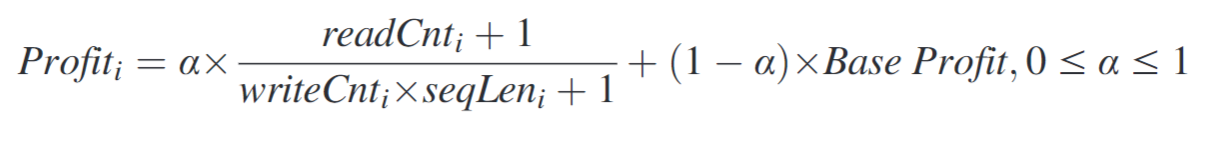
Profit data caching for hybrid disk

* Deceptive idleness
  + 上一個request和下一個request的距離是完全相反方向，以至於為了服務多個request，讀寫頭必須來來回回一直做尋找的動作，十分耗費seeking overhead
* Anticipatory scheduler
  + 若當下request位置與下一個request位置隔很遠，就先暫停，等到之後遇到附近的request，再去服務那個request
* CFQ
  + 藉由各個process獨立的queue，來實現共享頻寬的目的
    - 首先，使用類似RR的方式，在多個queue(process)之間做切換
    - 使用one-way elevator algorithm來做搜尋，以減少seeking time
    - 用anticipatory scheduler解決deceptive idleness問題
    - 一般來說，會使用同步更新的方式來處理連續性的read request，因此下一個request會等上一個request完成才開始做
    - 當current request queue is empty就會暫停服務
* Select only high-benefit value block to cache in flash memory
* benefit value，會考慮【randomness/ sequentiality】【read-intensiveness/write-intensiveness】【recency】
  + 藉由【大量read/write operation】來掌握【read/write-intensiveness】
  + 藉由【sequence length】來掌握【randomness/sequentiality】
    - 這是根據random access 通常會屬於short sequence，而large sequence通常會屬於sequential access這樣的特性
* 
  + 依照profit 的公式來看，讀很多，寫很少，會造成最佳的profit value，理由同上
  + 寫入的數目會乘上寫入的長度，是因為考慮到random/sequential的關係
* HA-CFQ
  + If request access HDD
    - 如果current request queue is empty，HA-CFQ會到next request queue找下一個request
    - 若下一個request是對HDD做存取，則採用CFQ方式來減少seeking time，但若是對SSD做存取，就不暫停了，直接服務該request🡪因為SSD架構不同，不會有seeking time，而CFQ的方式是為了減少seeking time而設計的，但這種方式對於SSD來說是沒有意義的。
    - 若資料是對SSD做存取，那CFQ的做法就會變成單純浪費時間於等待，不會有什麼改善
  + If request access SSD(有點不懂)
    - 在結束這個 request後，去【尋找】下一個request(also access SSD)，直到兩種狀況，才會停止【尋找】
      1. 附近的request在預期的結束時間內到來
      2. 下個request(並非附近的request)是對HDD做存取
         * 在第二種狀況下，假設累積的service time of these flash memory request超過預期的結束時間，這時候，HA-CFQ才會去服務該request
         * 否則，若累積的service time of these flash memory request沒有超過預期的結束時間，就按照CFQ方式去服務附近的request節省seeking time
         * 結論:可節省不必要的I/O
* HA-CFQ節省的時間
  + Tw=actual time interval=min(Tx,Ty)
  + Tx=anticipation time
  + Ty=inter-arrival time=Tn-Tc
    - Tn=next close-by request arriving time
    - Tc=the time the scheduler uses to handle the last request