Reinforcement Learning-based Optimization for Solid State Disks

July 12, 2019

Sungjoo Yoo

Seoul National University

http://cmalab.snu.ac.kr

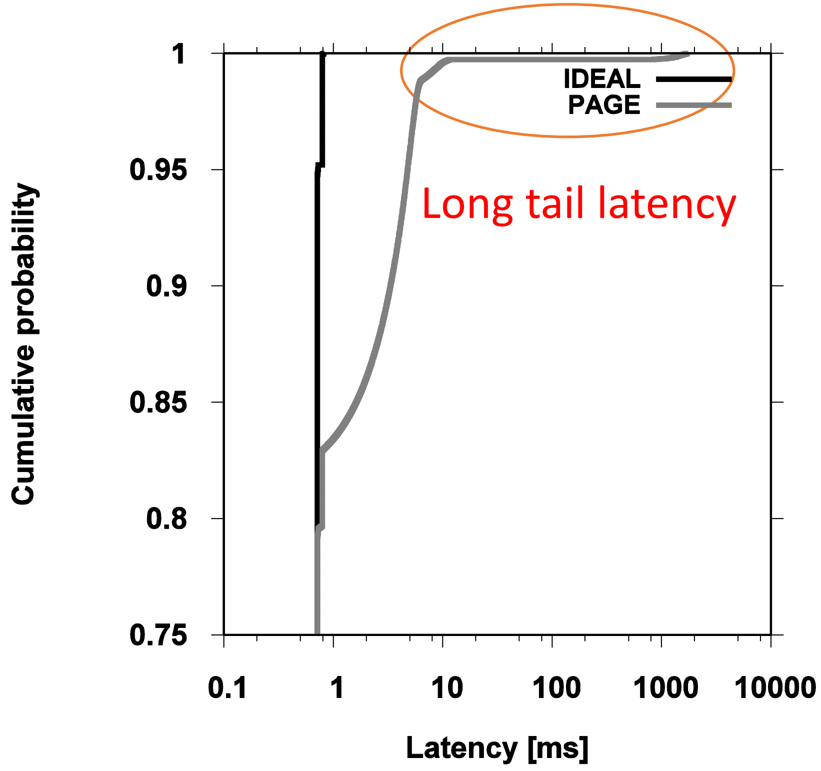
Agenda

- Our problem and solution overview
  - Long tail latency in SSD
  - Reinforcement learning (RL)
- Small Q table-based solution to reduce long tail latency
- Q table cache to exploit a large number of states at small cost
- Summary

The Tail at Scale, Communication of ACM, 2013

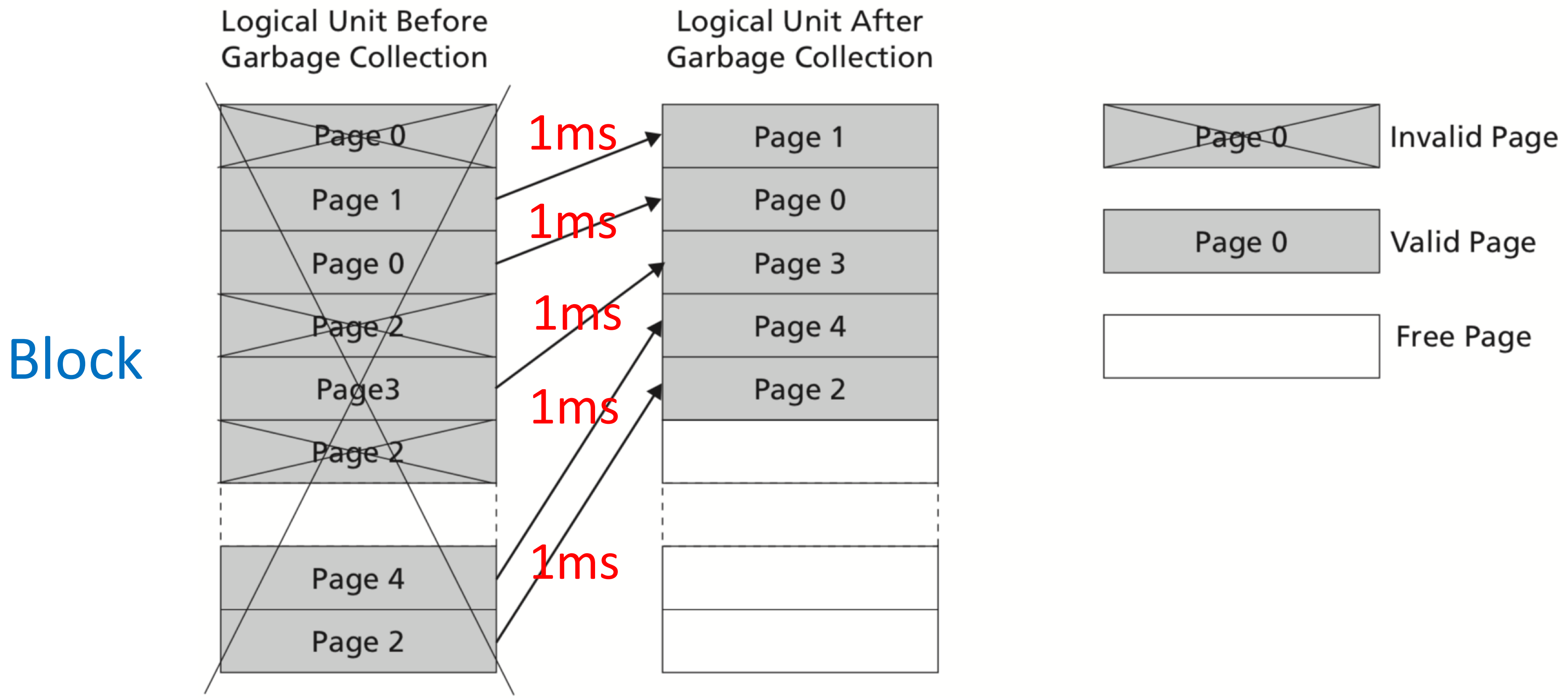
Long Tail Latency Problem in SSD

“If your read is stuck behind an erase you may have to wait 10s of milliseconds. That’s a 100x increase in latency variance”



Garbage Collection in Flash-based Storage

- In order to reclaim a block, page copy and block erase are needed
- Plane conflict during page copy can delay the service of subsequent requests on the plane



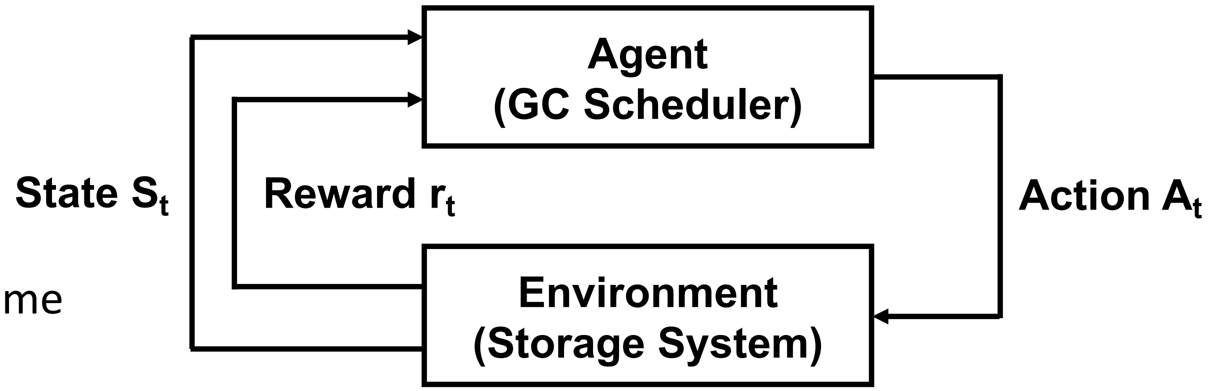
Our Approach

- Our goal is to reduce the latency incurred by garbage collection (GC)
- We aim at exploiting idle time to run partial GC operations
  - Partial GC = copying a few pages
- Reinforcement learning can
  - Learn system behavior, e.g., write intensive phase, idle time pattern, …
  - Make a choice of how many pages to copy for each of idle periods

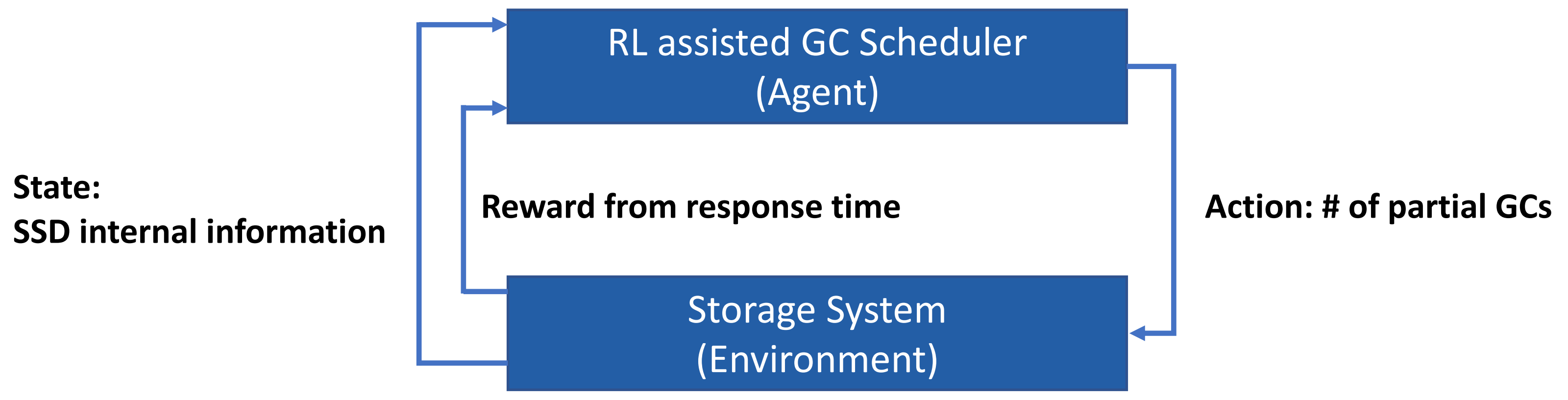
Reinforcement Learning

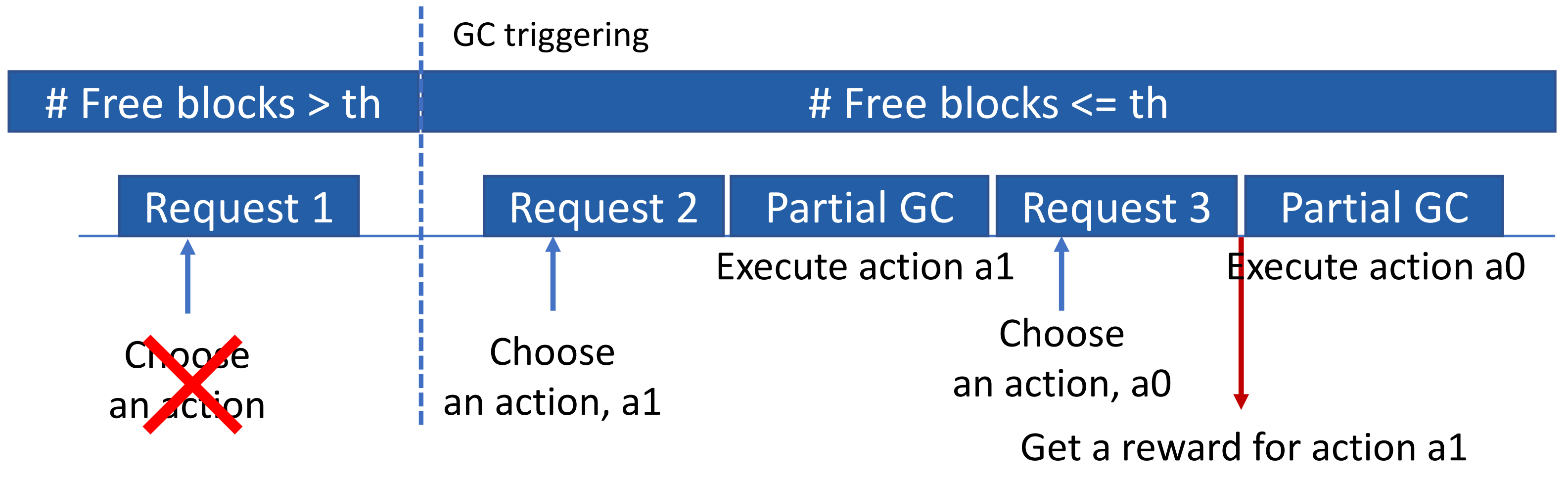
- “**Reinforcement learning** (**RL**) is an area of machine learning inspired by behaviorist psychology, concerned with how software agents ought to take *actions* in an *environment* so as to maximize some notion of cumulative *reward*.”

- State (S_t): a set of environment states
- Action (A_t): a set of actions of agent
- Reward (r_t): reward associated with last action
- Policy (π): agent’s way of action selection at a given time
- At each time step t
  - Agent: executes action A_t
    receives state S_t
    receives reward r_t
  - Environment: receives action A_t
    emits state S_t+1
    emits reward r_t+1



Solution Overview





[Kang, 2017]

Table Contains Q-Values, Expected Cumulative Reward

- States
  - Previous inter-request interval
  - Current inter-request interval
  - Previous action
- State binning
  - 68 states

Q table

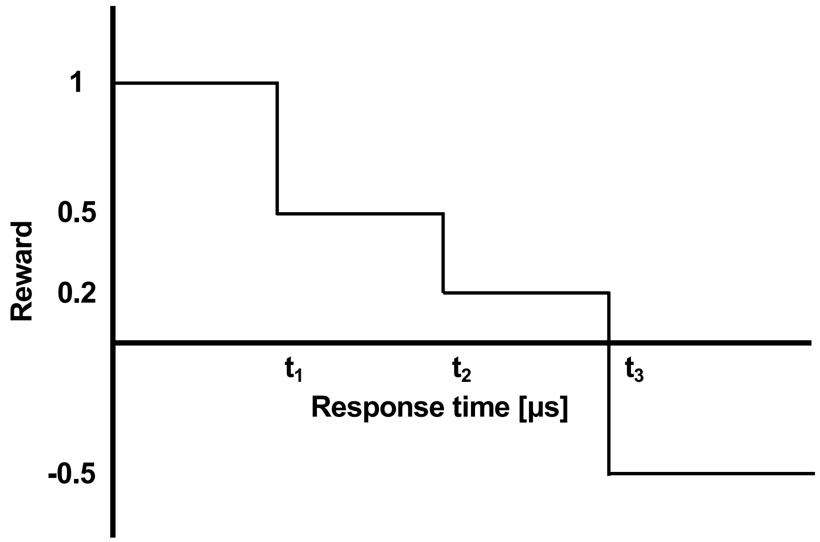
	Action 0	Action 1	Action 2
State 1			
State 2			
State 3			
⋮	⋮	⋮	⋮

Previous inter-request interval [μs]	Previous action	Current inter-request interval [μs]
< 100	< max action/2	< 100
		< 500
		...
		> 100000
	> max action/2	...
> 100	...	...
		...
	> max action/2	> 100000

Reward

[Kang, 2017]

- Assign the larger reward to the smaller latency
- Three thresholds
  - Adjust based on the distribution of response time
  - t_1, t_2, and t_3: 70^{th}, 90^{th}, and 99^{th} percentiles of the response time, respectively



RL-Assisted GC Scheduling (RLGC)

[Kang, 2017]

- Actions
  - # of partial GC operations
  - 0 ~ 2 page copies
- GC trigger threshold
  - # free blocks <= 10
- Exploitation and Exploration balance
  - ε-greedy technique

    ε= 0.8 for the first 1000 GC operations to perform aggressive exploration

    ε= 0.01 for the rest of period to exploit the learned policy

Q table

	Action 0	Action 1	Action 2
State 1			
State 2			
State 3			
⋮	⋮	⋮	⋮

Q Learning

- Q value, Q(s,a)
  - Expected cumulative reward from taking action a in state s

	Action 0	Action 1	Action 2
State 1			
State 2			
State 3			
⋮	⋮	⋮	⋮

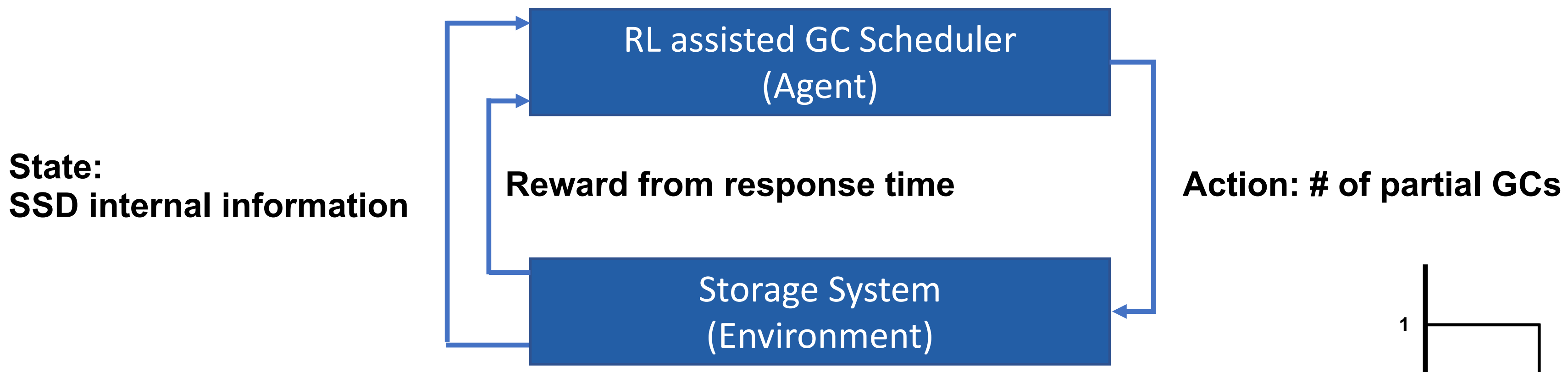
$$\boxed{Q(s_t, a_i)} = \max_{\pi} \mathbb{E}[r_t + \underbrace{\gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots}_{\textit{max } Q(s', a')} | \pi],$$

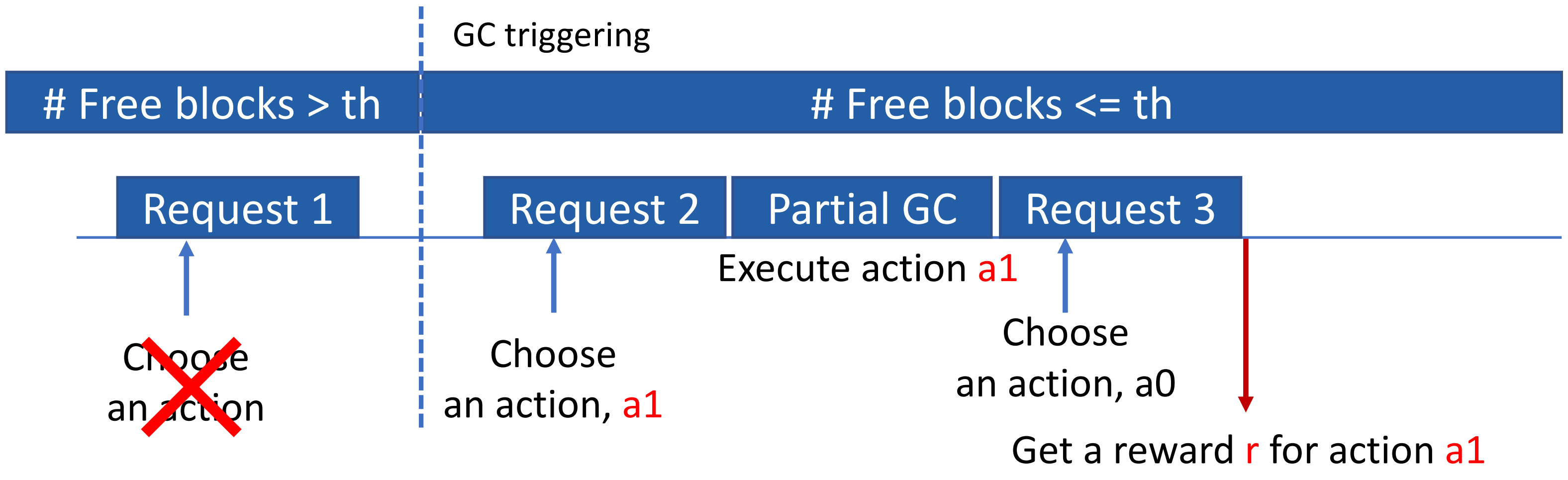
- Bellman equation
  - Q value will ultimately approach this expectation when an optimal policy is used
    - s' and a' are the next state and the action in the next state
  - r + γ max Q(s',a') works as the target Q value to learn in Q learning

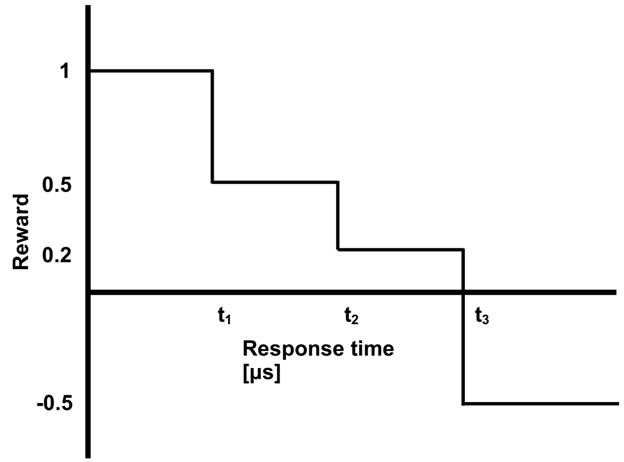
$$Q^*(s, a) = \mathbb{E}_{s' \sim \mathcal{E}} \left[r + \gamma \max_{a'} Q^*(s', a') | s, a \right]$$

Latency → Reward

[Kang, 2017]







Q Learning

- How to obtain max Q(s’,a’)?

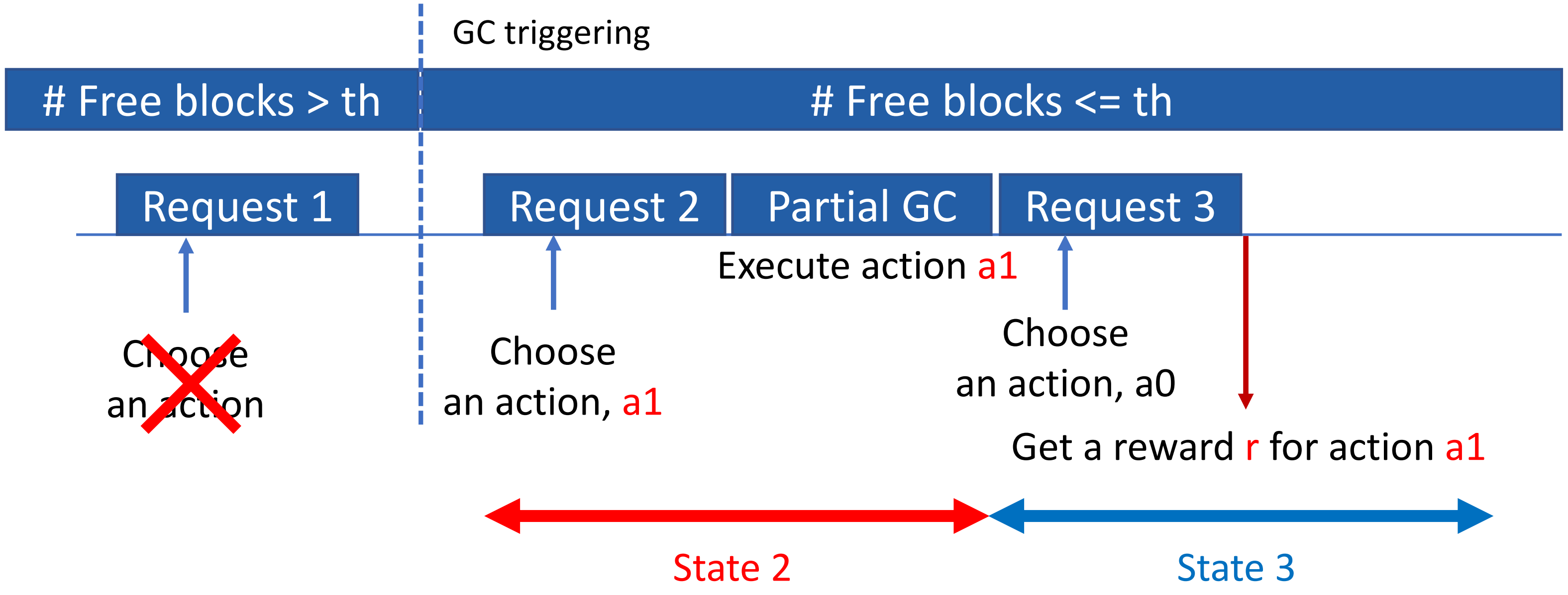
$$Q^*(s,a) = \mathbb{E}_{s' \sim \mathcal{E}}\left[r + \gamma \max_{a'} Q^*(s',a') | s, a\right]$$

- Boostrapping
  - The current largest Q value of the next state is used

	Action 0	Action 1	Action 2
State 1			
State 2			
State 3			
⋮	⋮	⋮	⋮

$$\boxed{Q(s_t, a_i)} = \max_{\pi} \mathbb{E}[r_t + \underbrace{\gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots}_{max\ Q(s_{t+1},\ a_i)} | \pi],$$

Find the largest Q value for state s3 in the Q-table (called *bootstrapping*)



Q Learning

- How to update Q value?

$$Q^*(s,a) = \mathbb{E}_{s'\sim\mathcal{E}}\left[r + \gamma \max_{a'} Q^*(s',a')|s,a\right]$$

	Action 0	Action 1	Action 2
State 1			
State 2			
State 3			
⋮	⋮	⋮	⋮

- Incremental update of Q value
  - Called time difference (TD) learning
  - Try to reduce the gap between the current target Q value, $r + \gamma$ max Q(s',a') and the current Q value, Q(s,a)

$$Q(s,a) = \boxed{Q(s,a)} + \alpha[\overbrace{r + \gamma \boxed{\max_{a'} Q(s',a')}}^{\textbf{Current target Q value}} - \overbrace{\boxed{Q(s,a)}}^{\textbf{Current Q value}}]$$

Experimental Setup

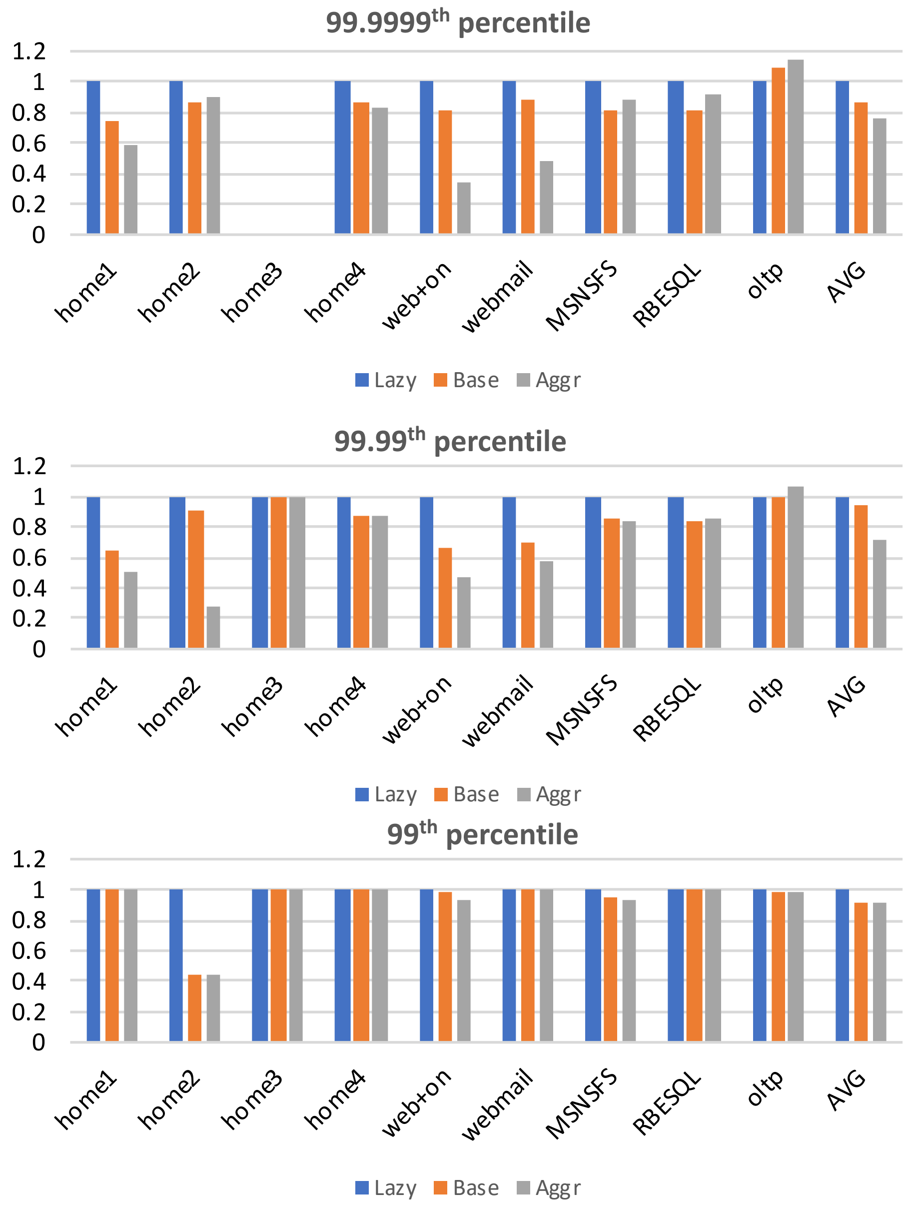
[Kang, 2017]

- Implemented our proposed method, LazyRTGC and page-level GC on FlashSim
- Results are normalized to the state-of-the-art method, LazyRTGC
- Workload: 8 real world workloads
  (6 from FIU, 2 from MS and 1 from filebench)

Experiments

[Kang, 2017]

- Long tail latency comparison (normalized to LazyRTGC)
- Average latency (99.9999^{th}, 99.99^{th}, 99^{th})
  - Ours (Base): 0.86×, 0.94×, 0.92×
  - **Ours (Aggr): 0.76×, 0.71×, 0.92×**



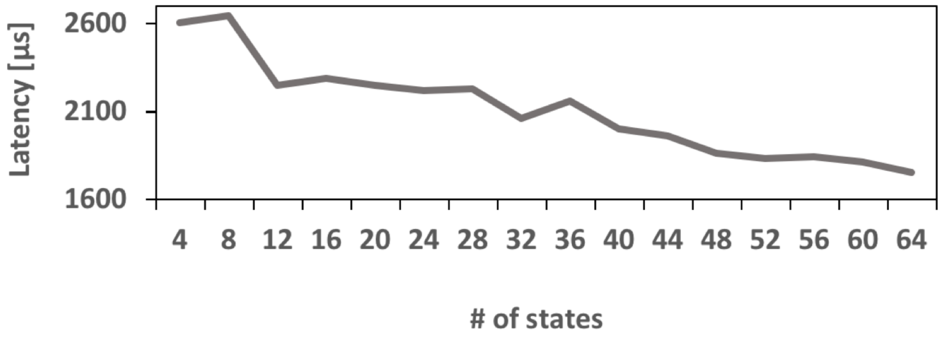
[Kang, 2018]

What if More States Are Used?

State information and # of bins

Information used for state	# of bins
Current (t) inter-request interval	32
Previous (t-1) inter-request interval	32
Previous (t-1) action (# of performed partial gc)	3
Previous (t-2) action (# of performed partial gc)	3
Previous (t-3) action (# of performed partial gc)	3
Previous (t-4) action (# of performed partial gc)	3
Previous (t-5) action (# of performed partial gc)	3
# of free blocks	12
Previous (t-1) request size	5
Previous (t-2) request size	5
Previous (t-1) valid page copy (performed or not)	2
Previous (t-2) valid page copy (performed or not)	2
Previous (t-1) block erase (performed or not)	2
Previous (t-2) block erase (performed or not)	2
Current (t) requested operation	2
Previous (t-1) requested operation	2
Previous (t-2) requested operation	2

Latency variation according to the number of states in home1



- The more state information (the more states), the better latency
- However, we need an extremely large Q table having 88 x 10^8 states

Locality in Visiting States

- There are a few frequently visited states
- They change across periods

Top rank states and access counts in home2

Period 1		Period 2		Period 3		Period 4	
State #	**Count**	**State #**	**Count**	**State #**	**Count**	**State #**	**Count**
199813153	3152	199803970	5779	424000545	1246	274455586	123
349109313	963	349133890	2524	199822369	328	200025122	88
274455585	853	424000545	132	423757951	321	199803969	88
423760929	849	199804036	55	423831585	279	199914530	86
199887871	627	423907361	49	274621473	181	199803938	85
199969825	593	423969825	48	423757857	127	199803937	79
199803937	543	199804063	35	199960609	122	274454562	77
199886881	464	199804003	24	274454563	111	199969857	77
274482209	323	199804899	22	199831585	98	274474049	73
349189153	300	199803999	22	274455585	84	274537506	73

Proposed Idea: Q-table Cache (QTC)

- Instead of having a large Q table
- Manage a small cache to keep recently visited states
  - 100 entries per action (3 tables for 3 actions, i.e., 0/1/2-page copy)
  - LRU (Least Recently Used) policy in replacement
- In case of inserting a new entry to Q-table cache
  - Q-value is initialized to 0 and 0-page copy is adopted

Conventional Q-table

	Action 0	Action 1	Action 2
State 1			
State 2			
State 3			
⋮	⋮	⋮	⋮

Required memory size: 98GB

Proposed Q-table cache

100 entries

Action 0		Action 1		Action 2	
State	Q-value	State	Q-value	State	Q-value
⋮	⋮	⋮	⋮	⋮	⋮

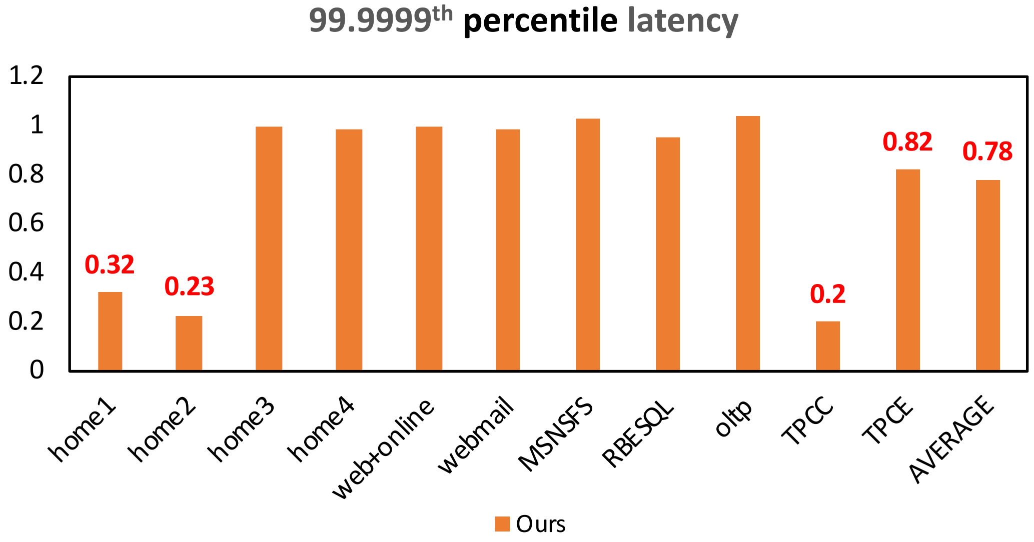
Required memory size: 2.34KB

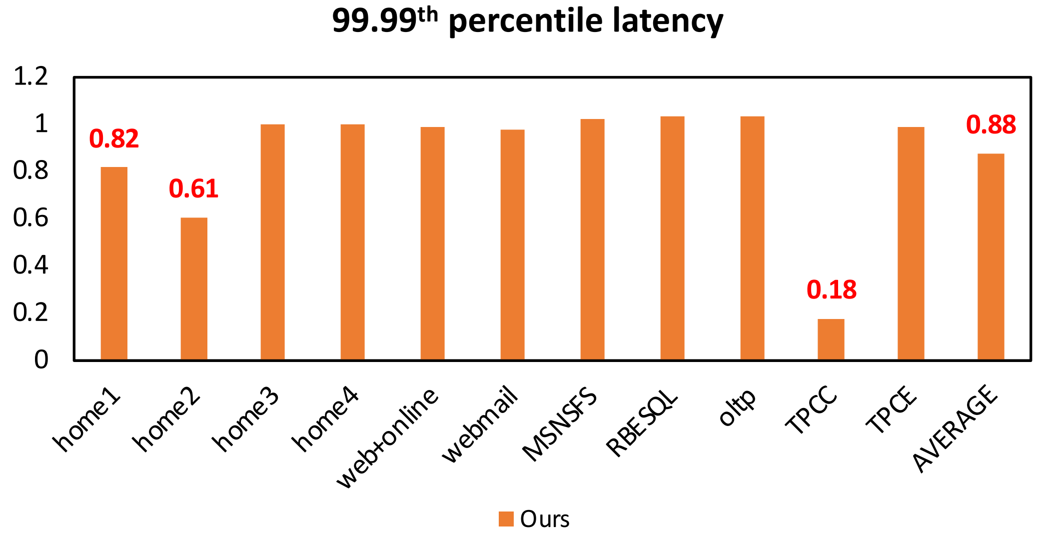
Experimental Setup

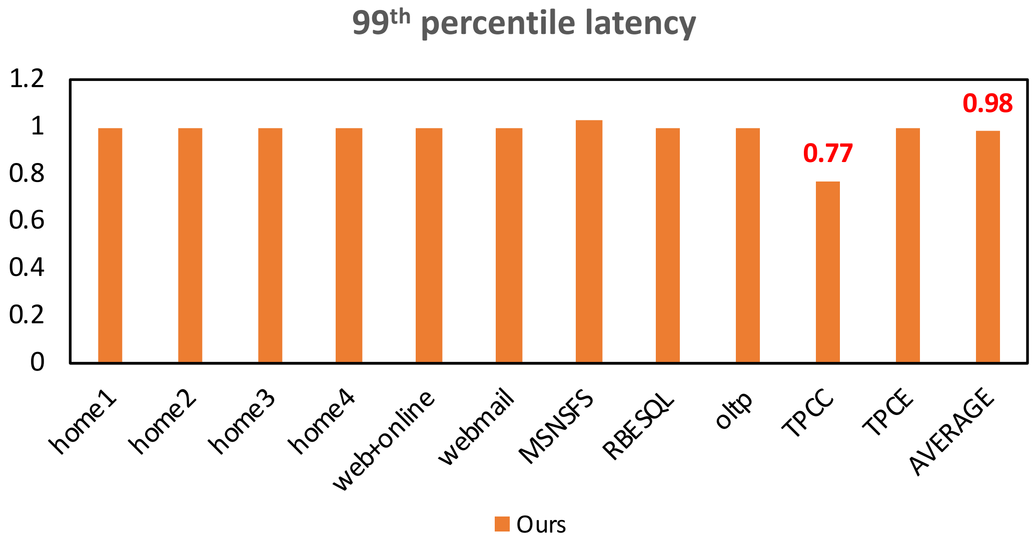
- FlashSim simulator
- 3D 128Gb, 3D 512Gb flash memories
- 11 workloads
  (home1, home2, home3, home4, webmail+online, webmail, MSNSFS, RBESQL, oltp, TPCC, TPCE)
- Compared with RLGC

[Kang, 2018]

Latency Comparison (3D NAND 512Gb)







- Average latency (99.9999th, 99.99th, 99th)
  - Base: 1×, 1×, 1×
  - Ours: **0.78×**, **0.88×**, **0.98×**

Normalized to the baseline RLGC

Summary

- Problem
  - Long tail latency reduction in SSD
- Q learning based solutions
  - Simple Q-table solution: 22~25% reduction
  - Q-table cache to exploit much more states: 11~25% further reduction
- Future work: Applying reinforcement learning to buffer management in SSD
  - Write back from buffer to Flash memory
  - Prefetch from Flash memory to buffer

Reference

- [Kang, 2017] W. Kang, D. Shin, S. Yoo, "Reinforcement Learning-Assisted Garbage Collection to Mitigate Long Tail Latency Problem," ACM Transactions on Embedded Computing Systems (TECS), Oct. 2017.
- [Kang, 2018] W. Kang, S. Yoo, "Dynamic management of key states for reinforcement learning-assisted garbage collection to reduce long tail latency in SSD," Proc. Design Automation Conference (DAC), June 2018.