CS294-1 On-line Computation and Network Algorithms Spring 1997

Lecture 3: January 28

Lecturer: Yair Bartal *Scribe: Brent Chun*

3.1 Paging

This lecture presented the problem of paging – or more generally that of cache replacement policies – as an example of where on-line algorithms can be used. We start with a definition of the problem and describe and analyze an optimal off-line algorithm. We then defined a class of on-line algorithms known as marking algorithms and showed that they achieve the best possible competitive ratio of k.

3.1.1 The Problem

The paging problem is defined as follows. There are two levels of memory: a fast memory M_1 consisting of k pages and a slow memory M_2 consisting of n pages ($k < n$). Pages residing in M_1 are a strict subset of the pages in M_2, all of which are accessible but only through M_1. If an access is made to a page contained in M_1, this occurs at zero cost. If an access is made to a page currently not in M_1, the desired page must first be brought in from M_2 at a cost of 1 before it can be accessed. This event is called a page fault. If M_1 is already full when a page fault occurs, some other page in M_1 must be evicted in order to make room in M_1. The problem we are trying to solve is how to choose a page to evict each time a page fault occurs in a way that minimizes the total number of page faults over time.

3.1.2 Paging Algorithms

3.1.2.1 An Optimal Off-line Algorithm

Algorithm LFD (Longest-Forward-Distance) is an optimal page replacement strategy. On each page fault, replace the page in M_1 that will be requested farthest out in the future.

Belady [Belady66] proved the following classic result.

Theorem 3.1 *LFD is an optimal page replacement policy.*

Lemma 3.2 *1. For any paging algorithm A and for all i, there exists an algorithm A' such that:*

- *Until the $(i-1)$st page fault, A' behaves exactly as A.*

- *Upon the ith page fault, A' evicts the page that is requested farthest in the future.*
- $\forall \sigma, \mathrm{A}'(\sigma) \leq \mathrm{A}(\sigma)$

Proof: Consider the cache configurations of both algorithms after the ith page fault. These would differ in exactly one page. Let the page that A' evicted (the one to be requested farthest) be p. Therefore there exists a set of $k-1$ pages X such that the contents of the cache of algorithm A is $X+p$ and that of A' is $X+q$. If $q=p$ then the algorithms are identical. Assume $q \neq p$. The two algorithms act the same until either p or q are requested. If at some point A evicts p then A' evict q and from there on they are identical. Otherwise by definition of p q will be requested before p. When it is requested, A replaces an element in its cache, say r by q while A' hits in its cache. Thus A incurrs a cost higher by 1 than A'. If $r=p$ then A and A' are identical from this point on, otherwise the cache of A is now $\tilde{X}+p$ and that of A' is $\tilde{X}+r$ for some set of $k-1$ page $\tilde{X}$. This continues until eventually p is requested. At that time A' faults and brings p in its cache in place of r. The cost for this has covered by the former fault that A icurred, and from this point on the algorithms are identical. ∎

3.1.2.2 On-line Algorithms

Competitive Analysis of paging algorithms was initiated by Sleator and Tarjan [ST85]. We first state a lower bound for deterministic on-line algorithms.

Theorem 3.3 *For any deterministic on-line algorithm A, the competitive ratio of* $A \geq k$.

Proof: We restrict the request sequence to a set of $k+1$ distict pages. It is easy to see that given any such sequence once LFD faults it evicts the page that will be requested farthest in the future and thus will not fault again before at least k more requests will be issued.

Lemma 3.4 $\forall \sigma$, *over* $k+1$ *pages*, $\mathrm{LFD}(\sigma) \leq \frac{|\sigma|}{k}$.

A lower bound follows since for any deterministic on-line algorithm A an adversary can construct a sequence where the next request is always the page that is not contained in the cache of A. Hence there exists a sequence σ such that $\mathrm{A}(\sigma) = |\sigma|$. ∎

A number of on-line algorithms for the page replacment problem are presented and can be divided into two categories: 1) those that are competitive (i.e. those which can be shown to do no worse that k times any other algorithm) and 2) those that are not competitive.

Competitive algorithms include "old-time favorites" such as LRU (Least-Recently-Used page is evicted) and the CLOCK algorithm (a variant of LRU that is easier to implement in practice). Non-competitive on-line algorithms included LFU (Least-Frequently-Used) and LIFO (Last-In-First-Out).

Other competitive algorithms: FIFO (First-In-First-Out) and FWF (Flush-When-Full).

3.1.3 Marking Algorithms

Marking algorithms are a class of on-line algorithms that can be shown to be k-competitive for the page replacment problem. In fact all the above mentioned competitive algorithms are either marking algorithms or behave as marking algorithms on sequences of requests where each requests is a fault for the algorithm (we may assume that any worst case sequence has this form).

We first define the following marking process with respect to a particualr request sequence.

- 1. Unmark all pages in the fast memory M_1.
- 2. Partition the sequence of accesses σ into phases, where each phase includes accesses to k *distinct* pages, and ends just before the $k+1$st distinct page is requested. Each new page that is accessed is marked.
- 3. At the end of a phase, unmark all pages in M_1.

KEY PROPERTY: A *Marking Algorithm* is one that never evicts a page which is already marked.

Theorem 3.5 *Any Marking algorithm is k-competitive.*

Proof: Obviously the cost incurred by a Marking algorithms is at most k per phase. We will show that for any adversary we can associate a cost of 1 per phase, which implies the theorem. Consider the i'th phase and let p be the first request in the phase. After the request the adversary must contain p in the cache. Now, upto and including the first request in the next phase there are at least k distinct pages requested and all distinct from p. Thus the adversary must have a page fault for at least one of these pages. ■

References

[Belady66] A Study of Replacement Algorithms for Virtual Storage Computers. In *IBM Systems Journal*, 5, pages 78-101, 1966.

[ST85] D.D. Sleator and R.E. Tarjan. Amortized Efficiency of List Update and Paging Rules. In *Communications of the ACM*, 28(2) pages 202-208, 1985.