Engineering Efficient Paging Algorithms

Andrei Negoescu

Goethe University Frankfurt

June 8, 2012

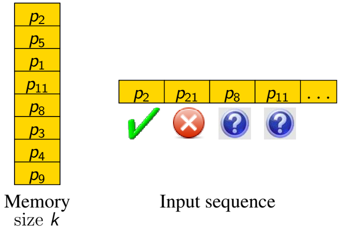
Joint work with G. Moruz, C. Neumann and V. Weichert

CENTER FOR MASSIVE DATA ALGORITHMICS

Deutsche Forschungsgemeinschaft

DFG

Paging



- Cache of size k
- Page requests online (no information about future)
- Page fault - required page not in memory
  - Bring it in memory; replace some other page
- Minimize # page-faults

Quality Aspects of Paging Strategies

1. **Theoretical worst case guarantees**
   - OPT is the optimal offline solution
   - A is c-competitive: $cost(A) \leq c \cdot cost(OPT) + b$
   - Good algorithms: many

2. **Empirical competitive ratio**
   - Ratio $cost(A)/cost(OPT)$ on real-world traces
   - LRU is good and simple
   - RDM, EELRU, FARL outperform LRU but are more complex

3. Today's talk: **Overall performance**
   - Consider: Page faults and runtime
   - Simple algorithms: LRU (few page faults), FIFO (very fast)
   - We show: RDM is a good alternative

Simple Paging Algorithms

LRU

- Replace the page least recently used
- Competitive ratio: k (optimal)
- Empirical competitive ratio: ≈ 4
- Can be implemented in $O(1)$ per request

FIFO

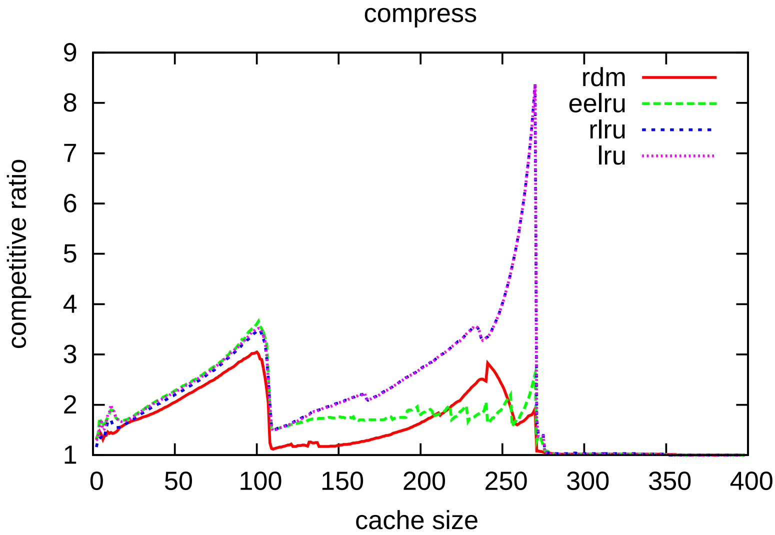
- Replace the page which entered first the cache
- Competitive ratio: k (optimal)
- Empirical performance: worse than LRU
- $O(1)$ per request, faster than LRU

RDM - Main Idea

[Moruz and N. '12]

- Priority assignment for page upon request
- Keeps OPTs cache content, given
  - Sequence seen so far
  - Priority based future prediction
- RDM - Priority assignment
  - Recency: current timestamp
  - Duration: priority $=$ time in OPTs cache
  - Mix: priority $= 0.8 \cdot \textit{recency} + 0.1 \cdot \textit{duration}$

Competitiveness of RDM



RDM - Layer Partition

[Koutsoupias and Papadimitriou '94]

- Tracks cache configurations of optimal solution
- Splits pageset in $k + 1$ disjoint layers $L_0, \ldots, L_k$
  - Implementation: $k + 1$ consecutive time intervals
- OPT caches at most i pages from $L_1 \cup \cdots \cup L_i$
- Update after each page request p
  - Find the layer containing p
  - Merge two consecutive layers
  - Create a new rightmost layer

Example $k = 4$: $\{1, 3, 5\}_0 \quad \{2\}_1 \quad \{6, 8\}_2 \quad \{4\}_3 \quad \{7\}_4$

RDM - Layer Partition

- Three types of requests
  - Revealed – for sure in OPT's cache (rightmost singletons)
  - OPTmiss – for sure not in OPT's cache (L_0)
  - Unrevealed – maybe in OPT's cache (or maybe not)

Example $k = 4$: $\{\boxed{1}, \boxed{3}, \boxed{5}\}_0 \quad \{\boxed{2}\}_1 \quad \{\boxed{6}, \boxed{8}\}_2 \quad \{\boxed{4}\}_3 \quad \{\boxed{7}\}_4$

RDM

- **Cache Update**
  - if $p \in M$ do nothing
  - if $p \notin M$
    - $p \in L_0$: evict page in M having smallest priority
    - $p \in L_i$ and $p \notin M$:
      - identify $j \geq i$ s.t. $|M \cap (L_1 \cup \cdots \cup L_j)| = j$;
      - evict page in this set having smallest priority
- **Layer update** (each request)

- **Runtime**
  - $\Theta(\log k)$ per request using balanced binary trees
  - High constants
  - Implementig it: very slow compared to LRU

Outline

- New compressed layer partition
- A simpler RDM implementation based on arrays
- Runtime experiments
  - RDM: array vs. tree implementation
  - RDM versus LRU and FIFO
  - Overall performance including page fault penalties

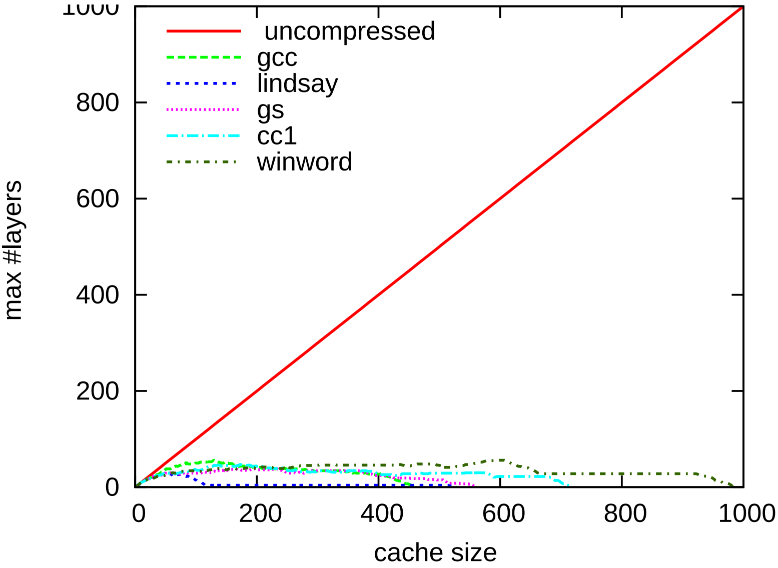
RDM - Compressed Layers

- Observations
  - $> 99\%$ of requests are revealed
  - Layer update faster if we decrease amount of non-empty layers
- New Layer Partition (compressed)
  - Still: OPT caches at most i pages from $L_1 \cup \dots \cup L_i$
  - Difference: No update on revealed requests
  - Difference: $\#$ Non-empty Layers $\leq k + 1$

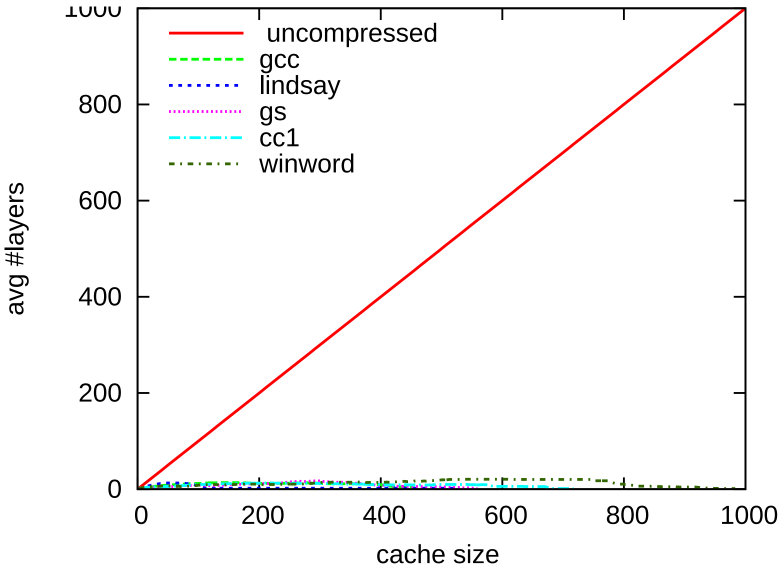
Old: $\{1, 3, 5\}_0 \quad \{2\}_1 \quad \{6, 8\}_2 \quad \{4\}_3 \quad \{7\}_4$

New: $\{1, 3, 5\}_0 \quad \{\}_1 \quad \{2, 6, 8\}_2 \quad \{\}_3 \quad \{4, 7\}_4$

#Non-empty Layers – Max



#Non-empty Layers – Average



RDM - Array Implementation

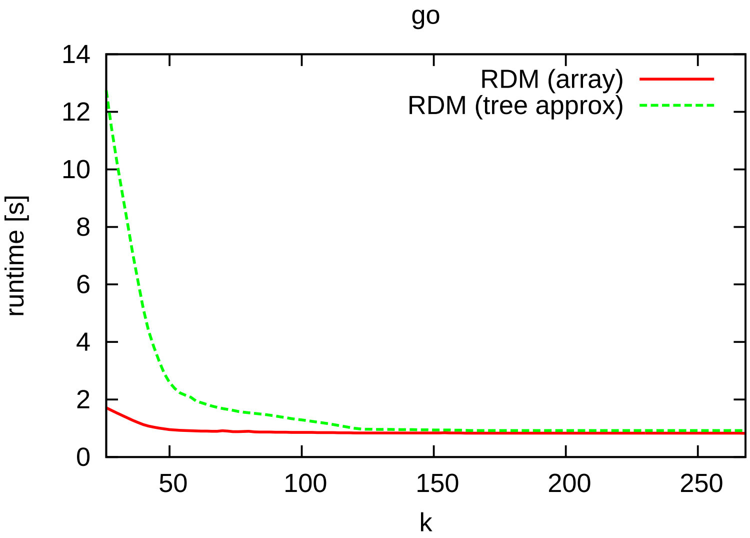
An alternative implementation using arrays

- Store $O(1)$ data per page, timestamp of last request in $O(1)$
- L non-empty layers: interval delimiters in a sorted array
- All operations done by linear traversal

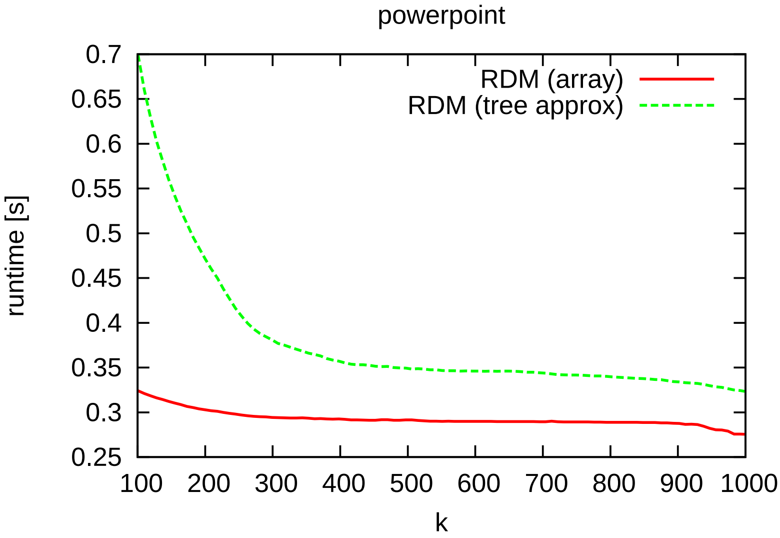
Upon request of page p

- $p \in M$ and revealed: do nothing $> 99\%$ requests $O(1)$
- $p \in M$ and not revealed: Update layers $O(L)$
- $p \notin M$ $O(k)$
  - ...

Runtime RDM implementations



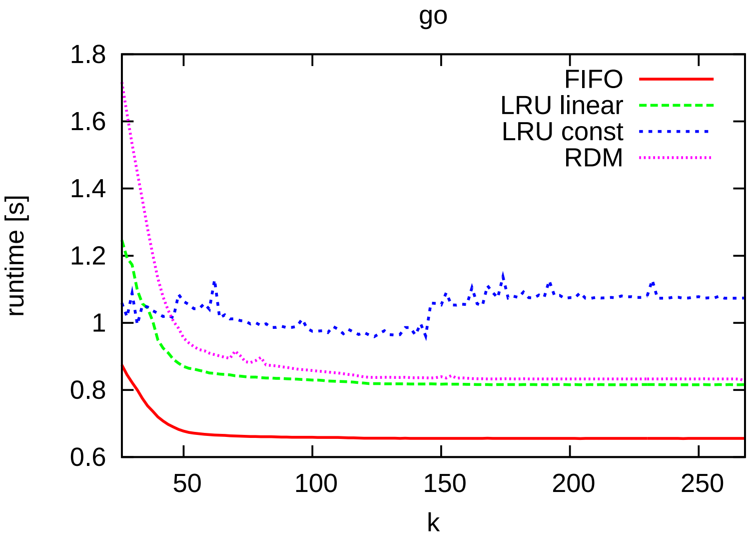
Runtime RDM implementations



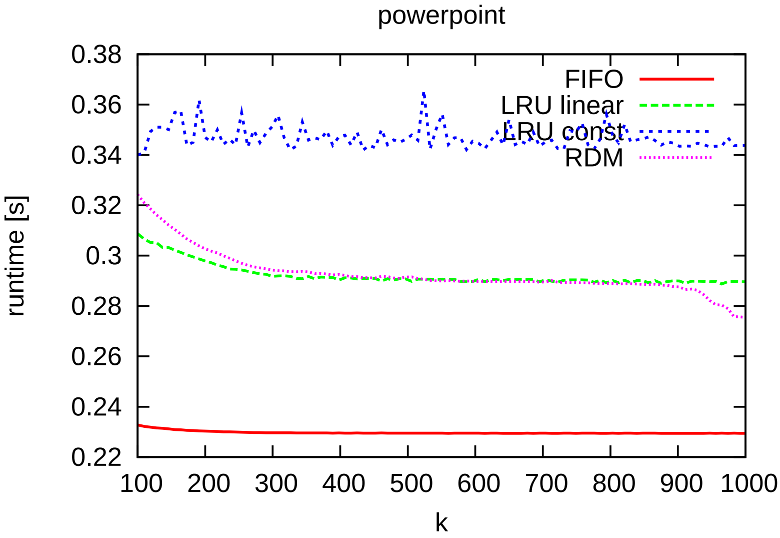
RDM, LRU, and FIFO

- FIFO
  - Circular array
  - $O(1)$ time per page request
- Constant time LRU
  - A recency list – updated for each request
  - $O(1)$ time per page request
- Linear LRU
  - An array storing pages in cache
  - For hits do nothing, misses take $O(k)$ time
- RDM
  - Array version

Runtime RDM vs LRU and FIFO



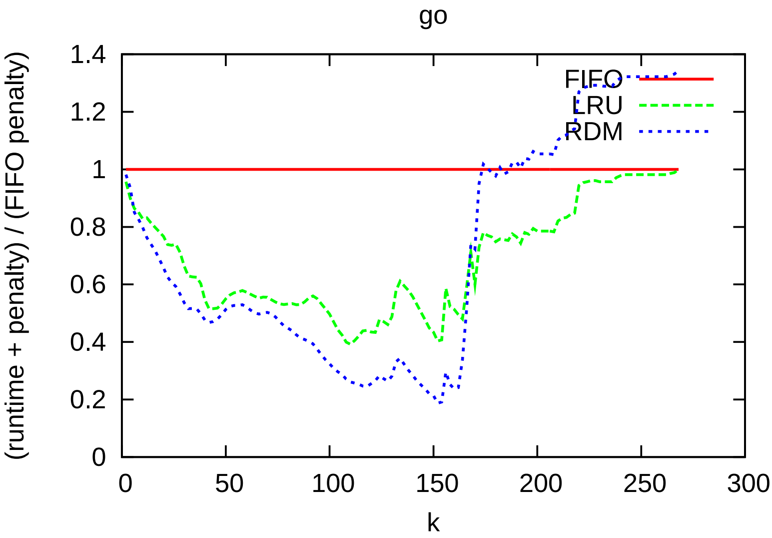
Runtime RDM vs LRU and FIFO



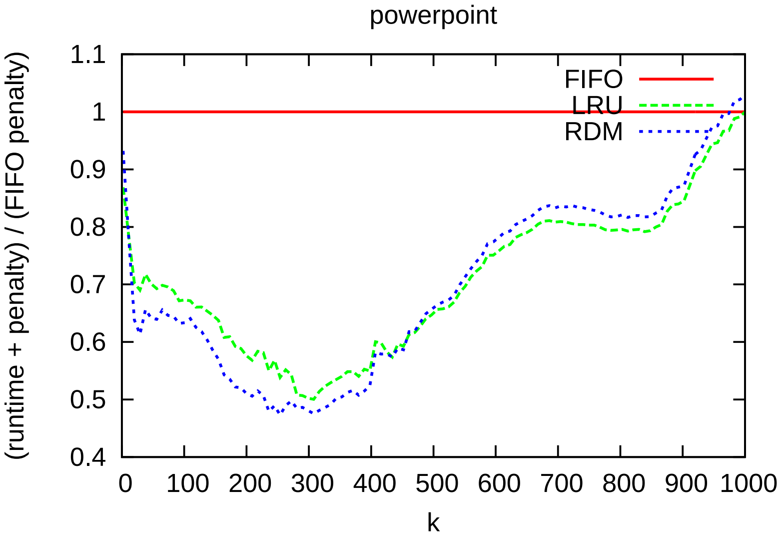
Overall Performance

- Assign page fault a cost of 9 ms
- Penalty $=$ #page faults $\times$ 9ms
- Total runtime $=$ Effective runtime $+$ penalty
- For the algorithms considered:
  - RDM: Effective runtime $+$ penalty
  - FIFO: 0 $+$ penalty
  - LRU: 0 $+$ penalty

Total Runtime (including penalties)



Total Runtime (including penalties)



Conclusions

- Compressed Layers lead to faster implementations
- FIFO is the fastest
- RDM implementation can compete with LRU
- RDM may be of practical interest