OnlineMIN: A Fast Strongly Competitive Paging Algorithm

### Paging and Competitive Analysis

#### Paging
- Setup: a cache of size $k$ and a memory of infinite size
- Process pages sequentially online (no information about future)
- Current page:
  - Cache hit – page is in cache: move to next page
  - Cache miss – page is not in cache: bring it in cache
  - Cache is full: evict some page to make room
- Objective: minimize #misses

#### Competitive analysis
- Compare online algorithm against optimal cost OPT
- An algorithm $A$ has competitive ratio $c$ if
  \[
  \text{cost}(A) \leq c \times \text{cost}(OPT)
  \]

### Previous work

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Comp. ratio</th>
<th>Space</th>
<th>Time per page</th>
</tr>
</thead>
<tbody>
<tr>
<td>LRU, FIFO</td>
<td>$k$</td>
<td>$O(k)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Mark</td>
<td>$2H_k$</td>
<td>$O(k)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Partition</td>
<td>$H_k$</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>Equitable</td>
<td>$H_k$</td>
<td>$O(k^2 \log k)$</td>
<td>$O(k^2)$</td>
</tr>
<tr>
<td>Equitable2</td>
<td>$H_k$</td>
<td>$O(k)$</td>
<td>$O(k^2)$</td>
</tr>
</tbody>
</table>

#### OnlineMIN
- $H_k$-competitive, $O(k)$ space, $O(\log k)$ time per page

### Selection Process

#### Selection process
- Assume pages have random priorities
- Build sets $C_0, \ldots, C_k$ as follows
  - $C_0 = \emptyset$
  - $C_i$ has the $i$ pages in $C_{i-1} + L_i$ having largest priorities
  - The cache of the algorithm is always $C_k$

#### Example ($k = 5$)
- $C_1 = \{8\}$, $C_2 = \{4,8\}$, $C_3 = \{8,9,10\}$, $C_4 = \{7,8,9,10\}$, $C_5 = \{2,7,8,9,10\}$

- Same distribution as Equitable2, and thus $H_k$-competitive!

### Roadmap

- Layer Partitioning
  - Random selection
  - Layer Partitioning
  - Random selection
  - Cache OnlineMIN
  - Cache Equitable
- $O(\log k)$ time
- Same probability distribution
- $O(k^2)$ time

### Implementation

- OnlineMIN
  - Upon processing page $p$:
    - Update cache if cache miss:
      - If $p$ in $L_0$, evict page in cache having smallest priority
      - If $p$ in $L_i$ ($i > 0$):
        - Find smallest $j > i$ s.t. first $j$ layers have $j$ pages in cache
        - Evict the page in the first $j$ layers having smallest priority
    - Update layers as previously described
- Analysis
  - $O(k)$ space per Equitable2, $O(\log k)$ time per smart data structures

### References