PEKING UNIVERSITY

Kinetic Modeling of Data Eviction in Cache

Xiameng Hu, Xiaolin Wang, Lan Zhou, Yingwei Luo

Peking University

Chen Ding

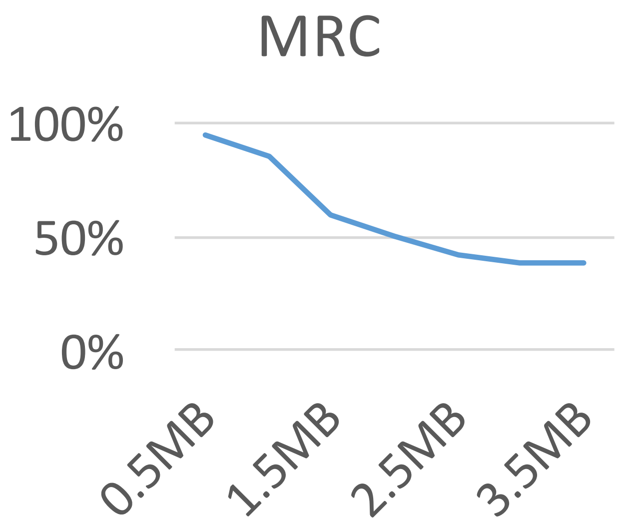
University of Rochester

Zhenlin Wang

Michigan Technological University

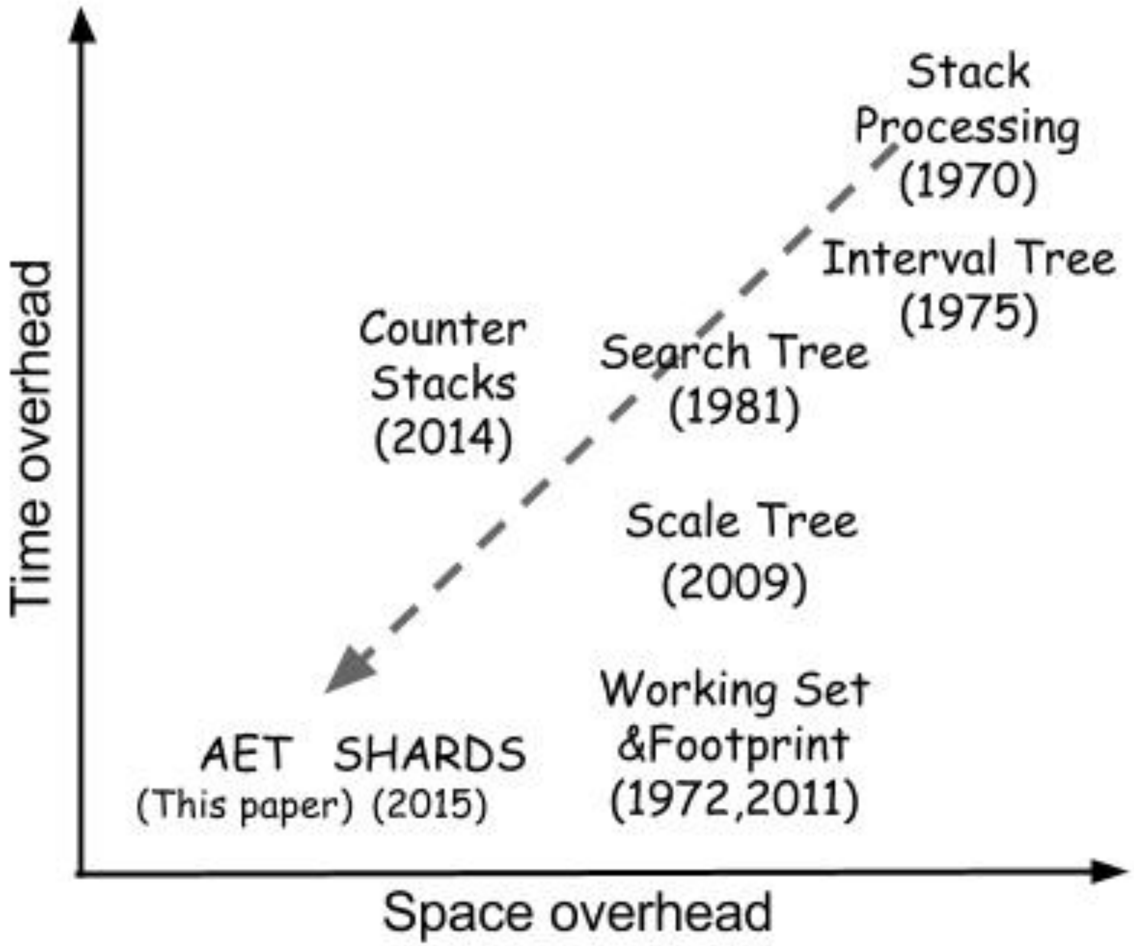
Background

- Miss Ratio Curve (MRC) is a powerful metric for cache optimization:
  - Allocation, Partition, Scheduling, QoS managing…
- Online MRC profiling techniques have been developed for decades.
- Ultimate goals:
  - Less space consumption.
  - Lower time complexity.



Background

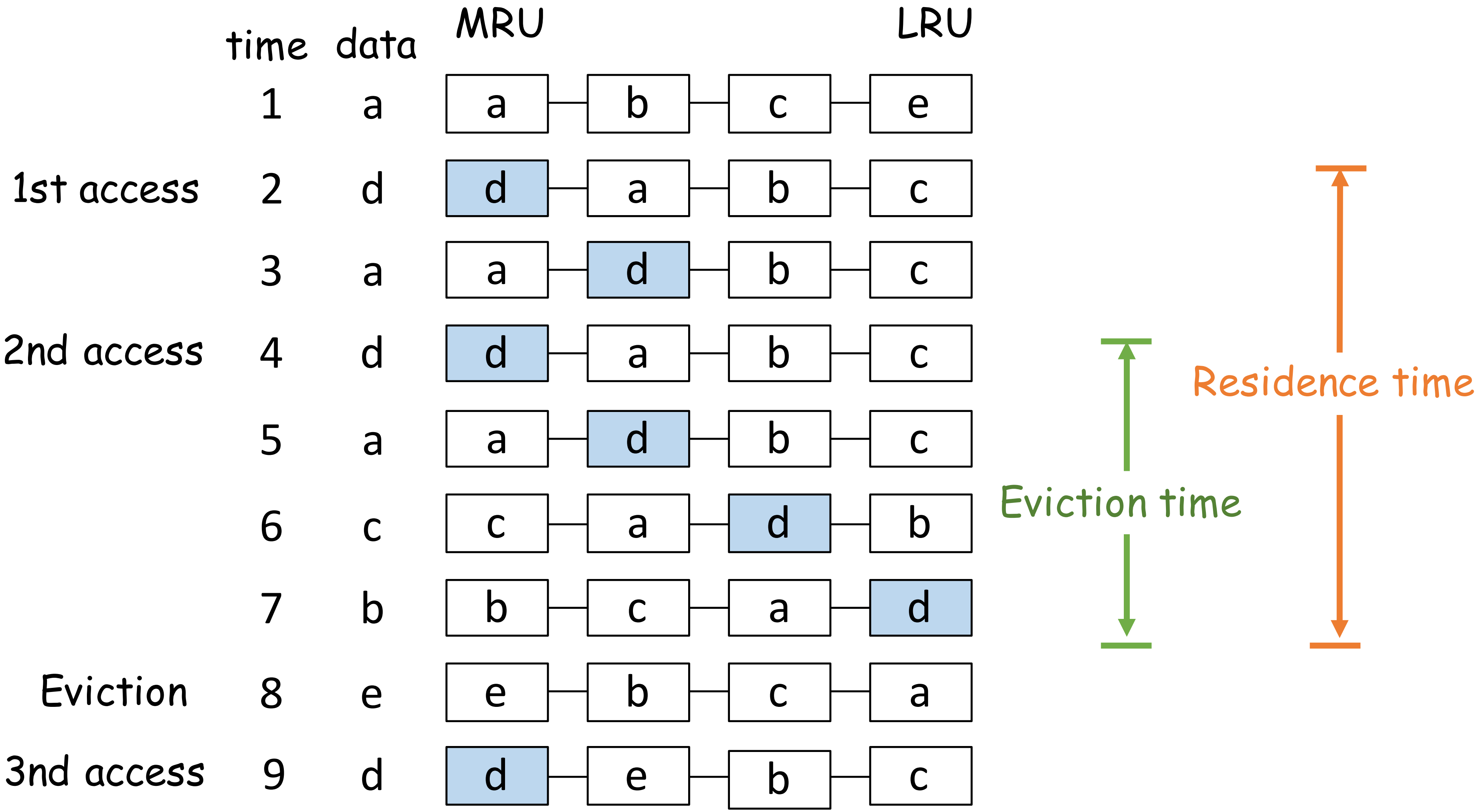
- A brief history of MRC techniques.



Our Model: Average Eviction Time

- Linear time
- Constant space
- Composability

Eviction Time



Eviction Time

- The *eviction time* is the time between the last access and the eviction.
- Property of eviction time:
  - If the *reuse time* of an access is larger than it's *eviction time*, it's a miss.
  - *Reuse time*: the time between an access and its next reuse. The reuse time of cold miss is defined as infinite.

Back to the example

		time	data	MRU			LRU
		1	a	a	b	c	e
Reuse time = ∞	Cold Miss!	2	d	d	a	b	c
		3	a	a	d	b	c
Reuse time = 2	Hit!	4	d	d	a	b	c
		5	a	a	d	b	c
		6	c	c	a	d	b
		7	b	b	c	a	d
		8	e	e	b	c	a
Reuse time = 5	Miss!	9	d	d	e	b	c

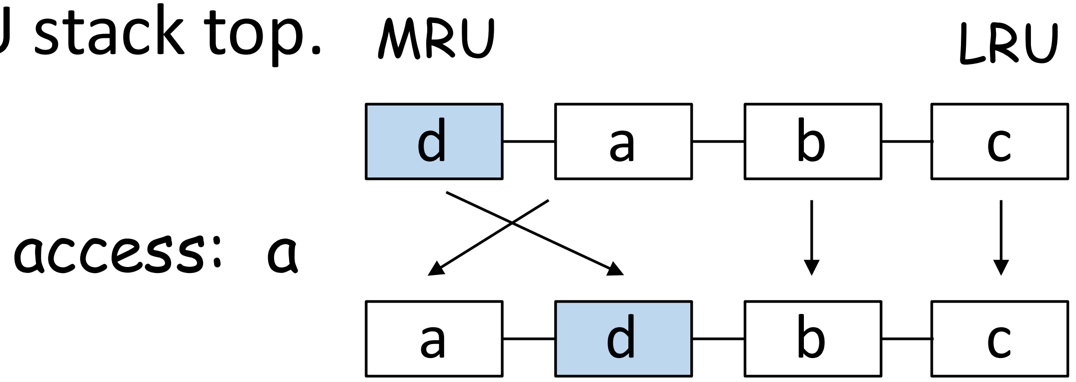
Eviction time = 4

Average Eviction Time

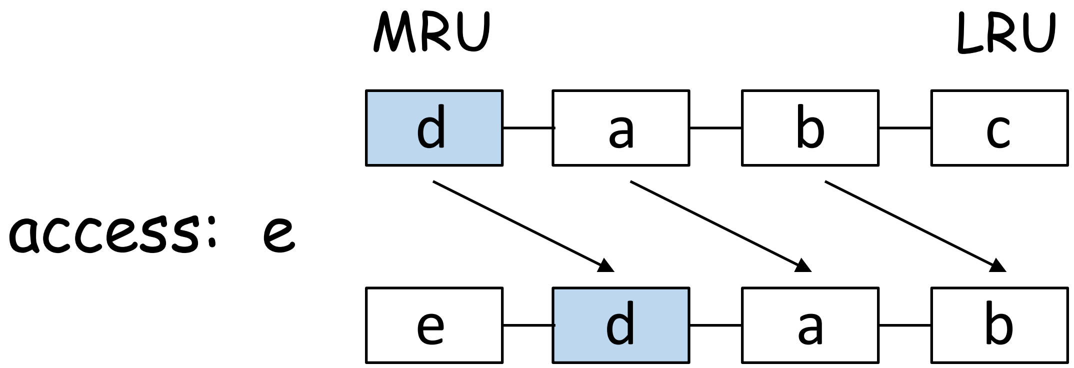
- *Average Eviction Time* (AET) is the mean eviction time of all data evictions in a fully associative LRU cache.
- We can assume all data references with a reuse time larger than AET are misses.

How to model AET?

- Move condition #1:
  - Cache hit inserts the *lower priority position* data to the LRU stack top.

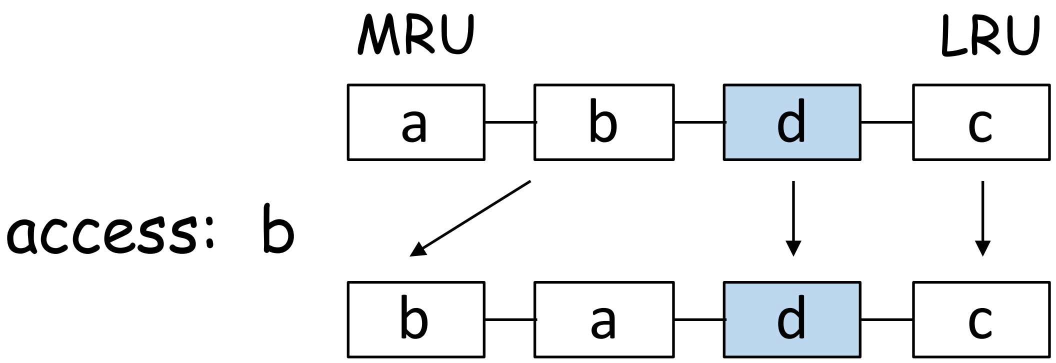


- Move condition #2:
  - Cache miss inserts a *missed* data to the LRU stack top.



How to model AET?

- Stay condition :
  - Cache hit inserts the *higher priority position* data to the LRU stack top.



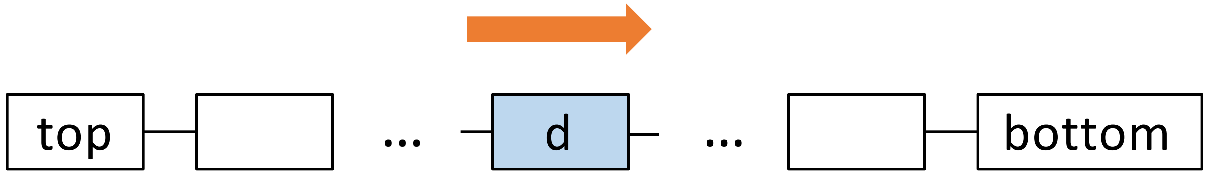
How to model AET?

- We define the *arrival time* T_m as the expected time it takes for an evicting data to reach the *m*-th position (from its last access).
- A data block at position m move one step down whenever the reuse time of current access is greater than the T_m.
- $P(t)$ is the probability for an access with a reuse time greater than t .
- The movement condition is now a probability. Every access , a data block at stack position m moves by $P(T_m)$.

Kinetic Model

- Data travels in one direction with changing speed:

$$V(t) = P(t)$$



- In general, if the time that evicting data already traveled is t, its' current evicting speed is $P(t)$.

Average Eviction Time

- ***Physics***: the integration of speed over time is travel distance.
- The length of LRU list is the travel distance of every eviction. Which is the cache size c.

$$\int_0^{AET(c)} P(t)dt = c$$

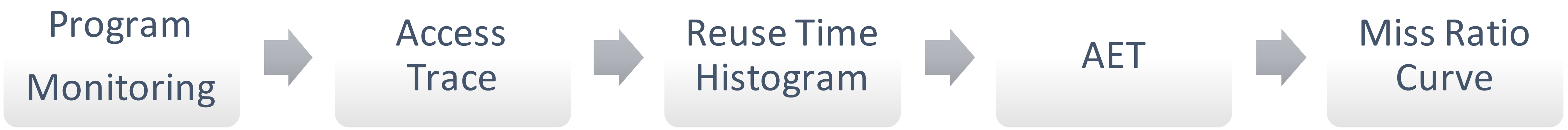
- With P, we calculate AETs of different cache sizes in linear time.
- P can be acquired online by monitoring the *reuse time histogram*.

From AET to MRC

- The miss ratio $mr(c)$ at cache size c is the probability that a reuse time is greater than the average eviction time $AET(c)$:

$$mr(c) = P(AET(c))$$

AET Design Overview



Random Sampling

- Randomly pick current accessed data to monitor its reuse time.
- The distance between two sampled is a random value.
- Constrain the random value range to control sampling rate.
- A hash table is required. It maintains current monitored data.
- The space consumption is linear but limited.

Reservoir Sampling

- To bound the space cost to constant. $O(1)$
- When the i-th sampled data arrives, reservoir sampling keeps the new data in monitoring set with probability $\min(1, k/i)$ and randomly discards an old data when the set is full.
- It ensures the equal probability for every sampled reuse to update reuse time histogram.
- While the number of samples be recorded is bounded.

AET in Shared Cache

- Composability： co-run behavior can be computed from the metric of solo-runs.
- When n programs share the cache of size c, all $n+1$ co-run $AETs$, $AET_i(c)$ for each program *i* and $AET(c)$ for the group, are the same:

$$AET_1(c) = AET_2(c) = \cdots = AET_n(c) = AET(c)$$

- Detailed modeling is described in paper.

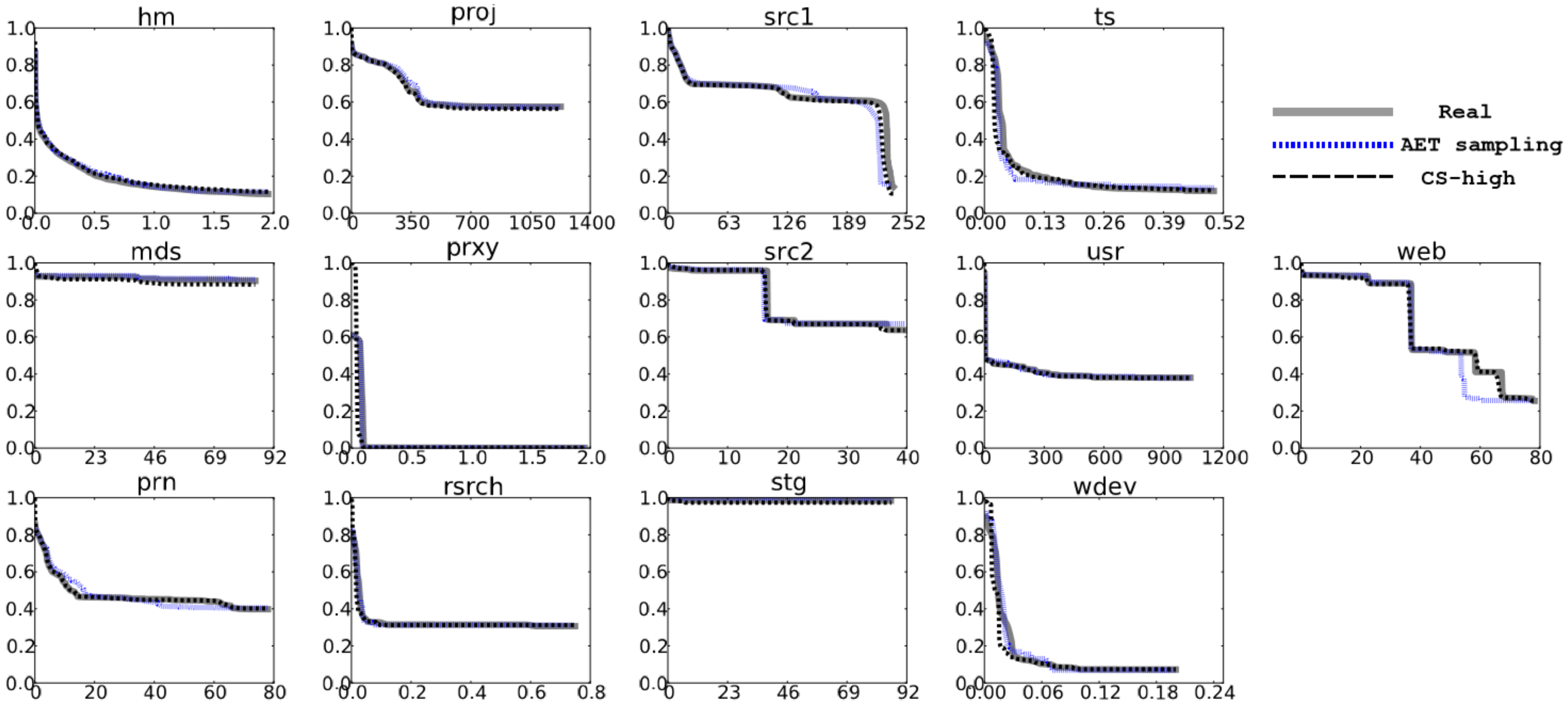
Evaluation

- AET vs Counter Stacks (OSDI’14)
- AET vs SHARDS (FAST’15)
- Shared Cache AET

AET vs Counter Stacks

- Counter Stacks:
  - Only requires extremely small space while maintaining an acceptable accuracy.
  - HyperLogLog counter to track reuse distance.
  - Balance accuracy and space by limiting the number of counters.
- Benchmarks:
  - Microsoft Research Cambridge (MSR) storage traces.
  - Configured with only read requests of 4KB cache blocks.

AET vs Counter Stacks



AET vs Counter Stacks

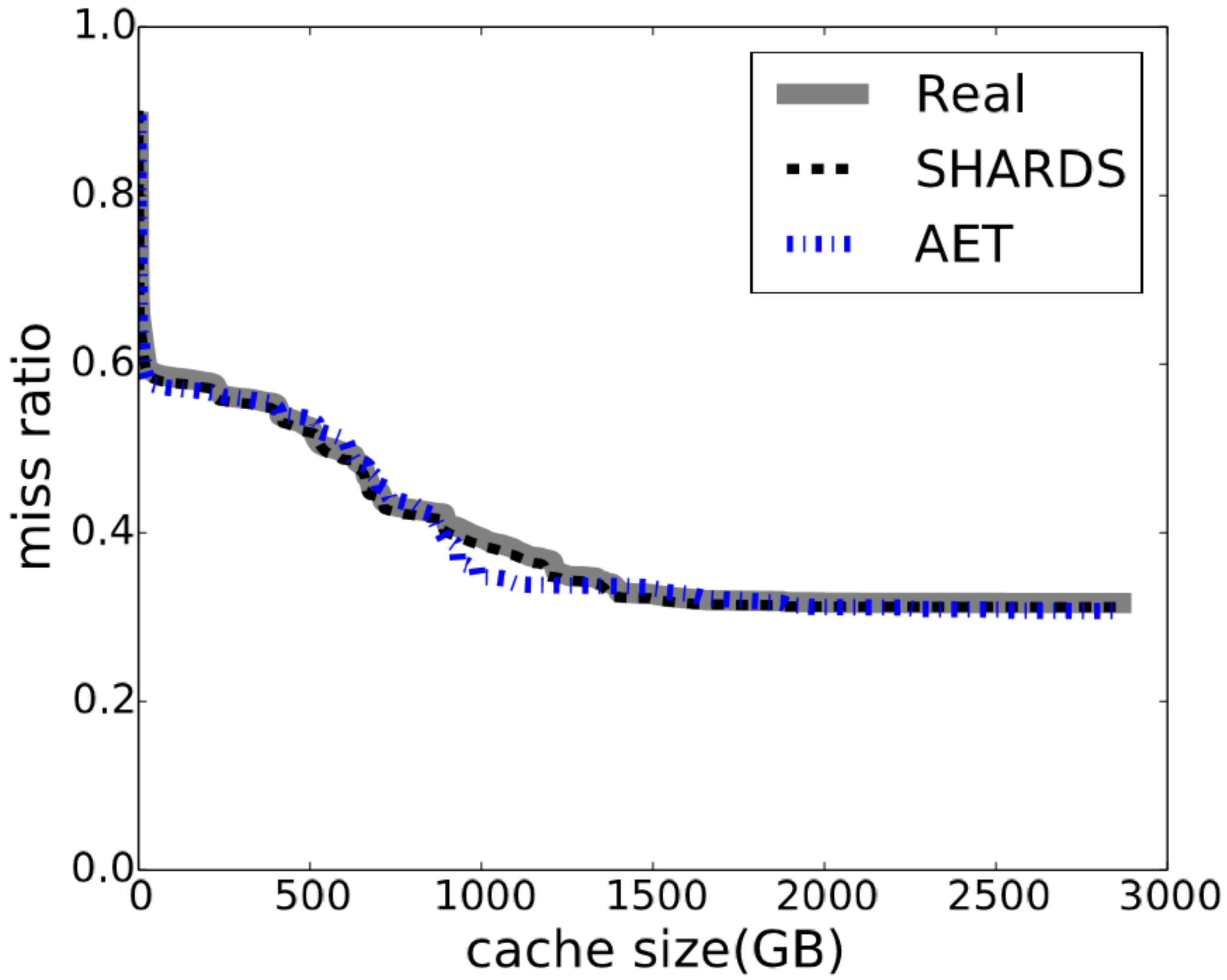
	AET Random Sampling $(1 * 10^{-5})$	**AET Reservoir Sampling 8k entries**	**Counter Stacks High fidelity (d = 1M, s = 60, δ = 0.02)**	**Counter Stacks Low fidelity (d = 1M, s = 3600, δ = 0.1)**
Mean Absolute Error	0.96%	1.12%	0.77%	1.26%
Average Space Cost	452KB	384KB	7363KB	1292KB
Average Throughput	63.99M reqs/sec	61.99M reqs/sec	1.73M reqs/sec	5.86M reqs/sec

AET vs SHARDS

- SHARDS:
  - hash-based spatial sampling
  - a splay tree to track the reuse distances of the sampled data.
  - Limits the space overhead to a constant by adaptively lowering the sampling rate.

- Benchmarks:
  - "master" MSR, which is a 2.4 billion-access trace combining all 13 MSR traces by ranking the time stamps of all accesses.

AET vs SHARDS



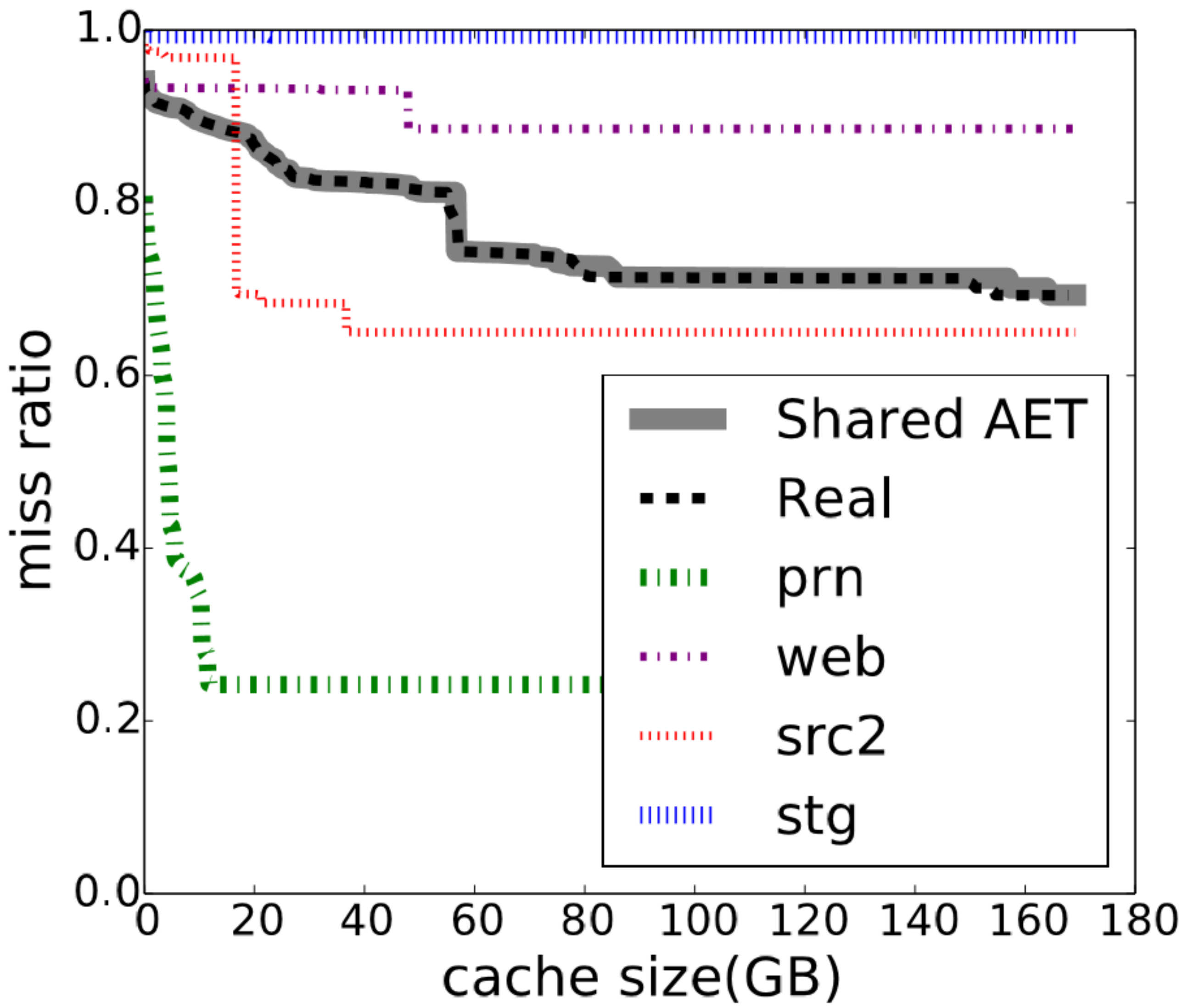
AET vs SHARDS

	AET Random Sampling $(1 * 10^{-5})$	AET Reservoir Sampling 8k samples	SHARDS 8k samples	Counter Stacks
Mean Absolute Error	1%	1%	0.6%	0.3%
Average Space Cost	1.7MB	1.4MB	2.3MB	80MB
Average Throughput	79M reqs/sec	66.6M reqs/sec	81.4M reqs/sec	3.2M reqs/sec

Shared Cache AET

- We choose Four MSR storage traces {prn, src2, web, stg} as a co-run group.
- Generate a combined trace from the four traces under equal speed assumption.
- We compare MRC composed by individual AET modeling of each trace, as well as the real MRC of the combined trace.

Shared Cache AET



Summary

- A new model to characterize cache behavior.
  - Enable fast MRC profiling with O(1) space and O(n) time.
  - Predict shared cache MRC without co-run testing.
  - Perfect for online deployment with limited overhead.

	Time complexity	Space complexity	Memory	Runtime	Composability	Correctness
Stack Processing	$O(NM)$	$O(N)$	10GB	> 1 day	No	accurate
Search Tree	$O(N \log M)$	$O(M)$	21GB	482 secs	No	accurate
Scale Tree	$O(N \log \log M)$	$O(M)$	17GB	333 secs	No	bounded err
Footprint	$O(N)$	$O(M)$	17GB	50 secs	Yes	conditional
Counter Stacks	$O(N \log M)$	$O(\log M)$	80MB	1034 secs	No	bounded err
SHARDS	$O(N)$	$O(1)$	2.3MB	29.6 secs	No	conditional
AET model	$O(N)$	$O(1)$	1.7MB	30.5 secs	Yes	conditional

PEKING UNIVERSITY

Thank you for your attention!

Q&A

Email: hxm@pku.edu.cn

AET vs StatStack

- StatStack:
  - Designed for CPU workloads.
  - It samples cache blocks and measures their reuse time using performance counters and watchpoints.
  - Reuse time histogram -> Reuse distance histogram.
- Benchmarks:
  - SPEC CPU2006, 30 benchmarks.
  - For each benchmark, we intercept 1 billion references from their execution using the instrumentation tool Pin.
  - We measure the cumulative distribution function (CDF) of absolute error of full-trace StatStack, full-trace AET, sampling AET.

AET vs StatStack