

Notation

- m = the number of memory chips
 - b = the number of distinct blocks on the disk
 - j = the number of DMA “slots”
 - n = the number of memory accesses
 - d = the number of disk blocks a memory chip can hold
- * Indexes start from 0

Equations

$$f_k(C_k, CT_k, r, i+1) = \begin{cases} f_k(C_k, CT_k, r, i) & \text{if } (i+1) - CT_k[S_{\text{busy}}] > \alpha \wedge a_{i+1} \notin C_k \wedge r \neq k \quad (1) \\ f_k(C_k, CT_k, r, i) + 1 & \text{if } (i+1) - CT_k[S_{\text{busy}}] \leq \alpha \wedge a_{i+1} \notin C_k \wedge r \neq k \quad (2) \\ f_k(C_k, CT'_k, r, i) + 1 & \text{if } a_{i+1} \in C_k \wedge r \neq k \quad (3) \\ f_k(C'_k, CT'_k, r, i) + 1 & \text{if } a_{i+1} \in C_k \in \text{repl}(C'_k, a_{i+1}) \wedge r = k \quad (4) \end{cases}$$

$$F(i+1) = \begin{cases} \sum_{k=1}^m f_k(C_k, CT_k, 0, i) + \min_{CT'_h} f_h(C_h, CT'_h, 0, i) & \text{if } a_{i+1} \in C_h \Leftrightarrow a_{i+1} \in C \quad (5) \\ \min_{C'_k, CT'_k, r' | r' \neq 0} \left(\sum_{k=1}^m f_k(C'_k, CT'_k, r', i) \right) & \text{if } a_{i+1} \notin C \end{cases}$$

Algorithm

There wasn't really an “algorithm” for the computation, so I created one that closely followed the algorithm given in *Power-Aware Storage Cache Management* (Zhu & Zhou).

```

for  $i = 0$  to  $n - 1$  do
  for  $k = 0$  to  $m$  do
    for all  $C_k$  such that  $C_k$  is a set of  $d$  disk blocks do
      if  $(i + 1) - CT_k[S_{\text{busy}}] > \alpha \wedge a_{i+1} \notin C_k \wedge r \neq k$  then
         $f_k(C_k, CT_k, r, i + 1) \leftarrow f_k(C_k, CT_k, r, i)$ 
      else if  $(i + 1) - CT_k[S_{\text{busy}}] \leq \alpha \wedge a_{i+1} \notin C_k \wedge r = k$  then
         $f_k(C_k, CT_k, r, i + 1) \leftarrow f_k(C_k, CT_k, r, i) + 1$ 
      else if  $a_{i+1} \in C_k \wedge r \neq k$  then
         $f_k(C_k, CT_k, r, i + 1) \leftarrow f_k(C_k, CT'_k, r, i) + 1$ 
         $CT'_k[S_{\text{free}}] \leftarrow (i + 1) + \theta_m$ 
         $CT'_k.S_{\text{free}} \leftarrow \{i | CT'_k[i]\} == \min(CT_k)$ 
      else if  $a_{i+1} \in C_k \in \text{repl}(C'_k, a_{i+1}) \wedge r == k$  then
         $f_k(C_k, CT_k, r, i + 1) \leftarrow f_k(C'_k, CT'_k, r, i) + 1$ 
      end if
    end for
  end for
  if  $a_{i+1} \in C_h \Rightarrow a_{i+1} \in C$  then
     $F(i + 1) \leftarrow \min_{CT'_h} f_h(C_h, CT'_h, 0, i) + \sum_{k=1 | k \neq h}^m f_k(C_k, CT_k, 0, i)$ 
  else if  $a_{i+1} \notin C$  then
     $F(i + 1) \leftarrow \min_{C'_k, CT'_k, r' | r' \neq 0} \left( \sum_{k=1}^m f_k(C'_k, CT'_k, r', i) \right)$ 
  end if
end for

```

Analysis

It should be clear that for each iteration of the innermost **for**-loop, the first two cases are $\mathcal{O}(1)$, since they both simply involve some evaluations and an assignment. Similarly, the third and fourth cases involve constant-time evaluations and assignments. These two cases also involve minimizations, but both are on the

order of the number of DMA “slots” (j). This inner `for`-loop is evaluated for each possible combination of cache configurations for a single memory chip. Since there are b distinct disk blocks, and each memory chip can hold d blocks, this gives a worst-case total of $b \times d \times j$ minimizations.

Of the next two cases, the second is more complex. To see this, note that both involve summations of f_k over all memory chips. The first of the two (from equation 5 above), though, involves a minimum which evaluates immediately. The minimum CT'_h is simply $CT_h[CT_h.S_{\text{free}}]$. As noted, the other case also has a summation, but its minimization is more complex. Specifically, the algorithm must look at all cache configurations and choices for r (which memory chip will be have a page replacement). r' can take on m values, i.e. the index of any memory chip. Each chip has b possible configurations after a replacement, so there are a total of $m \times b$ different cache configurations. Additionally, CT'_k can take on any of j different values, depending on which DMA slot is used in the memory access. Hence, $f_k(C'_k, CT'_k, r', i)$ can have a total of $m \times b \times j$ values. This summation ranges from 1 to m , so this portion of the algorithm runs in $\mathcal{O}(m^2 \times mbj) = \mathcal{O}(m^3bj)$.

Finally, the outer `for`-loop runs from n times, so the overall complexity of this algorithm is $\mathcal{O}(m^3bjn)$.