在空閒時或者剩餘空間壓力下才觸發垃圾回收。

5.4 預格式化

FTL 可能爲某種文件系統的寫入模式做優化，同時如果文件系統能得知 FTL 的一些具體參數（比如擦除段大小、 讀寫頁大小、隨機寫入優化區域），那麼可能更好地安排數據結構，和 FTL 相互配合。 F2FS 和 exFAT 這些文件系統都在最開頭的文件系統描述中包含了一些區域，記錄這些閃存介質的物理參數。 閃存盤出廠時，可能預先根據優化的文件系統做好格式化，並寫入這些特定參數。

5.5 TRIM 和 discard

另一種文件系統和 FTL 相互配合的機制是 TRIM 指令。TRIM 由文件系統發出，告訴底層閃存盤（或者別的類型的 thin provisioning 塊設備）哪些空間已經不再使用， FTL 接受 TRIM 指令之後可以避免一些數據搬運時的寫入放大。關於 TRIM 指令在 Linux 內核中的實現，有篇 The best way to throw blocks away 介紹可以參考。

考慮到 FTL 的上述地址映射原理， TRIM 一塊連續空間對 FTL 而言並不總是有幫助的。 如果被 TRIM 的地址位於正在以「段內地址映射」或「日誌式映射」方式打開的寫入段中，那麼 TRIM 掉一些頁面可能減少垃圾回收時搬運的頁面數量。但是如果 TRIM 的地址發生在已經垃圾回收結束的段中， 此時如果 FTL 選擇立刻對被 TRIM 的段執行垃圾回收，可能造成更多寫入放大， 如果選擇不回收只記錄地址信息，記錄這些地址信息也需要耗費一定的 Flash 寫入。 所以 FTL 的具體實現中，可能只接受 TRIM 請求中，整段擦除段的 TRIM ，而忽略細小的寫入頁的 TRIM 。

可見 FTL 對 TRIM 的實現是個黑盒操作，並且 TRIM 操作的耗時也非常難以預測，可能立刻返回， 也可能需要等待垃圾回收執行結束。

對操作系統和文件系統實現而言，有兩種方式利用 TRIM ：

1. 通過 discard 掛載選項，每當釋放一些數據塊時就執行 TRIM 告知底層塊設備。
2. 通過 fstrim 等外部工具，收集連續的空塊並定期發送 TRIM 給底層設備。

直覺來看可能 discard 能讓底層設備更早得知 TRIM 區域的信息並更好利用，但是從實現角度來說， discard 不光影響文件系統寫入性能，還可能發送大量被設備忽略掉的小塊 TRIM 區域。可能 fstrim 方式對連續大塊的區間執行 TRIM 指令更有效。

6 TL;DR 低端 vs 高端

標題中的疑問「SSD就是大U盤？」相信看到這裏已經有一些解答了。 即使 SSD 和U盤中可以採用類似的 NAND Flash 存儲芯片，由於他們很可能採用不同的 FTL 策略，導致在讀寫性能和可靠性方面都有不同的表現。（何況他們可能採用不同品質的 Flash ）。

如果不想細看全文，這裏整理一張表，列出「高端」閃存盤和「低端」閃存盤可能採取的不同策略。 實際上大家買到的盤可能處於這些極端策略中的一些中間點，市場細分下並不是這麼高低端分明。 比如有些標明着「爲視頻優化」之類宣傳標語的「外置SSD」，對消費者來說可能會覺得爲視頻優化的話一定性能好， 但是理解了 FTL 的差異後就可以看出這種「優化」只針對線性寫入，不一定適合放系統文件根目錄的文件系統。

參數	低端	高端
段大小	8MiB	128KiB
段地址映射	靜態段映射	日誌式映射

隨機寫入範圍	FTL元數據與FAT表區域	全盤
同時打開段數	4~8	全盤
物理段統計信息	無（隨機挑選空閒段）	擦除次數、校驗錯誤率等
擦寫均衡	動態均衡（僅寫入時分配新段考慮）	靜態均衡（空閒時考慮搬運）
寫入單元模式	TLC	長期存儲 MLC，模擬 SLC 日誌

介紹完閃存類存儲，下篇來講講文件系統的具體磁盤佈局，考察一下常見文件系統如何使用 HDD/SSD 這些不同讀寫特性的設備。