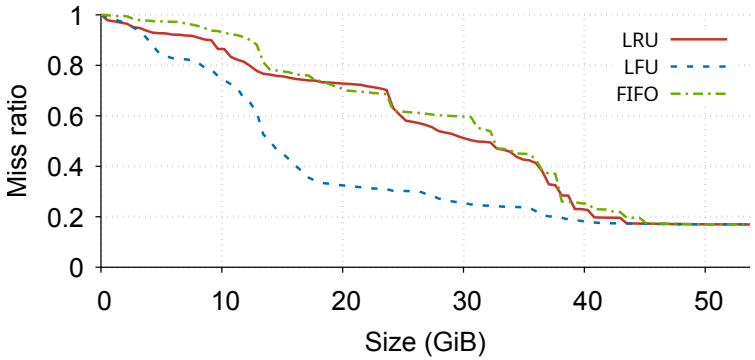
PaperCache: In-Memory Caching with Dynamic Eviction Policies

Kia Shakiba and Michael Stumm

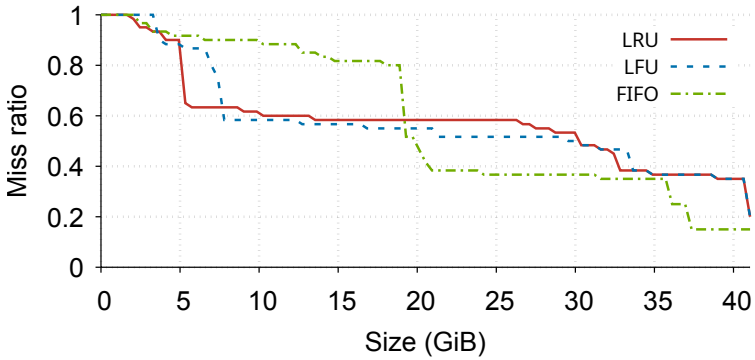
University of Toronto

Eviction policy performance



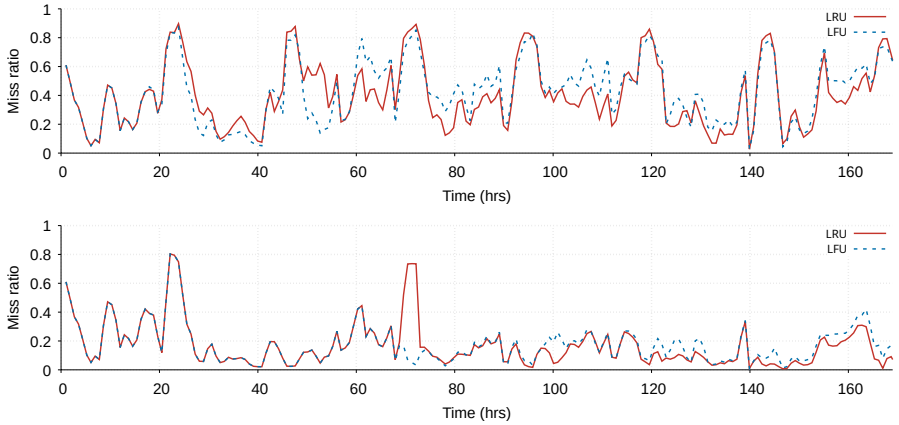
A cache's eviction policy significantly affects its performance.

Eviction policy performance



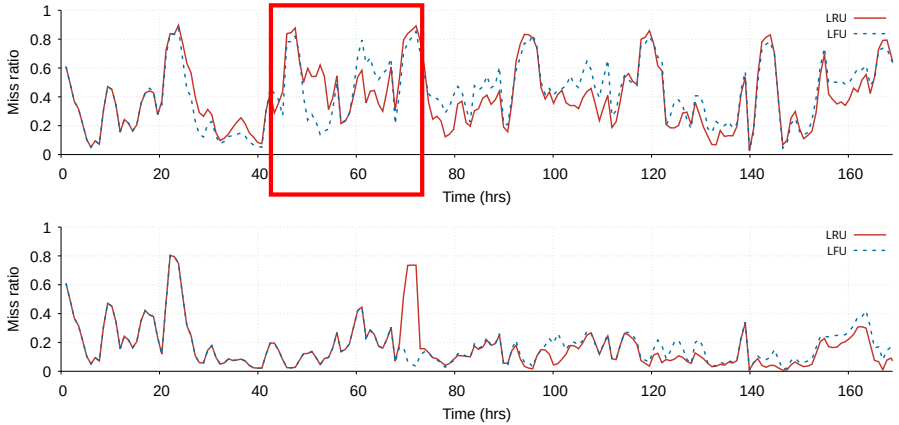
The optimal eviction policy depends on the cache's configured size.

Eviction policy performance



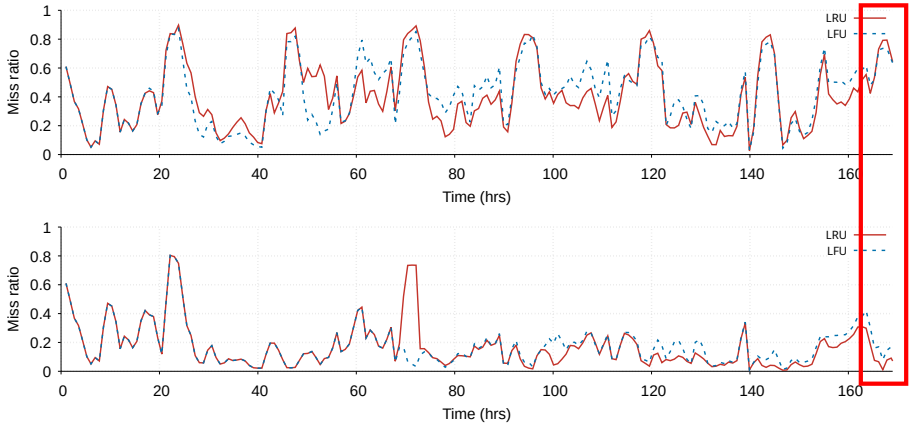
The optimal eviction policy changes over time.

Eviction policy performance



The optimal eviction policy changes over time.

Eviction policy performance



The optimal eviction policy changes over time.

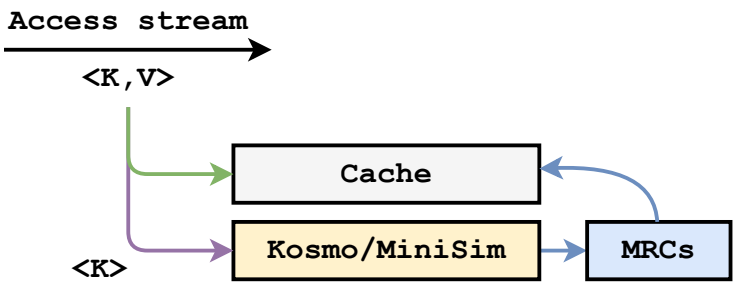
The current state-of-the-art

- Most caches have limited support of eviction policies
  - Memcached only supports LRU
  - Redis supports LRU and LFU
  - CacheLib supports many
- Eviction policies are typically statically configured
  - Redis is the only cache that can switch dynamically
- Recent modeling methods identify the optimal policy online
  - MiniSim (ATC'17), Kosmo (FAST'24)

Many eviction policies exist

Policy	**“Rule-of-Thumb”**
LRU	High data locality
LFU	Zipfian access patterns
FIFO	Large inter-arrival gaps
MRU	Cyclic or strict scanning access patterns
2Q	Scanning access patterns
S3-FIFO	“One-hit-wonders”
Arc	Scanning access patterns

How can we use MRCs?

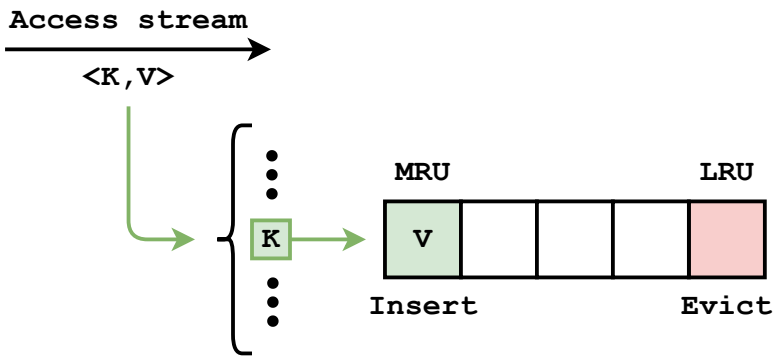


Ideally, MRCs can be used to periodically reconfigure caches.

The problem

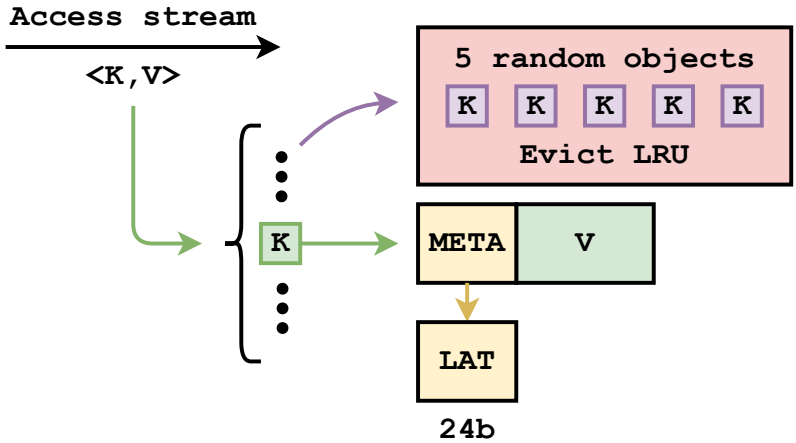
- The optimal eviction policy for a workload changes over time
- We can generate policy-specific MRCs efficiently online
- We need a cache that can:
  1. Continuously identify the optimal eviction policy
  2. Switch to said policy at runtime without negatively affecting performance

The issue with switching eviction policies



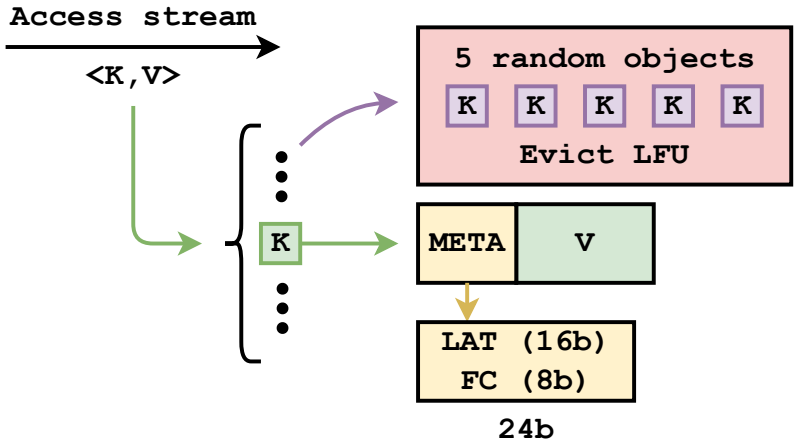
Why can't modern caches switch eviction policies at runtime?

Evictions in Redis



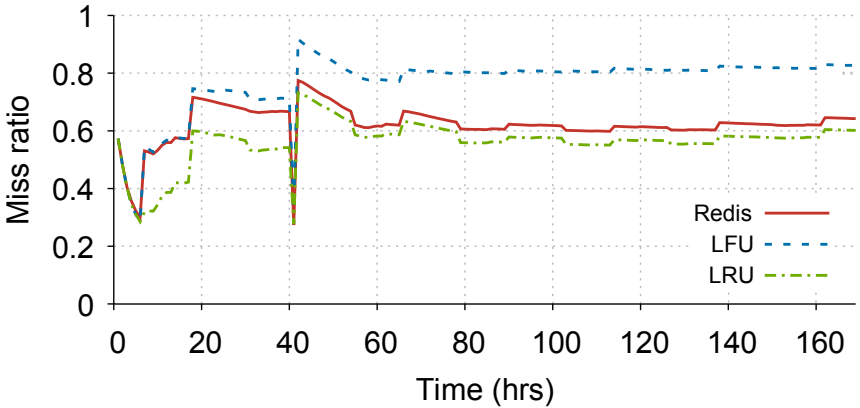
Redis uses low-overhead approximations of eviction policies.

Switching eviction policies in Redis



Switching between *simple* eviction policies is trivial in Redis.

Switching eviction policies in Redis

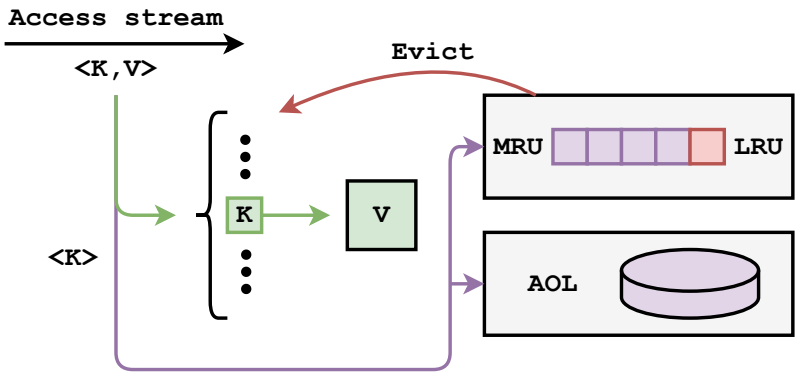


Redis switching from LFU to LRU at 40 hours.

PaperCache

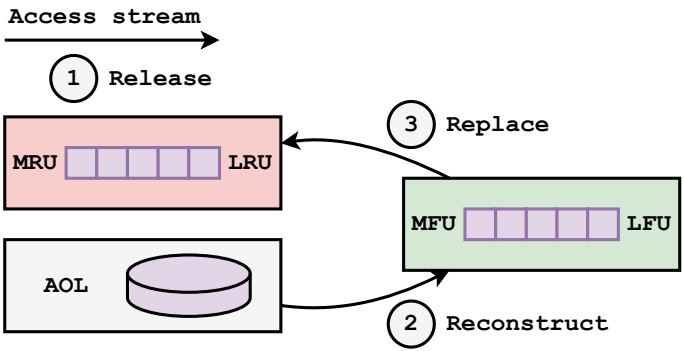
- PaperCache design components:
  - Maintain the full stack for the current eviction policy
  - For each access, save the metadata to an append-only-log
  - Maintain an approximate “MiniStack” in memory for each configured eviction policy
- Each design component operates on a separate thread

An overview of PaperCache



PaperCache handles its eviction policy stack off the main thread.

Switching eviction policies in PaperCache

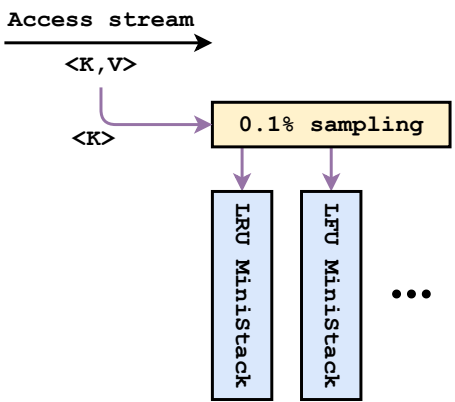


The AOL is used to reconstruct eviction policy stacks at runtime.

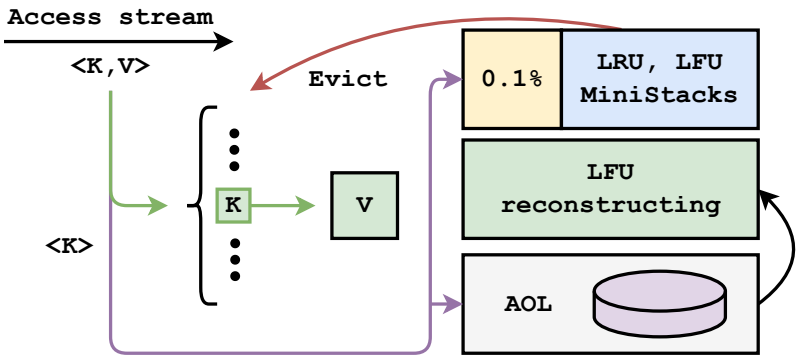
How do we evict objects during a stack reconstruction?

MiniStacks

- Maintain an approximate stack in memory for each policy
- MiniStacks perform evictions during a policy switch
- MiniStacks are 1000× smaller than FullStacks.



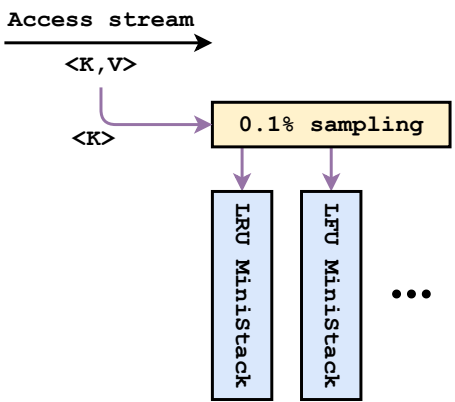
PaperCache during a policy switch



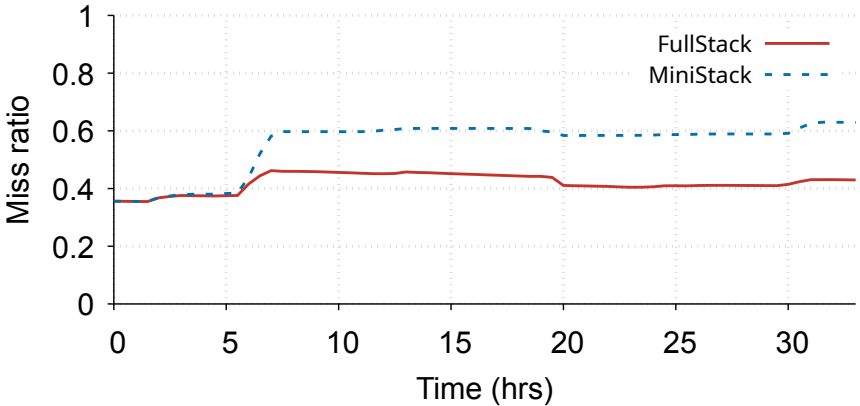
MiniStacks perform evictions during a policy switch.

MiniStack efficacy duration

- MiniStacks may shrink over time during evictions
- If a MiniStack is empty, resort to random evictions
- How long can we depend on a MiniStack for our evictions?

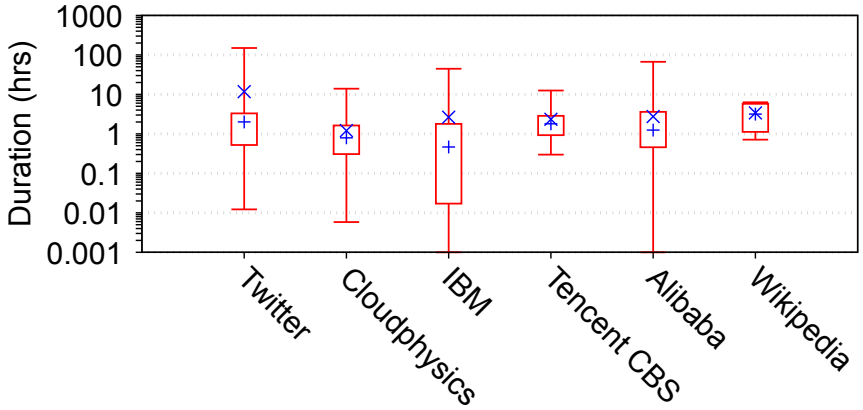


MiniStack efficacy duration



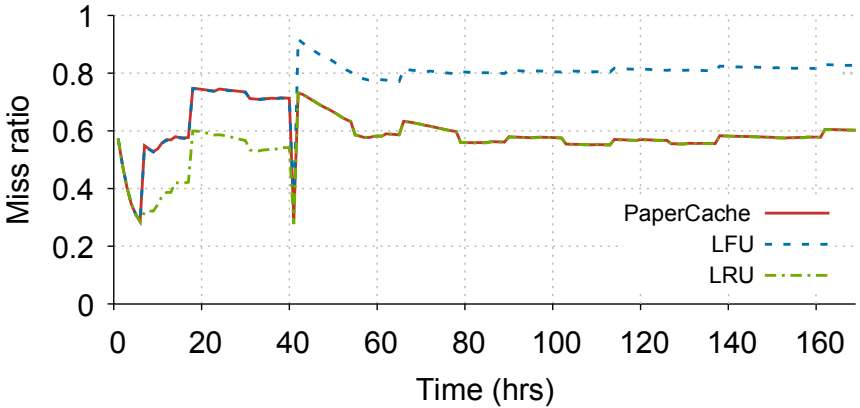
How long can a MiniStack be used before its performance degrades?

MiniStack efficacy duration



How long can a MiniStack be used before its performance degrades?

Switching eviction policies in PaperCache



PaperCache switching from LFU to LRU at 40 hours.

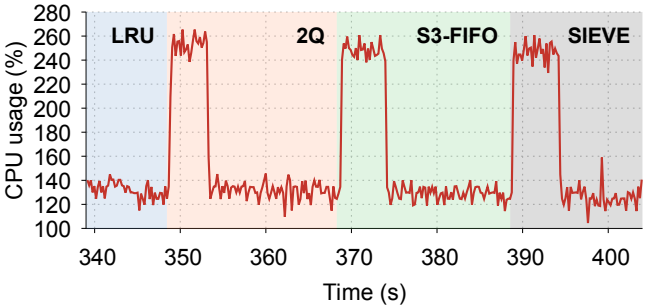
PaperCache memory overhead

	1,000	10,000	100,000	1,000,000
Redis v8.0.1	15MiB	43MiB	176MiB	270MiB
PaperCache	12MiB	53MiB	195MiB	459MiB
	−20%	+23.3%	+10.8%	+70%

PaperCache has higher memory overhead than Redis.

PaperCache latency and CPU usage

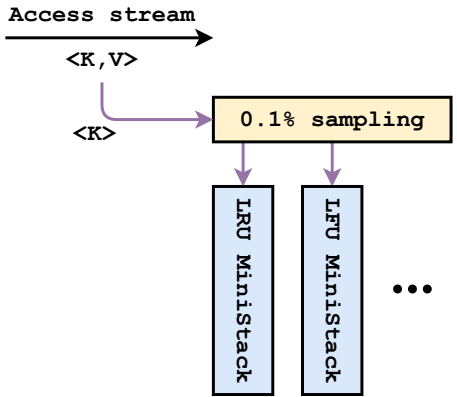
	p99 (μs)		p99.9 (μs)		p99.99 (μs)	
	GET	**SET**	**GET**	**SET**	**GET**	**SET**
Redis v8.0.1	116	595	298	1,327	600	1,889
PaperCache	66	523	177	806	417	1,070
	−43.1%	−12.1%	−40.6%	−39.3%	−30.5%	−43.4%



How do we know when to switch eviction policies?

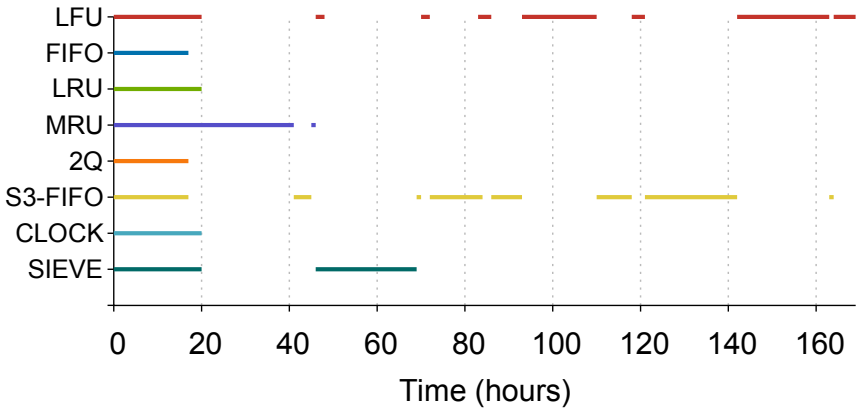
Leveraging MiniStacks for miss ratio reporting

- MiniStacks used in MiniSim to approximate miss ratios
- We can leverage MiniStacks to track miss ratios
- No need to generate full MRCs



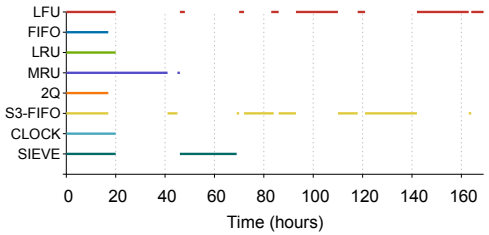
Is this really worthwhile?

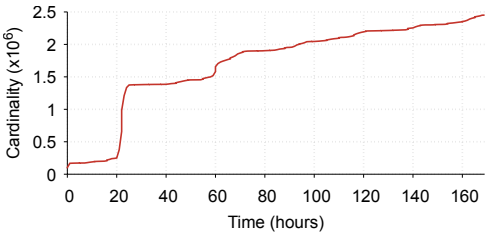
Auto-switching eviction policies in PaperCache



Instances where each PaperCache has the lowest miss ratio.

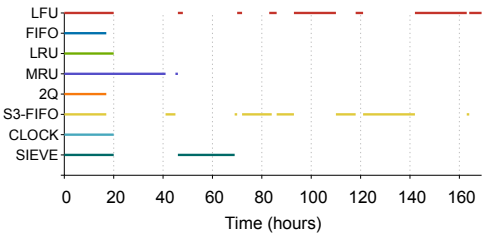
Auto-switching eviction policies in PaperCache

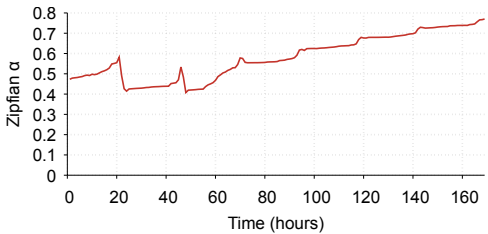




How does the workload's cardinality affect the optimal miss ratio?

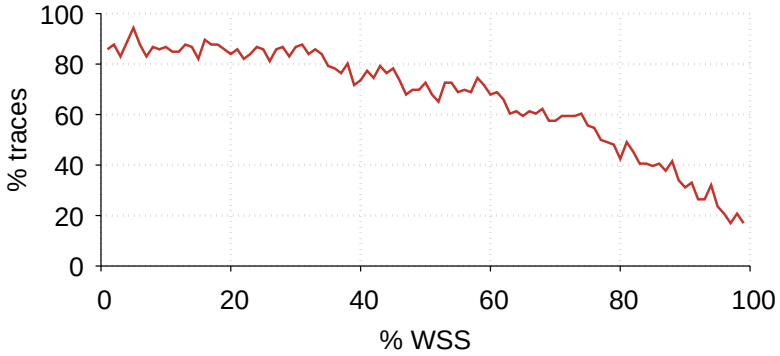
Auto-switching eviction policies in PaperCache





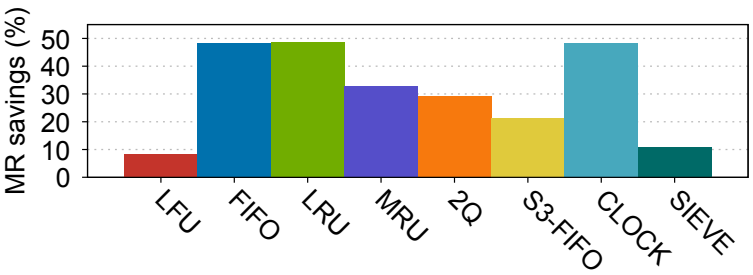
How does the workload's Zipfian behavior affect the optimal miss ratio?

Auto-switching eviction policies in PaperCache



The % of traces where the optimal eviction policy changes at least twice.

PaperCache: Performance benefits



How much can PaperCache lower the miss ratio?

Conclusion

- The optimal eviction policy of a workload changes over time
- In-memory caches lack the ability to adapt to these changes
- PaperCache can switch between any eviction policy
- PaperCache unlocks eviction policy research directions
  - Targeted eviction policies
- https://papercache.io