UNIVERSITATEA DIN BUCUREȘTI

FACULTATEA DE MATEMATICĂ ȘI INFORMATICĂ

Paging with Dynamic Memory Capacity

– Summary –

Student
Radu Ștefan-Octavian

Scientific coordinator
Prof. Dr. Alexandru Popa

Bucharest, December 2022

Peserico tackles in this work the **Paging Problem**, with emphasis on a system where the memory capacity can change over time. He analyses the performance of various classic algorithms using **competitive analysis**. Considering the fact that systems where the memory capacity fluctuates are seen increasingly often (cloud-computing, memory-optimized and multicore processors), the questions posed in this piece of work are deemed relevant.

This work shows that an algorithm which performs **optimally** (within a factor of 2) in the fixed-memory setting, can perform **suboptimally** by an arbitrarily large factor if the memory fluctuates. Despite this, there are *classic* algorithms which perform surprisingly well when memory capacity changes, despite not explicitly taking the fluctuations into account. The conclusion of the analysis is that knowledge of the future is not a requirement for achieving good performance for the *Paging Problem* [3].

The Paging Problem [4]

Memory in current computing systems is usually organized in a hierarchy. Starting from the fastest, but also the most limited in capacity memory (cpu registers and cpu cache), going down to RAM and then even further to mass storage solutions (SSD, HDD, TAPE). As technology progressed the gap between the layers grew larger and larger. The paging problem is essential in maintaining a fast, efficient information flow from one layer to another. As a consequence, the performance of the whole system is dependent on the implemented solution for the paging problem.

The most popular method of studying the paging problem focuses on a two layer system: a layer with a smaller capacity of k pages and a second layer which higher, usually infinite. The pages of the second layer can only be accessed only if they are first copied onto the first layer. To do that, some pages from the first layer have to be "kicked out" (*evicted* from memory), to make room for the ones which are going to be copied. This event is called a *page fault.* The goal of the algorithm solving the *Paging Problem* is to minimize the amount of page faults that occur.

LFD (Longest Forward Distance) is been known to optimal for a long time now [1]. This is an *offline* algorithm (meaning that is uses information from the future to compute the solution), so it's useful more like a point of reference when computing the *efficiency* of other algorithms. The algorithms that have true practical applications are *online* algorithms. One very popular method of evaluating their performance, that is also used in this work, is the *competitive analysis* framework.

Competitive analysis [5]

Competitive analysis [2] is a method of evaluating online algorithms by comparing them with an *optimal* offline algorithm for the same problem. An algorithm is said to be *competitive* if the ratio between its performance and the performance of the offline algorithm is bounded. This ratio is called the *competitive ratio* of the algorithm in the *competitive analysis* framework.

When looking at paging algorithms, an (h, k)-*competitive ratio* of ρ means that the online algorithm analysed, which runs on a system with k memory capacity, will perform at most ρ times the number of *faults* (plus a number d of faults unrelated to the input data), as an optimal offline algorithm, with memory capacity $h \leq k$. In this case, the ratio $\frac{h}{k}$ is called the *resouce augumentation.*

In the classic paging problem it has been determined that the optimal (h, k)-*competitive ratio* for deterministic algorithms is $\frac{k}{k-h+1}$. LRU (the least recently used page is evicted first), FWF (evict everything when memory is full), FIFO (first loaded page is evicted first), CLOCK (give each page a second chance before evicting) are deterministic algorithms which have been proven to be optimal, thus having a competitive ratio of $\frac{k}{k-h+1}$. RAND (evicting random pages) is also optimal. All of this results consider an *adapting adversary* (which is considered throughout the paper), that can adapt the input based on the choices of the paging algorithm. Using the competitive framework we can grasp the time and space overhead of the offline algorithms compared to the optimal. For example, by having $h = \frac{k}{2}$, we get a competitive ratio of $\frac{k}{k-\frac{k}{2}+1} < 2$ which tells us that the optimal online algorithms will not perform worse than the optimal offline algorithm would on a system with **half the memory** and **twice the access cost**.

The effect of small memory fluctuations [6]

To prove that the *Paging Problem* in a context with dynamic memory capacity if relevant, the hybrid **LFRU** (Least Frequently / Recently Used) is analysed. It starts out as **LRU** and switches to **LFU** after a palindromic sequence of pages causes more page faults in the second half than in the first half. If the opposite is true, it switches back to *LRU*.

In practice *LRU* performs well on inputs where data is reused frequently in a certain span of time (*temporal locality*), but performs bad on streams of data which have no *temporal locality* and for which every new access is a page fault. For this exact reason it is a good idea to use hybrid algorithms which account for both situations to counteract the shortcomings of using only **LRU**.

LFRU is a good example of such a hybrid algorithm, even though it may seems artificial at a first glance. In this paper it's been proven that **LFRU** has an optimal (h, k)-*competitive ratio* of $\frac{k}{k-h+1}$ when analysed in the fixed memory scenario. The problem appears when analysing the algorithm in the dynamic memory scenario. It's been proven that **LFRU** is not (h, k)-*competitive* for any $h \geq 3$ and any arbitrarily large value for k in a scenario where memory fluctuates even with only one page. That means that regardless or memory capacity, the algorithm will have a fault rate arbitrarily larger than the optimal offline algorithm would, even on a system with only 3 available pages in memory.

To understand this at an intuitive level think of the following request sequence, where p_i^l signifies l subsequent requests for p_i and $3m$ is the initial memory capacity:

$$\pi =< p_1, p_2, p_3^l, \cdots, p_{3m}^l, p_1, p_2, p_3, \cdots, p_{3m}, p_{3m}, \cdots, p_2, p_1 >$$

Now consider that for the last $3m$ requests in π, memory capacity drops by one and is thus equal to $3m - 1$. As a result of the capacity change, p_1 will be evicted. In order to load p_1 at the end of sequencd π, p_2 will be evicted. Since the last $6m$ requests of π form a palindrome, and a *page fault* took place on the last request, the condition for switching to **LFU** is met. If $\pi' =< (p_2, p_1)^{l-1} >$ is the following request sequence, the algorithm will continue in the **LFU** behaviour. At first p_2 is not loaded so there will be a fault after which p_1 is evicted. Next will follow p_1 which was just evicted and another fault will occur after which p_2 will be evicted, etc. For every following request p_i will evict p_{3-i} because p_1 and p_2 have frequencies smaller with $l - 1$ compared to the rest of p_i with $i \geq 3$. Thus, for π' there will be $2(l - 1)$ faults. An optimal algorithm which starts with capacity 3 and drops to 2 would incur at most $9m$ faults for the requests in sequence π and no additional requests during π', keeping p_1 and p_2

in memory. Since l can be chosen arbitrarily large, **LFRU** can perform arbitrarily worse than the optimal algorithm. In conclusion, **LFRU** does not have a bound $(3, 3m)$-*competitive ratio*, regardless of how large m is chosen.

Addressing even adversarial fluctuations [7]

Looking at the previous result, it is surprising to find out that *classic* paging algorithms can perform very well in the *Dynamic Paging Problem* even though they don't explicitly take memory fluctuations into account. The first remarkable result is that **LFD** is optimal, in the sense that its usage results in the minimal number of faults, even in a dynamic memory context. This means that for any page fault it is always best to evict a *far* page. To aid the proof the following concepts are introduced: any page the accessed before another page already in memory is a *close* page, and otherwise it is a *far* page. The proof considers an algorithm ALG that evicts a *close* page at a point t in time. It is provable that by changing its behaviour to instead evict a *far* page, we can achieve the same number of page faults in the end.

Since we're mostly concerned with online algorithms in practice, these are considered next. It is proven that many already well-documented algorithms bear a dynamic *competitive ratio* almost as good as $\frac{k}{k-h+1}$ that is equal to:

$$\rho EL(h,k) = \max_{k' \leq k, k \in \mathrm{N}} \frac{k'}{k' - \lfloor h\frac{k'}{k} - \frac{h}{k} \rfloor}$$

Some algorithms that are proved in this piece of work to achieve the $\rho EL(h,k)$ dynamic *competitive ratio* are all the **marking algorithms** (which include **FWF**, **MARK**, **LRU**) and all the *dynamically conservative* algorithms. If we define w as the number of distinct pages in a generic subsequence of requests, then an algorithm is *dynamically conservative* if it never incurs more than w faults on a short subsequence of length w. This class is narrower than the one of *conservative algorithms*, but includes **LRU**, **CLOCK** and **FIFO**. ρEL is also achieved by **RAND**.

It has also been proven that ρEL is a lower bound for the (h,k)-*competitive ratio* of any dynamic online algorithm.

Conclusions [8]

In this paper it's been proven that an algorithm with good performance in a constant memory environment is not guaranteed to perform well in an environment where memory can fluctuate. Moreover, it's been proven that for fluctuations of only 1 page, the classic algorithms can perform arbitrarily worse than the optimal. Despite this, it's been shown that there are a number of *simple* and well-documented algorithms that perform remarkably well in the dynamic memory context. This is an interesting conclusion to arrive at because none of these algorithms were designed to take memory fluctuations into account. This shows that even though knowledge of future page requests can offer an advantage (such as in the case of **LFD**), knowledge of future memory fluctuations does not provide any kind of advantage. This conclusion can be used as an argument for decoupling resource allocation from capacity management in the design of memory architectures which would lead to a less complex product.

The *competitive analysis* framework fails to distinguish between the performance of **LRU**, **FIFO** and more naive algorithms such as **FWF** and **RAND** in the dynamic memory setting.

Peserico suggests that further work in this direction could tackle the problem of differentiating between them. He also suggests investigating a way of benchmarking such algorithms in a *black box* approach.

Finally, it is emphasised that the concept of *"dynamic resources"* does not only appear in the *Paging Problem* and that other areas of research (such as *online scheduling* and *call admission*) could benefit from similar approaches. It would also be interesting to identify classes of problems that share the property of having (almost) optimal solutions, that do not depend on knowledge of future resource fluctuations.

Bibliography

[1] Laszlo A. Belady. “A study of replacement algorithms for a virtual-storage computer.” In: *IBM Systems journal* 5.2 (1966), pp. 78–101.

[2] Anna R Karlin, Mark S Manasse, Larry Rudolph, and Daniel D Sleator. “Competitive snoopy caching.” In: *Algorithmica* 3.1 (1988), pp. 79–119.

[3] Enoch Peserico. “Paging with dynamic memory capacity.” In: *arXiv preprint arXiv:1304.6007* (2013).

[4] Enoch Peserico. “Paging with dynamic memory capacity.” In: *arXiv preprint arXiv:1304.6007* (2013), p. 3.

[5] Enoch Peserico. “Paging with dynamic memory capacity.” In: *arXiv preprint arXiv:1304.6007* (2013), pp. 3, 4.

[6] Enoch Peserico. “Paging with dynamic memory capacity.” In: *arXiv preprint arXiv:1304.6007* (2013), pp. 8–11.

[7] Enoch Peserico. “Paging with dynamic memory capacity.” In: *arXiv preprint arXiv:1304.6007* (2013), pp. 11–14.

[8] Enoch Peserico. “Paging with dynamic memory capacity.” In: *arXiv preprint arXiv:1304.6007* (2013), pp. 22–23.