Adaptive Caching by Refetching

Manfred K. Warmuth

University of California at Santa Cruz, USA

http://www.cse.ucsc.edu/~manfred

August 12, 2003 - HP Labs - Palo Alto

Joint work with:
Robert B. Gramacy and Scott A. Brandt

Thanks to Systems Group at UCSC:
Ahmed Amer, Ismail Ari, Darrel D. E. Long, Ethan L. Miller

Caching

- Whenever small, fast memory
  and larger, slower secondary memory
- Keep objects in faster memory
  which likely to be needed again soon
  - Hit if requested object resides in cache
  - Miss otherwise

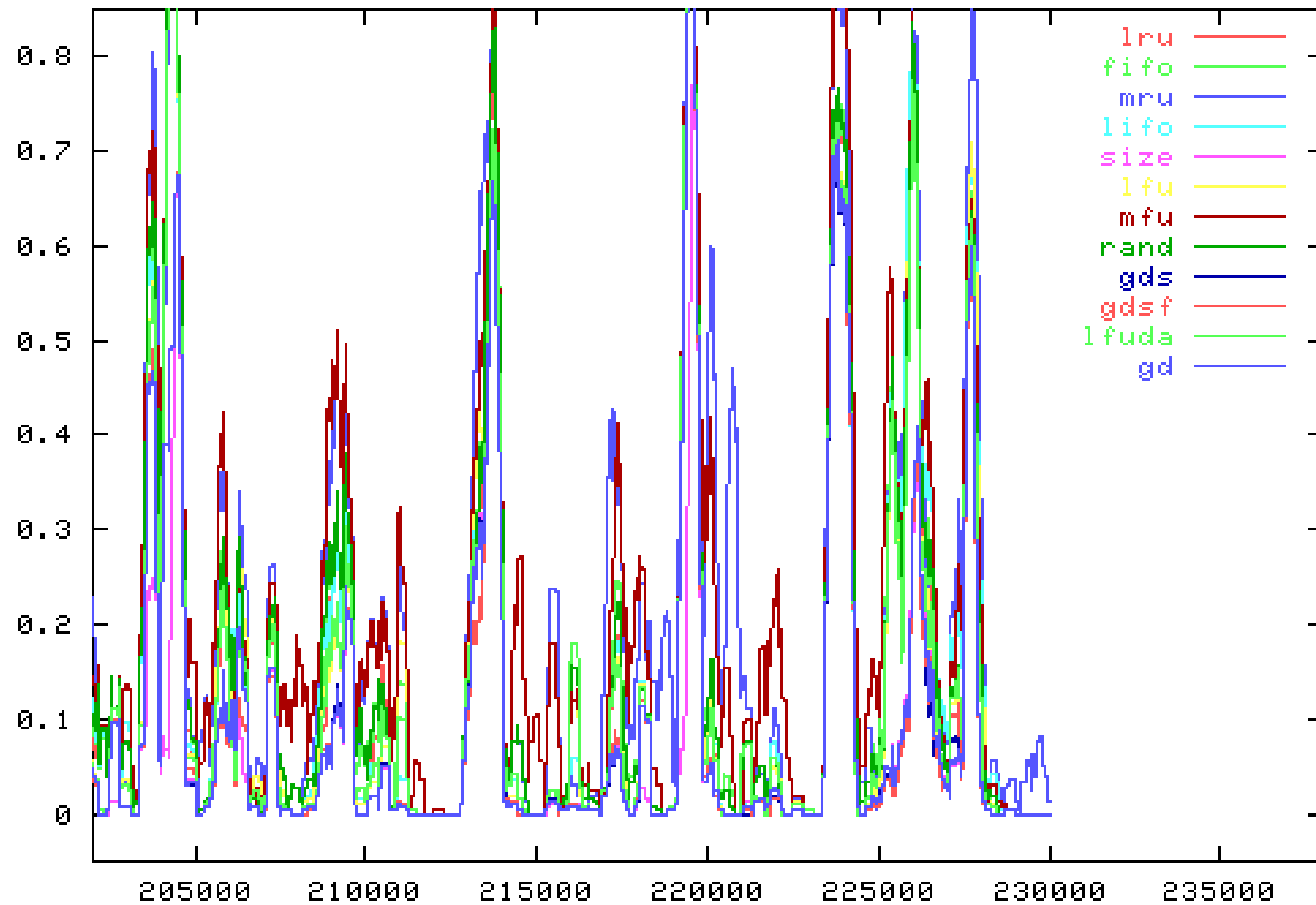
Caching Policies

- Decides which objects to discard
  to make room for new requests
- 7 common policies: LRU, RAND, FIFO, LIFO, LFU and MFU
- 5 fancy recent policies: SIZE, GDS, GD*, GDSF, LFUDA
- Criteria:
  - Recency and frequency of access
  - Size of objects
  - Cost of fetching object from secondary memory
- De facto standard: LRU

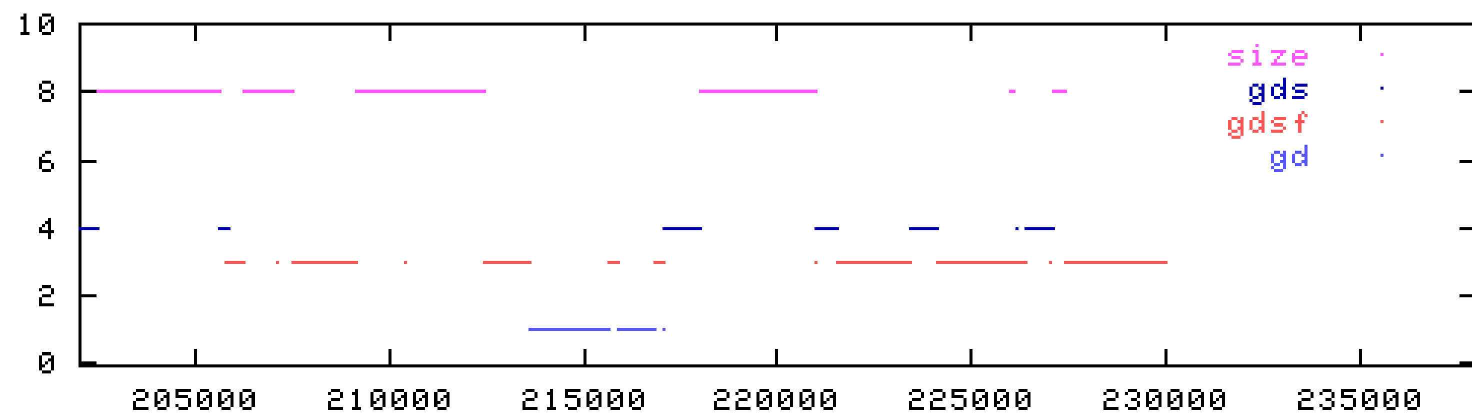
Which Policy to Choose?

- For which situation?
  - Disk access on PC
  - Web proxy access via browser
  - File server on local network
  - Middle of the night - during backup
  - Application as well as time dependent
- Choosing one is suboptimal

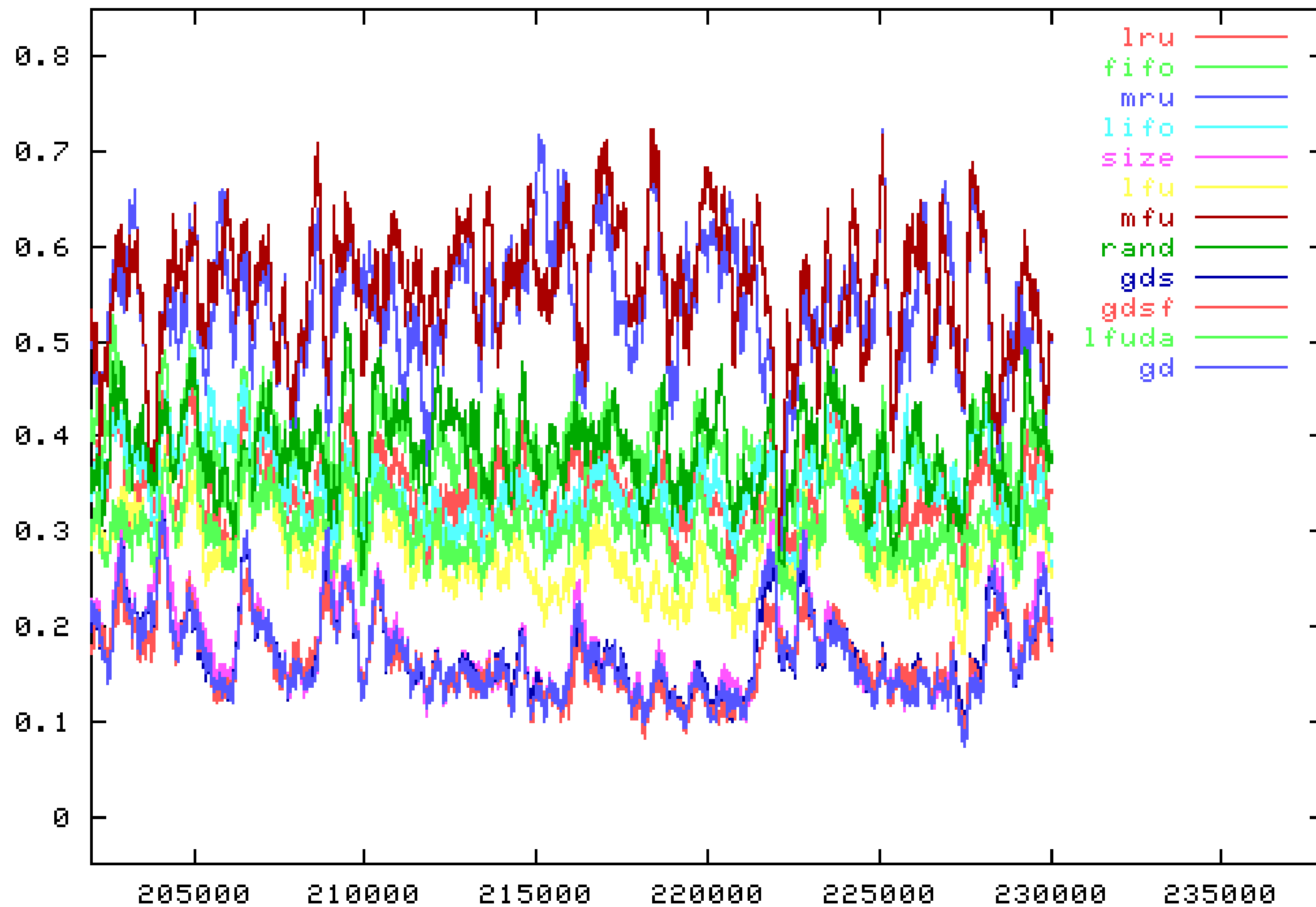
Characteristics Vary with Time



Best Policy Varies with time



Randomly Permuted Request Stream

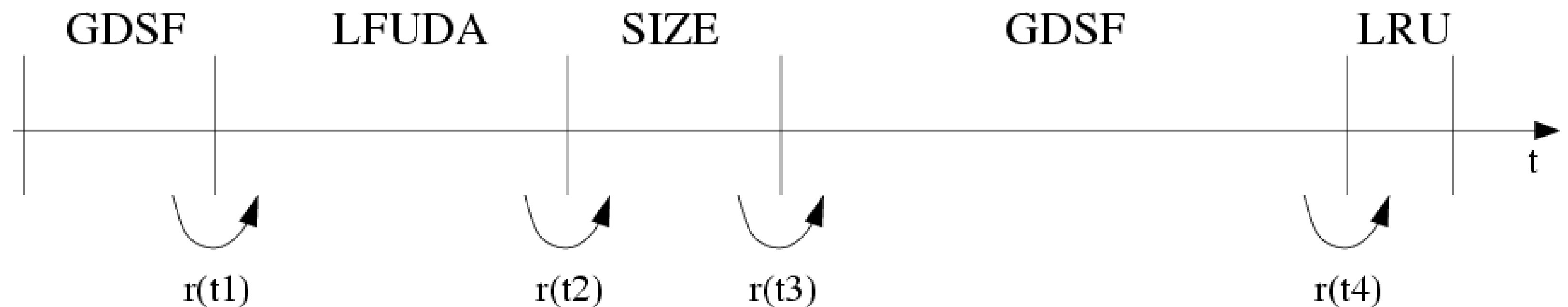


Want "Adaptive" Policy

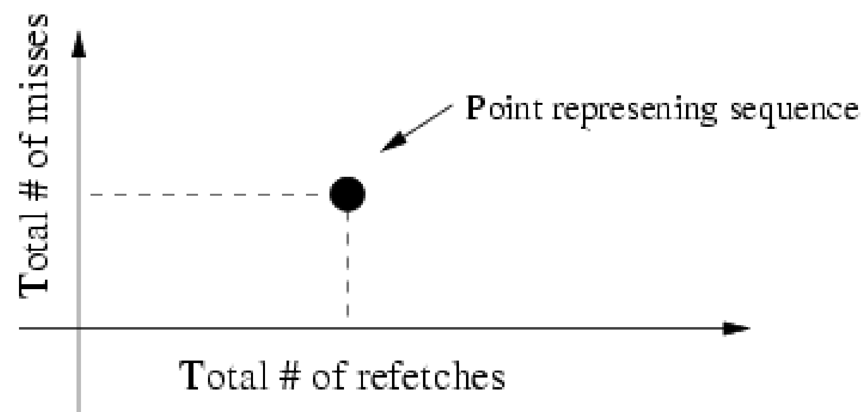
- Good compared to off-line comparator
  - **BestFixed**: a posteriori best of 12 policies on entire request stream
  - **BestRefetching**(R):
    minimum number of misses with at most R refetches in any sequence of switching policies

Refetches & Policy Switches

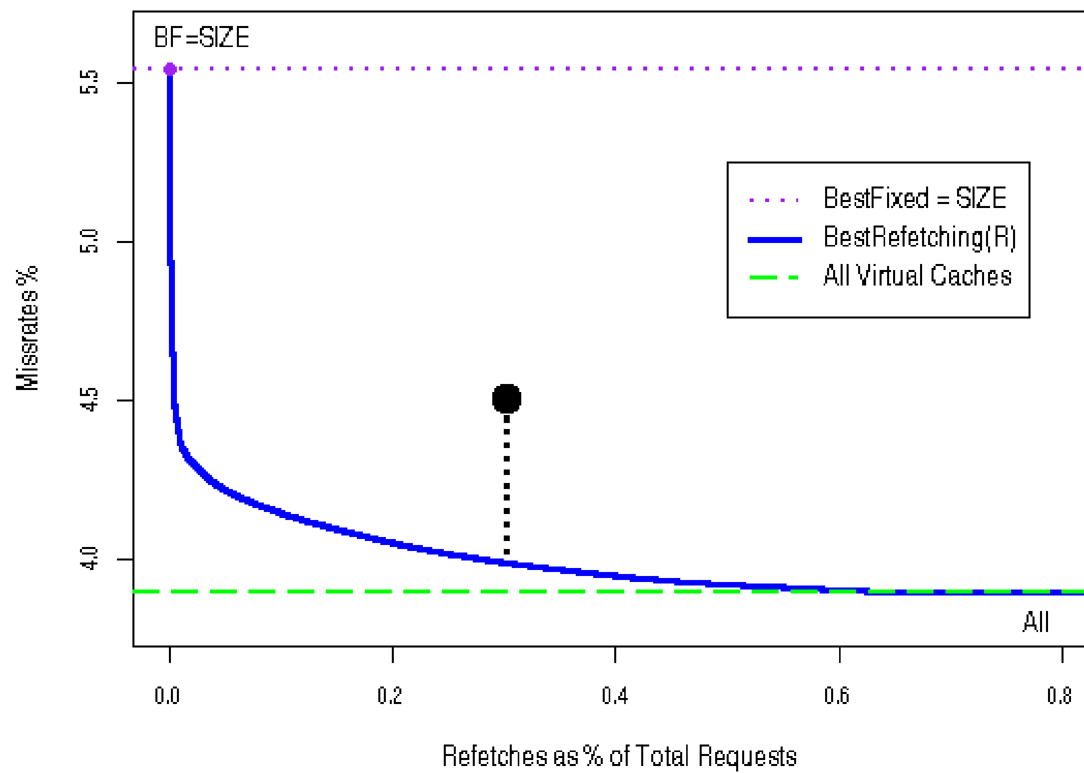
Off-line comparator: All sequences of the form



We plot miss rate v.s. refetches:



BestRefetching(R)

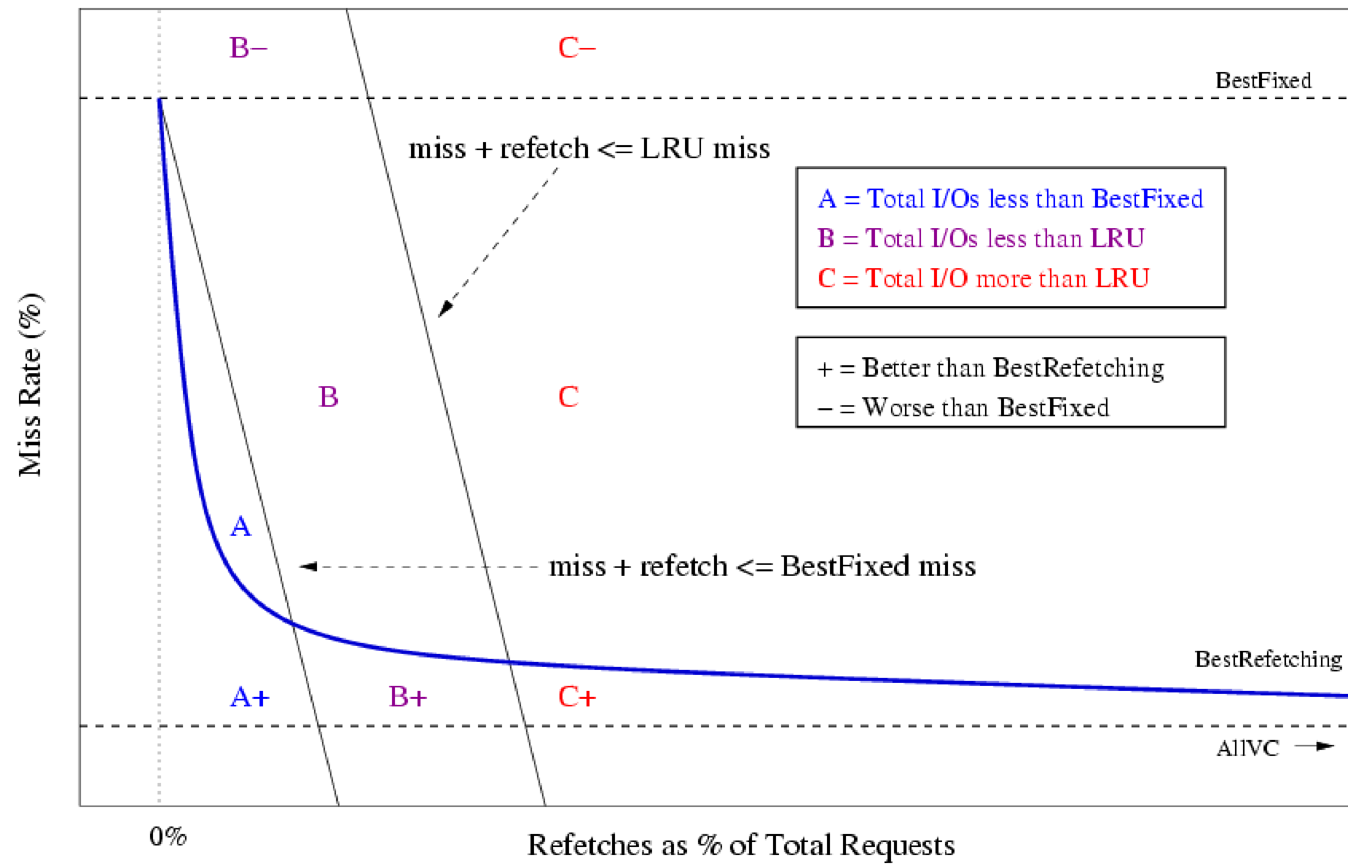


Dynamic programming in time $O(RN^2T)$

Goal for On-line policies

- Beat BestFixed
- Get close to BestRefetching
- Reduce I/O's and end-user latency

Score Card

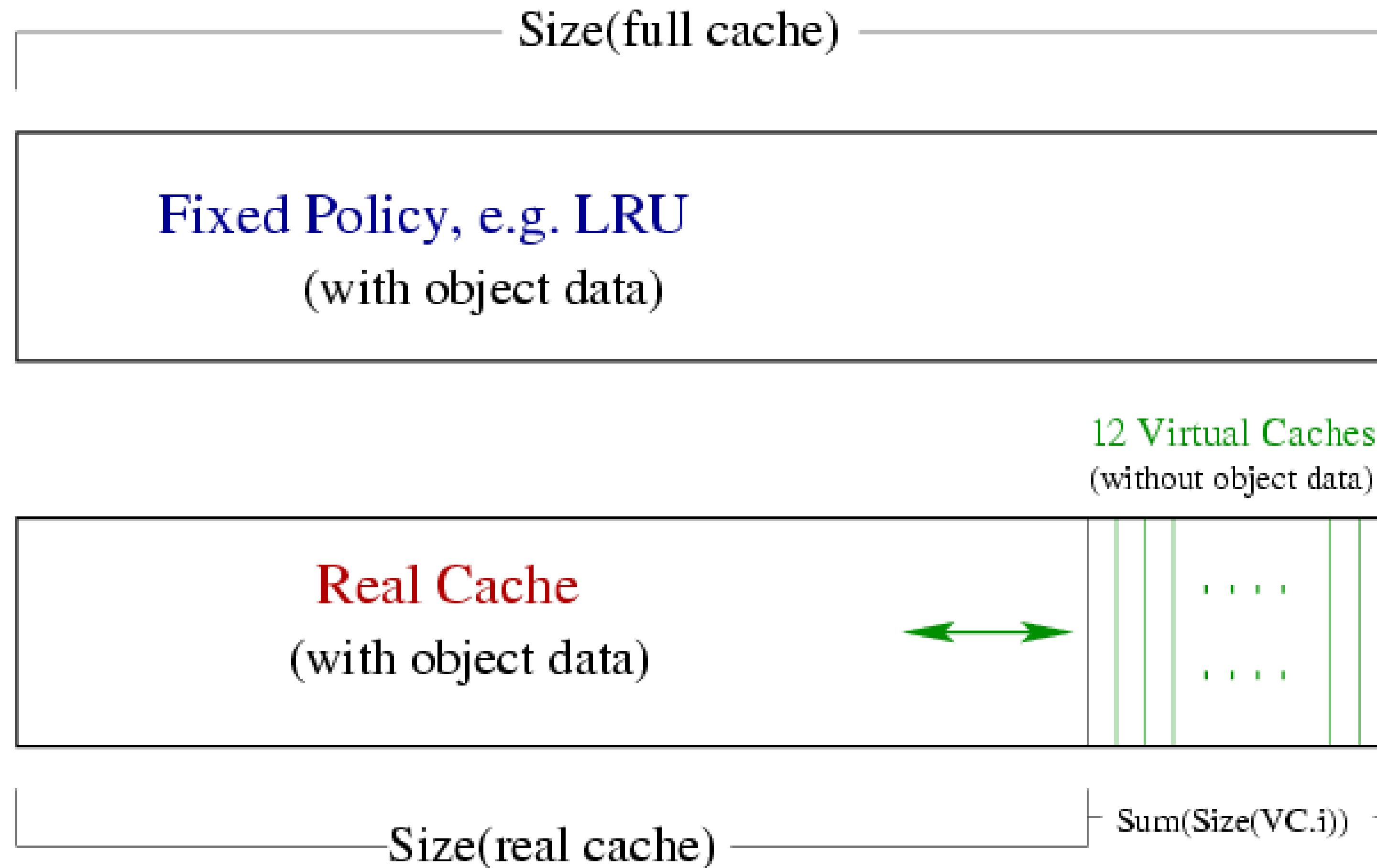


Key Idea: Virtual Caches

- Simulates a cache for each baseline policy
- Per object keep only (ID, size and calculated priority)
- Maintenance cost negligible
- Observe current miss rates of all 12
- Virtual Caches reside in the total cache space:

$$\text{Size(real cache)} = \text{Size(full cache)} - \sum_{i=1}^{12} \text{Size}(\text{VC}_i).$$

Virtual Caches



Window Algorithm

- Real cache governed by currently best policy
- Best means lowest number of hits in window of W (say 300) requests
- Works reasonably well - but
  - Hard to tune the window size
  - $O(NW)$ Additional space required for N policies.

Better Master Policy

- Use Expert Framework from On-line learning
- Maintain one weight w_i for each base policy / expert
- w_i is estimate of current relative performance of policy i
- Weights updated after each request:
  - Loss update punishes policies quickly that score misses
  - Share update [LW94,HW98,BW01]
    Keeps weights of poor policies from becoming too small
    Helps recovery

Fixed Share to Uniform Past

Loss Update:

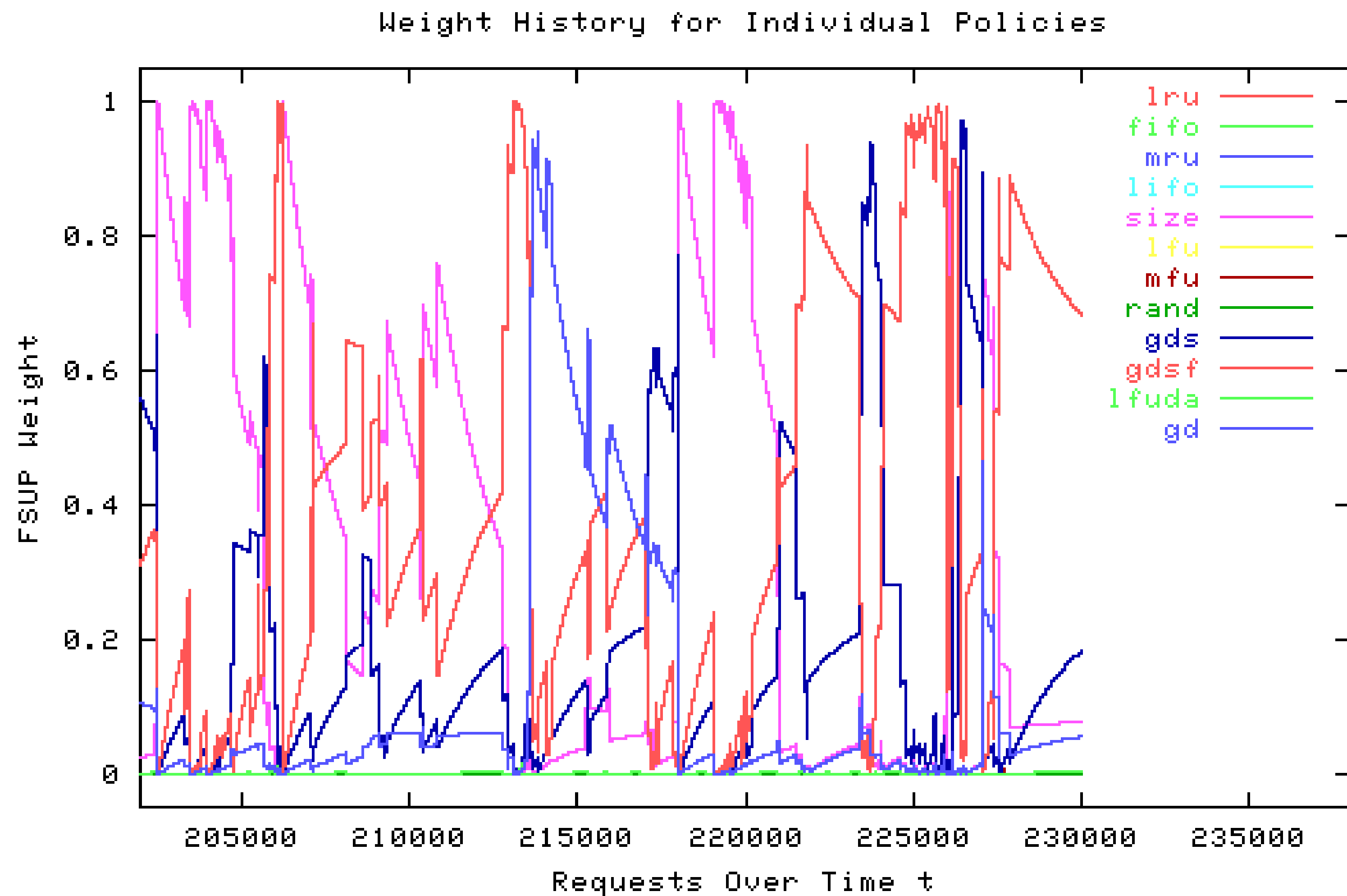
$$w'_{t,i} = \frac{w_{t,i}\beta^{\,\mathrm{miss}_{t,i}}}{\text{normaliz.}}, \quad \beta \in (0,1)$$

Share Update:

$$\mathbf{w}_{t+1} = (1-\alpha)\,\mathbf{w}'_t + \alpha\,\mathbf{r}_{t-1},$$
$$\text{where } \mathbf{r}_{t-1} = \sum_{q=1}^{t-1} \mathbf{w}'_q/(t-1)$$

- Prevents weights that did well in past from becoming too small
  Helps when these weights need to recover

Weights of baseline policies under FSUP



Digression

More on On-line Learning and Share Updates

On-line Learning

	experts						
	E_1	E_2	E_3	E_n	prediction	*true label*	loss
day 1	1	1	0	0	0	1	1
day 2	1	0	1	0	1	0	1
day 3	0	1	1	1	1	1	0
day t	$x_{t,1}$	$x_{t,2}$	$x_{t,3}$	$x_{t,n}$	$\hat{y}_t$	y_t	$(y_t - \hat{y}_t)^2$

- Choose comparison class of predictors (experts)
- Master Algorithm combines predictions of experts
- $\mathbf{x}_t$ vector of expert's predictions

Protocol of Master Algorithm

Loop for each trial $t = 1, .., T$

Get next instance	$\mathbf{x}_t$
Make prediction	$\hat{y}_t$
Get label	y_t ("true outcome")
Incur loss	$L(\hat{y}_t, y_t)$

- No statistical assumptions on the data

Goal

- Do well compared to the best off-line comparator / best expert

What kind of performance can we expect ?

- $L_{1..T,A}$ be the total loss of algorithm A
- $L_{1..T,i}$ be the total loss of i-th expert E_i

- Form of bounds

$$\forall S: \quad L_{1..T,A} \leq \min_i \left(L_{1..T,i} + \underbrace{c \ \log n}_{\text{bits}} \right)$$

where c is constant

- Bounds the loss of the algorithm relative to the loss of best expert

Algorithm that Achieves Bound

- Master algorithm predicts with weighted average

$$\hat{y}_t = \mathbf{w}_t \cdot \mathbf{x}_t$$

- The weights are updated according to the Loss Update

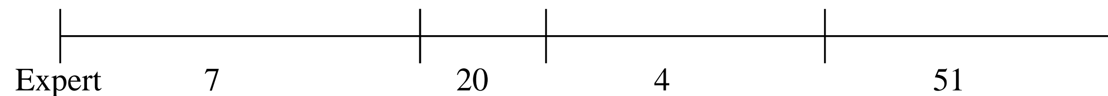
$$w_{t+1,i} := \frac{w_{t,i} \ \ e^{-\eta \ L_{t,i}}}{\text{normaliz.}}, \quad e^{-\eta} = \beta$$

where $L_{t,i}$ is loss of expert i in trial t

→ Weighted Majority Algorithm [LW89]

→ Generalized by Vovk [Vovk90]

What if Comparator Changes with Time ?



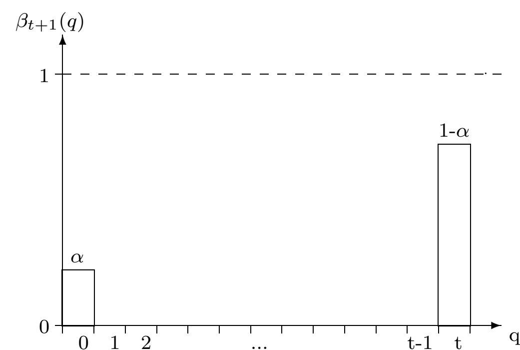
- Off-line algorithm partitions sequence into sections and chooses best expert in each section
- Goal:
  Do well compared to the best off-line partition
- Problem:
  Loss Update learns too well
  and does not recover fast enough

Mixing Update

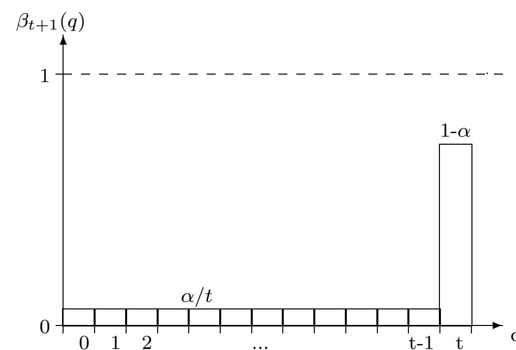
- Predict $\hat{y}_t = \mathbf{w}_t \cdot \mathbf{x}_t$
- Loss Update $w'_{t,i} = \frac{w_{t,i} e^{-\eta L_{t,i}}}{\text{normaliz.}}$
- Mixing Update

$$\mathbf{w}_{t+1} = \sum_{q=0}^{t} \beta_{t+1,q} \mathbf{w}'_q, \quad \text{where} \quad \sum_{q=0}^{t} \beta_{t+1} = 1$$

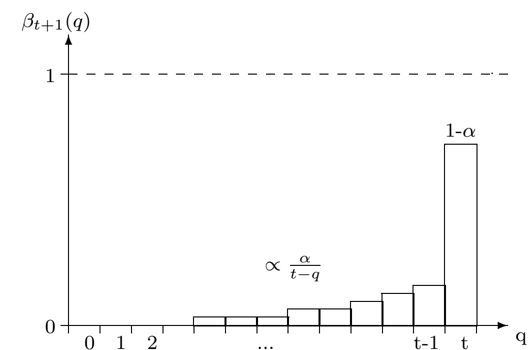
- Mixing schemes



FS to Start Vector

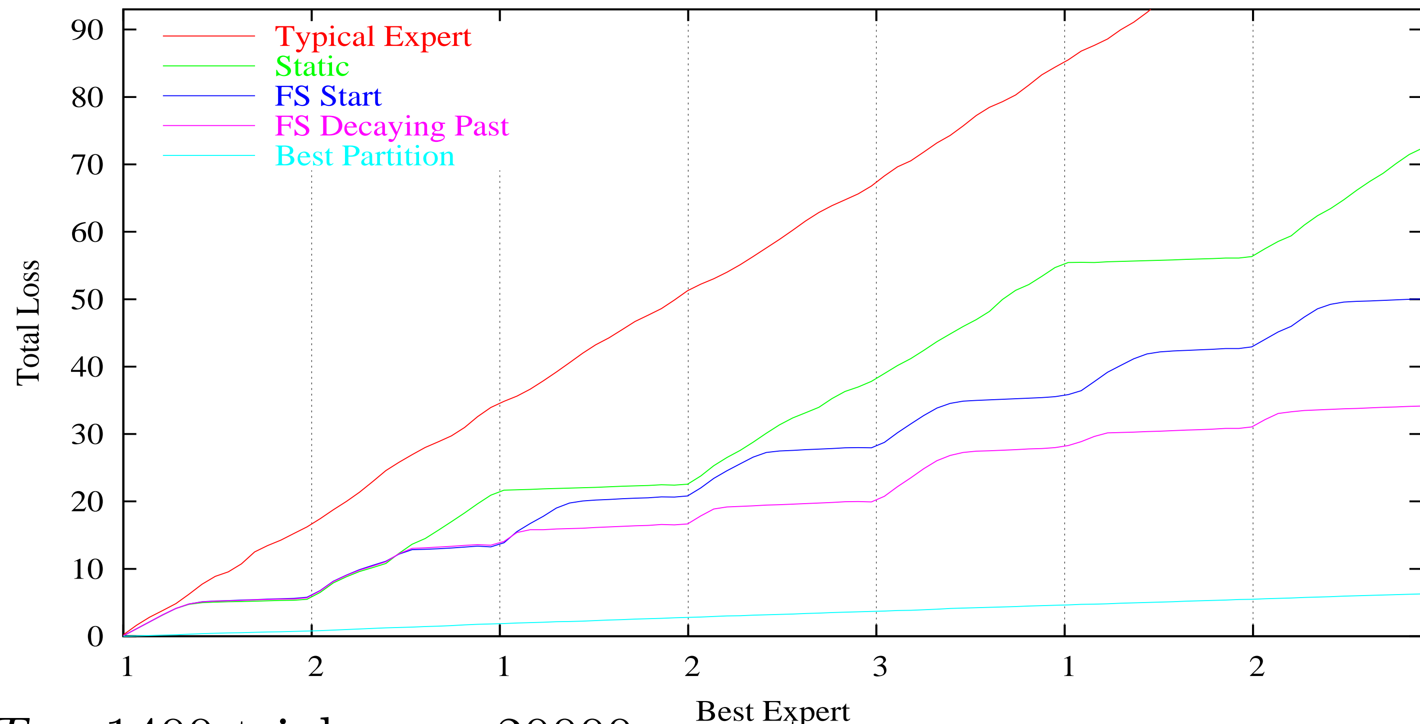


FS to Uniform Past



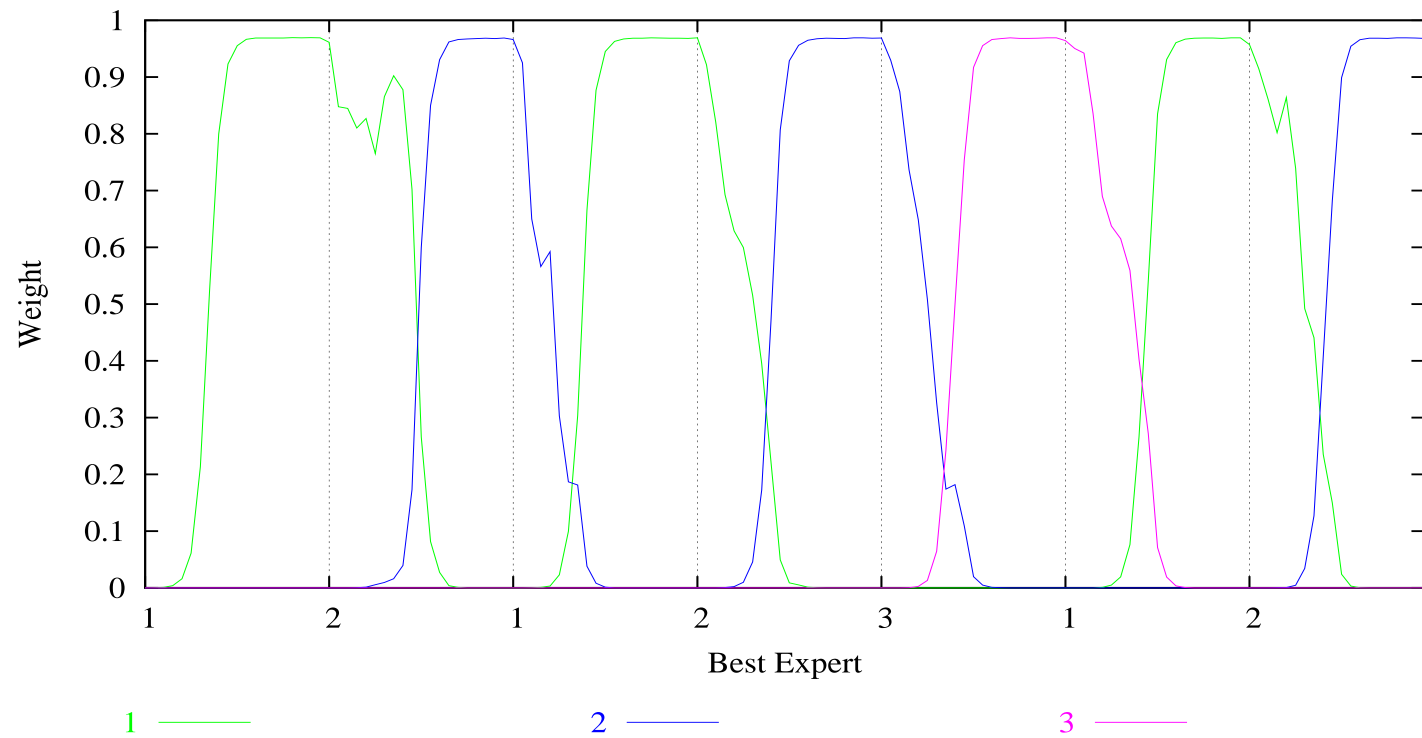
FS to Decaying Past

Total Loss Plots



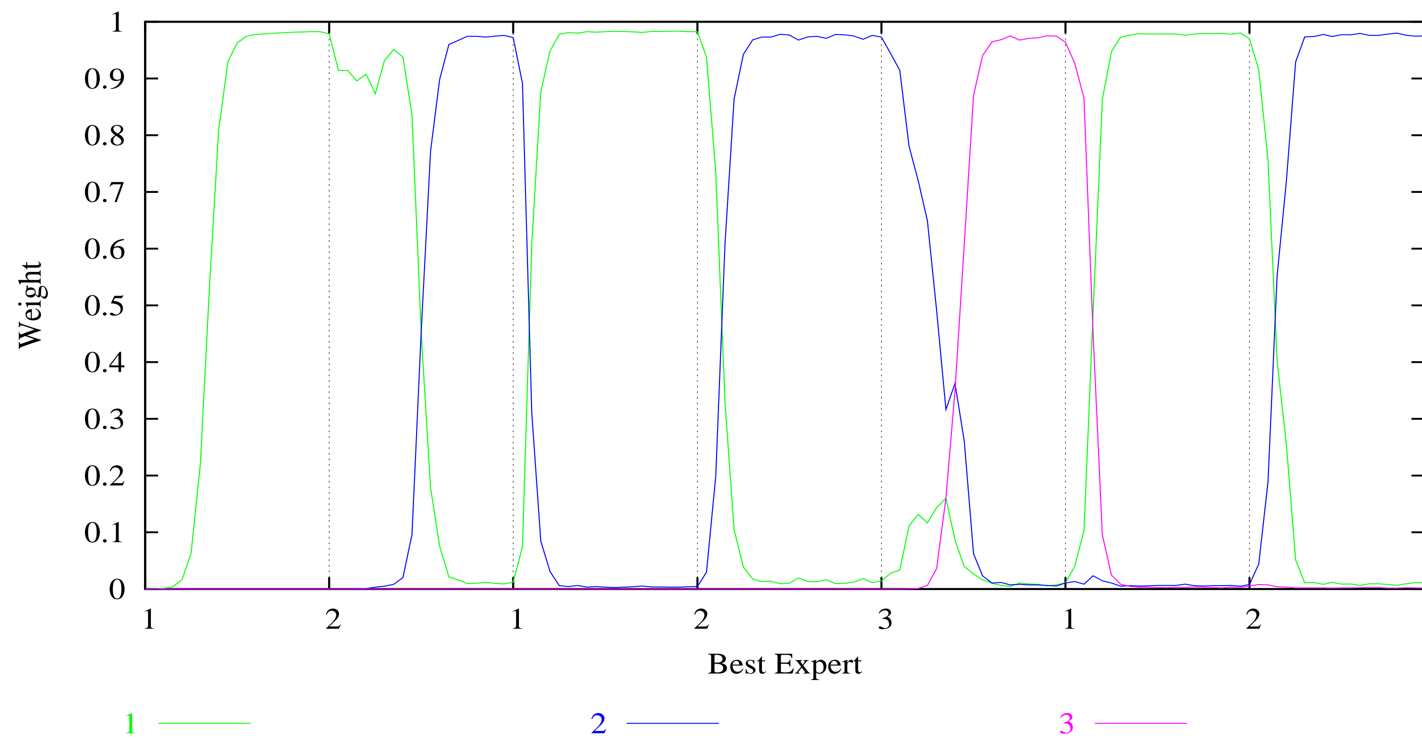
- $T = 1400$ trials, $n = 20000$ experts
- $k = 6$ shifts (every 200 trials)

Weights of Fixed Share to Start Vector Alg.



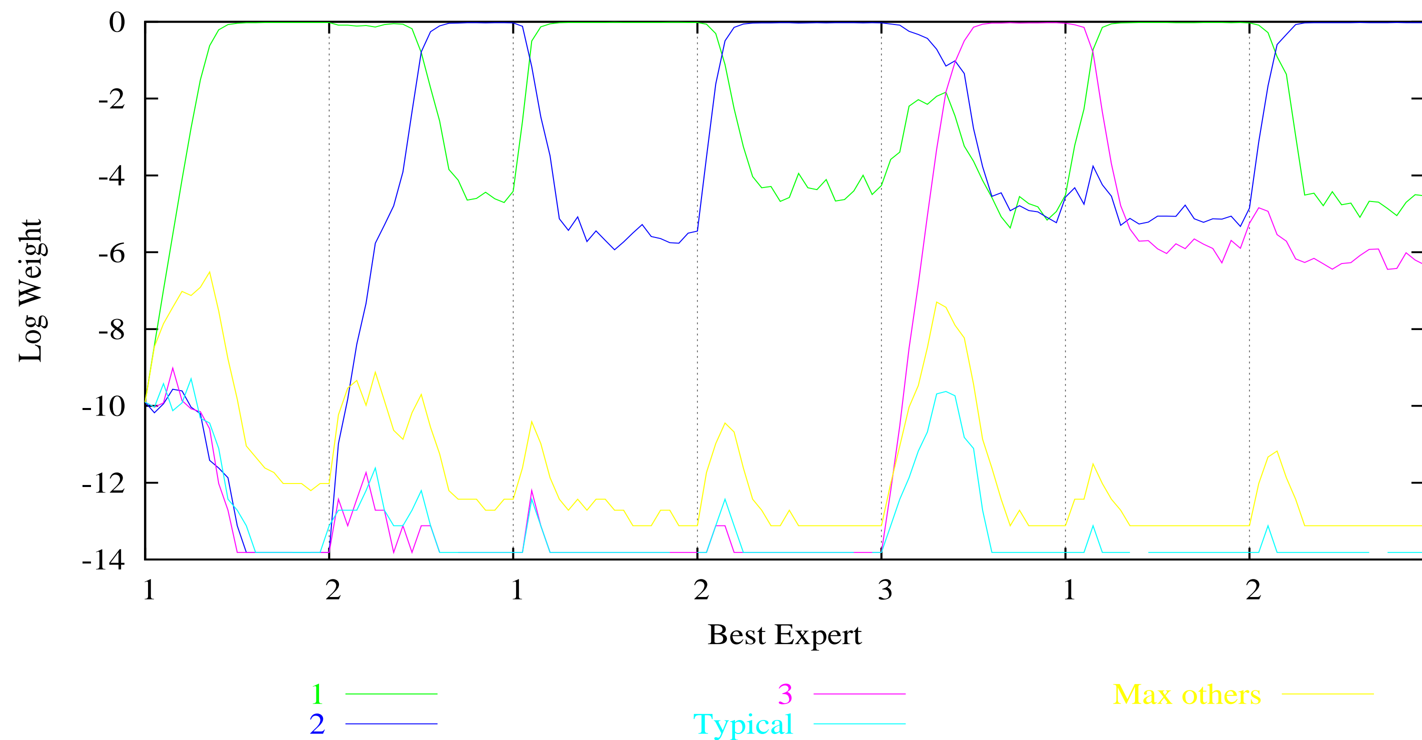
Weights of Fixed Share to Decaying Past Alg.

- Improved recovery when expert used before

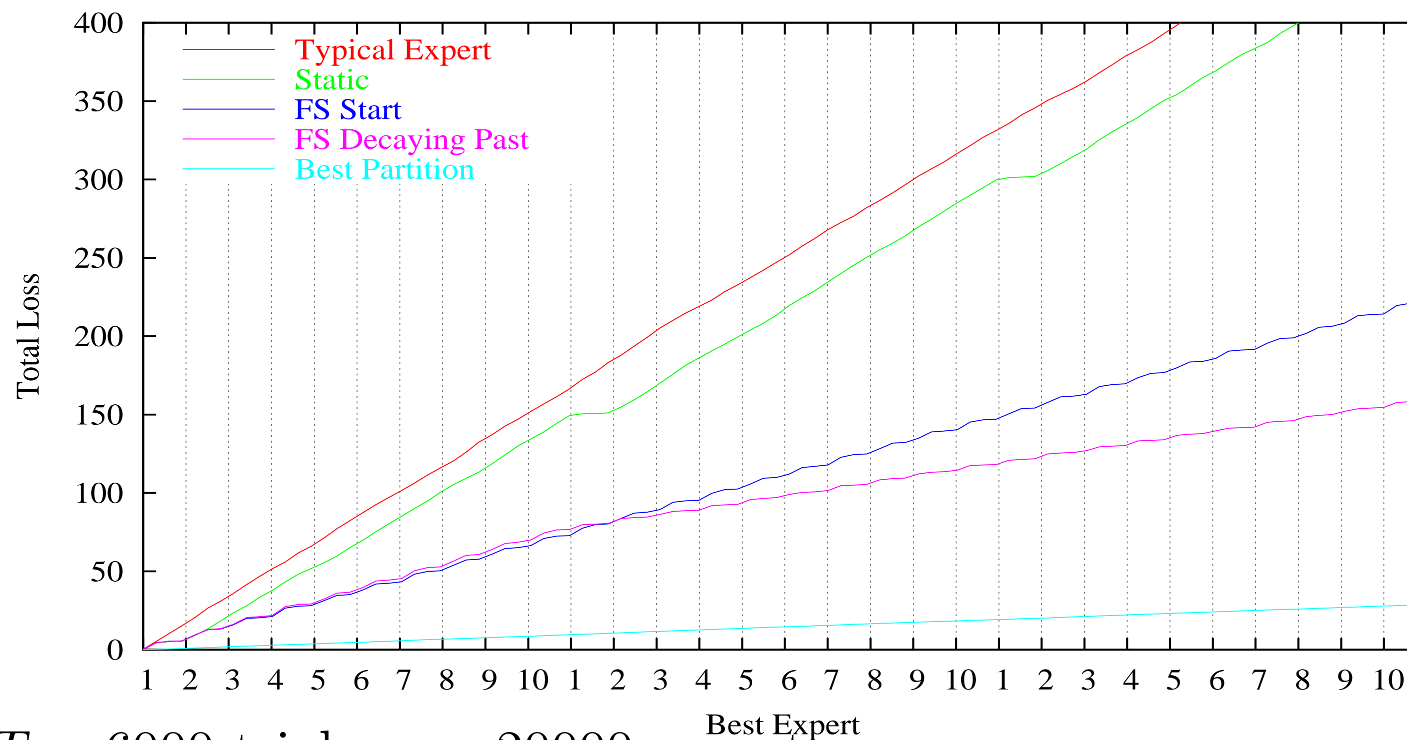


Fixed Share to Decaying Past - Log Weights

- Past good experts remain at higher level



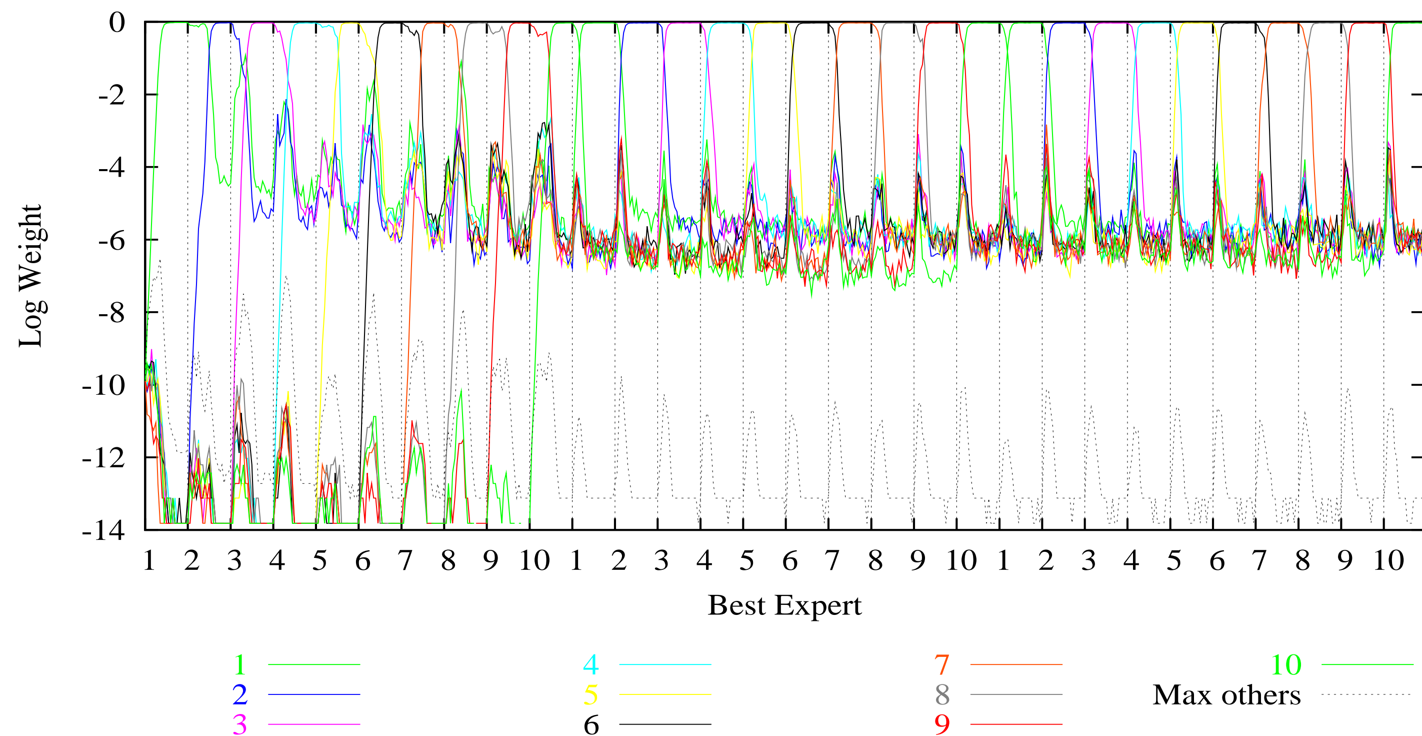
More Experts Remembered



- $T = 6000$ trials, $n = 20000$ experts
- $k = 29$ shifts (every 200 trials)

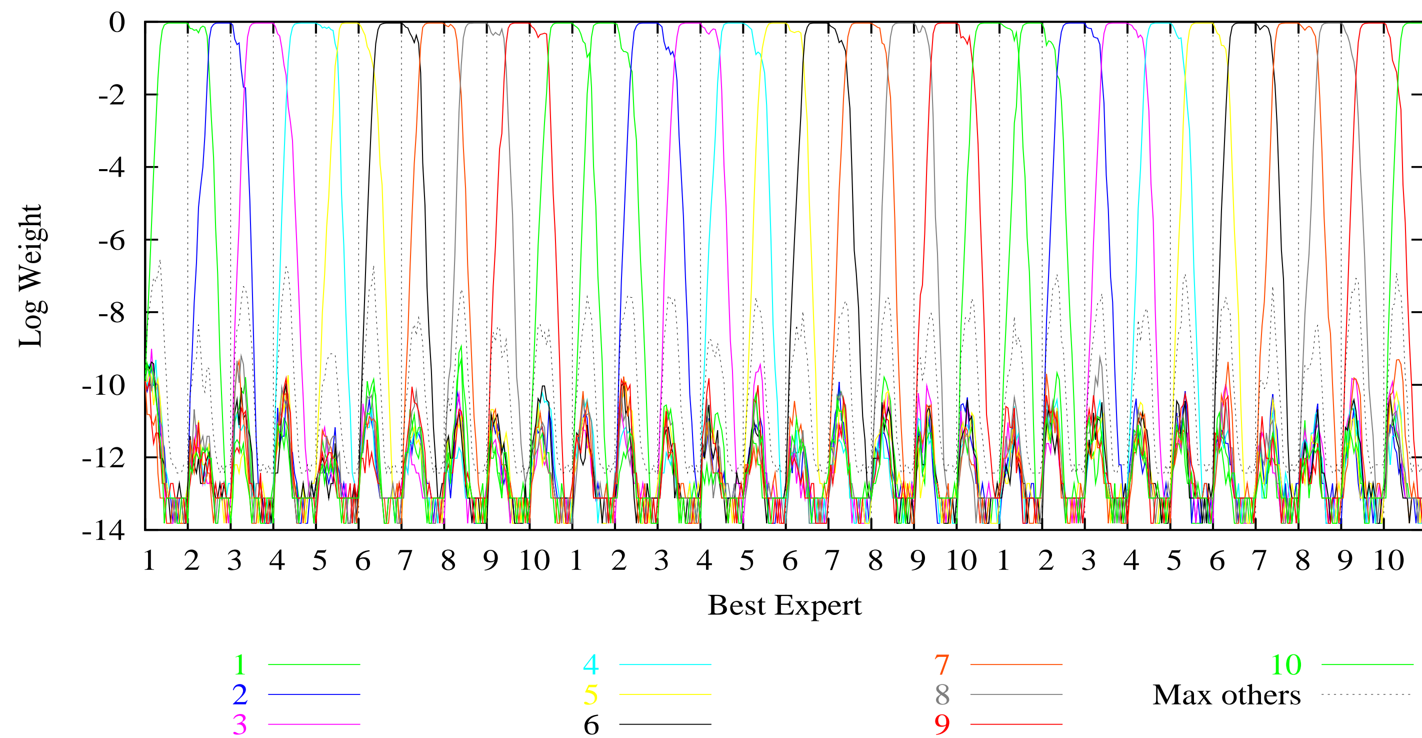
Fixed Share to Decaying Past - Log Weights

- Past good expert are cached



Fixed Share to Start Vector - Log Weights

- No memory



Relative Loss Bounds

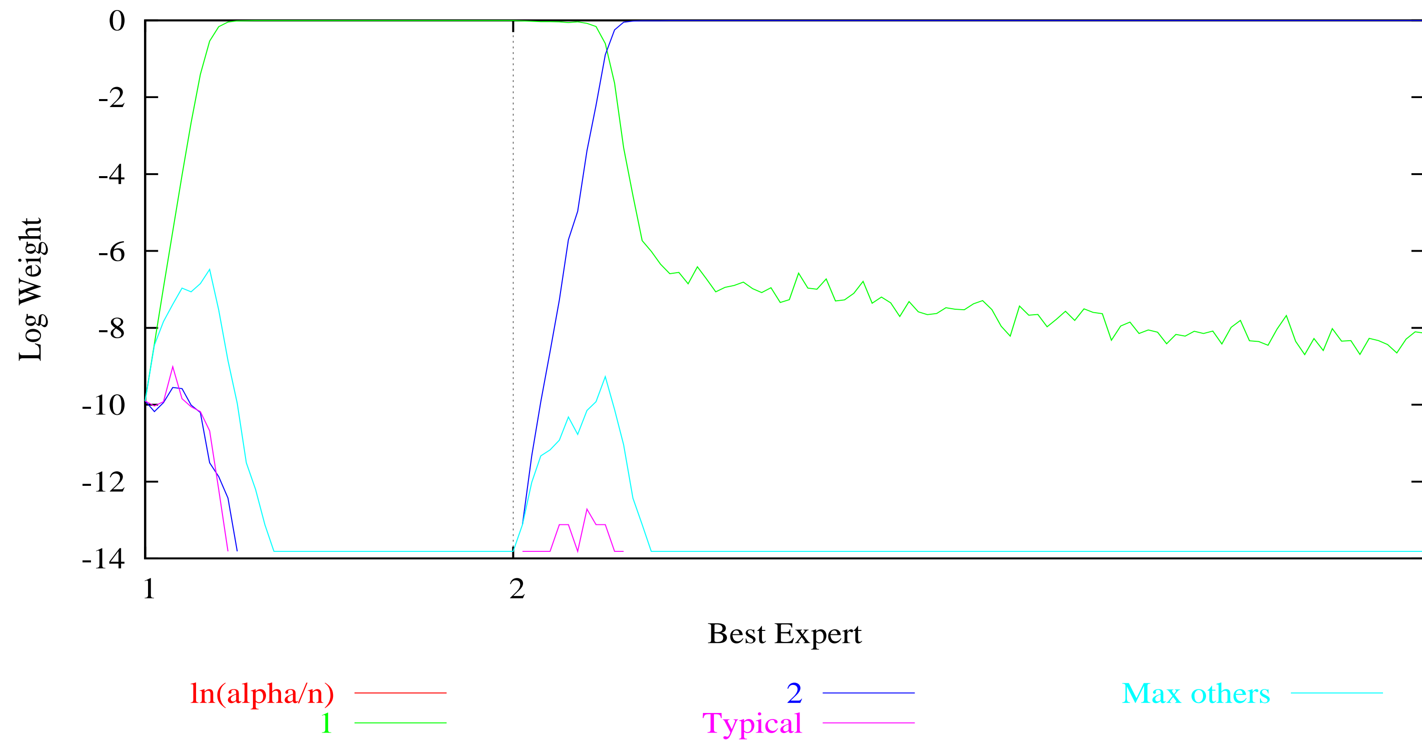
- Always have the form

$$L_{1..T,A} \leq \min_{P} \left(L_{1..T,P} + O(\text{\# of bits for } P) \right)$$

→ Boundaries are encoded twice

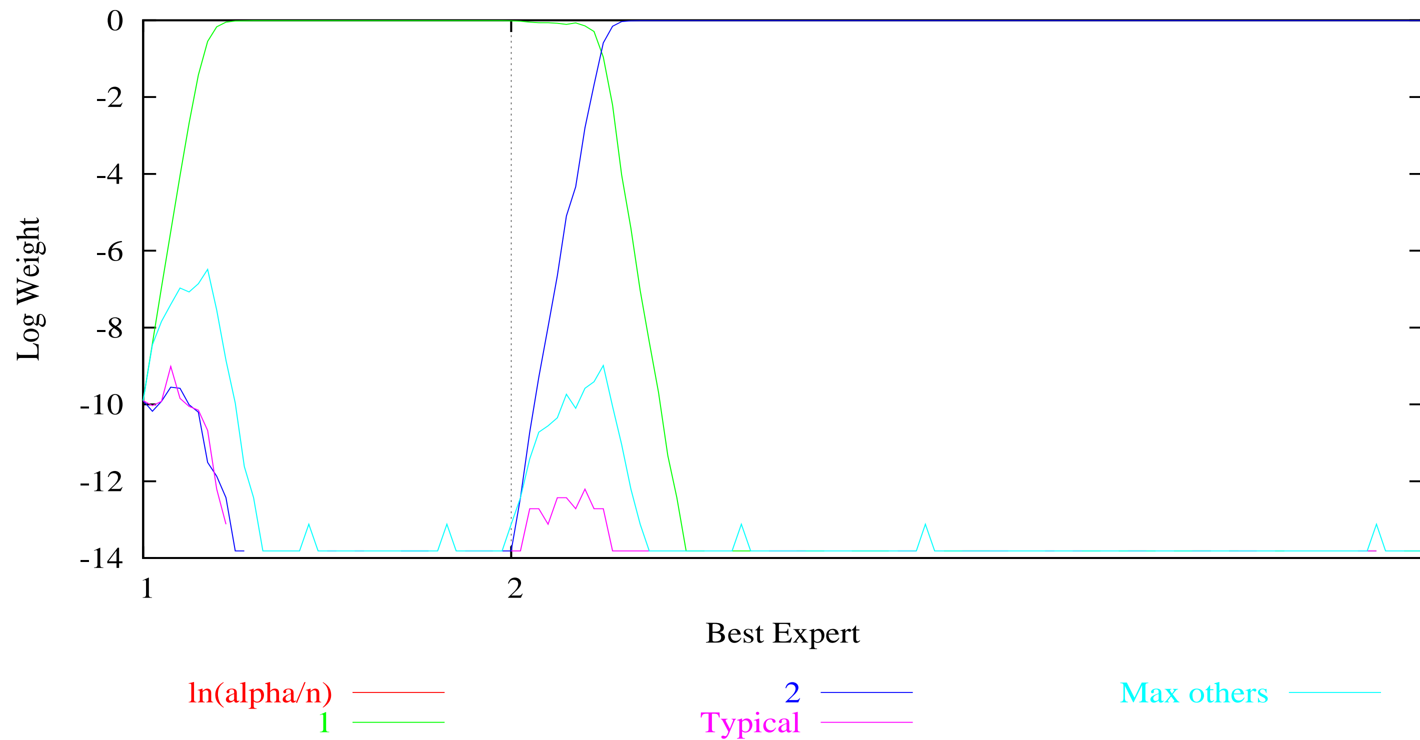
→ Off-line problem NP-complete

Fixed Share to Decaying Past - Log Weights



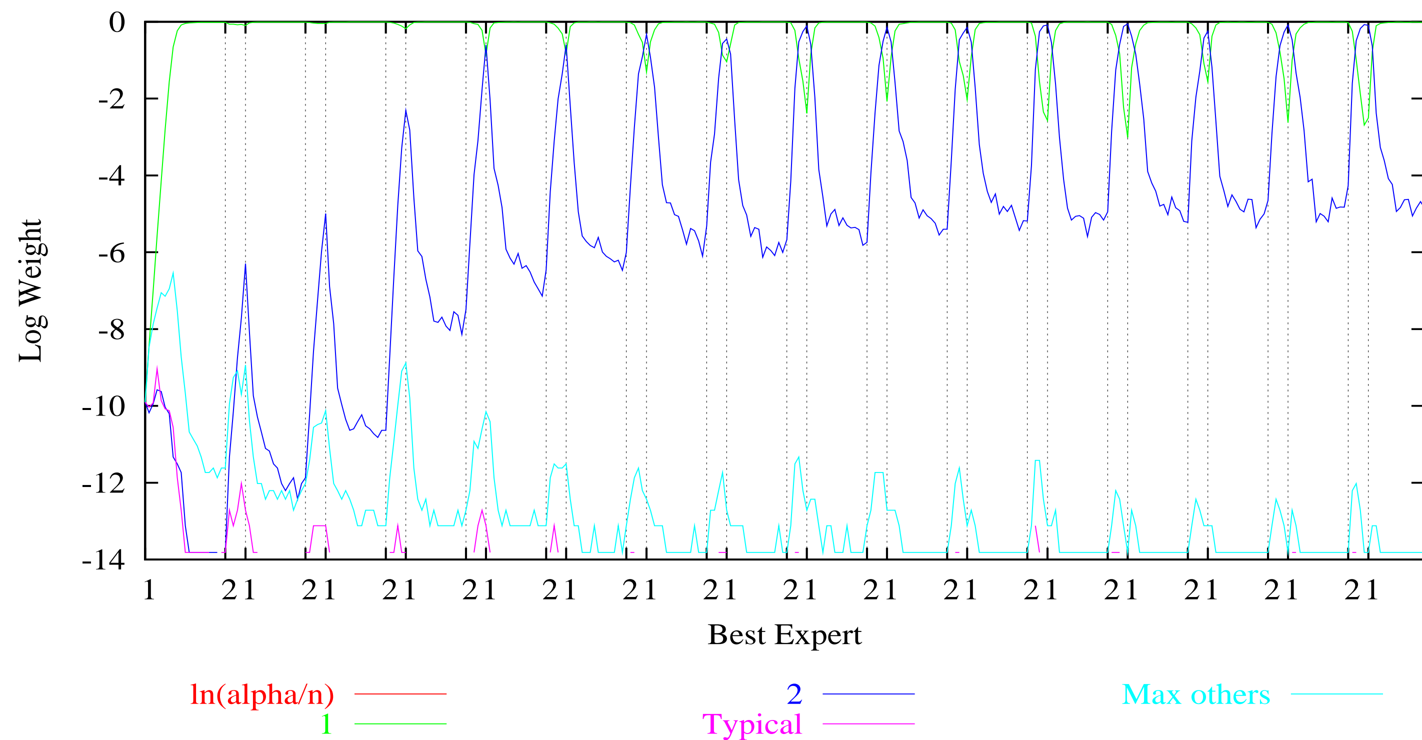
- Larger alpha gives better long-term memory

Fixed Share to Start Vector - Log Weights



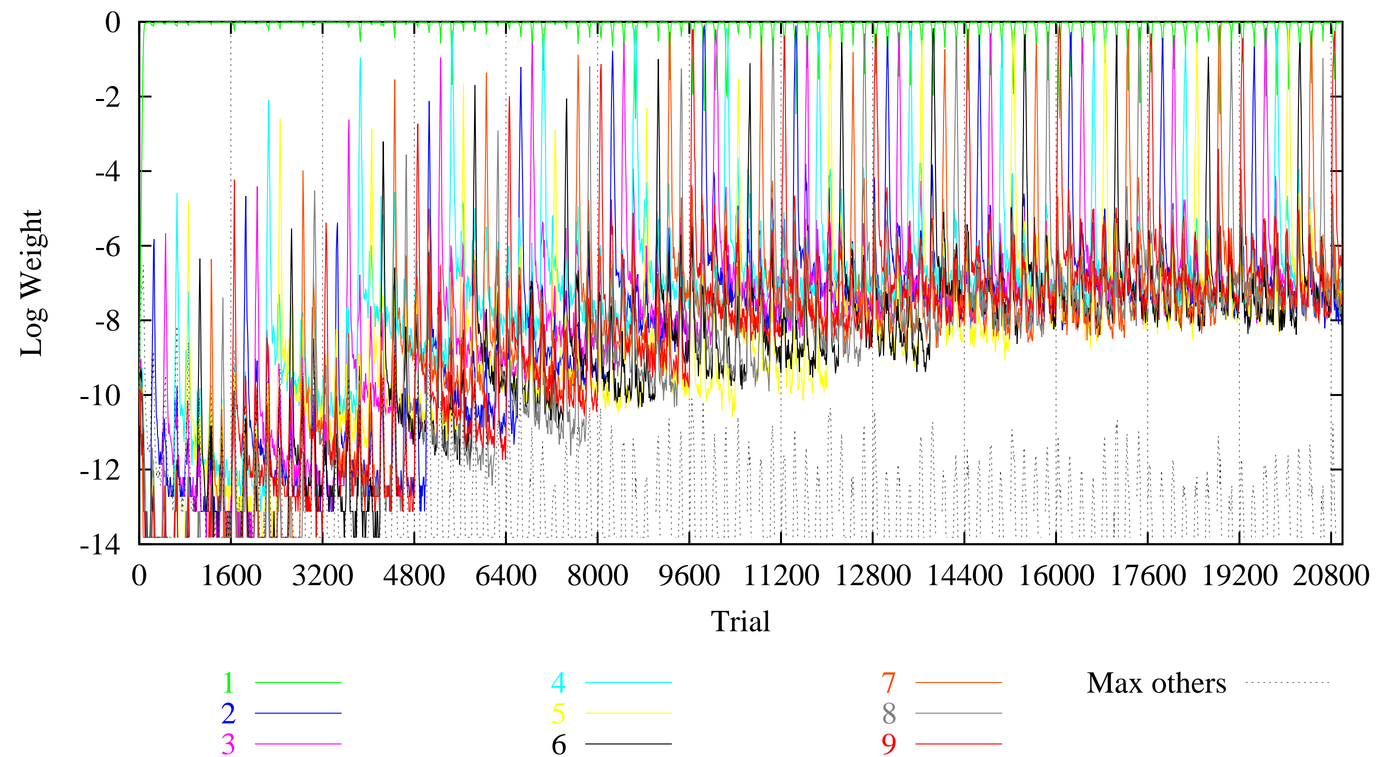
Memory from many short sections accumulates

- Fixed Share to Decaying Past - Log Weights



Bigger memory

- Cycling thru 10 different short sections



Back to Caching

- Share-update crucial
- Fixed Share to Uniform Past cheap one of the best
- Bounds do not apply but we are using recovery properties
  - parameter settings (α, β or η) not crucial
  - fix at

$$\beta = 1/e \quad \alpha = 0.005$$

Master Policy Protocol

- Process request on virtual caches
- Apply Loss and Share Updates
- Process request on real cache
  - based on combined weightings of all caches
- Refetch objects into real cache (if desired)

Virtual Cache Rankings

- Priorities induce ranks over virtually cached objects:

object	o_{12}	o_7	o_2	o_{22}	o_3	o_6	o_2	o_{15}	o_9
priority	31.2	30.2	24.1	17.1	9.3	8	4.1	2.5	1.2
rank	9	8	7	6	5	4	3	2	1

- o_9 first discarded
- o_{12} last

Master Rank

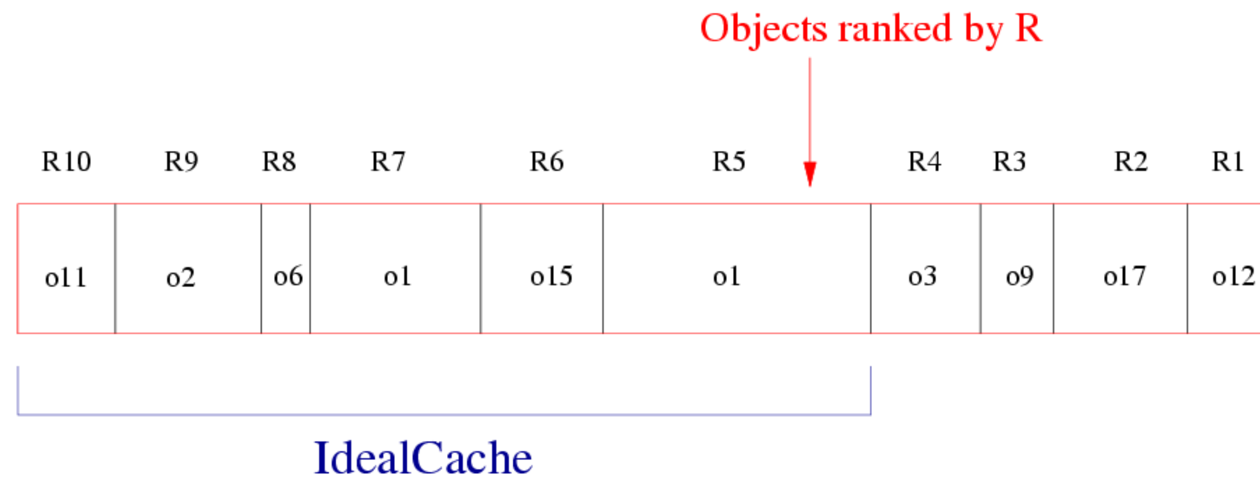
- Master priority P constructed from weights and ranks of virtual caches

$$P_o = \begin{cases} \sum_{n: o \in \mathrm{VC}_n} w_n r_{n,o} & \text{if } \exists n : o \in \mathrm{VC}_n \\ 0 & \text{if } \forall n : o \notin \mathrm{VC}_n \end{cases}$$

- R is corresponding master rank

Ideal Cache

- Highest ranked objects fill the ideal cache to capacity
- IdealCache =



Managing Real Cache: Instantaneous Rollover

- Keep RealCache = Ideal Cache
  i.e. Refetch all $o \in \text{IdealCache} - \text{RealCache}$
- Too much refetching

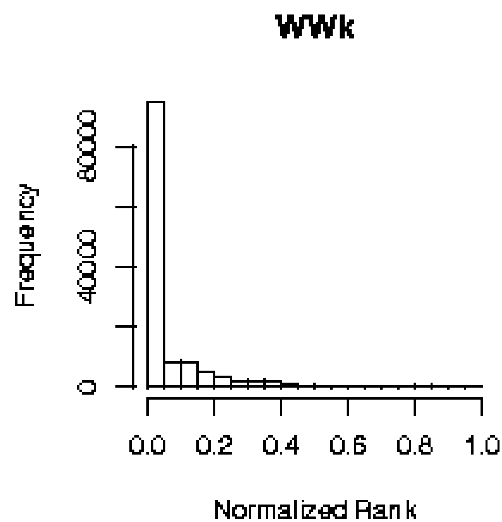
Demand Rollover

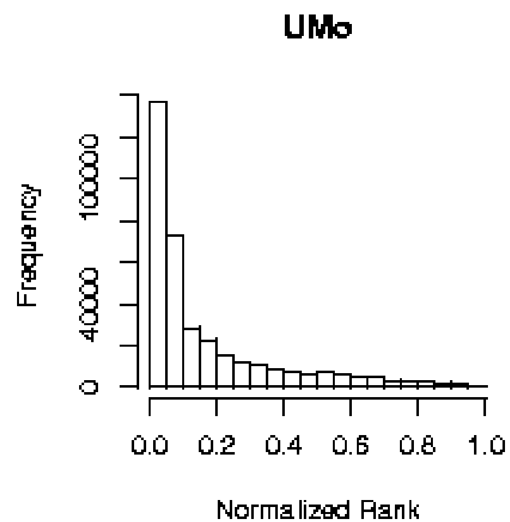
- Lowest R-ranked objects are discarded to make room for a new request
- No refetching

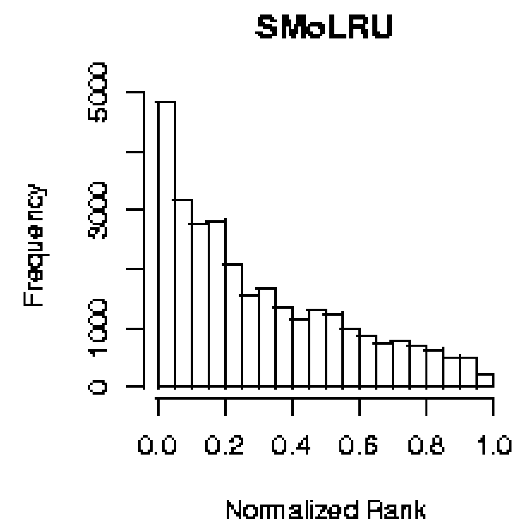
Compromise: Background Rollover

- Refetch objects
  $o \in \text{IdealCache} - \text{RealCache}$
  when system is idle
- Model idleness as Poisson process
  - Draw $d \sim \text{Pois}(\lambda)$
  - Refetch (at most) d objects $o \in \text{IdealCache} - \text{RealCache}$

Smart Refetching







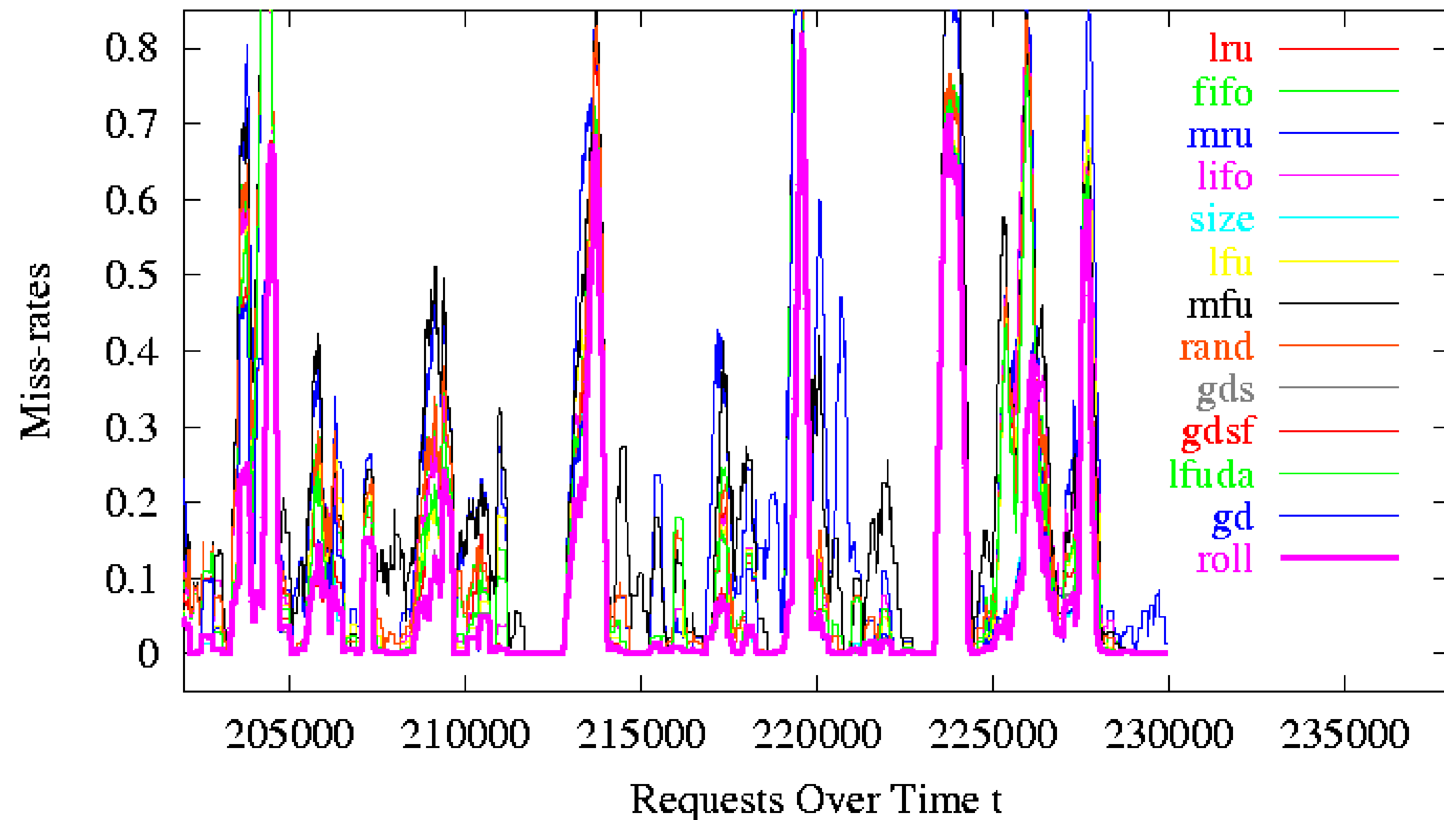
- Most hits in real cache have high R-rank
- Refetch only top 40-60% of R-ranked objects

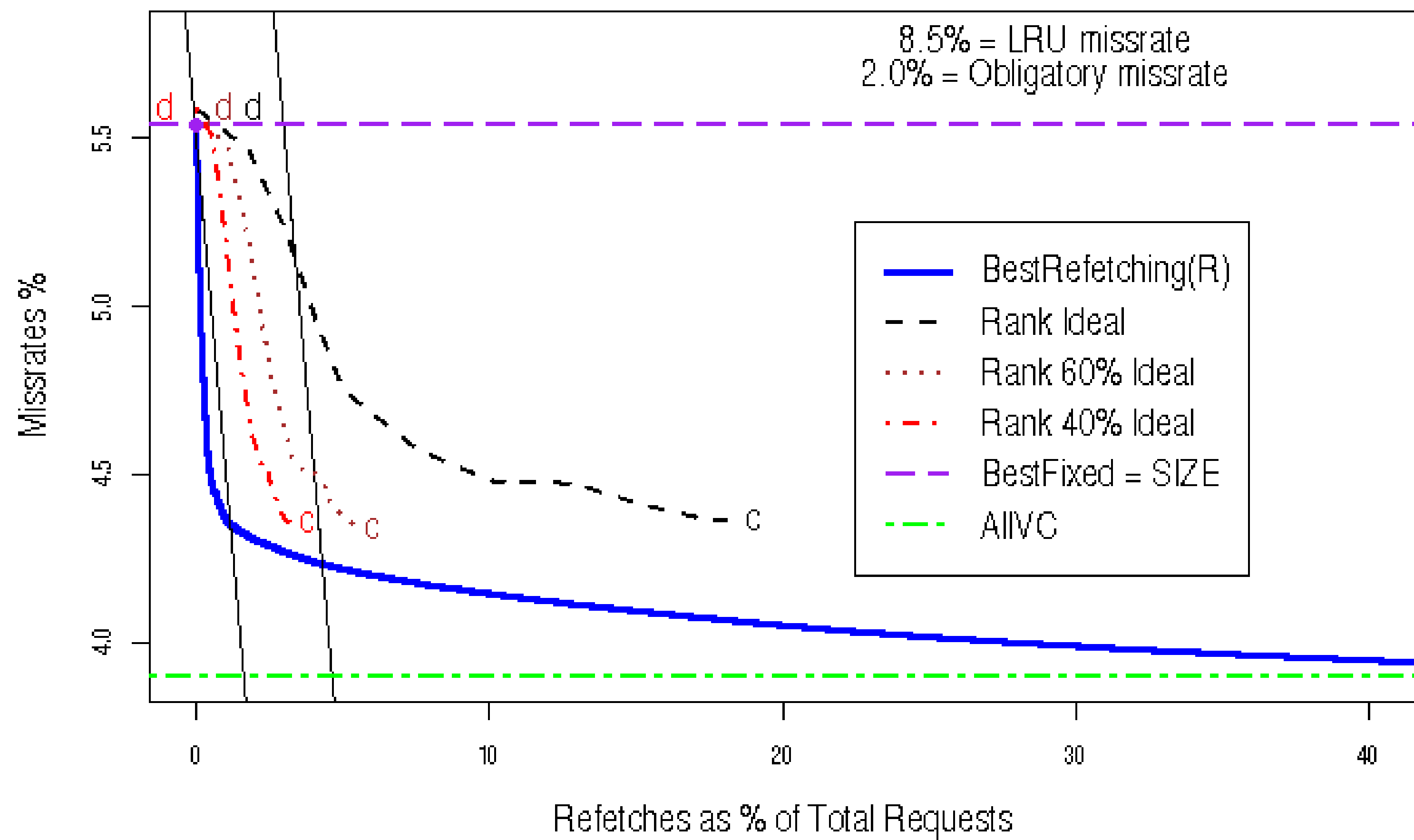
Experimental Results: Filesystem Data

Dataset:	Work-Week (WWk)	User-Month (UMo)	Server-Month-LRU (SMoLRU)
#Requests	138k	382k	48k
Cache size	900KB	2MB	4MB
%Skipped	6.5%	12.8%	15.7%
# Compuls	0.020	0.015	0.152
LRU Miss Rate	0.166	0.076	0.870
BestFixed Pol / MR	SIZE 0.055	GDS 0.075	GDSF 0.399
%<LRU	36.8%	54.7%	54.2%

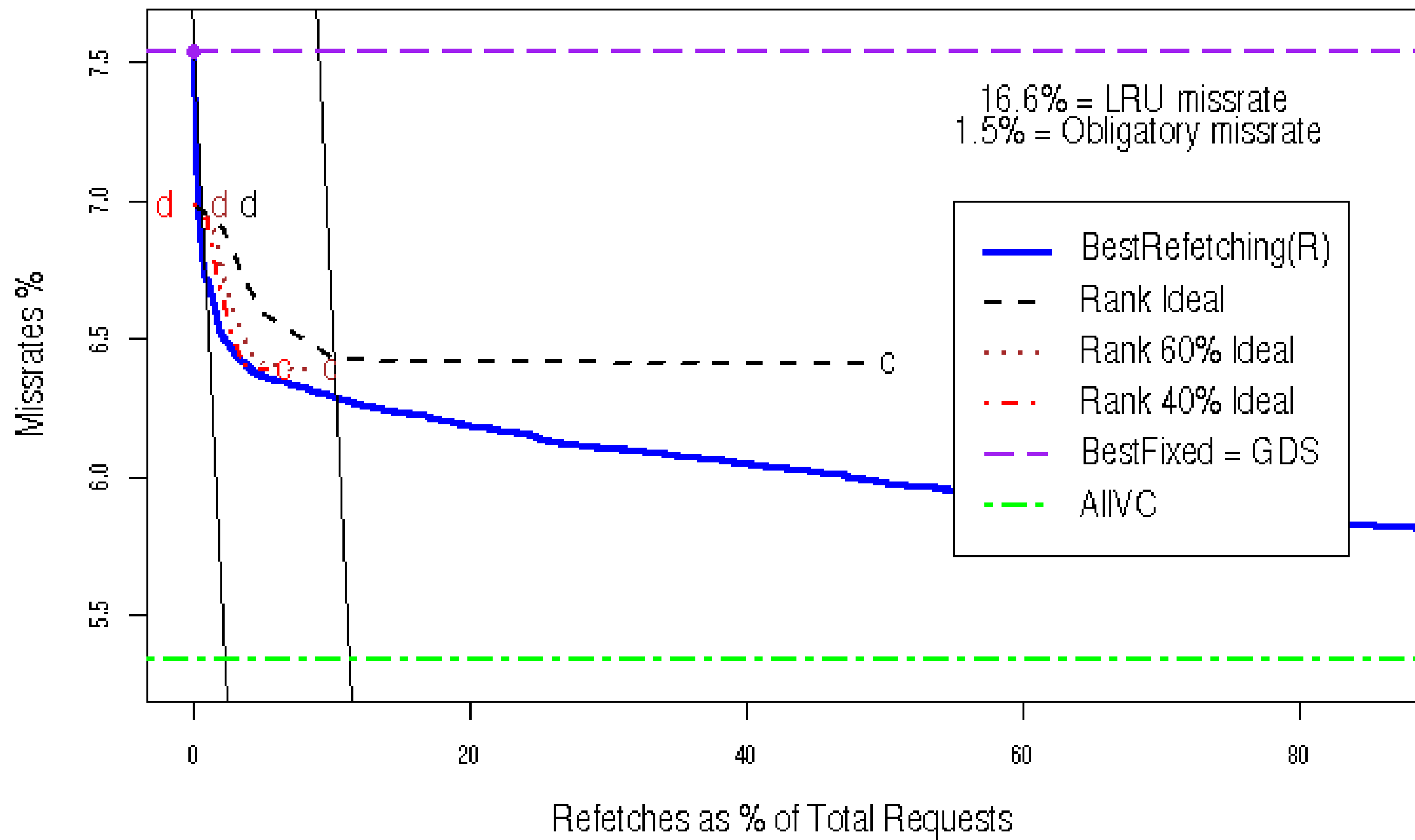
CMU DFStrace

Demand Rollover "Tracks" best policy

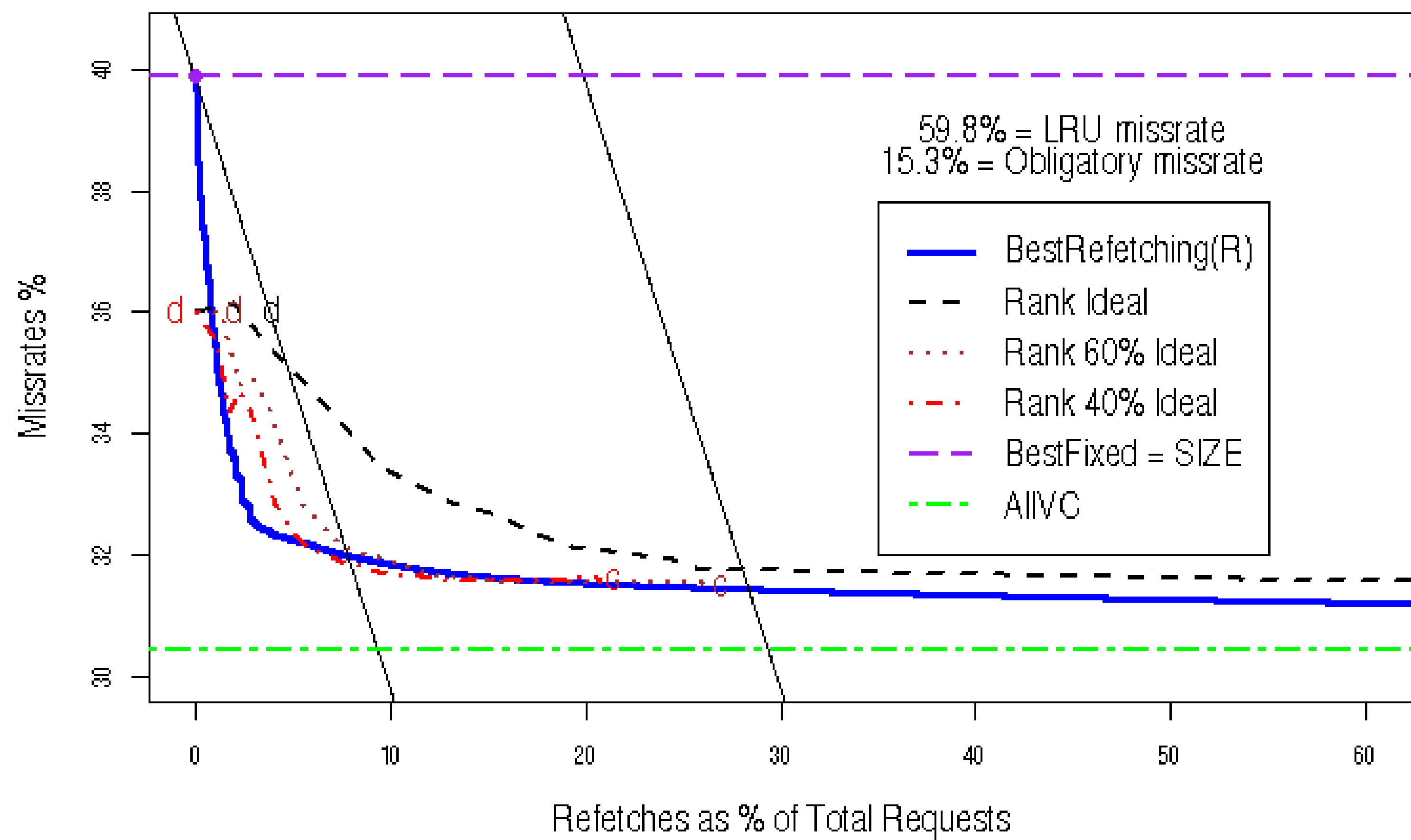



WWk
WWk Master and Comparator Missrates
8.5% = LRU missrate
2.0% = Obligatory missrate
d
d
d
c
c
c
BestRefetching(R)
Rank Ideal
Rank 60% Ideal
Rank 40% Ideal
BestFixed = SIZE
AllVC
Missrates %
5.5
5.0
4.5
4.0
0
10
20
30
40
Refetches as % of Total Requests

UMo



SMoLRU



Summary

- Demand Rollover is already as good or better than BestFixed
- Small amounts of refetching always beats Best Fixed
  - 15-22% fewer misses than BestFixed
  - 45-70% fewer misses than LRU
- Can be as good as BestRefetching
  - always less I/O's than LRU
  - can result in less I/O than BestFixed

Conclusion

- Operating Systems have many parameter tweaking problems suitable for on-line learning
- Previous work using same updates:
  - Tuning time-out for spinning down disk of a PC [HLSS00]
  - Load balancing between processors [BB97]
  - Tracking with GPS

Too expensive?

- Not for web caching and filesystem's caching
- Not clear for paging
- Implement in Linux kernel

Two approaches

- Use existing caching strategies as experts
- Use set of fine-grained experts
  from which all existing caching policies are built
- Machine Learners will get interested if there are realistic benchmark data sets