CS787: Advanced Algorithms	**Scribe:** Jennifer Cao, Sihan Liu
Lecture 14: Paging	**Date:** October 22, 2019

Last time we introduced online algorithm by conducting competitive analysis for two simple examples. For the *ski-rental problem*, a deterministic algorithm has a competitive ratio of 2, while it can be improved by a roughly $\frac{e}{e-1}$-competitive randomized algorithm. Another one is *load balancing problem*, and we gave a 2-competitive deterministic algorithm.

14.1 Paging Problem

In the computer system, there is a small fast memory(the "cache") and a big slow memory(the "disk")[1]. Each time when a program sends read or write request to data on "pages" in disk, it can be accessed directly when the page is also in the cache, which is a "cache hit". Otherwise we have a "cache miss", then we must fetch the upcoming page from the disk and put it in the cache, and a previous page in the cache needs to be evicted if the cache is already full. A "cache miss" is also known as "page fault". It's obvious that cache misses slow down a program because a page needs to be feched from a large slow memory. Hence the goal of paging is to minimize the number of cache misses.

The *online paging problem* is described as following:

Assume the disk has capacity of N pages, and the cache has capacity of size $k < N$ pages at a time. Consider a *request sequence* of pages $a_1, a_2, ..., a_N$, and we do not know what page it is at the next time step, thus we can apply an online algorithm in this case. If the page is already in the cache, zero cost incurred. Otherwise, we need to bring a_t into the cache, and evict one page out of the cache if it is full. In this case, one unit of cost is occurred.

14.2 Deterministic Algorithms

14.2.1 Common Deterministic Algorithms

There are many common deterministic strategies for paging problem.

- **LRU(Least Recently Used)** Evict a page in the cache that has not been used for the longest time.
- **LFU(Least Frequently Used)** Evict a page in the cache that has been used least frequently.
- **FIFO(Fist In First Out)** Treat the cache as a queue. Each time when the cache is full, evict the head of the queue and enqueue the new page.

For example, consider a sequence $a, b, b, b, c, c, c, d, d, a, e, f$, and capacity of the cache is $k = 4$. See Figure 1, at first a, b, c, d are put into the cache successfully, as there are only four distinct pages before e appears. To bring page e into the cache, one prevous page must be evicted. At this time step, FIFO evicts a as it is the head of current queue. LRU evicts b because b is the least recently used. While LFU kicks out a, which is used only twice and all other pages have been used three times.

a			
a	**b**		
a	**b**		
a	**b**		
a	*b*	**c**	
a	*b*	**c**	
a	*b*	**c**	
a	*b*	*c*	**d**
a	*b*	*c*	**d**
a	*b*	*c*	*d*

$\downarrow$

FIFO

e	b	c	d
e	**f**	c	d

LRU

a	**e**	c	d
a	e	**f**	d

LFU

e	b	c	d
f	b	c	d

Figure 1: Comparison of FIFO, LRU, LFU strategies.

14.2.2 Competitive Analysis

Among above three strategies, LRU and FIFO are k-competitive, and LFU has an unbounded competitive ratio.

Before we start proving the competitive ratio of LRU and FIFO, let's first provide an example to see why LFU is not competitive. Assume a cache has contained page 1 to page $k-1$, and all these pages has been requested for m times. Then we only request page k and page $k+1$ alternatively for m times. In this case, LFU encounters $2m$ cache misses, because the strategy is kicking out page $k+1$ when requesting k, and kicking out page k when requesting page $k+1$, as the frequency of both pages is less than m. Considering the optimum algorithm, it only has 1 cache miss, which happens when requesting page $k+1$.

Lemma 14.2.1 *LRU and FIFO are k-competitive[2].*

First we need to define the denotion *phase*. Assume the request sequence is divided into phases $p_1, p_2, \ldots$, each *phase* is a partition that consists of k-faults. Formally, phase 1 begins at page 1, and phase i begins at the first time we see k faults after phase $i-1$ has begun.

Claim 14.2.2 *For FIFO, phase p_i contains k distinct cache faults[3].*

Proof: Let a be the first page that causes fault in p_i. Because the cache has capacity of size k, by FIFO we know that there are $k-1$ pages needs to be evicted before page a. Thus, to encounter a second cache miss caused by page a, a needs to stay in the cache after $k-1$ distinct faults has happened. ■

Proof of Lemma 14.2.1: We need to prove that the optimum algorithm must make at least 1 mistake in an arbitrary phase.

For FIFO strategy, as Claim 14.2.2 holds, OPT must have at least 1 cache miss. That is because the cache cannot load both the last page in p_{i-1} and all k distinct pages in p_i.

The proof of LRU is similar, but the claim above does not strictly hold in this case. The proof can be divided into two cases. When there are k distinct cache misses during phase p_i, the proof is the same as the case in FIFO.

If there is some page a is missed twice during phase p_i, then a must be put in and kicked out once before. At the first time when a is put in, page a is the most recently used page. Hence, when kicking out page a, there are $k-1$ pages in the cache has been requested after a was put in. At the time when a is evicted, there is another distinct page that is different from previous $k-1$ requests and page a, so there are $k+1$ distinct pages has been requested in phase p_i. Thus, every time there are $k+1$ distinct pages, OPT has at least one mistake. ■

Lemma 14.2.3 *No deterministic algorithm for paging is better than k-competitive, where k is the size of the cache[4].*

Proof: Fix a deterministic online paging algorithm ALG and $k+1$ pages, and assume that there are k pages in the cache. As we already know the replacement policy in ALG, an adversary can be constructed to request a page not in the cache at each step. Thus, ALG fails at all T steps, where T is the length of the adversary requests.

Consider an offline greedy optimal strategy OPT. Suppose OPT encounters a mistake at step t, it evicts page that is not requested in the following $k-1$ steps. Thus, OPT has at most one cache miss in k consecutive requests. For a request sequence of length T, OPT $\leq \lceil \frac{T}{k} \rceil$. Therefore, ALG $= k \cdot$ OPT $- \Theta(1)$. ■

Lemma 14.2.3 implies that FIFO and LRU archieve the best competitive ratio for deterministic paging algorithm. However, randomized online algorithm can considerably beat k-competitive deterministic paging algorithms, and can give a tight competitive ratio of $\log k$.

14.3 Randomized Algorithms

To illustrate the benefit of randomization, take the marking algorithm as an example.

14.3.1 The Marking Algorithm

Take every k distinct pages as a phase. the online marking algorithm deals with a request sequence in phases. At the beginning of a phase, all pages are initialized as unmarked. Whenever a page is requested:

step 1. Mark the page.

step 2. If the page is not in the cache:

- Randomly kick out an unmarked page from the cache.
- If all pages in the phase has been marked, clear all marks.

14.3.2 Competitive Analysis

Lemma 14.3.1 ([5] Theorem 1) *The marking algorithm is* $\log k$*-competitive.*

Proof: Consider an arbitrary phase p_i. Let set S_i be the group of pages in the cache before phase i.

Then in phase i, there are two types of pages:

- If page $a \in S_i$, call the page *clean*;
- Otherwise, i.e., $a \notin S_i$, call the page *stale*.

ALG. First consider when we are using the marking algorithm, the expected number of cache misses during phase i. Let c_i be the total number of clean pages in phase i, and suppose at time step t, we see c clean pages and s stale pages. The expected cost of the request is $\frac{c}{k-s} \leq \frac{c_i}{k-s}$, as there are c clean pages(not in the cache) distributed uniformly among $k-s$ remaining pages, and c is bounded by c_i. Actually, the inequality shows that the sequence with the highest expected cost is the one which first request all the c_i clean pages and then the stale pages.

As the number of stale pages in phase p_i is $k - c_i$, The probability that a new stale page is missed at step $c_i + 1$ is

$$\mathbf{Pr}[\text{a new stale page is missed at step } c_i + 1] = \frac{\binom{k-1}{k-c_i}}{\binom{k}{k-c_i}} = \frac{c_i}{k}$$

The probability a new stale page is missed at step $c_i + 2$ is

$$\mathbf{Pr}[\text{a new stale page is missed at step } c_i + 2] = \frac{\binom{k-2}{k-c_i-1}}{\binom{k-1}{k-c_i-1}} = \frac{c_i}{k-1}$$

By induction, the expected total cost $C(ALG)$ is the sum of expected cache misses of stale and clean pages:

$$C(ALG) = \frac{c_i}{k} + \frac{c_i}{k-1} + \frac{c_i}{k-2} + \cdots + \frac{c_i}{c_i+1} + c_i \leq c_i H_k = \Theta(c_i \log k)$$

Where $H_k = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{k} \approx \log k$.

OPT. To inspect the amortized number of faults made by OPT within a phase, let Φ_i denote the number of different pages OPT has before phase p_i, i.e., $\Phi_i = |S_i^{OPT} - S_i|$.

First, we claim that the total cost $C(OPT)$ during phase p_i is at least $c_i - \Phi_i$, where c_i is the number of clean pages in phase p_i. That is because among all the clean pages, at most Φ_i

has already been in the cache. Moreover, we know that there are at least Φ_{i+1} pages not in the cache at phase $i+1$ by definition, thus $C(OPT) \geq \Phi_{i+1}$.

Thus, the total cost of OPT is

$$C(OPT) = \max\left(c_i - \Phi_i, \Phi_{i+1}\right) \geq \frac{c_i - \Phi_i + \Phi_{i+1}}{2}$$

When this is summed over all phases, the Φ_i and Φ_{i+1} telescope, and we get

$$C(OPT) \geq \frac{1}{2} \sum c_i$$

.

Thus, we can conclude that marking algorithm is $\log k$-competitive.

■

References

[1] T. Roughgarden, "Cs264: Beyond worst-case analysis lecture# 1: Three motivating examples," 2014.

[2] D. D. Sleator and R. E. Tarjan, "Amortized efficiency of list update and paging rules," *Communications of the ACM*, vol. 28, no. 2, pp. 202–208, 1985.

[3] D. Komm, *An introduction to online computation: determinism, randomization, advice.* Springer, 2016.

[4] S. Albers, *Competitive online algorithms.* Citeseer, 1996.

[5] A. Fiat, R. M. Karp, M. Luby, L. A. McGeoch, D. D. Sleator, and N. E. Young, "Competitive paging algorithms," *Journal of Algorithms*, vol. 12, no. 4, pp. 685–699, 1991.