OPTIMAL PAGING

Burt Rosenberg
University of Miami

Introduction and definitions

Page replacement strategies are responsible for the movement of pages between disk and memory. Let $\mathcal{P}$ be the set of *pages* in the virtual memory. They represent fixed sized blocks of contiguous addresses in the virtual address space. As the program runs, it will call upon addresses in its virtual memory. The computer will keep a sequence of working sets in memory in order to match the requirements of the program. That is, so that the page containing the address the program requires is in the working set. If the required page is not in the working set, a *page fault* occurs and computation is halted until the page is brought into the working set. Typically, this means some other page must leave the working set.

We model this with two sequences. The *page sequence* is a finite string,

$$P = p_1, p_2, \ldots, p_n, \text{ where for all } i,\ p_i \in \mathcal{P}.$$

As the program runs, it references virtual addresses according to the page sequence. The *paging strategy* is a finite string of working sets,

$$W = W_1, W_2, \ldots, W_n, \text{ where for all } i,\ W_i \subset \mathcal{P},$$

such that $|W_i| \leq k$, because core memory is limited, and W_i and W_{i+1} differ by "very little." If a page p appears in W_i but not in W_{i+1}, page p is said to have been *paged out at time* i. If a page p appears in W_{i+1} but not in W_i, page p is said to have been *paged in at time* i. In each step of the computation we will allow one page in and one page out. The *paging cost* is the number of page ins in the paging strategy.

REMARK: We are counting paging cost as if page outs are free. In reality, page outs are free if the page has not be modified, else they cost the same as page ins. It is interesting to consider whether counting this cost gives different conclusions.

It has been claimed that an optimal paging strategy is to page in only at the moment a page is actually needed (to wait for a page fault) and to page out the page whose first reuse is the farthest in the future, [1, 3, 5, 7]. The first rule is called *demand paging*. In this paper, we give the name *farthest out* to the second rule.

The supposition of demand paging means that the sequence of working sets begins with $W_1 = \{ p_1 \}$. At first, only page-in's occur and the working sets grow in size until the reach their maximum allowed size k. After which, either $W_{i-1} = W_i$ or $p_i \notin W_{i-1}$ in which case W_i differs from W_{i-1} in this manner: p_i is paged-in at time i and for some $p \in W_{i-1}$, p is paged-out.

The first reuse of a page $p \in W_i$, denoted $\rho(p, i)$, is defined as,

$$\rho(p, i) = \min \{ j > i \mid p_j = p \}.$$

If the set is empty, we will let $\rho(p, i) = \infty$. Formally stated, the rule of farthest out states that the page to throw out of W_i is the one among all $p \in W_i$ maximizing $\rho(p, i)$.

Farthest out is best

Assume we are given a demand paged paging strategy W for a page sequence P. If W is not farthest out, let i by the smallest integer such that W_i breaks the farthest out rule. That is, $a, b \in W_{i-1}$, and $\rho(b, i-1) > \rho(a, i-1)$, however a is paged out at time i.

We modify paging strategy W by replacing the page out of a by that of b, transforming W it into a strategy W'. That is, for all $j < i$, let $W'_j = W_j$, and,

$$W'_i = (W_i \setminus \{ b \}) \cup \{ a \}.$$

We next give directions for the construction of W'_j when $j > i$.

The idea is that the sequence W' continues by imitating W except in situations where this is impossible. There are exactly two such situations. The first is if W requires that b be paged out, the second is if W requires that a be paged in. It might happen that W simultaneously requires that b be paged out and a be paged in. As the reader shall see, once page b is paged out, the two strategies W and W' will converge. The role of page a, however, will change hands during the transformation. We say that a is the *excess page* and denote it by x. Here is a summary of the transformation:

- If W_j brings in $p \neq x$ and throws out b, then W'_j brings in p and throws out x:

$$W_{j-1} \xrightarrow[\searrow b]{p \searrow} W_j \implies W'_{j-1} \xrightarrow[\searrow x]{p \searrow} W'_j$$

- If W_j brings in x and throws out b the W'_j does nothing:

$$W_{j-1} \xrightarrow[\searrow b]{x \searrow} W_j \implies W'_{j-1} \longrightarrow W'_j$$

- If W_j brings in x and throws out $y \neq b$, in order that W' be demand paged, we are required to do nothing:

$$W_{j-1} \xrightarrow[\searrow y]{x \searrow} W_j \implies W'_{j-1} \longrightarrow W'_j$$

  This leaves us with:

$$W'_j = (W_j \setminus \{ b \}) \cup \{ y \}.$$

  Hence y is the new excess page. We set x equal to y.

The first two cases are terminal in that once invoked, $W_j = W'_j$ and so we complete the construction of W' by setting $W_k = W'_k$ for all $k > j$. The third case saves us a page-in but is not terminal. Eventually, either a terminal case will arise or we will hit the page b in the page sequence. At that point we cash in the saved page-in by throwing out the excess page and paging in b.

$$W_{j-1} \longrightarrow W_j \implies W'_{j-1} \xrightarrow[\searrow x]{b \searrow} W'_j$$

This sets $W_j = W'_j$ and hence the construction of W' is complete.

The strategy W' has no more paging cost than W. It differs from W in that the first exception to the farthest out rule occurs deeper into the page sequence. We repeat the transformation to form W'', W''', and so on. Since

the page sequence is finite, this cannot go on forever, and must stop when there are no more exceptions to the farthest out rule on which to apply the transformation. At this point we have a demand paged W^* which obeys the farthest out paging strategy. Since the cost of W^* is not more than that of W, we have shown that no paging strategy is better than farthest out.

REMARK: Note that the *number* of exceptions to the farthest out rule might remain the same after the transformation. For this reason we resort to the idea of always "pushing the exception to the right."

Conclusions

We looked at the classic problem of optimal paging and proved that under the demand paging strategy, farthest out is optimal. Most important next question is to show that demand paging is optimal. For this, it is likely that our model should be broadened. We should not require that every page out accompany a page in, only that the total number of page ins never exceed by a constant amount the number of page outs. This is also the most likely set-up in which to consider the problem of costly page-outs. A modified page must be written to secondary memory whereas a "clean" page does not. This considered, farthest out is no longer optimal. Could it be that the following strategy is optimal:

> If page p is the farthest out, and q is the next farthest out, page out p if p is clean or q is modified. Otherwise page out q.

What does this mean for actually algorithms in light of the fact that the future is not known. An interesting theorem presented in [5] is that the optimal algorithm gives the same paging cost when run either forward or backward over the paging sequence. Therefore, although we can't know what to do next, we can know how well or how poorly we are doing it.

The field of paging is quite active with many new papers, [2, 4, 6, 8]. A new version of the problem deals file servers. Workstations request files from the servers. The servers might move the files among themselves to that files tend to be near the users.

References

[1] L. A. Belady, "A study of replacement algorithms for a virtual-storage computer," IBM Systems Journal, vol. 5, No. 2, pp. 78–101, 1966.

[2] A. Borodin, S. Irani, P. Raghavan, and B. Schieber, "Competitive paging with locality of reference," In the Proceedings of the 23ird Annual ACM Symposium of Theory of Computing, 1991. Pp. 249–259.

[3] P. J. Denning, "Virtual Memory," Computing Surveys, vol. 2, pp. 153–189, Sept. 1970.

[4] A. R. Karlin, M. S. Manasse, L. Rudolph, and D. D. Sleator, "Competitive snoopy caching," Algorithmica, Vol. 3, No. 1, 1988. Pp. 70–119.

[5] R. L. Mattson, J. Gecsei, D. R. Slutz, and I. L. Traiger, "Evaluation techniques for storage hierarchies," IBM Systems Journal, vol. 9, no. 2, pp. 78–117, 1970.

[6] D. D. Sleator, and R. E. Tarjan, "Amortized efficiency of list update and paging rules," Communications of the ACM, Vol. 28, Feb. 1985. Pp. 202–208.

[7] Andrew S. Tanenbaum, *Modern Operating Systems,* Prentice-Hall, 1992.

[8] N. Young, "Competitive paging as cache-size varies," In the Proceedings of the Second ACM-SIAM Symposium on Discrete Algorithms, 1991. Pp. 241–250.