Paging with Dynamic Memory Capacity

Enoch Peserico
Univ. Padova, Italy
enoch@dei.unipd.it

Abstract

We study a generalization of the classic paging problem that allows the amount of available memory to vary over time – capturing a fundamental property of many modern computing realities, from cloud computing to multi-core and energy-optimized processors.

It turns out that good performance in the "classic" case provides no performance guarantees when memory capacity fluctuates: roughly speaking, moving from static to dynamic capacity can mean the difference between optimality within a factor 2 in space and time, and suboptimality by an arbitrarily large factor. More precisely, adopting the competitive analysis framework, we show that some online paging algorithms, despite having an optimal $(h, k)-$competitive ratio when capacity remains constant, are not $(3, k)-$competitive for any arbitrarily large k in the presence of minimal capacity fluctuations.

In this light it is surprising that several classic paging algorithms perform remarkably well even if memory capacity changes adversarially – in fact, even without taking those changes into explicit account! In particular, we prove that LFD still achieves the minimum number of faults, and that several classic online algorithms such as LRU have a "dynamic" $(h, k)-$competitive ratio that is the best one can achieve without knowledge of future page requests, *even if one had perfect knowledge of future capacity fluctuations.* Thus, with careful management, knowing/predicting future memory resources appears far less crucial to performance than knowing/predicting future data accesses.

We characterize the optimal "dynamic" $(h, k)-$competitive ratio exactly, and show it has a somewhat complex expression that is almost but not quite equal to the "classic" ratio $\frac{k}{k-h+1}$, thus proving a strict if minuscule separation between online paging performance achievable in the presence or absence of capacity fluctuations.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases paging, cache, adaptive, elastic, online, competitive, virtual, energy

Digital Object Identifier 10.4230/LIPIcs.STACS.2019.56

Related Version Some of these results were presented at the poster session of SIGMetrics'13 [25]. An older version of this work can be found on arXiv [26].

Acknowledgements We are indebted to Federica Bogo and Michele Scquizzato for their relentless encouragement and a number of fruitful discussions on this topic; and to the anonymous referees of STACS 2019 for their insightful, in-depth review of our work.

1 Introduction

This work examines a generalization of the classic paging problem that allows the amount of available memory to vary over time. After briefly reviewing the paging problem (Subsection 1.1) this section motivates paging with dynamic capacity (Subsection 1.2) and provides an overview of our results and of the organization of the rest of the article (Subsection 1.3).

1.1 The paging problem

The memory/data storage system of modern computing devices is almost always organized as a hierarchy of several layers of progressively larger capacity but also higher access cost (in terms of both time and energy); efficiently orchestrating the flow of information across the memory hierarchy is crucial for performance. The most widely used model for studying this *paging* problem is that of a two-layer system: a smaller *memory* layer with a capacity of k *pages* (data blocks), and a larger layer of infinite capacity whose pages can only be accessed by first copying them into memory – an operation termed a *(page) fault.* Given any sequence of pages that must be accessed in order, an algorithm for the paging problem must choose which page(s) to "evict" from memory, whenever a new page must be copied into it, so as to minimize the total number of faults.

The simple algorithm LFD (Longest Forward Distance) that evicts the page accessed furthest in the future has long been known to be optimal [4]. However, paging is often studied as an *online* problem, i.e. an algorithm can decide evictions only on the basis of past requests. A popular framework for evaluating the performance of online paging algorithms is that of *competitive analysis* [20]. A paging algorithm is said to have an $(h, k)-$*competitive ratio* of (no more than) ρ if, for every request sequence, it incurs in expectation with a memory of capacity k at most ρ times as many faults as an optimal offline algorithm incurs with a memory of capacity $h \leq k$, plus a number of faults independent of the request sequence. The ratio $\frac{k}{h}$ is called the *resource augmentation.* Resource augmentation and competitive ratio capture, respectively, the space and access cost overheads incurred by an online algorithm.

Many simple, deterministic algorithms including LRU[1], FIFO[2], FWF[3] and CLOCK[4] have an (h, k)-competitive ratio of $\frac{k}{k-h+1}$ [30, 8]; and the same ratio holds for RAND[5] [29]. This ratio is optimal for deterministic algorithms, and even for randomized ones if page requests can depend on previous choices of the paging algorithm (the "adaptive adversary" model [29] which we adopt throughout the article[6]). Since $\frac{k}{k-k/2+1} < 2$, many simple online algorithms never fare worse than the optimal offline algorithm would on a memory system with half the capacity and twice the access cost. This justifies the use of competitive analysis for preliminary performance evaluation of paging algorithms. Its "worst-case" approach may be somewhat pessimistic, but it is not overly so for many popular online paging algorithms – for which it provides guarantees of performance within a factor 2 of the optimal *under any workload* (in terms of faults and required memory capacity). In contrast, the finer granularity evaluation provided by experimental benchmarking is inevitably tied to specific workloads.

[8] and [10] provide two excellent surveys of the many variants of competitive analysis for the paging problem: these include somehow limiting the choice of the adversarial request sequence [9, 13, 14, 21], amortizing the performance evaluation over a spectrum of sequences [1, 2, 5] or of memory capacities [32], considering pages of different size and access cost [19, 32], and accounting for the non-zero cost of non-fault requests [31].

[1] Least Recently Used – evict the least recently accessed page.
[2] First In First Out – evict the page brought least recently into memory.
[3] Flush When Full – evict all pages whenever memory is full and space is needed.
[4] Mark any page accessed; to evict a page, cycle through pages, unmarking those found marked and evicting the first found unmarked.
[5] Evict a page chosen uniformly at random.
[6] More precisely, in the *online* adaptive adversary model, the choices of the reference offline algorithm can depend only on the *past* random choices of the online algorithm; in the *offline* adaptive adversary model they can depend on the random choices of the online algorithm over the entire request sequence. The bounds above – and in fact all the bounds in this article – hold for both models, with one exception: the upper bound on the competitive ratio of RAND above, and the corresponding upper bound we provide for RAND in Theorem 10, only hold in the adaptive online model.

1.2 Memory capacity often varies over time

In all variants of the paging problem, until a few years ago memory capacity was always fixed throughout the request sequence. This no longer reflects many computing realities. In a cloud computing environment, the amount of physical memory available to an individual virtual machine varies considerably over time depending on the virtual machine's load and on the number, load and relative class of service of other virtual machines hosted on the same hardware. Even on a simple PC, most modern operating systems have and use the option of declaring some critical virtual pages temporarily "unswappable", pinning them in main memory and thus reducing the amount of main memory available to user processes. Memory fluctuations also take place when considering the cache-RAM interface – in which case memory represents cache memory and pages represent cache lines. In many multi-core processor designs cache capacity is partitioned *dynamically* between different cores [28]. And low-power chip designs can often dynamically disable underutilized portions of the cache to save energy [18], again resulting in a capacity that can vary over time.

Note that, although there exists a large body of work on servicing the same request sequence with a policy that is simultaneously "good" on memories of different [32] and perhaps unknown [15] *but unchanging* capacity, allowing capacity to vary dynamically during the course of the computation is an entirely different problem; as we shall see, a solution to the former does not guarantee even an approximate solution to the latter.

A seminal work considering memory fluctuations in a hierarchical memory system is [3], that presents efficient algorithms for problems such as sorting or FFT assuming each algorithm can explicitly choose which pages to keep in memory. A slightly different approach is that taken by the recent literature on *cache-adaptive* algorithms for sorting, FFT and many other problems [6, 7], where management of the memory is devolved to an "automatic optimal" page replacement policy; the choice is justified by showing that a LRU policy is $O(1)$-competitive with $O(1)$ resource augmentation.

Our work instead focuses on the page replacement policy itself, assuming as in classic paging that the request sequence is provided by an adversary. The adversary also controls how memory capacity fluctuates between 1 and k pages; these fluctuations may be either known beforehand to the paging algorithm (an "offline" problem), or unknown until they take place (an "online" problem). We stress that, when comparing different paging algorithms, we take a slightly different approach from that of [7]: we assume identical/proportional memory capacity when servicing the same page request, whereas [7] assumes identical/proportional capacity after an identical/proportional number of faults. As [7] itself notes when referencing this work (the preliminary version of which predates [7]), both models are natural and reflect a different emphasis on what may initiate capacity changes.

Another related, but fundamentally different, paradigm is that of *RAM rental* [11, 23], where memory capacity fluctuates *under control of the paging algorithm* and the goal is to minimize a linear combination of average capacity and fault rate over time. In practice there are very strong constraints on the set of admissible capacity values, on how they can change over time, and on their relative costs (which may themselves fluctuate). Also, a number of architectural approaches (e.g. [12]) decouple the portion of the system responsible for page replacement from that responsible for capacity allocation. Then, assuming as we do that capacity fluctuations are not controlled by the paging algorithm (in fact, that they may be unknown beforehand and even chosen adversarially) leads to a more robust evaluation of page replacement policies.

1.3 Our results

The rest of this article is organized as follows. Section 2 introduces some formalism and terminology. In particular, it extends the notion of “online vs. offline” problem to encompass the extra dimension of future memory capacity, and it extends the notion of $(h,k)-$resource augmentation to the dynamic capacity scenario (in a nutshell, restricting the offline algorithm to at most a fraction $\frac{h}{k}$ of the online algorithm’s *current* memory capacity).

Section 3 shows the existence of online paging algorithms that have an (optimal) (h,k)-competitive ratio of $\frac{k}{k-h+1}$ in the “classic” paging model, and yet are no longer $(3,k)-$competitive *for any arbitrarily large* k if their memory capacity is subject to single page fluctuations. This very negative result provides strong justification for our inquiry, as one cannot infer performance in the presence of (even minimal) memory fluctuations from performance in their absence.

In this light, it is surprising that many well-known algorithms perform remarkably well in the presence of memory fluctuations *even if those fluctuations are chosen adversarially.* In Section 4 we show that the classic LFD algorithm remains strictly optimal for all possible memory capacity fluctuations even though it does not explicitly take those fluctuations into account (i.e. it is an online algorithm in terms of memory fluctuations). We also show that in the dynamic capacity framework every online algorithm that is either marking [14] like LRU, FWF or MARK[7], or *dynamically conservative* (a simple refinement of the notion of “conservative algorithm”[33]), like LRU, CLOCK or FIFO, has an $(h,k)-$competitive ratio no larger than $\rho_{EL}(h,k) = \max_{k' \leq k, k' \in \mathcal{N}} \frac{k'}{k' - \lfloor h\frac{k'}{k} - \frac{h}{k} \rfloor}$[8]. Exactly the same bound holds for RAND (against an online adaptive adversary).

Section 5 analyses $\rho_{EL}(h,k)$. We show that it is a lower bound to the $(h,k)-$competitive ratio achievable by any online paging algorithm in the presence of memory fluctuations, proving the optimality of marking and dynamically conservative algorithms. We also show that $\rho_{EL}(h,k)$ almost, but not quite, matches the “classic” bound of $\frac{k}{k-h+1}$ on the $(h,k)-$competitive ratio. More precisely, $\rho_{EL}(h,k)$ is always less than $(1+\frac{1}{k})$ times the classic $\frac{k}{k-h+1}$ ratio – and if $h > k - \sqrt{k}$ the two quantities actually coincide. However, $\rho_{EL}(h,k)$ is also at least $1 + (\frac{1}{k} - \frac{2}{k^2})$ times as large as $\frac{k}{k-h+1}$ for any odd h and $k = 2h$, which proves a strict if minuscule separation between performance achievable in the presence and absence of memory fluctuations.

Section 6 briefly looks at the implications of our results for the RAM rental problem. In a nutshell, since many simple replacement are near optimal regardless of capacity fluctuations, RAM rental is simplified into the problem of just choosing a “good” capacity sequence without worrying about replacement.

Finally, Section 7 summarizes our results and looks at their significance and at possible directions of future work.

2 Some formalism/terminology

We can easily extend the notion of request sequence $\sigma = r_1, r_2, \ldots$ to the case of memory fluctuations. We simply assume that, interleaved with standard page requests, it is possible to have two additional types of requests, *growths* and *shrinks*. On a growth, memory capacity increases by 1 page; on a shrink, it decreases by 1 – and if the memory was full a page must

[7] Mark any page accessed; evict a random unmarked page, first unmarking all pages if all are marked.
[8] The “EL” in ρ_{EL} stood for “elastic” in an early, poster version of this work titled “Elastic paging” [25].

be evicted. We assume that initially memory capacity is 0. Throughout the rest of the article, we denote a growth request by the symbol $+$ and a shrink request by the symbol $-$, and we denote k consecutive growths / shrinks by $+^k$ and $-^k$. Thus, a standard request sequence $p_1, \ldots, p_n$ on a memory of capacity k simply becomes $+^k, p_1, \ldots, p_n$ in the more general dynamic capacity framework.

The request sequence automatically induces a *page sequence* $\pi =< p_1, p_2, \cdots >$ (the sequence of requested pages $p_1, p_2, \ldots,$ as in the classic paging problem) and a *capacity sequence* $\mu = m_1, m_2, \ldots$ where m_i is the memory capacity immediately before the request for p_i (i.e. it is equal to the number of $+$s minus the number of $-$s in the request prefix ending with p_i). Note that the presence of growths and shrinks introduces a second aspect of "onlineness". More formally:

▶ **Definition 1.** *A paging algorithm ALG is online relative to the page sequence if its eviction choices before servicing a request are independent of any future page requests; otherwise it is offline relative to the page sequence. Similarly, ALG is online relative to the capacity sequence if its eviction choices before servicing a request are independent of any subsequent growths and shrinks; otherwise it is offline relative to the capacity sequence. ALG is a fully online, partially offline and fully offline paging algorithm if it is online relative to (respectively) both, one, or neither of the page and the capacity sequence.*

Thus, in the dynamic capacity model, all well-known paging algorithms such as LRU, FIFO, FWF, CLOCK, RAND and MARK are fully online, and LFD is partially offline, being offline relative to the page sequence but online relative to the capacity sequence.

We can easily extend the notion of $(h, k)-$competitive ratio to the dynamic capacity model by comparing the cost (i.e. number of faults) incurred by an online algorithm whose memory capacity never exceeds k to the cost incurred by an offline algorithm whose memory capacity never exceeds $\frac{h}{k}$ times that of the online algorithm. More formally, denote by OPT the optimal offline algorithm, and by $c_{ALG}(\pi, \mu)$ the cost incurred by an algorithm ALG when servicing a page sequence $\pi = p_1, \ldots, p_n$ with a capacity sequence $\mu = m_1, \ldots, m_n$. Also, given a capacity sequence $\mu = m_1, \ldots, m_n$ and a non-negative number a, denote by $\lfloor a \cdot \mu \rfloor$ the capacity sequence $m'_1, \ldots, m'_n$ with $m'_i = \lfloor a \cdot m_i \rfloor$. Then:

▶ **Definition 2.** *A paging algorithm ALG has a* dynamic $(h, k)-$*competitive ratio of (at most)* ρ *if there exists some constant* d *such that, for any page sequence* $\pi = p_1, \ldots, p_n$ *and any capacity sequence* $\mu = m_1, \ldots, m_n$ *such that,* $\forall i$, $m_i \leq k$*:*

$$c_{ALG}(\pi, \mu) \leq \rho \cdot c_{OPT}(\pi, \lfloor \frac{h}{k} \cdot \mu \rfloor) + d$$

Note that the dynamic $(h, k)-$competitive ratio of an algorithm is always an upper bound to its $(h, k)-$competitive ratio. Thus online paging with dynamic capacity is in some sense "harder" than classic online paging, and no online algorithm can have a dynamic $(h, k)-$competitive ratio lower than the "classic" ratio $\frac{k}{k-h+1}$.

3 Minimal capacity fluctuations can lead to arbitrarily large performance degradation

This section shows that there exist online paging algorithms *that do not depend explicitly on memory capacity*, and that have an optimal (h, k)-competitive ratio in the classic setting of fixed memory capacity, but are not competitive at all, even with arbitrary resource augmentation, when faced with even slight fluctuations in memory capacity. Consider the online paging algorithm LFRU (Least Frequently / Recently Used) that starts as LRU and then alternates between LFU and LRU – switching from LRU to LFU after any palindrome subsequence incurring more faults in its second half, and switching from LFU to LRU after any palindrome subsequence incurring more faults in its first half:

Algorithm 1 LFRU: service $p_0, \dots, p_n$ as follows.

```
at $p_0$ POLICY ← LRU

for $i = 1 \dots n$ do

    if at $p_i$ POLICY = LRU AND $\exists j < i$:
    $< p_j \dots p_i >$ is palindrome AND faults($p_j \dots p_{\lfloor \frac{i+j}{2} \rfloor}$) < faults($p_{\lceil \frac{i+j}{2} \rceil} \dots p_i$)
    then at $p_{i+1}$ POLICY ← LFU

    else if at $p_i$ POLICY = LFU AND $\exists j < i$:
    $< p_j \dots p_i >$ is palindrome AND faults($p_j \dots p_{\lfloor \frac{i+j}{2} \rfloor}$) > faults($p_{\lceil \frac{i+j}{2} \rceil} \dots p_i$)
    then at $p_{i+1}$ POLICY ← LRU

end for
```

We would convince the reader that LFRU, while undoubtedly artificial and difficult to implement in practice, is not too different from many real-world paging heuristics designed for static memory capacity (note that the behaviour of LFRU, like that of LRU and LFU, does not depend explicitly on memory capacity). In fact, pure LRU tends to be outperformed in practice by various LRU/LFU hybrids [22, 24]. The main reason is the common coexistence of "local" or "temporal" computations sporting a high degree of temporal locality and data reuse, and "streaming" computations that access long sequences of sequential data without any temporal locality. In such cases, under LRU and similar policies such as CLOCK, streaming data not only gain no benefit from being kept in the fast memory layer (since every new access is a fault) but actively *pollute* it, forcing the eviction of temporal data and preventing the temporal computation from deriving more than a minimal benefit from the fast memory layer. One possible solution is to combine LRU with eviction schemes biased, like LFU, against data that have no reuse history even if their last (and only) access was very recent. And since LRU performs best when future requests are a "mirror image" of the past, it may seem reasonable to switch to it when such palindrome sequences sport good data-reuse behaviour, and switch to LFU when such palindrome sequences exhibit sport poor data-reuse behaviour – which is what LFRU does.

It turns out that LFRU has an optimal $(h, k)-$competitive ratio in the classic paging model where memory capacity is fixed. At the same time, even if faced with capacity fluctuations of just a single page, and even if allowed the use of an arbitrarily large amount of memory, LFRU's fault rate can be arbitrarily larger than that of an offline algorithm running with just 3 pages of memory. More formally we prove:

▶ **Theorem 3.** *LFRU has an $(h, k)-$competitive ratio equal to $\frac{k}{k-h+1}$ if memory capacity is constant, but has no finite dynamic $(h, k)-$competitive ratio for any $h \geq 3$ and any arbitrarily large k.*

Proof. Let us first prove that LFRU has an $(h, k)-$competitive ratio equal to $\frac{k}{k-h+1}$ if memory maintains an arbitrary but fixed capacity k. We need only prove that, as long as LFRU keeps behaving as LRU, on no page request sequence a palindrome subsequence incurs more faults in its second half: then LFRU keeps behaving exactly as LRU and shares its (h, k)-competitive ratio of $\frac{k}{k-h+1}$.

Consider a palindrome page subsequence $\pi = p_{i_1}, \dots, p_{i_\ell}, p_{i_\ell}, \dots, p_{i_1}$ of even length 2ℓ, containing $\lambda \leq \ell$ distinct pages $p_1, \dots, p_\lambda$. Note that, if $\lambda \leq k$, by the end of the first half of π the λ most recently requested pages are $p_i, \dots, p_\lambda$, which are then in memory and prevent any fault from taking place during the second half of π. Then, we need only consider the case $\lambda > k$.

Let us focus on the first half of π. For each distinct page, we analyse separately the first request to it (which we call a *cold* request), and the remaining requests, if any (which we call *hot* requests). The i^{th} cold request is certainly a fault for any $i > k$, since at least k distinct pages have been requested before it in $p_{i_1}, \dots, p_{i_\ell}$; so the number of cold requests incurring faults is at least $\ell - k$. Let us now look at hot requests, and let r_i be the number of hot requests of p_i in the first half of π. For $1 \leq i \leq \lambda$ and $1 \leq j \leq r_i$, let D_i^j be the set of distinct pages requested between the j^{th} hot request for p_i and the previous request for p_i, inclusive (so D_i^j always includes p_i). Then the j^{th} hot request to p_i is a fault if and only if $|D_i^j| > k$, and the total number of faults in $p_{i_1}, \dots, p_{i_\ell}$ is:

$$f_\pi^{\frac{1}{2}} \geq (\ell - k) + |\{(i,j) : |D_i^j| > k\}| \tag{1}$$

Let us now focus on the second half of π. Again, we divide requests for any distinct page into a cold request (the first) and hot requests (subsequent ones, if any). The first k cold requests of $p_{i_\ell}, \dots, p_{i_1}$ are for the last k distinct pages requested in $p_{i_1}, \dots, p_{i_\ell}$, which are then present in memory at the beginning of $p_{i_\ell}, \dots, p_{i_1}$. So in $p_{i_\ell}, \dots, p_{i_1}$ none of the first k cold requests incurs a fault, yielding and at most $\lambda - k$ faults on cold requests. Let us now look at the hot requests of $p_{i_\ell}, \dots, p_{i_1}$; those for p_i are obviously r_i, as in the first half of π. For $1 \leq i \leq \lambda$ and $1 \leq j \leq r_i$, let $\bar{D}_j^i$ be set of distinct pages between the j^{th} hot request for p_i and its previous request, including p_i itself; then the j^{th} hot request for p_i is a fault if and only if $|\bar{D}_i^j| > k$, and the total number of faults in $p_{i_\ell}, \dots, p_{i_1}$ is:

$$\bar{f}_\pi^{\frac{1}{2}} \leq (\ell - k) + |\{(i,j) : |\bar{D}_i^j| > k\}| \tag{2}$$

It is crucial to observe that, since π is palindrome, $\bar{D}_j^i = D_{r_i - j+1}^i$. Then $|\{(i,j) : |\bar{D}_i^j| > k\}| = |\{(i,j) : |D_i^j| > k\}|$ and $f_\pi^{\frac{1}{2}} \geq \bar{f}_\pi^{\frac{1}{2}}$. The analysis is virtually identical for palindrome subsequences of odd length; and thus with static memory capacity LFRU incurs no more faults on the second half of any palindrome subsequence than in the first half and has an $(h,k)-$competitive ratio equal to $\frac{k}{k-h+1}$.

To prove that LFRU can incur arbitrarily more faults than an optimal offline algorithm OPT when memory capacity fluctuates – even if OPT is limited to a capacity fluctuating between capacity 3 and 2, while LFRU's fluctuates between $3m$ and $3m-1$ for an arbitrarily large m – we show how LFRU can be coaxed into, and kept in, LFU behaviour, and how that behaviour can result in arbitrarily more faults than OPT even with arbitrarily larger capacity.

Denoting by r^m the concatenation of m consecutive copies of r, consider a page sequence formed by a prefix $\pi_1 =< p_1, p_2, p_3^\ell \dots, p_{3m}^\ell, p_1, p_2, p_3, \dots, p_{3m}, p_{3m}, \dots, p_2, p_1 >$ followed by a suffix $p_2 = (p_2, p_1)^{\ell-1}$. Assume LFRU's memory capacity while servicing π_1 remains equal to $3m$ except for the last $3m$ requests, during which it drops by 1 to $3m-1$; and assume it then remains equal to $3m-1$ while servicing π_2. It is immediate to see that when capacity drops p_1 is evicted, and that the last $6m$ requests of π_1 form a palindrome subsequence experiencing a fault (only) on the last request. Thus, on that request, LFRU switches to LFU behaviour, and evicts p_2 (which, like p_1, has experienced $\ell - 1$ fewer requests than every other page p_i, $i \geq 3$). Then LFRU services $\pi_2 = (p_2, p_1)^{\ell-1}$ with memory capacity $3m - 1$

by alternatively evicting p_2 and p_1 in turn, since p_1 and p_2 remain until the end of π_2 the two pages having experienced the fewest requests so far; thus LFRU incurs at least $2\ell - 2$ faults to service π_2.

An optimal algorithm (or even just LRU) with memory capacity 3 throughout all but the last $3m$ requests of π_1, and with memory capacity 2 thereafter, would instead incur no more than $3m + 3m + 3m = 9m$ faults during π_1, and no faults at all during π_2 (retaining only p_1 and p_2 in memory). Thus, since ℓ can be chosen arbitrarily larger than m, LFRU cannot have a finite $(3, 3m)-$competitive ratio for any arbitrarily large m. ◀

4 Dealing with adversarial fluctuations efficiently – and "implicitly"

In the light of Theorem 3 it may be surprising that many well-known "good" paging algorithms still perform remarkably well in the dynamic capacity setting – even though they do not take memory fluctuations into explicit account. It is very easy to prove:

▶ **Theorem 4.** *LFD incurs the minimal number of faults on any request sequence.*

Proof. We can safely ignore algorithms leaving unoccupied space in memory after an eviction, as such an eviction could be delayed without incurring additional faults. Let a page be *close* if it will be accessed before another page currently in memory, *far* otherwise. LFD is the algorithm evicting no close pages. We prove the theorem showing that one can always eliminate the earliest close eviction without altering previous evictions or increasing the number of faults.

Let p be the close page evicted earliest, at time t, by an algorithm ALG servicing a request sequence. Consider the algorithm $\overline{ALG}$ that operates as ALG until t, when it instead evicts a far page $\overline{p}$, and then operates as follows. Denote by M and $\overline{M}$ the sets of pages respectively in ALG's and $\overline{ALG}$'s memory. When both ALG and $\overline{ALG}$ must incur an eviction, $\overline{ALG}$ evicts the same page as ALG if possible; otherwise, when $\overline{ALG}$ must incur an eviction, it evicts a page not in M (as soon as $\overline{M} = M$, ALG and $\overline{ALG}$ coincide). After t, let t' be the time of the first request or eviction of either p or $\overline{p}$. Until t' ALG and $\overline{ALG}$ incur exactly the same faults and evictions, and thus $M \setminus \overline{M} = \{\overline{p}\}$ and $\overline{M} \setminus M = \{p\}$. At t' $\overline{ALG}$ evicts p if and only if ALG evicts $\overline{p}$ – in which case M and $\overline{M}$ converge. Otherwise p is requested at t' and ALG, but not $\overline{ALG}$, incurs a fault; and since $\overline{ALG}$ never evicts a page unless ALG also has evicted it, $|\overline{M} \setminus M|$ never increases after t, and drops to 0 no later than the first fault incurred by $\overline{ALG}$ and not by ALG. In both cases $\overline{ALG}$ incurs no more misses than ALG. ◀

It is interesting to note that Theorem 4 yields as an immediate corollary Theorem 4.1 in [17] – in a nutshell, for a given, dynamically changing, partition of the memory space between different processes, using LFD for each process on its own partition yields the minimum *total* number of faults. It is not, however, immediately obvious that the result in [17] implies the thesis of Theorem 4. Furthermore, the result in [17] is only stated, and not proved – the proof is deferred to the full version of the article because of its complexity compared to the "classic" proof of LFD's optimality.

Let us now focus on *online* paging algorithms. It turns out that the dynamic $(h, k)-$competitive ratio achievable by many well-known online algorithms is almost, but not quite, as good as the "plain" $(h, k)-$competitive ratio $\frac{k}{k-h+1}$ – and in particular equal to:

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

The formula for ρ_{EL} is more complex, but vaguely reminiscent of the formula for the "classic" (h,k)-competitive ratio; and indeed it is easy to verify that for $h = k$ both equal k. A detailed analysis of the behaviour of ρ_{EL}, including a proof that it is a lower bound on the dynamic $(h,k)-$competitive ratio of any online algorithm, can be found in the following Section 5. The remainder of this section is devoted to proving that a dynamic $(h,k)-$competitive ratio $\rho_{EL}(h,k)$ is indeed achieved by all marking algorithms[9] (including MARK, LRU and FWF), by RAND, and by all *dynamically conservative* algorithms. The latter form a class of algorithms that is slightly narrower than that of conservative algorithms[10][33] but still includes LRU, FIFO and CLOCK. The cornerstone of the analysis lies in the notion of *short subsequence*, which is the "correct" extension of the concept of $k-$phase to dynamic capacity:

▶ **Definition 5.** *Given a generic (sub)sequence of consecutive requests, its* width *is the number of distinct pages in it.*

▶ **Definition 6.** *Consider a generic request sequence σ, and a subsequence σ' of consecutive requests in σ (including page requests, growths and shrinks). σ' is* short *if, for every prefix π of σ', the width of π does not exceed the memory capacity* at the end *of π.*

▶ **Definition 7.** *A* dynamically conservative *algorithm never incurs more than w faults on any short subsequence of width w.*

Note that a dynamically conservative algorithm is also always a conservative algorithm according to the definition of [33] since with a memory of fixed capacity k every short subsequence involves access to at most k pages, and thus incurs at most k faults. The converse is not true: LFRU from Section 3 is conservative but not dynamically conservative. However, we can easily prove:

▶ **Theorem 8.** *LRU, FIFO and CLOCK are dynamically conservative.*

Proof. It is not difficult to verify that all three algorithms have the following property: if a page p is brought into memory at time t, and a page p' already in memory at time t and is never accessed again, then p' will be evicted before p. This holds for LRU because p is more recently accessed than p'. It holds for FIFO because p' entered the memory before p. It holds for CLOCK because after t the unmark/evict process will encounter p' before encountering p – thus either evicting or at least unmarking p' before unmarking p, and thus certainly evicting it before evicting p. Then none of the three algorithms evicts a page accessed during a short sequence before the end of the sequence (since there is always sufficient memory to hold all pages accessed during the sequence), and thus none can incur more faults than the width of the sequence. ◀

The main result of this section is then:

▶ **Theorem 9.** *The dynamic $(h,k)-$competitive ratio of any online paging algorithm that is either marking or dynamically conservative is no larger than $\rho_{EL}(h,k)$.*

Proof. Let us begin with marking algorithms. The proof bears some resemblance to that of the static case, with a number of subtle but profound differences. One such difference is that, instead of partitioning the request sequence into maximal length phases each involving

[9] A marking algorithm marks a page in memory whenever it accesses it, never evicts a marked page, and unmarks all pages if all are marked and one must be evicted (e.g. in response to a fault or a shrink).

[10] A conservative algorithm never incurs more than k faults on a sequence of accesses involving at most k distinct pages and a memory of capacity k.

access to k distinct pages, we partition it into maximal short sequences $\pi_1, \ldots, \pi_n$ where π_i is the longest short sequence beginning immediately after the end of π_{i-1}. Furthermore, the strategy of analysing each short subsequence in isolation does not work, and we can only bound the ratio over the whole sequence, through careful accounting and a potential argument.

Denote by w_i the width of π_i. We can assume without loss of generality that the request sequence ends with a page request, so $w_i > 0 \; \forall i$. Note that, for $i > 1$, the first request $\pi_{i,1}$ of π_i must be either a shrink or a request for a page not in π_{i-1}; in the first case we say that π_{i-1} is *capacity bound*, in the second that it is *page bound*.

It is easy to verify by simultaneous induction the following claims hold for all i:

1. All pages in memory are unmarked when $\pi_{i,1}$ is serviced.
2. Every one of the w_i pages accessed during π_i (and no other page) remains marked and thus in memory until the end of π_i.
3. Immediately before $\pi_{i+1,1}$ is serviced, the memory is full and holds w_i pages, all marked.

Claim 1 holds trivially for $i = 1$. If Claim 1 holds for i, Claim 2 also holds for i, since until the end of π_i the memory is large enough to accommodate all pages accessed so far during π_i, which are the only ones marked. If Claim 2 holds for i, Claim 3 also holds for i, since immediately before $\pi_{i+1,1}$ is serviced the memory capacity exactly matches the number of distinct pages accessed in π_i. If Claim 3 holds for i, Claim 1 holds for $i + 1$ (proving the inductive step), since the first request of π_{i+1} must be either a shrink or a request for a page not in π_i, and thus causes all pages in memory to become unmarked.

From Claim 2 it is obvious that a marking algorithm incurs a number of faults at most equal to w_i during short sequence π_i, for a total number of faults equal to at most:

$$c_{ALG} \le \sum_{i=1}^{n} w_i \tag{3}$$

Let us compute the number of faults incurred by any other algorithm $\overline{ALG}$ with a memory of capacity at most $\frac{h}{k}$ times that of the marking algorithm, in the interval π'_i from immediately after the first request σ of π_i is serviced, to immediately after the first request of π_{i+1} is serviced or to the end of the request sequence if $i = n$. Let r_i be equal to 1 if π_i is page bound, and to 0 if it is capacity bound, for $1 \le i < n$, and let $r_0 = 0$ and $r_n = 0$. Remember that the first request of a short phase π_i is a shrink if π_{i-1} is capacity bound, and a page not in π_{i-1} if π_{i-1} is page bound – and a growth if $i = 1$. Denoting by w'_i the number of distinct pages in π'_i after removing the page involved in the first request of π_i if any (i.e. if π_{i-1} is page bound), we can then write for $1 \le i \le n$:

$$w'_i = -r_{i-1} + w_i + r_i \tag{4}$$

The subset of these pages in the memory of $\overline{ALG}$ immediately before servicing the first request of π'_i is then at most:

$$m_i = \begin{cases} 0 & \text{if } i = 1, \\ \lfloor \frac{h}{k}(w_{i-1} - 1) \rfloor & \text{if } i > 1. \end{cases} \tag{5}$$

Equation 5 is immediate if $i = 1$ or if π_{i-1} is capacity bound - since then the first request of π_i shrinks the memory available to ALG from w_{i-1} to $w_{i-1} - 1$. If instead π_{i-1} is page bound, of the $\lfloor \frac{h}{k} w_{i-1} \rfloor$ pages $\overline{ALG}$'s memory can hold, one must be the first page of π_i that has just been requested and that does not contribute to m_i – leaving only $\lfloor \frac{h}{k} w_{i-1} \rfloor - 1 \leq \lfloor \frac{h}{k}(w_{i-1} - 1) \rfloor$. Then the total number of faults incurred by $\overline{ALG}$ is at least:

$$\sum (w'_i - m_i) \geq \sum_{i=1}^{n} (-r_{i-1} + w_i + r_i) - \sum_{i=2}^{n} \lfloor \frac{h}{k}(w_{i-1}) - 1) \rfloor \geq \sum_{i=1}^{n} (w_i - \lfloor \frac{h}{k}(w_i - 1) \rfloor) \quad (6)$$

Remembering that both w_i and $w_i - \lfloor \frac{h}{k}(w_i - 1) \rfloor$ with $h \leq k$ are positive, and that $\forall a, b, c, d > 0$ we have that $\frac{a+b}{c+d} = \frac{c}{c+d} \cdot \frac{a}{c} + \frac{d}{c+d} \cdot \frac{b}{d} \leq \max(\frac{a}{c}, \frac{b}{d})$, the dynamic $(h, k)-$competitive ratio of ALG is at most:

$$\frac{\sum_{i=1}^{n} w_i}{\sum_{i=1}^{n} (w_i - \lfloor \frac{h}{k}(w_i - 1) \rfloor)} \leq \max_i \frac{w_i}{w_i - \lfloor \frac{h}{k}(w_i - 1) \rfloor} \leq \max_{k' \in \{1, \ldots, k\}} \frac{k'}{k' - \lfloor h \frac{k'}{k} - \frac{h}{k} \rfloor} \quad (7)$$

This proves the theorem for marking algorithms. The proof for dynamically conservative algorithms proceeds identically, except for the fact that in this case one can immediately obtain, from Definition 7, the bound given by Equation 3 on the cost incurred by the online algorithm. ◀

The analysis of RAND faces similar difficulties similar to those in the proof of Theorem 9 in terms of "compartimentalization of costs"; but they can be addressed in a different way due to the randomized nature of the algorithm, by exploiting its lack of memory. In this sense it may be somewhat surprising that *exactly* the same bound obtained in Theorem 9 also applies to RAND, particularly because in the (very slightly different) cache-adaptive model RAND is not competitive at all, regardless of resource augmentation[11]. Instead, we prove:

▶ **Theorem 10.** *RAND's dynamic $(h, k)-$competitive ratio is no larger than $\rho_{EL}(h, k)$ in the adaptive online adversary model.*

Proof. We cannot apply the "classic" paging analysis of RAND due to the fact that, if $h < k$, cache shrinks may not be "synchronized" and RAND may incur shrinks when the optimal offline algorithm OPT does not. Instead of comparing the number of faults c_{RAND} and c_{OPT} incurred, respectively, by RAND and OPT, we then begin by comparing the number of *page evictions* e_{RAND} and e_{OPT}. For simplicity, assume that, after any given request (for a page, or for a capacity change), the request is served in the following order. OPT performs any page eviction; then it loads into memory any requested page not yet there; then RAND does the same; finally, OPT adjusts its memory capacity, and then RAND does the same. Note that since capacity is adjusted one page at a time (see Section 2), evictions are always performed one page at a time.

Let the *garbage* of RAND at any given point in time be the set G of pages in its memory and not in the memory of OPT. First of all, note that G can increase only when OPT incurs an eviction (and at most by 1 page for each eviction), since RAND never brings into memory a page not requested by OPT – which at that point must then be in OPT's memory.

[11] This can be easily seen by the reader familiar with the cache-adaptive model. Consider, for an arbitrarily large $c \geq 2$, the memory sequence $p_{2c}, \ldots, p_0, (p_1, p_0)^{3c^2}$. OPT can service the sequence with memory 2 incurring only $2c + 1$ faults. RAND with memory $2c$ has a non-zero probability of evicting p_1 on every request for p_0 and viceversa, and thus incurring more than $3c^2 > (2c + 1)c$ faults – so in the worst case it completes the sequence after OPT even with the advantage of $c-$speed and $c-$memory augmentation. While this simple example leverages the worst-case accounting of the standard cache-adaptive model, even more sophisticated accounting "in expectation" faces similar difficulties – informally because in the cache-adaptive model even a minuscule probability of missing a shrink deadline can be catastrophic.

Immediately before RAND incurs an eviction, its memory must be full; denote by k' and h' the memory capacity of RAND and OPT at that point. If the eviction is the result of a shrink, then the number of pages in RAND's memory that are *not* garbage are at most:

$$h' = \lfloor \frac{h}{k}(k'-1) \rfloor \tag{8}$$

Note that at this point OPT has adjusted its memory capacity to the shrink but RAND has not. If the eviction is the result of a page fault, then the requested page at this point is in OPT's memory but not in RAND's, and the number of pages in RAND's memory that are *not* garbage are at most:

$$h' - 1 = \lfloor \frac{h}{k}k' \rfloor - 1 \leq \lfloor \frac{h}{k}(k'-1) \rfloor \tag{9}$$

Thus the probability that, when RAND incurs an eviction, $|G|$ decreases by 1 is at least:

$$p_{RAND} = \min_{k' \in \{1,\ldots,k\}} \frac{k' - \lfloor \frac{h}{k}(k'-1) \rfloor}{k'} \tag{10}$$

and at any given time we have that, in expectation:

$$|G| \leq e_{OPT} - e_{RAND} \cdot p_{RAND} \tag{11}$$

Appending to any request sequence sufficient shrinks to bring *RAND*'s memory capacity to 0 obviously brings $|G|$ to 0, without increasing the number of *faults* incurred by *RAND* or *OPT*. For any algorithm that evicts a single page at a time, when the memory holds no pages the number of faults and evictions incurred must coincide. Setting $|G|$ to 0, as well as $c_{OPT} = e_{OPT}$ and $c_{RAND} = e_{RAND}$, in Equation 11 then yields for RAND a dynamic $(h,k)-$competitive ratio equal at most to:

$$\frac{c_{RAND}}{c_{OPT}} \leq \frac{1}{p_{RAND}} = \max_{k' \in \{1,\ldots,k\}} \frac{k'}{k' - \lfloor h\frac{k'}{k} - \frac{h}{k} \rfloor} \tag{12}$$

◀

5 An exact characterization of the competitive ratio

The upper bound $\rho_{EL}(h,k)$ obtained in Section 4 for the dynamic $(h,k)-$competitive ratio of many online paging algorithms is actually tight. In fact, no paging algorithm that is online relative to the page request sequence can achieve a better $(h,k)-$competitive ratio, *even if it has from the start full knowledge of the entire capacity sequence.* More formally we can prove:

▶ **Theorem 11.** *No paging algorithm that is online relative to the page sequence has a dynamic $(h,k)-$competitive ratio (against any online or offline adaptive adversary if randomized) lower than:*

$$\rho_{EL}(h,k) = max_{k' \in \{1,\ldots,k\}} \frac{k'}{k' - \lfloor h\frac{k'}{k} - \frac{h}{k} \rfloor}$$

Proof. Let $\overline{k} = argmax_{k' \in \{1,\ldots,k\}} \frac{k'}{k' - \lfloor h\frac{k'}{k} - \frac{h}{k} \rfloor}$, and let *ALG* be a generic paging algorithm online relative to the page sequence. Consider a request sequence $\sigma_n =< +^{(\overline{k}-1)}, \pi_1, \ldots, \pi_n >$, where:

$$\pi_i =< +^{(k-\overline{k}+1)}, p_{i,1}, -^{(k-\overline{k}+1)}, \ldots, +^{(k-\overline{k}+1)}, p_{i,\overline{k}}, -^{(k-\overline{k}+1)} > \tag{13}$$

and $p_{i,j}$ is any one page, from the set $p_1, \dots, p_{\overline{k}}$, that is not in ALG's memory just before it is requested – note that immediately before any page request ALG's memory holds at most $\overline{k} - 1$ pages, so there always exists one such page. ALG then incurs a fault on every page request, for a total number of faults equal to:

$$c_{ALG}(\sigma_n) = n \cdot \overline{k} \tag{14}$$

Also, note that the capacity sequence associated to σ_n does not depend on ALG, so we can freely assume that the design of ALG incorporates full knowledge of it.

Consider an offline algorithm $\overline{ALG}$ with access to a memory that has at most $\frac{h}{k}$ times the capacity of ALG's at any given time; in particular, $\overline{ALG}$'s memory capacity grows to h immediately before any page request, and immediately afterwards drops to capacity:

$$\lfloor \frac{h}{k}(\overline{k} - 1) \rfloor = \lfloor h\frac{\overline{k}}{k} - \frac{h}{k} \rfloor < h \tag{15}$$

$\overline{ALG}$ can easily maintain in its "permanent" $\lfloor h\frac{\overline{k}}{k} - \frac{h}{k} \rfloor$ memory locations the $\lfloor h\frac{\overline{k}}{k} - \frac{h}{k} \rfloor$ pages with most *expected* accesses in σ_n, incurring for each only one initial fault. Note that the total number of accesses to these pages is, in expectation, at least $n\overline{k} \cdot \frac{\lfloor h\frac{\overline{k}}{k} - \frac{h}{k} \rfloor}{\overline{k}} = n\lfloor h\frac{\overline{k}}{k} - \frac{h}{k} \rfloor$. Every other page, when requested, is brought into the "temporary" location(s) immediately eliminated by the following shrink. $\overline{ALG}$ then incurs an expected number of faults equal to:

$$c_{\overline{ALG}}(\sigma_n) \leq \lfloor h\frac{\overline{k}}{k} - \frac{h}{k} \rfloor + n(\overline{k} - \lfloor h\frac{\overline{k}}{k} - \frac{h}{k} \rfloor) \tag{16}$$

Then the competitive ratio of ALG can be no lower than:

$$\lim_{n\to\infty} \frac{c_{ALG}(\sigma_n)}{c_{\overline{ALG}}(\sigma_n)} = \lim_{n\to\infty} \frac{n\overline{k}}{\lfloor h\frac{\overline{k}}{k} - \frac{h}{k} \rfloor + n(\overline{k} - \lfloor h\frac{\overline{k}}{k} - \frac{h}{k} \rfloor)} = \max_{k' \in \{1,\dots,k\}} \frac{k'}{k' - \lfloor h\frac{k'}{k} - \frac{h}{k} \rfloor} \tag{17}$$

It is important to observe that, if ALG is randomized, $\overline{ALG}$ need only know ALG's probabilistic behaviour to choose which pages to keep in its own memory; and it can choose which page to request next based only on ALG's current memory contents. Thus the lower bound we proved holds for deterministic and randomized algorithms both in the adaptive offline and in the adaptive online adversary models. ◀

As noted in Section 4 the expression of the optimal dynamic (h, k)-competitive ratio $\rho_{EL}(h, k)$ appears considerably more complex than, but vaguely reminiscent of, that of the "classic" bound on the (h, k)–competitive ratio, $\frac{k}{k-h+1}$. It is natural to ask whether the two are actually different, and if so to what extent. We show that $\rho_{EL}(h, k)$ *is*, in fact, a factor $\approx 1 + \frac{1}{k}$ larger for some "natural" values of h and k – though it is never more than a factor $1 + \frac{1}{k}$ larger, and actually coincides with $\frac{k}{k-h+1}$ if h is equal or very close to k. This is stated more formally in the following two theorems:

▶ **Theorem 12.** *For any odd h and $k = 2h$, $\rho_{EL}(h, k) \geq (1 + \frac{1}{k} - \frac{2}{k^2})\frac{k}{k-h+1}$.*

Proof. For any integer $i \geq 0$, choosing $h = 2i + 1$, $k = 2h$, and $k' = k - 1$, we obtain immediately:

$$\rho_{EL}(h, k) \geq \frac{4i + 1}{4i + 1 - \lfloor (2i + 1)\frac{4i+1}{4i+2} - \frac{2i+1}{4i+2} \rfloor} = \frac{4i + 1}{4i + 1 - \lfloor \frac{4i+1}{2} - \frac{1}{2} \rfloor} = \frac{4i + 1}{2i + 1}$$
$$= \frac{k - 1}{\frac{k}{2}} = \frac{k - 1}{\frac{k}{2}} \cdot \frac{\frac{k}{2} + 1}{k} \cdot \frac{k}{k - h + 1} = (1 + \frac{1}{k} - \frac{2}{k^2})\frac{k}{k - h + 1} \tag{18}$$

◀

▶ **Theorem 13.** $\frac{k}{k-h+1} \leq \rho_{EL}(h,k) < (1+\frac{1}{k})\frac{k}{k-h+1}$ *for all* $h,k \in \mathbb{Z}^+$ *with* $h \leq k$, *and* $\rho_{EL}(h,k) = \frac{k}{k-h+1}$ *if* $k \geq h > k - \sqrt{k}$.

Proof. It is immediate to verify that, for $k' = k$:

$$\rho_{EL}(h,k) \geq \frac{k'}{k' - \lfloor h\frac{k'}{k} - \frac{h}{k} \rfloor} = \frac{k}{k-h+1} \tag{19}$$

And since, if $k' \leq h$, we have that:

$$\frac{k'}{k' - \lfloor h\frac{k'}{k} - \frac{h}{k} \rfloor} \leq \frac{k'}{k' - (h\frac{k'}{k} - \frac{h}{k})} = \frac{k'\frac{k}{k'}}{k'\frac{k}{k'} - h\frac{k'}{k}\frac{k}{k'} + \frac{h}{k}\frac{k}{k'}} \leq \frac{k}{k-h+1} \tag{20}$$

then values of $k' \leq h$ can be disregarded in the max operation. To prove that, for all $h \leq k$, $\rho_{EL}(h,k) \leq (1+\frac{1}{k})\frac{k}{k-h+1}$, note that:

$$\begin{aligned} \rho_{EL}(h,k) &= max_{k' \in \{1,\ldots,k\}} \frac{k'}{k' - \lfloor h\frac{k'}{k} - \frac{h}{k} \rfloor} \\ &\leq max_{k' \in \{1,\ldots,k\}} \frac{k'}{k' - (h\frac{k'}{k} - \frac{h}{k})} = \frac{k}{k - (h\frac{k}{k} - \frac{h}{k})} = \frac{k}{k-h+\frac{h}{k}} \end{aligned} \tag{21}$$

Then we obtain:

$$\frac{\rho_{EL}(h,k)}{\frac{k}{k-h+1}} \leq \frac{k-h+1}{k-h+\frac{h}{k}} = 1 + \frac{\frac{k-h}{k}}{k-h+\frac{h}{k}} = 1 + \frac{1}{k + \frac{h}{k-h}} < 1 + \frac{1}{k} \tag{22}$$

To prove that $\rho_{EL}(h,k)$ coincides with $\frac{k}{k-h+1}$ for $h > k - \sqrt{k}$ let us rewrite h and k' as $h = k - a$ and $k' = k - b$, with $a > b$ and $a, b \in \mathbb{Z}_0^+$. We obtain:

$$\begin{aligned} \rho_{EL}(h > (k-\sqrt{k}), k) &= \max_{\sqrt{k}>a>b} \frac{k-b}{k-b-\lfloor \frac{(k-a)(k-b)}{k} - \frac{k-a}{k} \rfloor} \\ &= \max_{\sqrt{k}>a>b} \frac{k-b}{k-b-\lfloor k-(a+b)+\frac{ab}{k} - 1 + \frac{a}{k} \rfloor} \\ &= \max_{\sqrt{k}>a>b} \frac{k-b}{k-b-k+(a+b)+1-\lfloor \frac{a(b+1)}{k} \rfloor} \\ &< \max_{\sqrt{k}>a>b} \frac{k}{k-h+1-\lfloor \frac{a(b+1)}{k} \rfloor} = \frac{k}{k-h+1} \end{aligned} \tag{23}$$

where the last equality follows from the fact that, since $a = k-h < \sqrt{k}$ and $b \leq a-1 < \sqrt{k}-1$, then $a(b+1) < k$. ◀

The complex expression of $\rho_{EL}(h,k)$ is in part due to the "rounding" of the memory capacity of the optimal offline algorithm. However, it is important to note that this rounding is not sufficient to explain why $\rho_{EL}(h,k)$ can be strictly *larger* than the "classic" ratio $\frac{k}{k-h+1}$ obtained when capacity is fixed at its maximum value: at smaller capacities rounding can only favour the online algorithm, and for any fixed ratio $\frac{k'}{h'}$, $\frac{k'}{k'-h'+1}$ strictly decreases with k', again favouring the online algorithm at smaller capacities. *Capacity fluctuations (rather than simply the choice between different, constant capacities) are then the source of the separation* between $\rho_{EL}(h,k)$ and the "classic" (h,k)-competitive ratio $\frac{k}{k-h+1}$.

6 Decoupling replacement from capacity in RAM rental

The results from Section 4 can be readily applied to the RAM rental problem, in which a paging algorithm ALG can choose the capacity sequence (with maximum capacity k), and the cost it incurs and must minimize on a request sequence σ is:

$$R^k_{ALG}(\sigma) = \sum_{i=1}^{|\sigma|}(\alpha f(i) + \beta w(i)) \tag{24}$$

where $w(i)$ is the capacity when serving the i^{th} request of σ, and $f(i)$ is 1 if that request is a fault and 0 otherwise. The fundamental consequence of our results from Section 4 is that to a large extent the replacement policy can be decoupled from the choice of capacities. More precisely, Theorem 9 yields:

▶ **Corollary 14.** *Consider a paging algorithm ALG, servicing each request σ_i of a sequence σ with capacity $w(i) \leq h$ and an arbitrary (even offline) replacement policy; and a second paging algorithm ALG′ servicing σ_i with capacity $2w(i)$ and a replacement policy that can be any marking or dynamically conservative algorithm. Then, for any choice of α, β and $w(\cdot) \leq h$:*

$$R^{2h}_{ALG'}(\sigma) \leq 2 \cdot R^h_{ALG}(\sigma). \tag{25}$$

which follows immediately from the fact that the sum of all faults incurred by ALG' is at most twice that by ALG as long as ALG' maintains twice the capacity of ALG. *In other words, RAM rental is all about choosing the correct capacity at any given time; and any of the "classic" replacement policies analysed in the previous section will be close to optimal for any choice of α, of β, and of the capacity sequence.*

7 Conclusions

Good performance in the case of constant memory capacity provides no performance guarantees whatsoever in the case of fluctuating memory capacity: moving from a scenario where capacity remains constant to one where it can fluctuate by a single page can mean the difference between performance optimal within a factor 2, and performance suboptimal by an arbitrarily large factor. This suggests the need of extreme caution when evaluating with classic methodologies the performance of paging algorithms meant for memory systems with dynamic capacity.

A counterpoint to this very "negative" result is that several extremely simple classic paging algorithms achieve optimal or nearly optimal performance even in the dynamic capacity framework - which is surprising because none of those algorithms is designed to take memory capacity fluctuations into explicit account. Counterintuitively then, while knowledge of future page requests provides an advantage, knowledge of future memory capacity does not. A practical corollary is that, in the design of memory architectures, one can efficiently decouple the problem of allocating memory resources to different cores/processes/threads from the problem of managing the allocated memory – greatly simplifying system design and analysis and providing a strong (a posteriori!) theoretical justification for the exokernel approach [12].

As in classic paging, in the dynamic capacity framework competitive analysis fails to distinguish between the performance of LRU, of FIFO, and of more naive algorithms such as RAND or FWF – at least without resorting to more sophisticated approaches such as access graphs. While each of these algorithms is still guaranteed to outperform an optimal offline algorithm (and thus any other online algorithm) whose memory system has half the capacity and twice the access cost, there are probably differences within those factors of 2 that would be important to characterize in practice. It is by no means clear whether the winner in the dynamic capacity scenario would be the same as in the classic one, or whether models designed a posteriori to explain the superiority of e.g. LRU over FIFO would still provide correct predictions.

In this sense we are not aware of any experimental benchmarks specifically designed to assess the impact of memory capacity fluctuations. A fundamental obstacle in their development seems to be the difficulty of characterizing "typical" fluctuation patterns encountered in practice. An interesting line of inquiry would be to investigate whether one can obtain, from the performance numbers of a black box algorithm under a small "basis" of specific fluctuation patterns, sufficient information to compute a good assessment of the algorithm's performance numbers under any other pattern.

Finally, the impact of resource fluctuation on how efficiently a task can be solved is a line of inquiry obviously not limited to memory management. There are a number of other problems where the amount of resources available can realistically vary over time. Examples include call admission [16] (with variable circuit capacity) and the numerous variants of online scheduling [27] (with e.g. variable number or speed of servers). In addition to studying each problem individually, it would be extremely interesting to identify broad classes sharing similar characteristics. For example, which problems can be solved optimally or almost optimally without knowledge of the amount of resources available in the future (as in the case of paging with dynamic memory capacity)?

References

1 Susanne Albers, Lene M. Favrholdt, and Oliver Giel. On paging with locality of reference. *J. Comput. Syst. Sci.*, 70(2):145–175, 2005. doi:10.1016/j.jcss.2004.08.002.

2 Spyros Angelopoulos, Reza Dorrigiv, and Alejandro López-Ortiz. On the separation and equivalence of paging strategies. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2007, New Orleans, Louisiana, USA, January 7-9, 2007*, pages 229–237, 2007. URL: http://dl.acm.org/citation.cfm?id=1283383.1283408.

3 Rakesh D. Barve and Jeffrey Scott Vitter. A Theoretical Framework for Memory-Adaptive Algorithms. In *40th Annual Symposium on Foundations of Computer Science, FOCS '99, 17-18 October, 1999, New York, NY, USA*, pages 273–284, 1999. doi:10.1109/SFFCS.1999.814599.

4 Laszlo A. Belady. A Study of Replacement Algorithms for Virtual-Storage Computer. *IBM Systems Journal*, 5(2):78–101, 1966. doi:10.1147/sj.52.0078.

5 Shai Ben-David and Allan Borodin. A New Measure for the Study of On-Line Algorithms. *Algorithmica*, 11(1):73–91, 1994. doi:10.1007/BF01294264.

6 Michael A. Bender, Erik D. Demaine, Roozbeh Ebrahimi, Jeremy T. Fineman, Rob Johnson, Andrea Lincoln, Jayson Lynch, and Samuel McCauley. Cache-Adaptive Analysis. In *Proceedings of the 28th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA 2016, Asilomar State Beach/Pacific Grove, CA, USA, July 11-13, 2016*, pages 135–144, 2016. doi:10.1145/2935764.2935798.

7 Michael A. Bender, Roozbeh Ebrahimi, Jeremy T. Fineman, Golnaz Ghasemiesfeh, Rob Johnson, and Samuel McCauley. Cache-Adaptive Algorithms. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 958–971, 2014. doi:10.1137/1.9781611973402.71.

8 Allan Borodin and Ran El-Yaniv. *Online computation and competitive analysis.* Cambridge University Press, 1998.

9 Allan Borodin, Sandy Irani, Prabhakar Raghavan, and Baruch Schieber. Competitive Paging with Locality of Reference. *J. Comput. Syst. Sci.*, 50(2):244–258, 1995. `doi:10.1006/jcss.1995.1021`.

10 Joan Boyar, Sandy Irani, and Kim S. Larsen. A Comparison of Performance Measures for Online Algorithms. *Algorithmica*, 72(4):969–994, 2015. `doi:10.1007/s00453-014-9884-6`.

11 Marek Chrobak. SIGACT news online algorithms column 17. *SIGACT News*, 41(4):114–121, 2010. `doi:10.1145/1907450.1907547`.

12 Dawson R. Engler, M. Frans Kaashoek, and James O'Toole. Exokernel: An Operating System Architecture for Application-Level Resource Management. In *Proceedings of the Fifteenth ACM Symposium on Operating System Principles, SOSP 1995, Copper Mountain Resort, Colorado, USA, December 3-6, 1995*, pages 251–266, 1995. `doi:10.1145/224056.224076`.

13 Amos Fiat and Anna R. Karlin. Randomized and multipointer paging with locality of reference. In *Proceedings of the Twenty-Seventh Annual ACM Symposium on Theory of Computing, 29 May-1 June 1995, Las Vegas, Nevada, USA*, pages 626–634, 1995. `doi:10.1145/225058.225280`.

14 Amos Fiat, Richard M. Karp, Michael Luby, Lyle A. McGeoch, Daniel Dominic Sleator, and Neal E. Young. Competitive Paging Algorithms. *J. Algorithms*, 12(4):685–699, 1991. `doi:10.1016/0196-6774(91)90041-V`.

15 Matteo Frigo, Charles E. Leiserson, Harald Prokop, and Sridhar Ramachandran. Cache-Oblivious Algorithms. *ACM Trans. Algorithms*, 8(1):4:1–4:22, 2012.

16 Juan A. Garay, Inder S. Gopal, Shay Kutten, Yishay Mansour, and Moti Yung. Efficient On-Line Call Control Algorithms. *J. Algorithms*, 23(1):180–194, 1997.

17 Avinatan Hassidim. Cache Replacement Policies for Multicore Processors. In *ICS*, pages 501–509. Tsinghua University Press, 2010.

18 Houman Homayoun, Mohammad A. Makhzan, and Alexander V. Veidenbaum. Multiple sleep mode leakage control for cache peripheral circuits in embedded processors. In *Proceedings of the 2008 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems, CASES 2008, Atlanta, GA, USA, October 19-24, 2008*, pages 197–206, 2008. `doi:10.1145/1450095.1450125`.

19 Sandy Irani. Page Replacement with Multi-Size Pages and Applications to Web Caching. *Algorithmica*, 33(3):384–409, 2002.

20 Anna R. Karlin, Mark S. Manasse, Larry Rudolph, and Daniel Dominic Sleator. Competitive Snoopy Caching. *Algorithmica*, 3:77–119, 1988. `doi:10.1007/BF01762111`.

21 Elias Koutsoupias and Christos H. Papadimitriou. Beyond Competitive Analysis. *SIAM J. Comput.*, 30(1):300–317, 2000.

22 Donghee Lee, Jongmoo Choi, Jong-Hun Kim, Sam H. Noh, Sang Lyul Min, Yookun Cho, and Chong-Sang Kim. LRFU: A spectrum of policies that subsumes the least recently used and least frequently used policies. *IEEE Trans. Computers*, 50(12):1352–1361, 2001.

23 Alejandro López-Ortiz and Alejandro Salinger. Minimizing Cache Usage in Paging. In *WAOA*, volume 7846 of *Lecture Notes in Computer Science*, pages 145–158. Springer, 2012.

24 Nimrod Megiddo and Dharmendra S. Modha. Outperforming LRU with an Adaptive Replacement Cache Algorithm. *IEEE Computer*, 37(4):58–65, 2004. `doi:10.1109/MC.2004.1297303`.

25 Enoch Peserico. Elastic Paging. *SIGMETRICS Perform. Eval. Rev.*, 41(1):349–350, June 2013. `doi:10.1145/2494232.2479781`.

26 Enoch Peserico. Paging with dynamic memory capacity. *CoRR*, abs/1304.6007, 2013. `arXiv:1304.6007v1`.

27 Kirk Pruhs. Competitive online scheduling for server systems. *SIGMETRICS Performance Evaluation Review*, 34(4):52–58, 2007. `doi:10.1145/1243401.1243411`.

28 Moinuddin K. Qureshi and Yale N. Patt. Utility-Based Cache Partitioning: A Low-Overhead, High-Performance, Runtime Mechanism to Partition Shared Caches. In *MICRO*, pages 423–432. IEEE Computer Society, 2006.

29 Prabhakar Raghavan and Marc Snir. Memory versus randomization in on-line algorithms. *IBM Journal of Research and Development*, 38(6):683–708, 1994.

30 Daniel Dominic Sleator and Robert Endre Tarjan. Amortized Efficiency of List Update and Paging Rules. *Commun. ACM*, 28(2):202–208, 1985. `doi:10.1145/2786.2793`.

31 Eric Torng. A Unified Analysis of Paging and Caching. *Algorithmica*, 20:194–203, 1998.

32 Neal E. Young. On-Line Caching as Cache Size Varies. In *Proceedings of the Second Annual ACM/SIGACT-SIAM Symposium on Discrete Algorithms, 28-30 January 1991, San Francisco, California, USA.*, pages 241–250, 1991. URL: `http://dl.acm.org/citation.cfm?id=127787.127832`.

33 Neal E. Young. The K-Server Dual and Loose Competitiveness for Paging. *Algorithmica*, 11:525–541, 1994.