Electronic Journal of SADIO
http://www.dc.uba.ar/sadio/ejs/
vol. 1, no. 1, pp. 21–36 (1998)

New Results on Fair Multi-threaded Paging

Alejandro Strejilevich de Loma

Departamento de Computación, Facultad de Ciencias Exactas y Naturales, Universidad de Buenos Aires, Pabellón I, Ciudad Universitaria, (1428) Capital Federal, Argentina; e-mail: asdel@dc.uba.ar.

Abstract

The Multi-threaded Paging problem (MTP) was introduced as a generalization of Paging for the case where there are many threads of requests. This models situations in which the requests come from more than one independent source. At each step it is necessary to decide which request to serve among several possibilities, and also (as in normal Paging) which page of fast memory to remove on a page fault. In the fair version of the problem any algorithm must guarantee that the next request of each thread will be served within a predetermined finite time.

In this paper we reduce the existing gaps between the known lower and upper bounds for the competitiveness of on-line algorithms for the fair version of MTP. We treat some particular situations, with finite and infinite input sequences. We prove higher lower bounds and present a new on-line algorithm. We close the gap for the case in which the cache can hold only one page; surprisingly, we obtain different bounds for even and odd number of sequences; we prove that any lazy algorithm achieves the on-line lower bound when the number of sequences is odd.

Research supported in part by EC project DYNDATA under program KIT, and by UBACyT projects "Algoritmos Eficientes para Problemas On-line con Aplicaciones" and "Modelos y Técnicas de Optimización Combinatoria". A preliminary version of this paper appeared in [11, 12].

1 Introduction

The *Paging* problem consists of managing a two-level memory, one of them with limited capacity and fast access (the cache) and the other one with slow access but potentially unlimited capacity. A *Paging algorithm* is faced with a sequence of page references. At each step the algorithm must ensure that the requested page is in fast memory, perhaps evicting another page to make room for the incoming one. A *page fault* occurs each time a page must be brought into fast memory. The goal of the Paging algorithm is to minimize the total number of page faults over the sequence of requests. An *on-line algorithm* for Paging must decide which page to evict without knowledge of future requests, while an *off-line algorithm* can decide based on the whole sequence. On-line algorithms for Paging have been studied from a *competitive analysis* point of view in [10], comparing their performance to that of the optimal off-line algorithm. In that work it is shown that, if the cache can hold k pages, no deterministic on-line algorithm can be better than k-competitive, that is, guarantee less than k times the optimal off-line number of page faults on every input; besides, it is shown that known on-line algorithms such as Least-Recently-Used (LRU) and First-In-First-Out (FIFO) achieve that bound.

The *Multi-threaded Paging* problem (MTP) was introduced as a generalization of Paging for the case in which there is not just one sequence of requests but possibly many threads [5, 6]. This models situations in which the requests come from more than one independent source, as in multi-tasking systems where all processes independently present their requests of memory pages to the manager of fast memory. At each step it is necessary to decide which request to serve among several possibilities, and also (as in normal Paging) which page of fast memory to remove on a page fault. The total number of page faults depends therefore not only on the strategy used to determine how each request is served but when (in which order) this is done. Moreover, in some cases even the set of requests that will eventually be served depends on the particular algorithm that is used. As it can be seen, in MTP there is no notion of "sequence of requests" but a more complex pattern that is not captured by the most general classes of on-line problems proposed in the literature (like Metrical Task Systems [4] or Request-Answer Games [2, 9]).

Two versions of MTP were presented in [6]. In the first one, the goal is just to minimize the number of page faults done while serving a set of w sequences of requests. In the second one, *fairness* restrictions are imposed, so any algorithm must guarantee that the next request of each thread will be served within a predetermined finite time. For each one of these two problems, finite and infinite sequences of requests can be considered. In [6] it was proved that the only case in which there exist competitive on-line algorithms for the fair version is when fairness restrictions are so tight that enforce serving one request of each thread in a cyclic way. For that case, a $(k + w)$-competitive on-line algorithm was proposed, and a lower bound of k for the competitiveness of any on-line algorithm was obtained.

In this paper we address some of the problems that were left open when MTP was introduced. More precisely, for some particular situations of the fair version of MTP we reduce the existing gaps between the known lower and upper bounds. We treat several cases: The first one is the case in which the cache can hold only one page ($k = 1$) and the number of sequences is even; we raise from 1 to $w + 1$ the known lower bound, and therefore the on-line algorithm from [6] is strongly competitive (optimal). In the second case again we have $k = 1$, but now we consider an odd w; in this case the new lower bound is w, and we prove that any algorithm that does not unnecessarily evict pages achieves that bound. The third case is when $w \leq k$; we present a $(k + 1)$-competitive on-line algorithm, reducing the existing gap to 1.

The case $k = 1$ models the scheduling problem in which a machine serves w sequences of tasks, where the cost of each task is insignificant or zero, and the cost of switching tasks is unitary. This is an extreme situation of the more general case $k < w$. We treat that extreme situation and the opposite case, that is, the case $w \leq k$.

Fiat and Karlin [7] have considered a version of Paging in which the input corresponds to a multi-pointer walk on an *access graph* [3]. Within that framework, there are multiple threads of requests, but they are seen as a unique sequence corresponding to an interleaved execution of the different threads. The way in which the threads are interleaved in [7] is decided in an earlier stage of the process. On the contrary, in MTP the algorithms are free to decide (up to a certain limit, in the case of fairness restrictions) how to do that. In other words, the algorithms for MTP act not only as fast memory managers but also as schedulers, while in the cited work the scheduling is implicitly supposed to be done somewhere else.

Recently, Alborzi et al. [1] have proposed a multi-threaded version of the 1-server problem. They consider a metric space in which a server moves at constant speed to satisfy requests generated by many clients; each request is satisfied when the server arrives to the location of the request, and then the corresponding client presents a new request in another place of the metric space. Although the 1-server problem is a generalization of Paging with $k = 1$, only finite input sequences are considered in [1], and fairness restrictions are not stated.

The remainder of this paper is organized as follows: In Section 2 we formally present the fair version of MTP, with finite and infinite input sequences, and detail the known results about it. In Section 3 we treat the case $k = 1$ and even w. In Section 4 we complete the case $k = 1$ by considering an odd w. In Section 5 we analyze the case $w \leq k$. We conclude in Section 6 by presenting some remarks.

2 Preliminaries

2.1 Description of the Problem

Imagine that w independent processes simultaneously present their requirements of pages of secondary memory that must be brought into fast memory. At each moment, the scheduler and fast memory manager can see only one request per process, precisely the first unserved request of the sequence of requests that the process presents. Serving the current request of a particular process allows the system to see the next request of that process. The system must decide at each step which request should be served, apart from deciding which page of fast memory to evict to make room for the incoming page if that is the case.

MTP allows the modeling of the above problem as follows. In the finite version, called Finite-MTP (FMTP), algorithms are faced to a certain number of finite sequences of requests that have to be served *completely*, that is, algorithms have to arrive to the end of each one of the sequences. FMTP is given by the universe or set of pages U and two positive integers k and w, the size of the cache and the number of sequences respectively. $C = \mathcal{P}_k(U)$ is the set of *configurations*, where $\mathcal{P}_k(U)$ is the power-set of U restricted to subsets of size k. $\sigma \in (U^*)^w$ is the *input tuple*, $\sigma = \sigma_1, \sigma_2, \dots \sigma_w$, where each σ_i is a sequence of requests. We can view σ as a set of sequences of requests; each request is an element of the universe U. The tuple of the j-th requests is called the j-th *row* of requests.

We can imagine that we have a pointer to the current position in each sequence σ_i. In a certain configuration c, the system can advance the pointer of some sequence σ_i such that u_i, the currently pointed page of σ_i, is present in c. Given an input tuple σ and an initial configuration, a *schedule* for σ is a sequence of pairs $< i_j, c_j >$, where $1 \leq i_j \leq w$, such that the currently pointed request of σ_{i_j} may be served in configuration c_j. The *cost* of such schedule is the summation over the sequence of the Hamming distance between successive configurations. At any stage of a schedule, any sequence σ_i whose last request has not been served is called *active*.

An algorithm for FMTP receives a tuple of sequences σ as input and produces as output a schedule for σ. An on-line algorithm must produce the schedule with the restriction that each configuration must be determined only as a function of the current tuple of requests and all the requests already served by the algorithm. On the contrary, an off-line algorithm can decide based on the entire input. An algorithm for FMTP is c-competitive if and only if there exists a constant D such that for any input tuple σ, we have $C_A(\sigma) \leq c \cdot C_{OPT}(\sigma) + D$, where $C_A(\sigma)$ is the cost incurred by the algorithm, and $C_{OPT}(\sigma)$ is the cost incurred by an optimal off-line algorithm.

The definition of FMTP is modified to get the infinite version of the problem, Infinite-MTP (IMTP). In IMTP the sequences of requests are infinite, that is, the input to an algorithm is $\sigma \in (U^\omega)^w$ instead of $\sigma \in (U^*)^w$, where Σ^ω denotes the set of infinite words over an alphabet Σ. In this case, although the sequences are infinite, the system is observed after a finite number ℓ of steps, when we compare the cost incurred by the distinct algorithms with the cost of an optimal off-line algorithm that advances at least the same number of steps. Notice that the value of ℓ is not known by on-line algorithms.

In a multi-threaded environment it is desirable to include fairness restrictions. In MTP these restrictions are modeled by considering, as part of the input of the problem, an integer t such that fair algorithms are restricted to act as follows: from the moment any request is seen till the moment that request is served, at most t other requests can be satisfied. Consequently, the notion of *t-fair* schedule is defined. This allows the definition of the Fair-MTP problem, the fair version of MTP. An algorithm for Fair-MTP must produce a t-fair schedule for the input tuple of sequences. Again there are finite and infinite versions of the problem, namely Fair-FMTP and Fair-IMTP.

2.2 Previous Results

In [6] it was proved that there is no competitive on-line algorithm for Fair-MTP with $t \geq w \geq 2$. This means that competitive on-line algorithms can exist in only two cases of Fair-MTP: $w = 1$ and $t = w - 1$. In the case of finite sequences of requests (that is, Fair-FMTP) there is no competitive on-line algorithm when $w \geq 3$, even with $t = w - 1$. This is because on-line algorithms can be forced to serve a first sequence of length 1; in practice this is equivalent to have $t = w$.

Among the cases where on-line competitiveness is possible, we can ignore the situation in which $w = 1$: with $w = 1$ fairness restrictions have no sense, and we are faced to regular Paging. On the other hand, the case $t = w - 1$ requires further analysis. In this situation the algorithms (on-line or not) must apply round-robin, i. e., serve one request of each sequence in a fixed order which is repeated over and over again. This particular case of Fair-MTP is closely related to normal Paging, since after an algorithm has chosen the order in which to serve the requests, we can think that there is only one sequence to be served, as it is the case in normal Paging. Nevertheless, the two problems are different, as we can see:

- In Fair-MTP we can say that any algorithm has served the same set of requests only after each w new requests (after each row of requests), not at every step as in normal Paging.
- The algorithms (of any type) can choose between $w!$ distinct orders of the sequences. A priori this is an advantage for off-line algorithms, because they can decide with information on the whole input tuple, while any choice made by an on-line algorithm can be fooled if the first two requests of all the sequences are to the same page. In other words, from a competitive analysis point of view, off-line algorithms can really decide in which

order to serve the sequences, while the choice of an on-line algorithm is an illusion.

- The on-line algorithms can see the following w requests to serve, not only one of them. More precisely, on-line algorithms can make use of a lookahead of size w. This is not a clear advantage for on-line algorithms, since this kind of lookahead can be easily neutralized by replacing each request with w requests to the same page.

A lower bound of k for the competitiveness of any on-line algorithm for Fair-MTP can be obtained by considering equal sequences, each one like the nemesis sequence used in the proof of the lower bound for normal Paging.

An on-line algorithm for Fair-MTP called Round-Robin–Flush-When-Full (RRFWF) was presented in [6]. The algorithm is based on Flush-When-Full (FWF), a very well known k-competitive on-line algorithm for Paging [8], although any deterministic marking algorithm for Paging can be used instead (for instance, LRU). FWF maintains a set of marked pages. Initially no page is marked. On each request, an unmarked page is evicted to make room for the requested page if necessary; in any case the requested page is marked. FWF works in *phases*, the first phase starting with the first request of the sequence and each new phase starting with the request that would have caused more than k pages to be marked (when the marks are deleted). It is easy to verify that FWF never faults twice on the same page during any given phase, which implies that its cost is at most k per phase; besides, if we consider all the requests of a phase plus the first request of the following phase, $k + 1$ distinct pages appear.

Algorithm RRFWF for Fair-MTP is described in Fig. 1. The algorithm works in "super-phases"; each super-phase consists of applying a phase of FWF to the sequence formed by taking in turn one request of each active sequence $\sigma_1, \sigma_2, \dots \sigma_w$, and then serving the next request and all the other pending requests in the same row; these additional requests are served by RRFWF in an arbitrary deterministic way. Clearly RRFWF is fair for any legal value of t. It is not difficult to show that it is $(k+w)$-competitive for Fair-IMTP with $t = w-1$. Algorithm RRFWF is $(k + w)$-competitive also for Fair-FMTP with $t = w - 1$, but of course when $w \leq 2$ only.

3 The Case $k = 1$ and Even w

In this section we will analyze the case of Fair-MTP in which the cache can hold only one page and the number of sequences is even. We will consider infinite and finite input sequences, and we will restrict our attention to the situations where on-line competitiveness is possible. As it is usually done in competitive analysis, we will compare on-line strategies against an *adversary* that chooses the input and serves it optimally.

```
While there is at least one request to be served do
      % a new super-phase starts
      Apply a phase of FWF to the sequence $\sigma^*$ formed by taking
      in turn one request of each active sequence $\sigma_1, \sigma_2, \ldots \sigma_w$
      Serve the next request of $\sigma^*$ (no matter how)
      Serve all the pending requests of $\sigma^*$ in the same row
      of the last served request (no matter how)
end While.
```

Figure 1: Algorithm Round-Robin–Flush-When-Full

3.1 Infinite Sequences

Till now the known lower bound for the competitiveness of on-line algorithms for Fair-IMTP with $t = w - 1$ was k; this bound becomes trivial when $k = 1$. We will now prove a higher lower bound for an even number of sequences, using that the adversary can choose an appropriate order for the threads.

Theorem 3.1.1 *No on-line algorithm for Fair-IMTP with $t = w - 1$, $k = 1$ and even w, is c-competitive if $c < w + 1$, even if we restrict the sequences of requests to be formed by at most 2 distinct pages.*

Proof Let A be any on-line algorithm and ADV an off-line adversary. We will show a set of sequences $\sigma_1, \sigma_2, \ldots \sigma_w$ for which A cannot behave better than the proposed bound.

Let $U = \{a_1, a_2\}$ be a set of 2 distinct pages, where a_1 is a page not in A's cache at the beginning. We will describe the input tuple by rows of requests. The first row is formed by page a_1 for the odd sequences $\sigma_1, \sigma_3, \ldots \sigma_{w-1}$, and by page a_2 for the even sequences $\sigma_2, \sigma_4, \ldots \sigma_w$. In the second row, all the requests are to page a_1. Each new pair of rows is like the previous one, but with pages a_1 and a_2 interchanged (see Fig. 2). Let $\ell = 2wn$ be the total number of requests to be served, where n is any positive integer. Without loss of generality assume that A serves one request of each sequence in the order $\sigma_1, \sigma_2, \ldots \sigma_w$ (otherwise, we add two initial rows with requests only to page a_2, and rearrange the sequences if necessary). Since $k = 1$, only one page can be held in fast memory at a time, and therefore A necessarily faults on each request of the odd rows and on each first request of the even ones; thus, its total cost is $C_A \geq (w + 1)n$.

The adversary can use any order in which the even sequences appear before the odd ones, e. g., $\sigma_2, \sigma_4, \ldots \sigma_w, \sigma_1, \sigma_3, \ldots \sigma_{w-1}$. In this way it groups the repeated requests, and faults only once every two rows. Then the total cost of the adversary is $C_{ADV} \leq n$, and we have for the competitive ratio

$$\frac{C_A}{C_{ADV}} \geq \frac{(w+1)n}{n} = w + 1 \ .$$

σ_1	$\cdots$	σ_{odd}	σ_{even}	$\cdots$	σ_w
a_1		a_1	a_2		a_2
a_1		a_1	a_1		a_1
a_2		a_2	a_1		a_1
a_2		a_2	a_2		a_2
a_1		a_1	a_2		a_2
a_1		a_1	a_1		a_1
$\vdots$		$\vdots$	$\vdots$		$\vdots$

Figure 2: Sequences used in the proof of Theorem 3.1.1

□

As we can see the above lower bound is tight, since algorithm RRFWF described in Section 2 is $(k+w)$-competitive for Fair-IMTP with $t = w-1$.

Corollary 3.1.2 *Algorithm RRFWF is strongly competitive for Fair-IMTP with $t = w-1$, $k = 1$ and even w.*

3.2 Finite Sequences

We can reproduce in the finite model the results we have obtained for infinite sequences. The following lower bound for Fair-FMTP with $t = w-1$ and $k = 1$ is only interesting when the number of sequences is $w = 2$.

Theorem 3.2.1 *No on-line algorithm for Fair-FMTP with $t = w-1$, $k = 1$ and even w, is c-competitive if $c < w+1$, even if we restrict the sequences of requests to be formed by at most 2 distinct pages.*

Proof The proof is the same of Theorem 3.1.1, except that we use finite sequences, all of them of the same length. □

Again the above lower bound is tight for the cases in which competitive on-line algorithms can exist, since algorithm RRFWF is $(k+w)$-competitive for Fair-FMTP with $t = w-1$ and $w \leq 2$.

Corollary 3.2.2 *Algorithm RRFWF is strongly competitive for Fair-FMTP with $t = w-1$, $k = 1$ and $w = 2$.*

4 The Case $k = 1$ and Odd w

In this section we will consider an odd number of sequences for Fair-MTP with $t = w-1$ and $k = 1$. This completes the analysis of Fair-MTP when the size of the cache is 1. As we will see, lower and upper bounds differ from the ones obtained for an even number of sequences. Nevertheless, again we will show an optimal on-line algorithm. Moreover, we will prove that any reasonable algorithm is optimal. The only case of Fair-FMTP with an odd number of sequences in which competitive on-line algorithms can exist is regular Paging ($w = 1$); so we will omit a useless discussion, and we will treat only the infinite model.

4.1 Infinite Sequences

We start with a lower bound for the competitive ratio of on-line algorithms.

Theorem 4.1.1 *No on-line algorithm for Fair-IMTP with $t = w - 1$, $k = 1$ and odd w, is c-competitive if $c < w$, even if we restrict the sequences of requests to be formed by at most 2 distinct pages.*

Proof If $w = 1$ then there is nothing to prove. For $w \geq 3$ the proof is as follows. The first $w - 1$ sequences are constructed as in Theorem 3.1.1. Sequence σ_w is a copy of σ_1. Without loss of generality assume again that the on-line algorithm A serves the sequences in the order $\sigma_1, \sigma_2, \dots \sigma_w$. In this situation A faults on each request of the odd rows (note that the even rows are redundant). Then its cost for $\ell = 2wn$ requests is $C_A \geq wn$. The adversary can group the repeated requests as in Theorem 3.1.1 faulting only once every two rows, with a total cost $C_{ADV} \leq n$. Therefore we have

$$\frac{C_A}{C_{ADV}} \geq \frac{wn}{n} = w \ .$$

□

In normal Paging with $k = 1$ there is no particular strategy that a good algorithm must follow: if the requested page is in the cache, then nothing is done; otherwise the page must be brought into fast memory, replacing the only page that is there. Any algorithm that does not unnecessarily evict pages is optimal. In Fair-IMTP with $t = w - 1$ and $k = 1$ the situation is the same, except that the algorithms can choose between $w!$ distinct orders of the sequences. We said that this choice is useless for an on-line algorithm, so it makes sense that any on-line algorithm is strongly competitive. We will now see that this is the case, at least for an odd number of sequences. We are able to prove that any lazy algorithm (on-line or not) for Fair-IMTP with $t = w - 1$, $k = 1$ and odd w, is w-competitive. A lazy algorithm is one which only evicts a page on a page fault, and in that case evicts exactly one page.

Theorem 4.1.2 *Any lazy algorithm for Fair-IMTP with $t = w - 1$, $k = 1$ and odd w, is w-competitive.*

Proof Let A be any lazy algorithm (on-line or not) and ADV an off-line adversary. To measure the costs of both algorithms we will split the rows of requests into disjoint *stages.* The first stage starts with the first row and ends with the first row in which the adversary has no cost. Each new stage starts with the row immediately after the last row of the previous stage, and ends with the next row in which again the adversary has no cost. We will consider that the i-th stage starts and ends with the p_i-th and the q_i-th rows of requests, respectively. We will denote by C_A^i and $C_A(j)$ the costs of A in the i-th stage and in the j-th row, respectively. A similar notation is valid for the adversary. Notice that $(\forall j)\, C_A(j) \leq w$.

Suppose that after ℓ requests the algorithms completed m stages and are currently in the $(m+1)$-st stage. Before we measure the costs of the algorithms, it is convenient to point out some facts about the stages we have defined. In the completed stages $(i \leq m)$, the last row verifies $C_{ADV}(q_i) = 0$, which implies that all the requests in the row are to the same page, and hence $C_A(q_i) \leq 1$; besides, both A and the adversary start the next stage with the same contents in the cache. On the other hand, in each row r but the last one $(p_i \leq r < q_i)$ we have $C_{ADV}(r) \geq 1$; this is also true for the completed rows of the $(m+1)$-st stage, because otherwise we would have an additional stage.

We will now analyze the costs in the different stages:

- In the first stage, all the rows but the last one verify $C_{ADV}(r) \geq 1$, and then we have
$$C_A^1 \leq wC_{ADV}^1 + w \ .$$
- In the last stage, all the rows except eventually the last one verify the same condition $C_{ADV}(r) \geq 1$, and then we have
$$C_A^{m+1} \leq wC_{ADV}^{m+1} + w - 1 \ .$$
- Let s be an intermediate stage $(2 \leq i \leq m)$ for which A does not pay in the last row of the stage. Again all the other rows verify $C_{ADV}(r) \geq 1$, and then we have
$$C_A^i \leq wC_{ADV}^i \ .$$
- We must now consider the costs in each intermediate stage $(2 \leq i \leq m)$ for which A pays in the last row of the stage. The only possibility is that $C_A(q_i) = 1$, since we know that $C_A(q_i) \leq 1$. First notice that the algorithms start the last row of the stage with different contents in their caches; this is because all the requests in that row are to the same page, while $C_A(q_i) \neq C_{ADV}(q_i)$ $(1 \neq 0)$. It follows that the stage has at least two rows of requests. We claim that among the rows that are not the last one of the stage, there exists at least one row which verifies $C_A(t_i) \leq w - 1$ or $C_{ADV}(t_i) \geq 2$. Suppose this is not true, i. e., for each row we have $C_A(r) = w$ and $C_{ADV}(r) = 1$, in particular for the first row of the stage.

Let a_i be the page in the caches of both A and the adversary when they start serving the stage. Since $C_{ADV}(p_i) = 1$, from the adversary's point of view, this first row is formed by zero or more requests to page a_i, followed by one or more requests to another page b_i ($b_i \neq a_i$). On the other hand, since $C_A(p_i) = w$, for A the row looks like alternating requests to pages a_i and b_i, starting and ending with b_i because A faults on each request and w is odd. Hence, once the first row of the stage is completed, we are as in the beginning of the stage, in the sense that both A and the adversary has the same contents in the cache (now it is page b_i). We can think about the second row of the stage like we did about the first, and so on, until we arrive to the last row of the stage. This last row starts with the same contents in the cache of both algorithms, but we said it cannot happen. This is a contradiction, and therefore our claim is true. If $C_A(t_i) \leq w-1$ we have

$$C_A(t_i) + C_A(q_i) \leq (w-1) + 1 = w \leq wC_{ADV}(t_i) =$$

$$= w\left[C_{ADV}(t_i) + C_{ADV}(q_i)\right] \ ,$$

while if $C_{ADV}(t_i) \geq 2$

$$C_A(t_i) + C_A(q_i) \leq w + 1 \leq 2w \leq wC_{ADV}(t_i) =$$

$$= w\left[C_{ADV}(t_i) + C_{ADV}(q_i)\right] \ .$$

Since the row we found verifies at least one of those conditions, and the other rows in the stage (except the last one) verify $C_{ADV}(r) \geq 1$, again we have

$$C_A^i \leq wC_{ADV}^i \ .$$

The total costs C_A and C_{ADV} are the sums of the costs in each stage, and hence

$$C_A = \sum_{i=1}^{m+1} C_A^i \leq \left(\sum_{i=1}^{m+1} wC_{ADV}^i\right) + 2w - 1 = wC_{ADV} + (2w-1) \ ,$$

that is, A is w-competitive. □

The preceding proof cannot be applied to an even number of sequences; the problem arises with the last kind of stages we considered, where the induction is not valid for an even w.

It is worthwhile to mention that the lower bounds shown in this and the previous section can be generalized to arbitrary values of k, so as to obtain a general lower bound of (roughly) w/k. This new lower bound is linear in the number of sequences, and when $w > k^2$ it is better than the already known lower bound of k.

```
While there is at least one request to be served do
      % a new super-phase starts
      Repeat
             Apply a phase of FWF to the sequence $\sigma^*$ formed by
             taking in turn one request of each active sequence
      Until  it is possible to serve with unitary cost
             the next request of $\sigma^*$ and all the pending requests
             in the same row of that request
      Serve the next request of $\sigma^*$ and all the pending requests
      in the same row of that request (with unitary cost)
end While.
```

Figure 3: Algorithm Check-Row–Flush-When-Full

5 The Case $w \leq k$

In this section we will analyze the case of Fair-MTP in which $w \leq k$. This case is the opposite situation to the extreme case of the two previous sections (where we considered $k = 1$). We will present a new on-line algorithm for Fair-MTP and we will show that for $w \leq k$ it behaves better than RRFWF, the known on-line algorithm for Fair-MTP.

5.1 Infinite Sequences

The new on-line algorithm for Fair-MTP is called Check-Row–Flush-When-Full (CRFWF), and it is described in Fig. 3. Algorithm CRFWF can be regarded as a modification of RRFWF: CRFWF works in super-phases and serves the requests row by row as RRFWF does; the difference is that in each super-phase RRFWF applies one phase of FWF and serves some additional requests (see Fig. 1), while CRFWF applies one or more phases of FWF until it is possible to serve the additional requests with unitary cost. As we will see now, CRFWF beats RRFWF in Fair-IMTP with $t = w - 1$ and $w \leq k$.

Theorem 5.1.1 *Algorithm CRFWF is $(k+1)$-competitive for Fair-IMTP with $t = w - 1$ and $w \leq k$.*

Proof Assume that after ℓ requests CRFWF completed m super-phases, each one having $p_i \geq 1$ phases $(1 \leq i \leq m)$, and is currently in the $(m+1)$-st super-phase with $p_{m+1} \geq 0$ phases completed in this last super-phase. To terminate a super-phase CRFWF entirely serves its last row; thus the adversary has served all the requests in the super-phases that CRFWF has completed. Consider the i-th of those super-phases $(1 \leq i \leq m)$. By definition of CRFWF, the cost of each phase is at most k, and then the cost for the super-phase is

$$C^i_{CRFWF} \leq p_i k + 1 \leq (k+1)p_i \ .$$

To analyze the adversary's cost in the super-phase, we associate to each odd phase an *xphase* (extended phase) as follows: we include in the xphase all the requests of the odd phase plus the next request served by CRFWF, and all the requests in the rows of the already included requests; note that we include in the xphase all or none of the requests of any given row. Being $w \leq k$, each phase has at least w requests, and then the xphases are disjoint. From the behavior of FWF we know that in each xphase at least $k+1$ different pages appear; this implies that the adversary must fault at least once in each xphase. If an xphase is not associated with the last phase of the super-phase, then the number of distinct pages is at least $k+2$ (because otherwise the super-phase would finish in that xphase); in these xphases the adversary must fault at least twice. If p_i is even, there is no xphase associated with the last phase of the super-phase; then the cost of the adversary in the super-phase is

$$C^i_{ADV} \geq 2\frac{p_i}{2} = p_i \ .$$

If p_i is odd, the last xphase corresponds to the last phase of the super-phase, and hence the cost is

$$C^i_{ADV} \geq 2\frac{p_i - 1}{2} + 1 = p_i \ .$$

In both cases we have for the costs in the super-phase

$$C^i_{CRFWF} \leq (k+1)p_i \leq (k+1)C^i_{ADV} \ .$$

In the $(m+1)$-st super-phase the cost of CRFWF is

$$C^{m+1}_{CRFWF} \leq p_{m+1}k + k \leq (k+1)p_{m+1} + k \ .$$

We can think about the adversary's cost in this super-phase as we did for the other super-phases. Nevertheless we must discard the last phase of the super-phase, because we do not know whether the adversary has served those requests. Thus we have

$$C^{m+1}_{ADV} \geq p_{m+1} - 1 \ ,$$

which implies

$$C^{m+1}_{CRFWF} \leq (k+1)p_{m+1} + k = (k+1)\left(p_{m+1} - 1\right) + 2k + 1 \leq (k+1)C^{m+1}_{ADV} + 2k + 1 \ .$$

Summing over all the super-phases we obtain that the total costs C_{CRFWF} and C_{ADV} verify

$$C_{CRFWF} = \sum_{i=1}^{m+1} C^i_{CRFWF} \leq (k+1)\sum_{i=1}^{m+1} C^i_{ADV} + 2k + 1 = (k+1)C_{ADV} + (2k+1) \ .$$

□

Table 1: Summary of previous and current results about Fair-IMTP (Im = Improved; Cl = Closed).

	Previous		Current			
	l.b.	u.b.	l.b.	u.b.	Im?	Cl?
$t \geq w$	∞	–	∞	–		$\surd$
$t = w - 1$, $k = 1$ and even w	1	$w + 1$	$w + 1$	$w + 1$	$\surd$	$\surd$
$t = w - 1$, $k = 1$ and odd w	1	$w + 1$	w	w	$\surd$	$\surd$
$t = w - 1$ and $w \leq k$	k	$k + w$	k	$k + 1$	$\surd$	
$t = w - 1$ and $w > k > 1$	k	$k + w$	k	$k + w$		

5.2 Finite Sequences

To conclude our analysis we will prove that algorithm CRFWF is $(k + 1)$-competitive also for Fair-FMTP with $w \leq k$. Of course we are restricted to the situations in which competitive on-line algorithms can exist.

Theorem 5.2.1 *Algorithm CRFWF is $(k+1)$-competitive for Fair-FMTP with $t = w - 1$, $w \leq k$ and $w \leq 2$.*

Proof With two active sequences CRFWF behaves as in the infinite version, and hence its cost is at most $k + 1$ times the optimal cost necessary to serve those requests. The performance of CRFWF is the same with only one active sequence. In any case the algorithm results $(k + 1)$-competitive. □

6 Concluding Remarks

Table 1 summarizes our results about Fair-IMTP, as well as the results already known; we assume $w > 1$ through the table. In the finite model the results are the same, except that no competitive on-line algorithm can exist if $w \geq 3$.

When the size of the cache is 1, having an even number of threads is slightly more difficult for on-line algorithms than the case in which the number of threads is odd. The difficulty probably depends not only on the parity of w itself, but on its relationship with the size of the cache. The fact that both w and k affect the behavior of on-line algorithms becomes clear when we compare the results for the two opposite situations we have considered: when $k = 1 < w$ there is a tight bound near to w, while if $w \leq k$ on-line competitiveness is between k and $k+1$. It seems that when the cache is small the adversary can take benefit of its decision about the order in which to serve the sequences; thus the performance of on-line algorithms get worse when the number of sequences increases. On the other hand, when the cache is big enough the permutation of the threads might have a local effect, and then the performance that on-line algorithms can achieve is similar to the one observed in normal Paging.

Although we improved previous results and we closed some gaps, there are still differences between current lower and upper bounds; a nice result would be to close those gaps. We proved that any lazy algorithm achieves the on-line lower bound if $k = 1$ and w is odd; we know that algorithm RRFWF is optimal when the number of sequences is even, but it would be interesting to analyze whether that general result is valid in this case too.

Several interesting research directions are possible. One of them is modeling fairness restrictions in a different way; perhaps alternative models allow the existence of more flexible competitive on-line algorithms. Distinct definitions of competitiveness may also be considered for the infinite problem, such as comparing the performances of the different algorithms in the limit, that is, when the number of served requests tends to infinity. Another interesting possibility is to analyze randomized algorithms for these problems.

A general observation is that multi-threaded environments seem a powerful modeling tool and a challenging research field.

Acknowledgments: I would like to thank Esteban Feuerstein for his encouragement and many helpful suggestions about this work.

References

[1] H. Alborzi, E. Torng, P. Uthaisombut, and S. Wagner. The k-client problem. In *Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 73–82, New Orleans, Louisiana, 5–7 January 1997.

[2] S. Ben-David, A. Borodin, R. Karp, G. Tardos, and A. Widgerson. On the power of randomization in online algorithms. Technical Report TR-90-023, ICSI, June 1990.

[3] A. Borodin, S. Irani, P. Raghavan, and B. Schieber. Competitive paging with locality of reference. In *Proc. of 23rd ACM Symposium on Theory of Computing*, pages 249–259, 1991.

[4] A. Borodin, N. Linial, and M. Saks. An optimal on-line algorithm for metrical task system. *J. ACM*, 39(4):745–763, 1992.

[5] E. Feuerstein. *On-line Paging of Structured Data and Multi-threaded Paging.* PhD thesis, Università degli Studi di Roma "La Sapienza", 1995.

[6] E. Feuerstein and A. Strejilevich de Loma. On multi-threaded paging. In *Proc. Seventh Annual International Symposium on Algorithms and Computation (ISAAC'96)*, volume 1178 of *Lecture Notes in Computer Science*, pages 417–426. Springer-Verlag, 1996.

[7] A. Fiat and A. Karlin. Randomized and multipointer paging with locality of reference. In *Proc. of 27th ACM Symposium on Theory of Computing*, 1995.

[8] A. Karlin, M. Manasse, L. Rudolph, and D. Sleator. Competitive snoopy caching. *Algorithmica*, 3:79–119, 1988.

[9] R. Karp. On-line algorithms versus off-line algorithms: how much is it worth to know the future? Technical Report TR-92-044, ICSI, July 1992.

[10] D.D. Sleator and R.E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of ACM*, 28:202–208, 1985.

[11] A. Strejilevich de Loma. New results on fair multi-threaded paging. In *Proc. First Argentine Workshop on Theoretical Informatics (WAIT'97)*, pages 111–122. SADIO, 1997.

[12] A. Strejilevich de Loma. New results on fair multi-threaded paging. Technical Report TR 97-005, Universidad de Buenos Aires, Departamento de Computación, July 1997. http://www.dc.uba.ar/people/proyinv/tr.html.