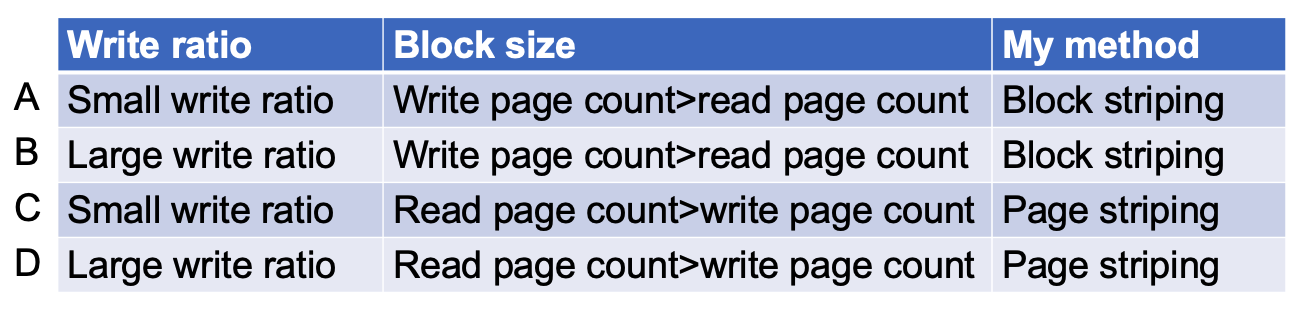
學姊(吳宜芬)程式：write buffer以logical block管理

學長(陳繹閔)程式：write buffer以physical block管理

完成進度：

* 從學長的程式抓取需要資訊，訓練model
  + All buffer
    - Duration model
  + Ignore min benefit block
    - Duration model
    - Buffer\_or\_not model

目標：

* 改disksim🡪依照每個block的benefit value（看過整個trace得到的）來踢資料
  + 其實應該是要將model預測好的結果拿去做，但我想先測試直接將模擬結果套入disksim看看表現如
* 實作部分
  + 將進入SSD.C的request資訊寫入txt
  + 利用建立好的model預測
  + 將預測的結果寫成txt，並從disksim讀入(從SSD.C的入口讀入)
  + 依照結果來實作
  + 看write count, total count, flush count…
* 比較兩者效能差異(在disksim)
  + Total hit ratio(write and read)
  + Write hit ratio
  + Flush block count(與block size成反比)
* Collect data from disksim
  + Result(再多測試幾個trace)
    - Write buffer以Logical block管理：資料比較集中， write buffer block number較多，從write buffer踢掉的資料不會再度被存取
    - Write buffer以Physical block管理：資料比較分散，write buffer block number較少，從write buffer踢掉的資料會再度被存取，因此會發生【一個page同時出現在多個block】的奇怪現象
      * 那是因為踢掉的資料再度被存取，因此再度被放入write buffer，但被放入不同的block內，而我抓取的是進入write buffer的block number，因此就會產生上述的現象
      * 猜測可能是因為資料太過分散，有點難以區分hot or cold data，因此容易不小心踢到hot data
* To do
  + 2/14
    - Striping要固定(可用學長的)，但是學長的striping是使用host資訊，而我只用SSD的資訊
    - Striping方式，要看當下write buffer內的狀況?還是整個歷史資訊?
      * 只看當下：當下被寫入達到某個threshold，就執行對應的striping，
      * 看整個歷史資訊：由於實際狀況可能是，先在write buffer寫入，然後到flash memory讀取，接著又跑到write buffer寫入，不斷反覆。採取歷史資訊的話，就會將整個納入考量。
  + 測試
    - 簡易版
      * Write buffer滿了，就踢一個下去
    - 進階版
      * Write buffer預留某個比例的空間(ex:10% or 15%...)，當write buffer容量小於該比例，就開始踢資料(在SSD idle時)……PS:踢資料是在background作業
    - 3/7更新：
      * Striping 方式除了考慮write count以外，還需要考慮size，否則大量的read/write只針對特定page的話，使用page striping不會有優勢（block striping也是一樣概念）
* 進度：
  + Striping的方式：
    - **如果write page count>read page count且 write ratio(per block)=>50%，就以block striping擺放**
    - **如果write page count<=read page count，就以page striping擺放**
    - Write page count=從放入write buffer到從write buffer踢掉，該block中有多少page被寫入
    - Read page count=同上，但是是計算read page
    - Write count=從放入write buffer到從write buffer踢掉，該block被寫入的次數
    - Read count=同上，但是是計算該block被讀取的次數
  + 
  + 狀況(A)：很常讀取，但是read page count比較少，代表read request集中在少數page中，即使採用page striping也用不到平行畫的好處，因此採用block striping
  + 狀況(B)：很常寫入，write page count又多，理所當然採用block striping
  + 狀況(C)：很常讀取，read page count又高，代表read request被平均分散在多個page中，此時採用page striping能有效用到page striping平行化的優勢，因此使用page striping
  + 狀況(D)：很常寫入，但是write page count很低，代表write request被集中在少數page中
    - 以block striping來說，因為write page count很低，因此會提高GC的時間成本
    - 以page striping來說，因為很少讀取，但是read page count又高，代表有可能無法完全利用到page striping平行化的優勢（比如說page striping一次可以讀8個channel，但實際上read request只來4個page）
    - 在這個狀況中，考量到『GC時間成本』與『read request的response time』，將『GC時間成本』優先考量，因此採用page striping