



# AMP: Program-Context Specific Buffer Caching

*Feng Zhou, Rob von Behren, Eric Brewer*


*University of California, Berkeley*

*Usenix tech conf 2005, April 14, 2005*

# Buffer caching beyond LRU

- Buffer cache speeds up file reads by caching file content
- LRU performs *badly* for large **looping accesses**

Access stream:            1            1 2 3 4 , Cache Size: 3

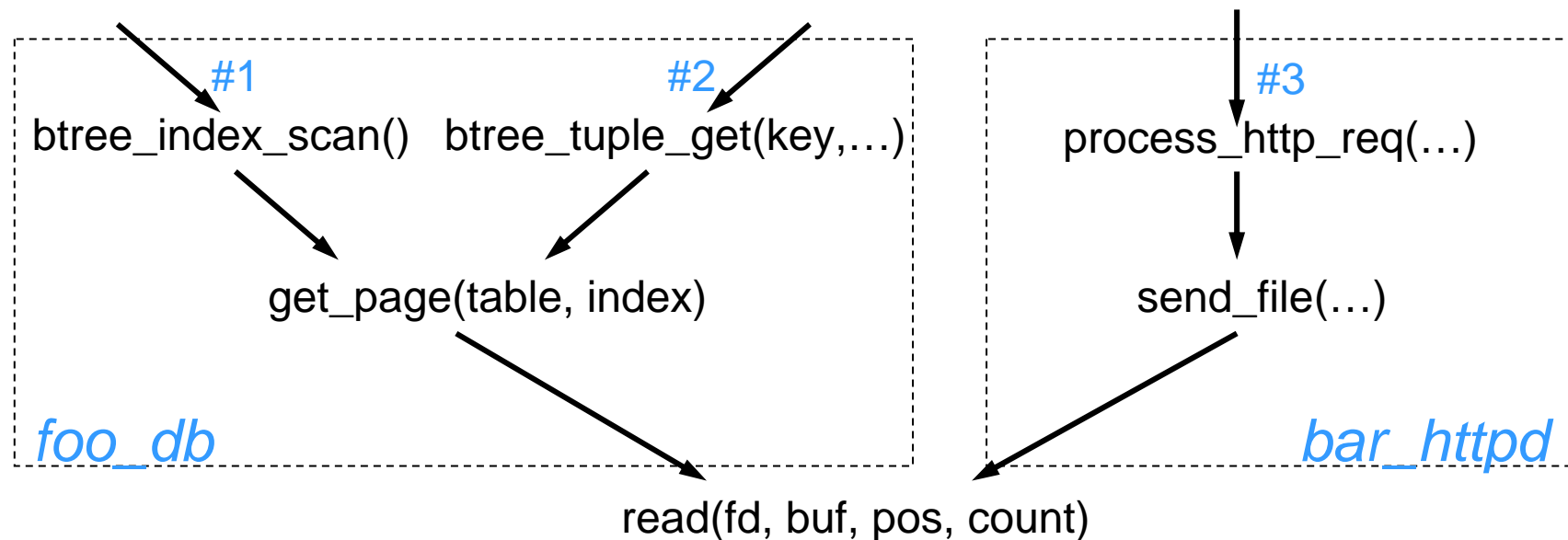


0 Hit Rate for any loop over data set larger than cache size

- DB, IR, scientific apps often suffer from this
- Recent work
  - Utilizing frequency: ARC (Megiddo & Modha 03), CAR (Bansal & Modha 04)
  - Detection: UBM (Kim et al. 00), DEAR (Choi et al. 99), PCC (Gniadv et al. 04)

# Program Context (PC)

- Program context: current program counter + all return addresses on the call stack



## Ideal policies

#1: MRU for loops

#2, #3: LRU/ARC for all others



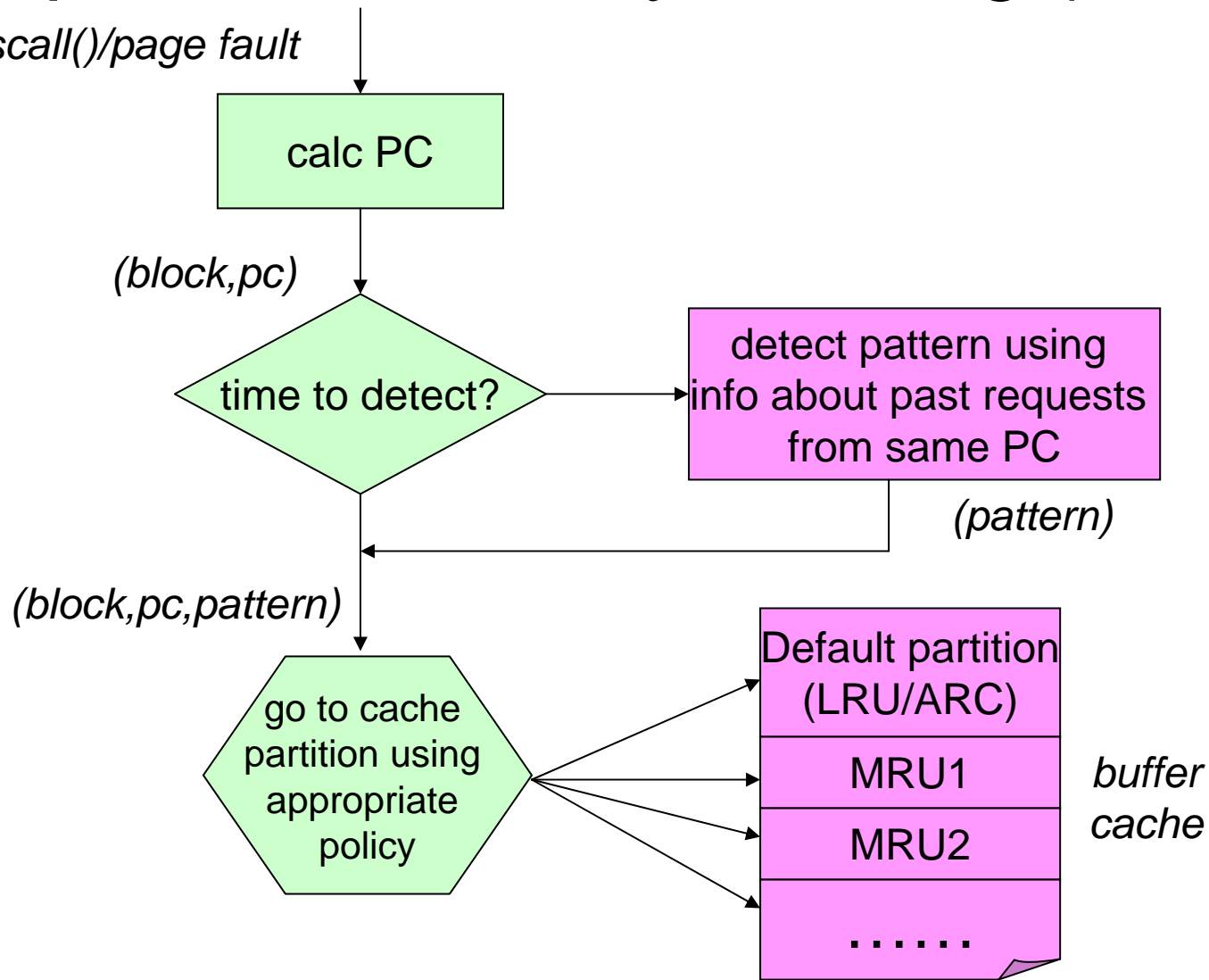
# Contributions of AMP

- PC-specific organization that treats requests from different *program contexts* differently \*
- Robust looping pattern detection algorithm
  - reliable with irregularities
- Randomized partitioned cache management scheme
  - much cheaper than previous methods

\* *Same idea is developed concurrently by Gniady et al (PCC at OSDI'04)*

# Adaptive Multi-Policy Caching (AMP)

*fs syscall()/page fault*





# Looping pattern detection

- Intuition:

- Looping streams always access blocks that has not been accessed for the longest period of time, i.e. the *least recently used* blocks.

1 2 3 1 2 3

- Streams with locality (temporally clustered streams) access blocks that has been accessed recently, i.e. *recently used* blocks.

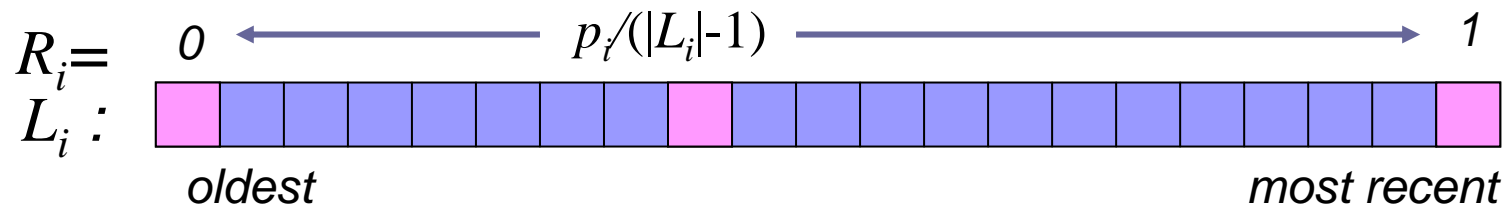
1 2 3 3 4 3 4

- What AMP does: measure a metric we call *average access recency* of all block accesses

# Loop detection scheme

- For the  $i$ -th access

- $L_i$ : list of all previously accessed blocks, ordered from the oldest to the most recent by their last access time.
- $p_i$ : position in  $L_i$  of the block accessed (0 to  $|L_i|-1$ )
- **Access recency**:  $R_i = p_i / (|L_i| - 1)$





## Loop detection scheme cont.

- *Average access recency*  $\bar{R} = \text{avg}(R_i)$
- Detection result:
  - *loop*, if  $\bar{R} < T_{loop}$  (e.g. 0.4)
  - *temporally clustered*, if  $\bar{R} > T_{tc}$  (e.g. 0.6)
  - *others*, o.w. (near 0.5)
- Sampling to reduce space and computational overhead



# Example: loop

- Access stream: [1 2 3 1 2 3]

$i$	block	$L_i$	$P_i$	$R_i$
1	1	empty	$\perp$	$\perp$
2	2	1	$\perp$	$\perp$
3	3	1 2	$\perp$	$\perp$
4	1	1 2 3	0	0
5	2	2 3 1	0	0
6	3	3 1 2	0	0

- $\bar{R} = 0$ , detected pattern is *loop*

# Example: non-loop

- Access stream: [1 2 3 4 4 3 4 5 6 5 6],  $\bar{R} = 0.79$

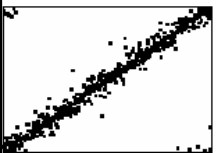




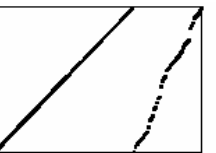

$i$	block	$L_i$	$p_i$	$R_i$
1	1	empty	⊥	⊥
2	2	1	⊥	⊥
3	3	1 2	⊥	⊥
4	4	1 2 3	⊥	⊥
5	4	1 2 3 4	3	1
6	3	1 2 3 4	2	0.667
7	4	1 2 4 3	2	0.667
8	5	1 2 3 4	⊥	⊥
9	6	1 2 3 4 5	⊥	⊥
10	5	1 2 3 4 5 6	4	0.8
11	6	1 2 3 4 6 5	0	0.8



# Randomized Cache Partition Management

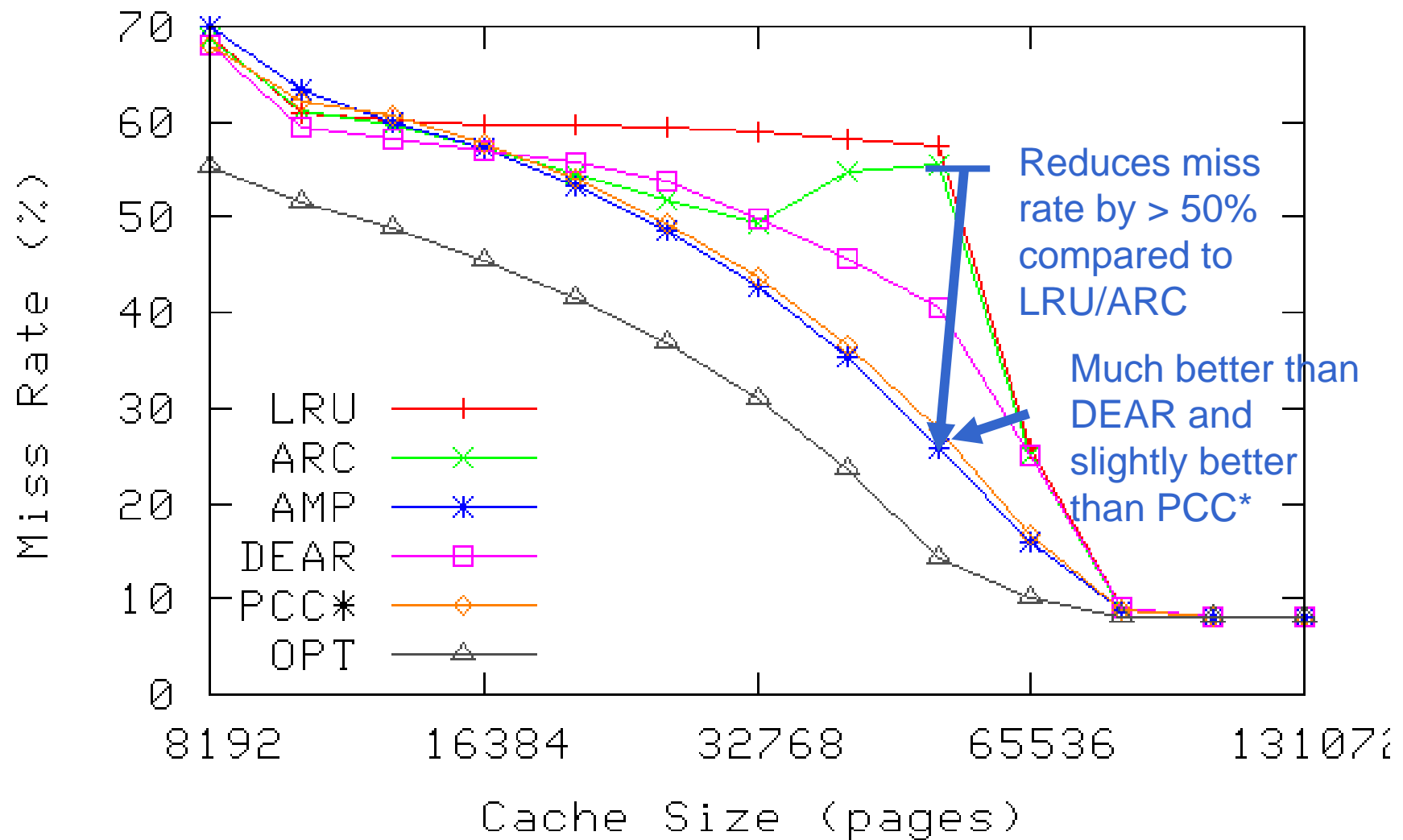
- Need to decide cache sizes devoted to each PC
- Marginal gain (MG)
  - the expected number of extra hits over unit time if one extra block is allocated
  - Local optimum when every partition has the same MG
- Randomized scheme
  - Expand the default partition by one if ghost buffer hit
  - Expand an MRU partition by one every  $loop\_size/ghost\_buffer\_size$  accesses to the partition
  - Expansion is done by taking a block from a random other part.
- Compared to UBM and PCC
  - $O(1)$  and does not need to find smallest MG

# Robustness of loop detection

							
AMP $\overline{R}$	tc 0.755	loop 0.001	loop 0.347	tc 0.617	loop 0.008	loop 0.010	other 0.513
DEAR	other	loop	<i>other</i>	other	loop	<i>other</i>	other
PCC	<i>loop</i>	loop	loop	<i>loop</i>	loop	<i>other</i>	<i>loop</i>

“tc”=temporally clustered  
**Colored** detection results are wrong  
 Classifying *tc* as *other* is deemed correct.

# Simulation: *dbt3 (tpc-h)*





# Implementation

- Kernel patch for Linux 2.6.8.1
- Shortens time to index Linux source code using glimpseindex by up to 13% (read traffic down 43%)
- Shortens time to complete DBT3 (tpc-h) DB workload by 9.6% (read traffic down 24%)
  
- <http://www.cs.berkeley.edu/~zf/amp>
- Tech report
- Linux implementation
- General buffer cache simulator