Locating Cache Performance Bottlenecks Using Data Profiling

Aleksey Pesterev Nickolai Zeldovich Robert T. Morris

Computer Science and Artificial Intelligence Lab
Massachusetts Institute of Technology

EuroSys 2010

DProf

Classical profilers attribute time to source lines.
DProf attributes (cache miss) time to data.

Sampling

Access Samples IBS/PEBS gather IP, cache level, and latency for random instructions; type deduced from address

Access Histories Debug registers gather all accesses to some memory.

What does DProf give you? (Middle layer)

Address Set Set of all addresses (and thus cache sets) used for objects of some type.

What does DProf give you? (Middle layer)

Address Set Set of all addresses (and thus cache sets) used for objects of some type.

Path Traces Graph that shows all possible flows of accesses to objects of some type, each with access latency, cache hit rate, etc.

DProf profiles

For each data type DProf can collect

Data Profile Cache miss rates (and bounce flag),

DProf profiles

For each data type DProf can collect

Data Profile Cache miss rates (and bounce flag),

Working Set Total size and count of objects in working set,

DProf profiles

For each data type DProf can collect

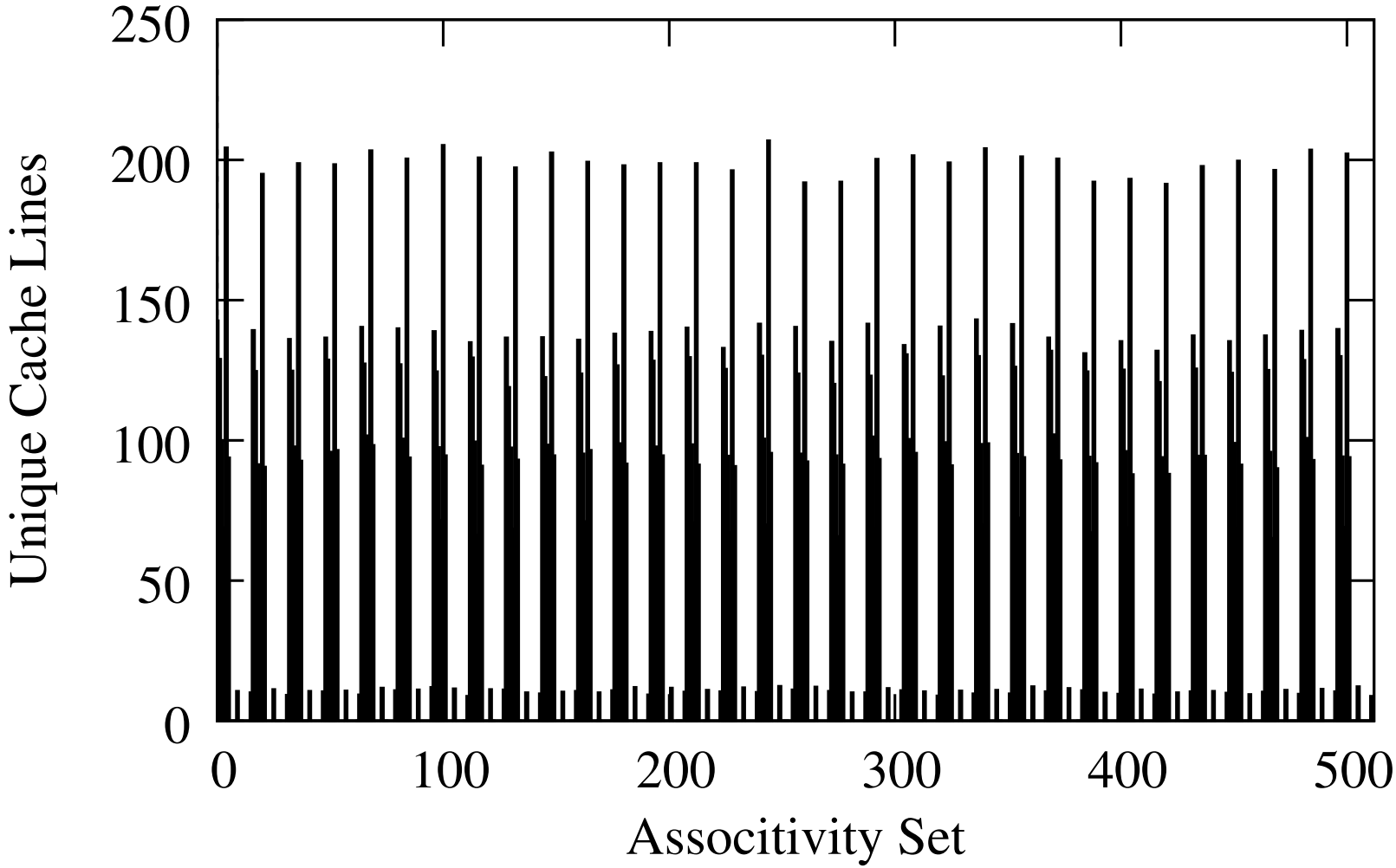
Data Profile	Cache miss rates (and bounce flag),
Working Set	Total size and count of objects in working set,
Miss Classification	$REASON cache miss rates,

DProf profiles

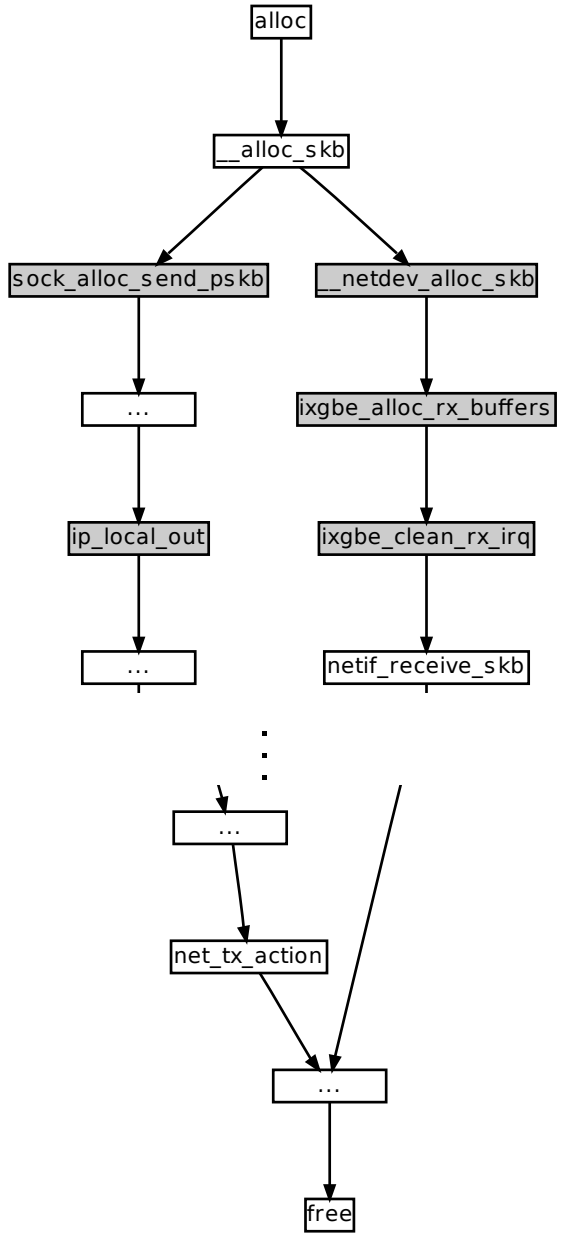
For each data type DProf can collect

Data Profile	Cache miss rates (and bounce flag),
Working Set	Total size and count of objects in working set,
Miss Classification	$REASON cache miss rates,
Data Flow	Common sequences of functions that reference objects of that type.

Working Set Example



Data Flow Example



Evaluation

Evaluation

16 core AMD with 10GB Ethernet + 16 load generating machines

Evaluation

16 core AMD with 10GB Ethernet + 16 load generating machines

Case Study 1 Fixing unintended data sharing between cores (true sharing cache miss) improved Memcached performance by 57%

Evaluation

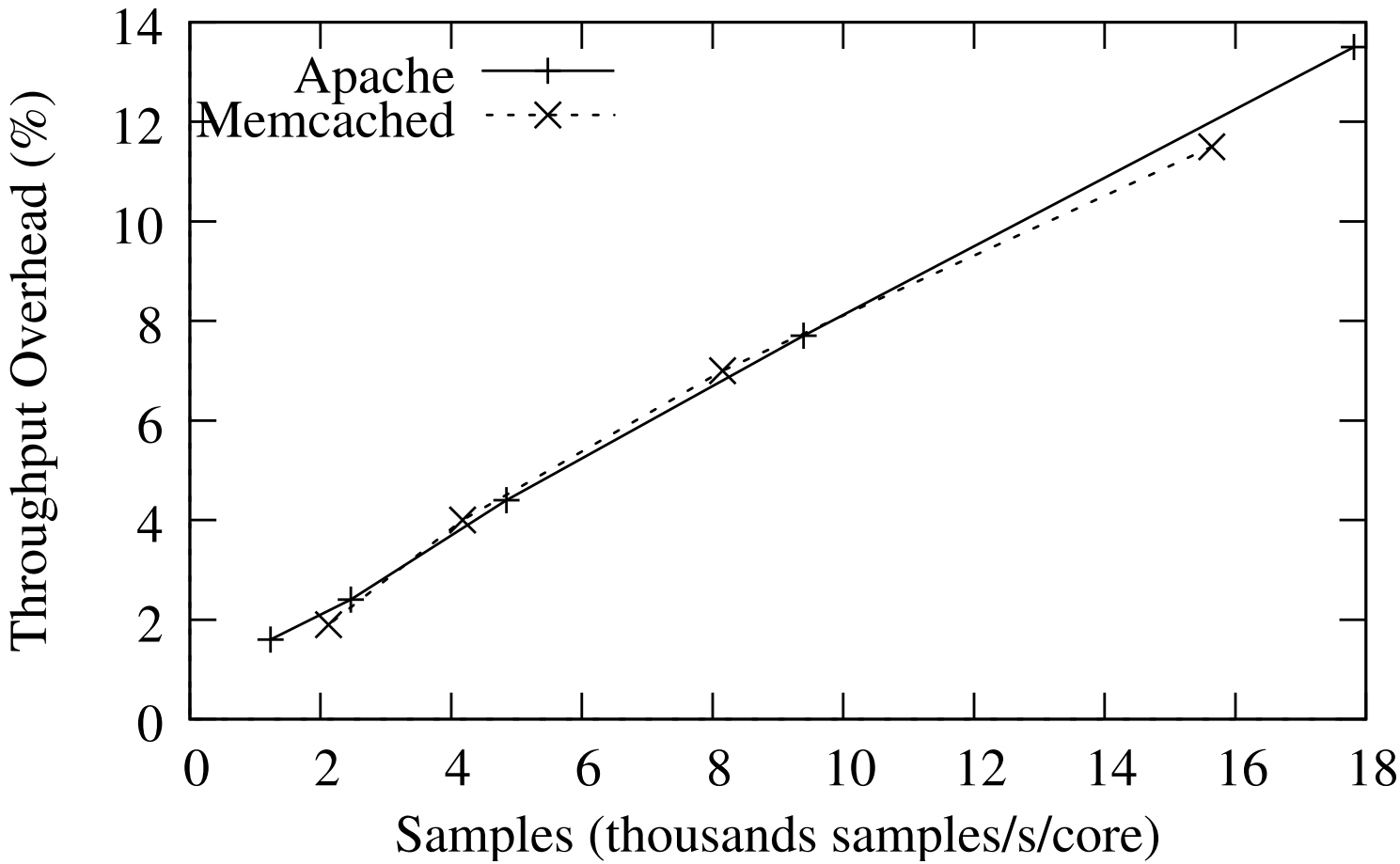
16 core AMD with 10GB Ethernet + 16 load generating machines

Case Study 1 Fixing unintended data sharing between cores (true sharing cache miss) improved Memcached performance by 57%

Case Study 2 Apache request serve rate dropped at high request generation rate. Limiting request queue fixed that (16% speedup).

Type Name	% of all L1 misses	% of all L1 misses
`tcp_sock`	11.0%	21.5%
`task_struct`	21.4%	10.7%
`net_device`	3.4%	12.0%
`size-1024`	5.2%	4.1%
`skbuff`	3.3%	3.7%

Evaluation – IBS overhead



Evaluation – Access history collection overhead

Benchmark	Data Type	Data Type Size (bytes)	Histories	Histories Sets	Collection Time (s)	Overhead (%)
memcached	size-1024	1024	8128	32	170	1.3
	skbuff	256	5120	80	95	0.8
Apache	size-1024	1024	20320	80	34	2.9
	skbuff	256	2048	32	24	1.6
	skbuff_fclone	512	10240	80	2.5	16
	tcp_sock	1600	32000	80	32	4.9

Table 10. Object access history collection times and overhead for different data types and applications.

Benchmark	Data Type	Elements per History	Histories per Second	Elements per Second
memcached	size-1024	0.3	53	120
	skbuff	4.2	56	350
Apache	size-1024	0.5	660	1660
	skbuff	4.8	110	770
	skbuff_fclone	4.0	4600	27500
	tcp_sock	8.3	1030	10600

Table 11. Average object access history collection rates for different data types and applications.

Change in Linux kernel for benchmarking?
Why use Linux in the first place?
Time for post-processing?
Are the case studies realistic?