



ARC: a self-tuning, low overhead replacement cache

Ohad Rodeh



Introduction

- This lecture is based on **ARC: A Self-Tuning, Low Overhead Replacement Cache**. Nimrod Megiddo, Dharmendra Modha. 2nd USENIX Conference on File and Storage Technologies (FAST 03), San Francisco, CA.

Main idea, combine LRU and LFU

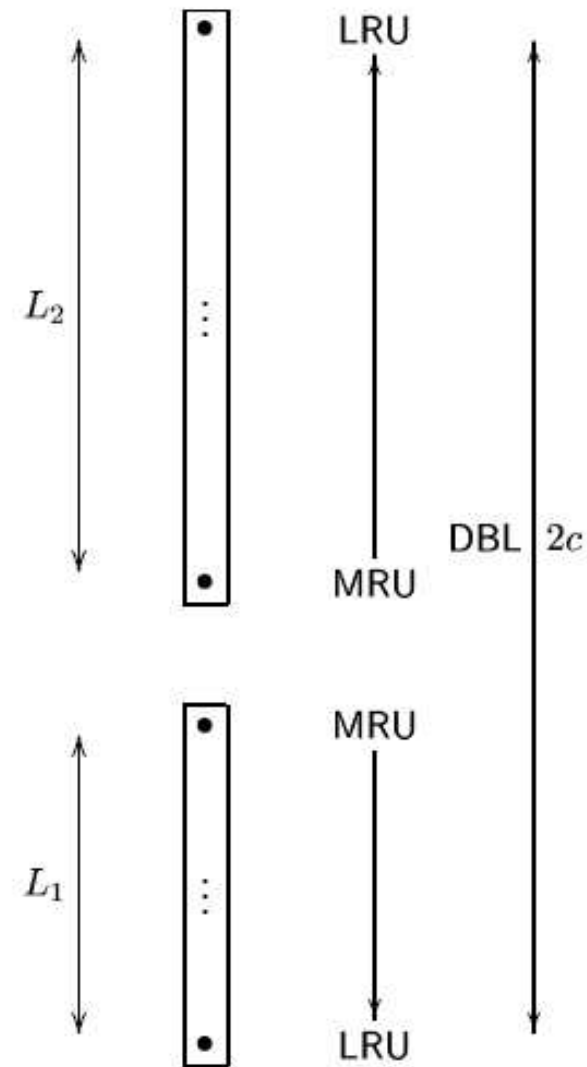
- Keep two lists, L_1 and L_2
- If the size of the cache is c then $|L_1| + |L_2| = 2c$
- The lists are larger than the amount of pages in cache
- There are ghost pages: book-keeping done for pages that are not actually in cache
- L_1 keeps track of recently accessed pages: recency
- L_2 keeps track of pages there were recently accessed twice (or more): frequency



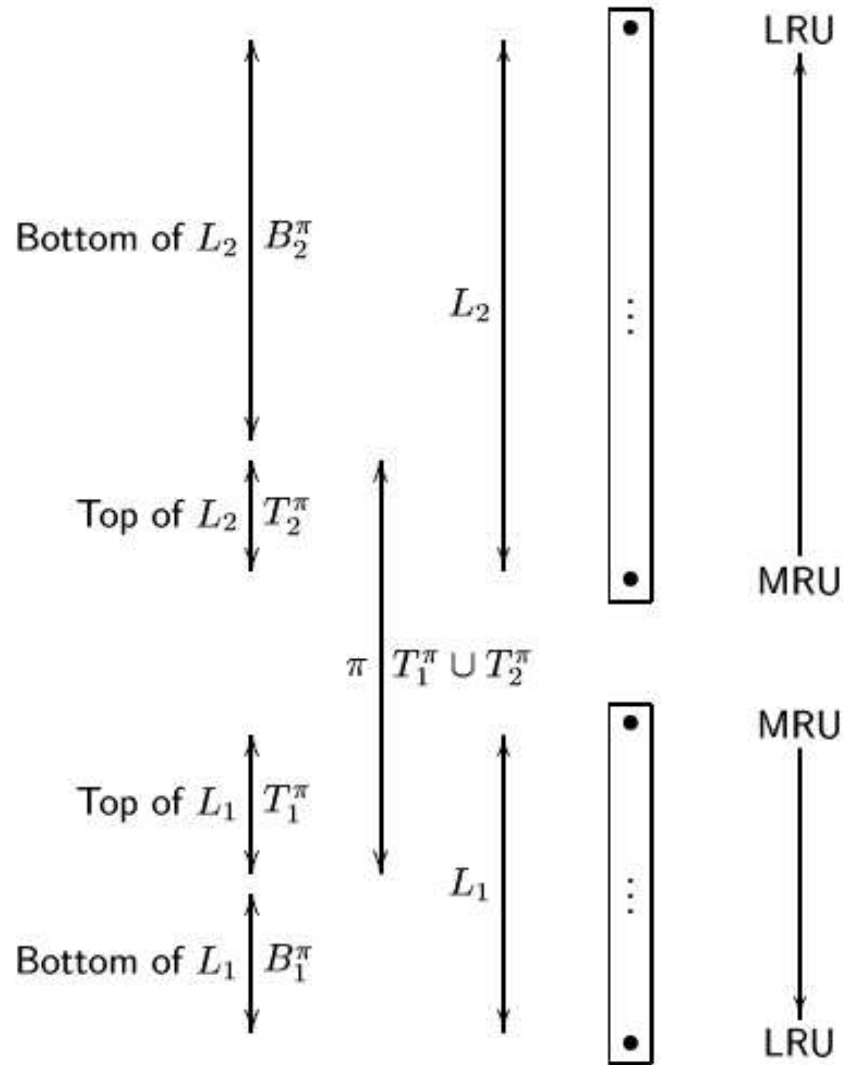
Main idea II

- Out of the total $2c$ pages, only c can be kept in cache
- The algorithm adaptively chooses how much to use from L_1 and how much from L_2

Two lists, L_1 and L_2



Lists are split into Top and Bottom



Fixed Replacement Cache

- The Fixed Replacement Cache policy $FRC(p, c)$: keep p pages in L_1 and $c - p$ pages in L_2
- Replacement rules, when lookup for page P:

If hit in B1, B2, T1, T2

- Move P to MRU position in T2

If miss in B1, B2, T1, T2

- Move P to MRU position in T1
- Get rid of the LRU page in either T1 or T2

Adaptation

- On a hit in B_1

If $|B_1| \geq |B_2|$, increment p by 1
Otherwise, increment by $|B_2|/|B_1|$

- Increments to p are bounded by the cap c

- On a hit in B_2

If $|B_2| \geq |B_1|$, decrement p by 1
Otherwise, decrement by $|B_1|/|B_2|$

- Decrements to p are bounded by zero

- The main idea is to adapt the parameter p according to what has worked better so far



Scan resistance

- Some workloads have a lot of one-time accesses to pages
- For example, b-tree leaves under a 100% random lookup workload
- ARC is resistant to such workloads
- It adapts and gives preference to the L_2 list

Workloads

- Several workloads were used:
 1. P1- P14 were traces on Windows NT workstations
 2. SPC: Standard industry block-storage workloads
 3. OLTP: Online transaction processing
- Writes were discarded, only reads were considered

Trace Name	Number of Requests	Unique Pages
OLTP	914145	186880
P1	32055473	2311485
P2	12729495	913347
P3	3912296	762543
P4	19776090	5146832
P5	22937097	3403835
P6	12672123	773770
P7	14521148	1619941
P8	42243785	977545
P9	10533489	1369543
P10	33400528	5679543
P11	141528425	4579339
P12	13208930	3153310
P13	15629738	2497353
P14	114990968	13814927
ConCat	490139585	47003313
Merge(P)	490139585	47003313
DS1	43704979	10516352
SPC1 like	41351279	6050363
S1	3995316	1309698
S2	17253074	1693344
S3	16407702	1689882
Merge (S)	37656092	4692924

CPU overhead

- Computational overhead:
 1. Run P9, see how long it takes (in seconds)
 2. ARC is very close to LRU
 3. All other algorithms are more expensive

c	LRU	ARC	2Q	LRU-2	LRFU		
					10^{-7}	λ 10^{-3}	.99
1024	17	14	17	33	554	408	28
2048	12	14	17	27	599	451	28
4096	12	15	17	27	649	494	29
8192	12	16	18	28	694	537	29
16384	13	16	19	30	734	418	30
32768	14	17	18	31	716	420	31
65536	14	16	18	32	648	424	34
131072	14	15	16	32	533	432	39
262144	13	13	14	30	427	435	42
524288	12	13	13	27	263	443	45



OLTP

- *OLTP*: Online transaction processing is an important workload
- Database receives operations (transactions) and performs them online
- 70% read, 30% write

OLTP performance comparison

- Results:

1. c is the cache size
2. The numbers are hit rates in percentage

c	OLTP ONLINE						OLTP OFFLINE			
	LRU	ARC	FBR	LFU	LIRS	MQ	LRU-2	2Q	LRFU	MIN
1000	32.83	38.93	36.96	27.98	34.80	37.86	39.30	40.48	40.52	53.61
2000	42.47	46.08	43.98	35.21	42.51	44.10	45.82	46.53	46.11	60.40
5000	53.65	55.25	53.53	44.76	47.14	54.39	54.78	55.70	56.73	68.27
10000	60.70	61.87	62.32	52.15	60.35	61.08	62.42	62.58	63.54	73.02
15000	64.63	65.40	65.66	56.22	63.99	64.81	65.22	65.82	67.06	75.13

- ARC performs better than all other online algorithms

Comparing LRU, ARC, and FRC

- ARC outperforms LRU for all workloads
- It is interesting to compare ARC to an offline version that compute the optimal p
- Mostly ARC is close to FRC
- Sometimes, it is better, because it can change p over the life of the workload

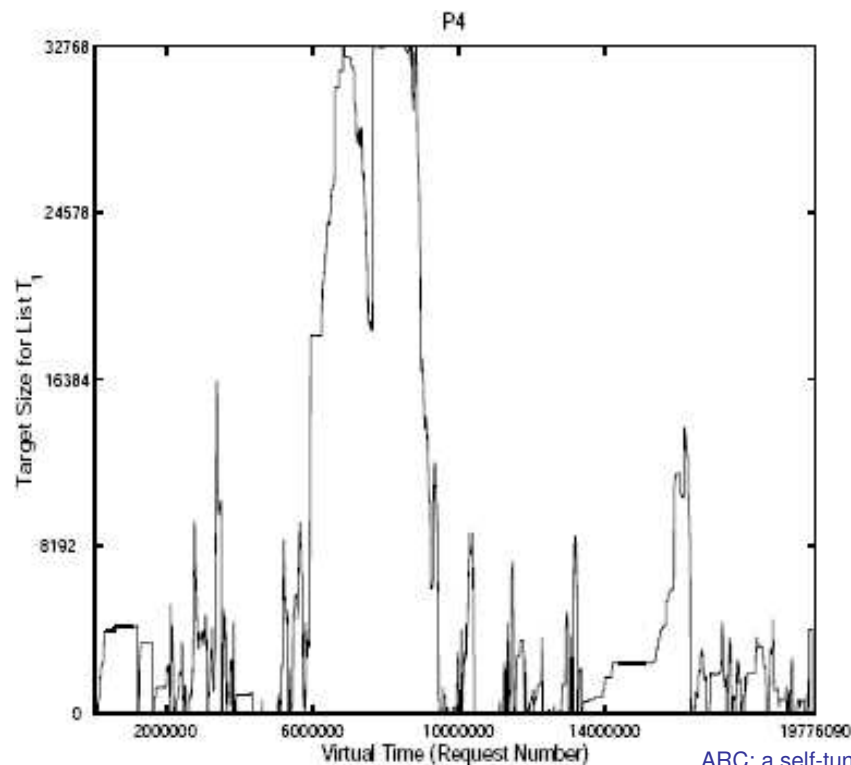
Comparison II

- Sometimes (P8) it is worse
- In P8 it is better to not continuously adapt
- Adaptation has a cost

Workload	c	space MB	LRU	ARC	FRC OFFLINE
P1	32768	16	16.55	28.26	29.39
P2	32768	16	18.47	27.38	27.61
P3	32768	16	3.57	17.12	17.60
P4	32768	16	5.24	11.24	9.11
P5	32768	16	6.73	14.27	14.29
P6	32768	16	4.24	23.84	22.62
P7	32768	16	3.45	13.77	14.01
P8	32768	16	17.18	27.51	28.92
P9	32768	16	8.28	19.73	20.28
P10	32768	16	2.48	9.46	9.63
P11	32768	16	20.92	26.48	26.57
P12	32768	16	8.93	15.94	15.97
P13	32768	16	7.83	16.60	16.81
P14	32768	16	15.73	20.52	20.55
ConCat	32768	16	14.38	21.67	21.63
Merge(P)	262144	128	38.05	39.91	39.40
DS1	2097152	1024	11.65	22.52	18.72
SPC1	1048576	4096	9.19	20.00	20.11
S1	524288	2048	23.71	33.43	34.00
S2	524288	2048	25.91	40.68	40.57
S3	524288	2048	25.26	40.44	40.29
Merge(S)	1048576	4096	27.62	40.44	40.18

Adaptation of p

- The graph shows adaptation of p during the P4 workload
- The cache size is 32768 pages





Summary

- ARC is efficient computationally
- Adapts well to varying workloads
- Outperforms LRU and most other algorithms