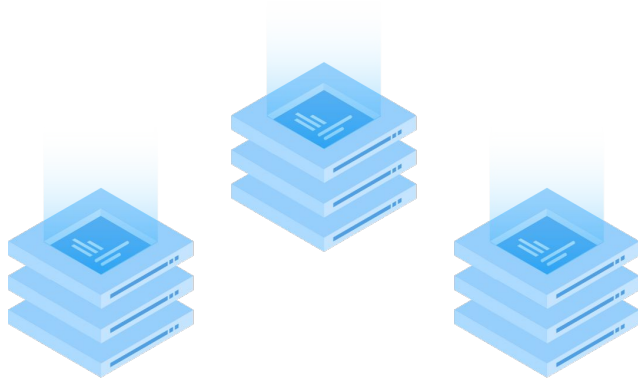


ARC: A SELF-TUNING, LOWOVERHEAD REPLACEMENT CACHE

By Zejun LI



Background



The real-life problem

- May contain long sequential I/Os or moving hot spots
- The frequency and scale of temporal locality may change with time
- The access pattern may fluctuate between
 - stable and repeating patterns
 - transient clustered references



Recency



LRU

- Always replaces the least recently used page
- Easy to implement and low overhead
- Only capture “recency”, no “frequency”



Frequency



LFU

- Assumes that each page reference is drawn in an independent fashion from a fixed distribution over the set of all pages
- Accumulates stale pages with high frequency counts that may no longer be useful



LRU-2 & 2Q

- For each page remember the last two times when it was requested, and to replace the page with the least recent penultimate reference
- Has two practical limitations
 - Needs to maintain a priority queue, requires logarithmic complexity
 - Contains at one crucial tunable parameter
- The algorithm 2Q mitigate the first limitation, but introduce a new parameter
 - The new parameter need a priori estimate of the miss rate



Hit ratio of LRU-2

c	CIP/ <i>c</i>								
	0.01	0.05	0.1	0.25	0.5	0.75	0.9	0.95	0.99
1024	0.87	1.01	1.41	3.03	3.77	4.00	4.07	4.07	4.07
2048	1.56	2.08	3.33	4.32	4.62	4.77	4.80	4.83	4.83
4096	2.94	4.45	5.16	5.64	5.81	5.79	5.70	5.65	5.61
8192	5.36	7.02	7.39	7.54	7.36	6.88	6.47	6.36	6.23
16384	9.53	10.55	10.67	10.47	9.47	8.38	7.60	7.28	7.15
32768	15.95	16.36	16.23	15.32	13.46	11.45	9.92	9.50	9.03
65536	25.79	25.66	25.12	23.64	21.33	18.26	15.82	14.99	14.58
131072	39.58	38.88	38.31	37.46	35.15	32.21	30.39	29.79	29.36
262144	53.43	53.04	52.99	52.09	51.73	49.42	48.73	49.20	49.11
524288	63.15	63.14	62.94	62.98	62.00	60.75	60.40	60.57	60.82

Recency And Frequency



LRFU

$$C(x) = \begin{cases} 1 + 2^{-\lambda}C(x) & \text{if } x \text{ is referenced at time } t; \\ 2^{-\lambda}C(x) & \text{otherwise,} \end{cases}$$

- $\lambda \rightarrow 0$ is LFU
- $\lambda \rightarrow 1$ is LRU



Adaptive Replacement Cache



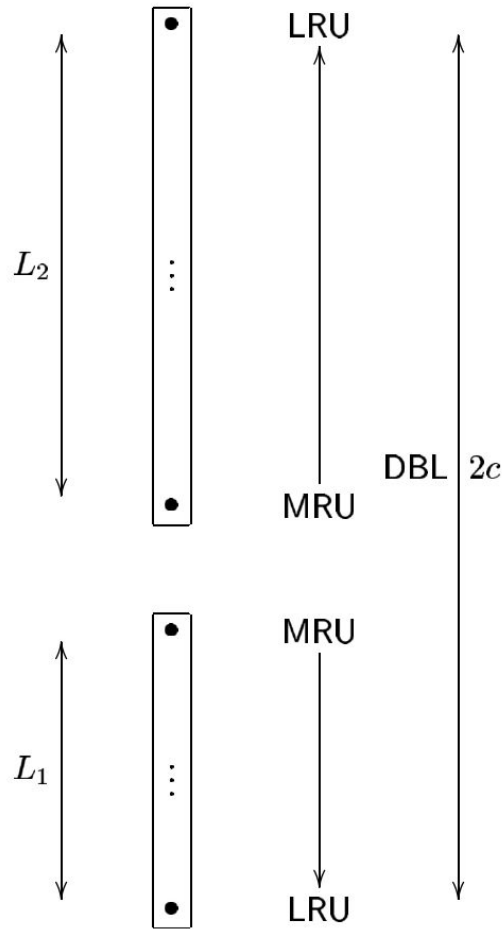
Highlights

- No tunable parameters
- Considering both recency and frequency
- Dynamically balancing between recency and frequency
- Has constant complexity per request
- Easy to implement
- ARC is scan-resistant
 - For a SPC1 like benchmark at 4GB cache, LRU get 9.19% and ARC achieves 20%



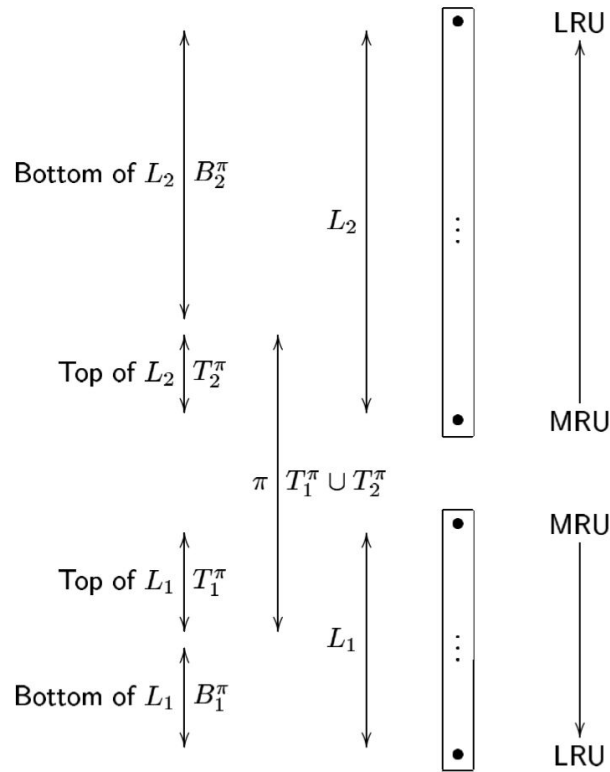
Starting with the DBL($2c$)

- L1 contains pages **accessed only once**
- $0 \leq |L_1| + |L_2| \leq 2c$, $0 \leq |L_1| \leq c$, $0 \leq |L_2| \leq 2c$.
- If x is in L1 or L2
 - Cache hit, make x the MRU page in L2
- If x not in DBL
 - If L1 has exactly c pages (**L1 is full**)
 - **Delete the LRU page in L1**
 - Make x the MRU page in L1
 - If L1 not full
 - If $|L_1| + |L_2| = 2c$ (**DBL cache full**), **delete LRU in L2** to make room for new page
 - Insert x as MRU page in L1



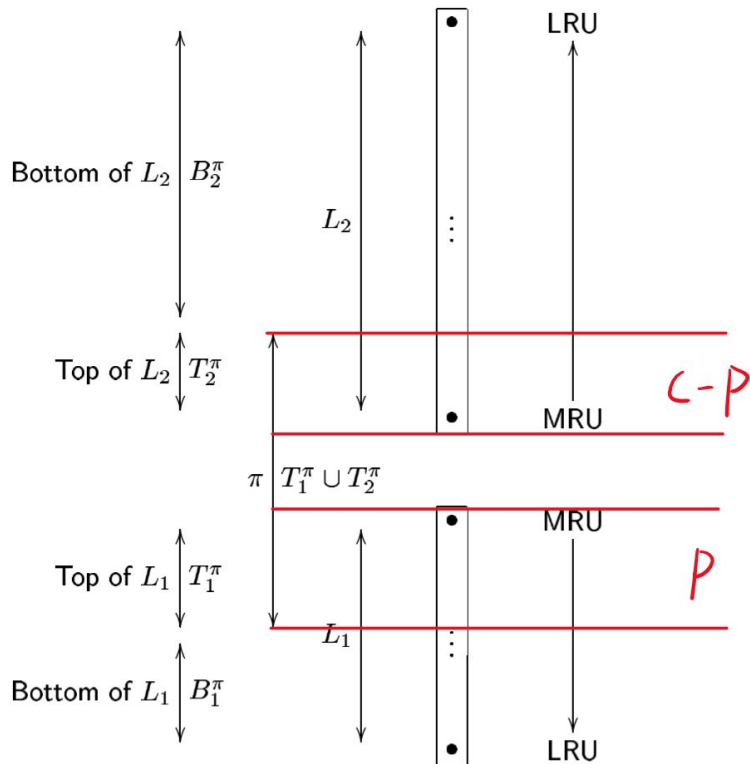
Let's discard the extra c pages

1. Divide list to two parts T and B
 - T hold the cached pages
 - B is **ghost list**, It just keep the metadata of recently evicted pages
2. If $|L_1| + |L_2| < c$, then both B_1 and B_2 is empty
3. If $|L_1| + |L_2| \geq c$, then $|T_1| + |T_2| = c$
4. The LRU page in T is **more recent** than MRU page in B
 - So when T_1/T_2 is full, we will remove the LRU page in T_1/T_2 and move it to the MRU position in B_1/B_2
 - This rule **keep the order of evict history**
5. ARC will change the size ratio between T_1 and T_2



Cache miss can hit a ghost

1. x is in T_1 or T_2 , **hit in ARC and BDL**
 - Move x to MRU position in T_2
2. x is in B_1 , miss in ARC and **hit in BDL**
 - Make p larger
 - Make room if ARC full
 - Move page to MRU position in T_2
3. x is in B_2 , miss in ARC and **hit in BDL**
 - Make p smaller
 - Make room if ARC full
 - Move page to MRU position in T_2



If miss in ghost list

1. If $|T1| + |T2| = c$ (**L1 is full**)
 - Delete LRU in **B1**
 - Make room if ARC full
 - Fetch x and move it to the MRU position in *T1*
2. If $|T1| + |T2| < c$
 - **if BDL is full**, delete LRU in **B2**
 - Make room if ARC full
 - Fetch x and move it to the MRU position in *T1*



How to discard page in ARC

- If $|T1| > p$, discard the LRU page in $T1$ (move it to $B1$)
- If $|T1| < p$, discard the LRU page in $T2$ (move it to $B2$)
- If $|T1| = p$, no hard limits, origin paper choose
 - The new page hit $B2$, reclaim $L1$
 - Otherwise, reclaim $L2$



The adaptation of p

If hit $B1$

Update $p = \min \{p + \delta_1, c\}$ where $\delta_1 = \begin{cases} 1 & \text{if } |B_1| \geq |B_2| \\ |B_2|/|B_1| & \text{otherwise.} \end{cases}$

If hit $B2$

Update $p = \max \{p - \delta_2, 0\}$ where $\delta_2 = \begin{cases} 1 & \text{if } |B_2| \geq |B_1| \\ |B_1|/|B_2| & \text{otherwise.} \end{cases}$



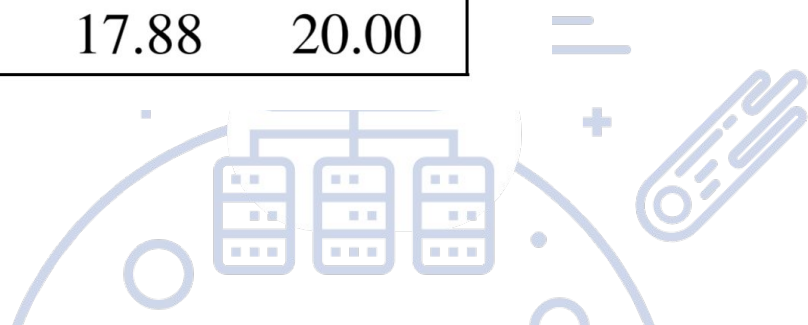
Evaluation



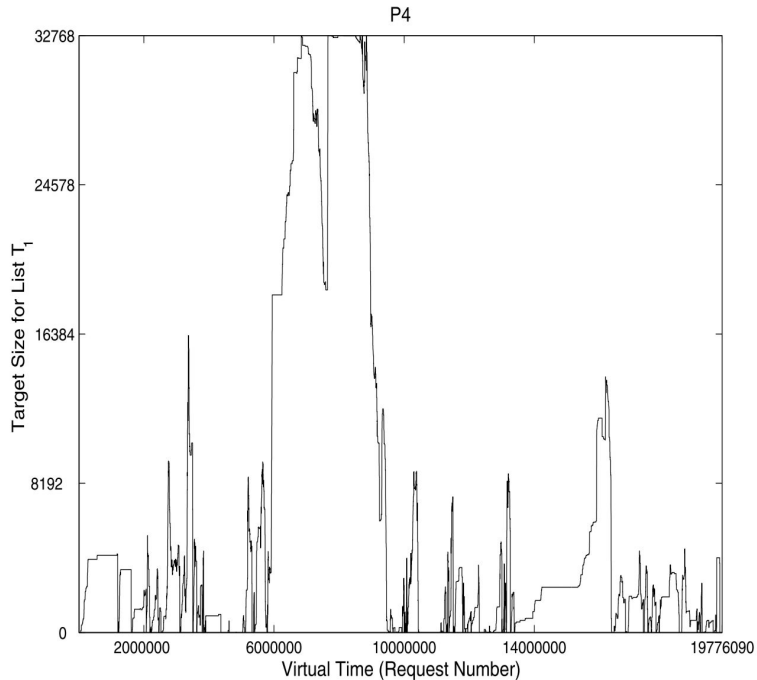
Long scan with random access

SPC1 like

c	LRU	MQ	2Q	ARC
	ONLINE			
65536	0.37	0.37	0.66	0.82
131072	0.78	0.77	1.31	1.62
262144	1.63	1.65	2.59	3.23
524288	3.66	3.72	6.34	7.56
1048576	9.19	14.96	17.88	20.00



Change in access pattern



c	P4		
	LRU	MQ	ARC
	ONLINE		
1024	2.68	2.67	2.69
2048	2.97	2.96	2.98
4096	3.32	3.31	3.50
8192	3.65	3.65	4.17
16384	4.07	4.08	5.77
32768	5.24	5.21	11.24
65536	10.76	12.24	18.53
131072	21.43	24.54	27.42
262144	37.28	38.97	40.18
524288	48.19	49.65	53.37

Thank You !

