After ARC : Consideration about Self-Tuning

Jaehwan Jeong
Department of Computer Science and Engineering
Seoul Nat'l Univ, Real-Time Ubiquitous Systems Lab
Seoul, Korea
jhjeong@rubis.snu.ac.kr

Tackhee Lee
Department of Computer Science and Engineering
Seoul Nat'l Univ, 3D Modeling and Processing Lab
Seoul, Korea
thlee@3map.snu.ac.kr

***Abstract*—This paper precisely speculates the probable enhancement of the adaptive replacement cache called ARC. More specifically, this paper introduces 3 tunable parameters which was fixed in previous adaptive replacment cache. We logically show the reasonable description that each parameter has potential increase factor for the hit ratio of disk cache. We collected several traces that are widely-used in research-area for effective evaluation in terms of practicality. Conclusionally, This paper aims at finding 1)effectiveness of ARC is still valid in current days 2)our speculation called "After-ARC" is reasonable. In order to find 1) and 2), We evaluate ARC and "After-ARC" with real-world benchmarks disk traces. Experimental results show that ARC is still valid in currently used benchmarks and our speculation "After ARC" shows some interesting findings through results compare to ARC.**

***Index Terms*—Storage management-Secondary Storage, Cache Replacement Policy, Disk Cache, Adaptive Replacement Cache**

I. Introduction

Caching is one of the oldest and most fundamental metaphor in modern computing. The main problem of cache management is to design a replacement policy that maximizes the hit rate measured over a very long trace subject to the important practical constraints of minimizing the computational and space overhead involved in implementing the policy. Generally, CPU cache, its replacement policy implementation is limited to be simple due to the hardware-bounded constraints. e.g. logic-circuit design, processor support, cache-associativity, etc. However, for disk cache, such constraints are normally relaxed compared to CPU cache. i.e. there exists a chance to improve the hit ratio by developing a good cache replacement policy. From 1960s to 2000s, many research paper has been published to solve this problem, But a well-known, folklore algorithm, called LRU(Least Recently Used) [1] showed a better performance by either hit ratio or tuning fashion. Even though there were some policies [2], [3] that show better hit ratio than LRU, they have critical weak points that they can outperform than LRU when some tunable paramter is fixed with some value. The existence of tuning-parameter seized them to overcome LRU policy due to the diversity of workload and computing environments since the value of paramter in each policy is highly depended on the workloads. e.g. database queries. To cope with this tuning-constraints, a novel cache replacement policy for disk cache called ARC(Adaptive Replacement Cache) [4] has been published in 2003. Unlike other policies, ARC has no tunable parameters since it adaptively changes the parameters itself during its execution by its learning rate. They showed significant improvement compared to LRU with respect to both hit ratio and tuning-constraints. Accordingly, ARC and its variants have been widely used in storage system nowadays. e.g. IBM storage controller DS6000, ZFS file system [5], PostgreSQL [6], VMware's vSAN [7].

Although the author of ARC [4] claims that it is "self-tunable", However, in our point of view, there still exists some tunable parameters. First, they use initial value of p as 0(which is the size of LRU portion in cache), but there's no guarantee that it is optimal for all workloads. Second, when they encounter phantom cache hit(i.e. actually miss, but exists in ghost cache), they use learning rate to increase/decrease size of LRU-LFU [1] portion respectively or vice versa. However, there is no proof that the value of learning rate is reasonable for all workloads. Lastly, they limit the maximum size of p as the cache size in ARC. However, this might loose full utilization of using ghost cache when they encountered a LRU-intensive workload. To investigate these conjectures, We first precisely speculate this observation. Second, we evaluate the hit-ratio of our "After-ARC" variation compared to original ARC on real-life benchmark disk I/O traces. Since various second-storages e.g. Flash-memory, SSD(Solid State Drive) have emerged today, it is not reasonable for only measuring the hit ratio of the disk cache. However, this consideration is beyond the scope of this paper. We only aim at comparing our proposed variation of ARC is valid compared to original ARC. In terms of that, our evaluation is still valid. Additionally, we verify that ARC is still effective on the workloads that we use these days because it has been 16 years passed since ARC has been published and there have been a lot of changes on I/O patterns in several workloads. Our experimental results say that proposed speculation can be used to improve the hit ratio of the cache and ARC still derives efficient hit ratio for the disk cache.

This paper is organized as follows. In Section 2, we survey related works. Then, Section 3 briefly explains the idea of adapative cache replacement policy which we want to precisely evaluate in terms of "self-tunable". In Section 4, we first explain our target traces i.e. real-life benchmarks that is widely used in research-area. And we also briefly describe about our simulation environment, and evaluates the experimental results. In Section 6, we discuss about the correctness of our consideration based on some interesting experimental results. Finally, Section 7 concludes the paper.

II. Related Works

For cache replacement policy, it is well-known that Belady's MIN [8] algorithm is offline optimal in terms of hit ratio. Belady's MIN replaces the page that has the greatest distance. The policy MIN provides an upper bound on the achievable hit ratio by any on-line policy.

A. Recency

The policy LRU [1] always replaces the least recently used page. LRU has several advantages, for example, it is simple to implement and responds well to changes in the underlying SDD(Stack Depth Distribution) model. However, it does not capture frequency. That is, in the long run, that each page is equally likely to be referenced and that therefore the model is useful for treating the clustering effect of locality but not the nonuniform page referencing. However, LRU still dominates among cache replacement policies because it is simple to implement and shows quite-good performance for all workloads.

B. Frequency

The Independent Reference Model(IRM) provides a workload characterization that captures the notion of frequency. Specifically, IRM assumes that each page reference is drawn in an independent fashion from a fixed distribution over the set of all pages in the auxiliary memory. The LFU [1] captures frequency but it has several drawbacks. It requires logarithmic implementation complexity in cache size, pays almost no attention to recent history, and does not adapt well to changing access patterns since it accumulates stale pages with high frequency counts that may no longer be useful.

C. Combining Recency and Frequency

Several algorithms has been published to capture both recency and frequency. For example, FBR(Frequency-Based Replacement) [9] policy maintains a LRU list, but divides it into three sections : new, middle, and old. The key idea known as factoring out locality was that if the hit page was in the new section then the reference count is not incremented. On a cache miss, the page in the old section with the smallest reference count is replaced. Drawbacks of FBR is that to prevent cache poluution due to stale pages with high reference count but no recent usage the algorithm must periodically resize all the reference counts. The algorithm also has several tunable parameters, namely, the sizes of all three sections, and some other parameteres C_{max} and A_{max} that control periodic resizing. Another example, LRFU(Least Recently/Frequently Used) policy [2] subsumes LRU and LFU. Main difference between LRFU and FBR is that LRFU frequently weights each page's age more than FBR, that is, LRFU weights each page's age on every page access, but FBR periodically resizes the page's age. Also, LRFU has tunable parameter λ and

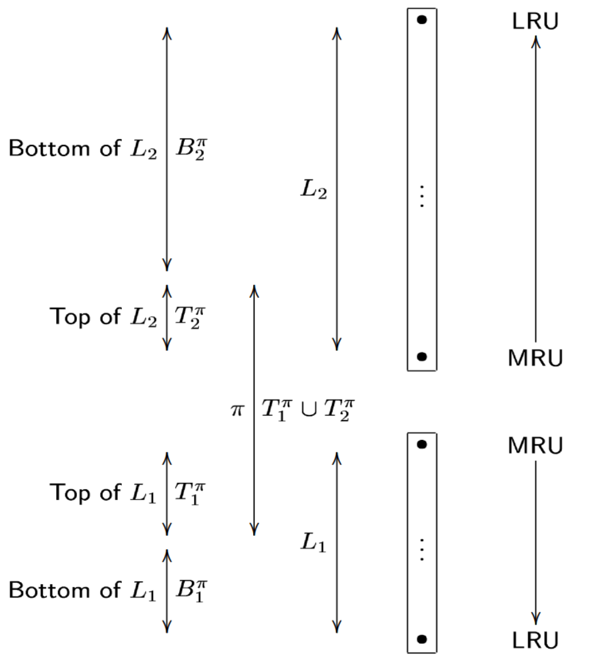
Bottom of L_2 B_2^π
Top of L_2 T_2^π
π $T_1^\pi \cup T_2^\pi$
Top of L_1 T_1^π
Bottom of L_1 B_1^π
L_2
L_1
LRU
MRU
MRU
LRU

Fig. 1: General structure of generic cache replacement policy

the performance of LRFU highly depends on λ. Now, we introduce ARC [4] in Section III. More algorithm exists such as LARC [10], SARC [11], CFLRU [12], AIP [13], RRIP [14], and etc. However, it is beyond the scope of this paper. If interested, refer these research papers.

III. Brief Review of ARC

We now show brief explanation about ARC [4]. Main difference between ARC and the other policies is that ARC adaptively changes its paramter. More specifically, the paramter of ARC is defined as the size of LRU/LFU portion within cache. We will show how it adaptively changes its size during runtime through Section A to C.

A. LRU and LFU

Many researchers attempted to make an algorithm that captures both recency and frequency. Since almost the whole workloads has both locality and frequency. So in order to the derive high hit ratio of the cache, For researchers, it was a fate to capture those properties together. However, all of them had critical drawbacks that they have to assign some value to their parameters in order to derive higher hit ratio, and it highly depends on the access pattern of certain workloads. Before describing ARC, We show a generic cache replacement policy which was used in ARC. Figure 1 shows the general structure of generic cache replacement policy.
This policy ensures that following invariants will always hold:

$$0 \leq |L_1| + |L_2| \leq 2c,\ 0 \leq |L_1| \leq c,\ 0 \leq |L_2| \leq 2c.$$

Also, in below conditions A.1-5 hold for generic cache replacement policy.

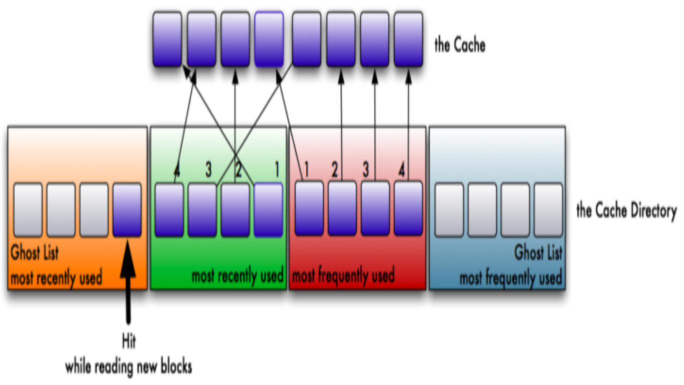

Fig. 2: A situation that phantom hit occurs

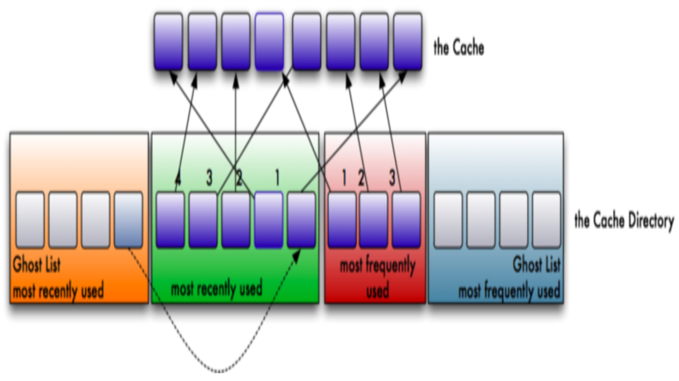

Fig. 3: Adaptive resizing from ARC

- A.1. The lists T_1^π and B_1^π are disjoint and, also, the lists T_2^π and B_2^π are disjoint, and $L_1 = T_1^\pi \cup B_1^\pi$ and $L_2 = T_2^\pi \cup B_2^\pi$.
- A.2 If $|L_1 \cup L_2| < c$, then both B_1^π and B_2^π are empty.
- A.3 If $|L_1 \cup L_2| \geq c$, then, together T_1^π and T_2^π contain exactly c pages.
- A.4 Either T_1^π(resp. T_2^π) or B_1^π (resp. B_2^π) is empty, or the LRU page in T_1^π (resp. T_2^π) is more recent than the mRU page in B_1^π (resp.B_2^π).
- A.5 For all traces and at each time, $T_1^\pi \cup T_2^\pi$ will contain exactly those pages that would be maintained in cache by the policy $\pi(c)$.

Key concept of this cache replacement policy is that L_1 holds pages that requested only once and L_2 holds pages that requested more than once. $|T_1 \cup T_2|$ is the actual cache and it keeps exactly the same size as cache size c. Compared to T_1 and T_2, B_1 and B_2 is used for the cache replacement management. i.e. ghost cache. In Figure 1, the ghost cache size is set to $2c$.

B. Ghost Cache

In ARC, they maintain a larger cache directory than that is needed to support the underlying cache. Such directories are known as a shadow cache or as a ghost cache. Previously, ghost caches have been employed in a number of cache replacement algorithms. ARC uses this ghost cache for adaptively determine the size of T_1 and T_2, B_1 and B_2 respectively.

C. Adaptive Replacement Cache

The main idea of ARC is quite simple. They maintain a ghost list i.e. B_1 and B_2. when they encountered the cache miss, they treat it as a actual-miss or phantom-hit (i.e. hit on B_1 or B_2). So if phantom-hit occurs, then ARC adaptively resizes the portion of T_1(i.e. phantom hit on B_1) or T_2(phantom hit on B_2). This is not actually accurate because some exceptions exist. Note that we describe brief the concept of ARC so if interested in more detailed algorithm, See [Fig 4 in [4]]. From Figure 2 and Figure 3, we can see the adaptive resizing of ARC, In Figure 2, We can see that actual cache size is 8. and ghost cache size is 16. Initial situation is same as following, $|T_1| = 4, |B_1| = 4, |T_2| = 4, |B_2| = 4$. Then phantom-hit occurs on B_1, So from that, ARC resizes the $|T_1|$ from 4 to 5, and $|T_2|$ from 4 to 3. $|B_1|$ and $|B_2|$ has same value in Figure 2 and 3. Note that this is a example for explaining ARC and it is not exactly same as ARC(i.e. a variant from ZFS-ARC) but the main idea(i.e. adaptive replacement) is same. Actually, various cases happen when a phantom-hit or actual miss occurs. See [Fig 4 in [4]].

IV. Consideration about "self-tuning"

In ARC, they actually assign p i.e. the maximum length of $|T_1|$ as 0 before starting, and they limits the maximum size of p with c i.e. actual cache size. Also, when phantom-hit occurs on B_1, they increase the size of T_1 from the following equation.

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

When phantom-hit occurs on B_2, they decrease the size of T_1 like Eq.(1), but differs with "+" to "-", and the inequality on Eq. (1) reverses from $\geq$ to $\leq$. Other things are exactly same. For simplicity, from now on, we will mark learning rate value as δ no matter whether phantom-hit occurs on B_1 or B_2. Eq. (1) intuitively involves the intention that if the following inequality $|B_2| \geq |B_1|$ holds, then there will be a lot of chances for encountering phantom-hit on B_1 again. So in order to avoid that situation again, ARC increase the size of p as $|B_2|/|B_1|$ not 1, and it is called learning rate.

A. Initial Value of p

In ARC, it starts algorithm with initial value p as 0, However, assigning p with value 0 does not guarantee that it is the best-choice. p indicates that ARC tries to re-size the length of T_1 to p as much as possible. Although ARC will adaptively change the parameter p during its runtime, initial value of p still can affect the hit ratio of the workload. So, we aim to evaluate the effectiveness of initial value p in terms of portion on hit ratio and the value itself by changing initial value of p

B. Limit value of p

In ARC, it limits the maximum size of p as c. This indicates that even though a certain workload is highly LRU-intensive, ARC cannot increase the size of LRU portion i.e. $|L_1|$ to $2c$. We guess that author's intention is 1)balancing the portion of LRU and LFU to 1:1. 2)It is meaningless because if phantom-hit occurs, then that page always goes to the MRU position of T_2, so it can be covered by T_2 even for the highly LRU-intensive workloads too. Although this is acceptable, There's no right answer on that. So, we aim to evaluate the effectiveness of the maximum value of p in terms of LRU-LFU portion by changing the maximum value of p.

C. Learning Rate

In ARC, if phantom-hit occurs, then they increase the size of p following as Eq. (1). Learning rate i.e. Eq. (1) indicates that ARC prefers to avoid the phantom-hit on the same hidden list i.e. B_1 or B_2. However, the value of learning rate, δ does not guarantee that it is the best-choice. So, we aim to evaluate the effectiveness of the learning rate by changing δ.

More specifically, First, we vary initial value of p as 0, $\frac{c}{2}$ and c. Second, we vary limit value of p as c, $\frac{1}{3}c$ $\frac{2}{3}c$, $\frac{4}{3}c$ and $\frac{5}{3}c$. Lastly, we vary the learning rate δ as ± 2, δ, $\log_2 \delta$ and 2^{δ}. Experimental results are shown in Section V.

V. EVALUATION

A. Collecting Benchmark Traces

Table I summarizes various traces that we used in this paper. These traces mainly include 3 traces. FIU Traces [15], MS Production server traces [16], MS Enterprise traces [16]. We use these traces because it is widely used for second storage research area, also it is available for free via SNIA/IOTTA(Storage Networking Industry Association's Input/Output Traces, Tools, and Analysis) [17].

Trace Name	Number of Request	Unique Pages
Exch	77304451	21724991
DAP-DS	11184349	5092722
MSN	11115258	7209398
RAD-AS	5529468	3323691
RAD-ES	39405995	20608323
HOMES	21163638	4760647
WEB-VM	14294158	549174
MAIL	14010588	1913909

TABLE I: Summary of disk traces

For all traces, the page size is 4KB. Also, all hit ratios reported in this paper are cold start. We will report hit ratios in percentages (%). We briefly explain the characteristics of each disk traces on the following subsections from 1) to 3). For all traces, We first append all logs of it and parsed disk I/O pattern from them.

1) Traces 1: MS Enterprise traces

- Exchange server (**Exch**)
  The Microsoft Exchange 2007 SP1 server is a mail server for 5000 corporate users. It is a 4-socket, dual-core system with 4 GB of memory. The storage consists of two 146 GB SAS drives in a RAID-1 configuration, six data arrays of fourteen 146 GB SAS drives, and two log arrays of eight 146 GB SAS drives configured as RAID-10. One trace covers a 5-hour peak load period on a weekday afternoon. Another trace covers a 24-hour weekday period. The traces are broken into 15-minute intervals.

2) Traces 2: MS Production Server traces

- Display Ads Platform data and payload servers(**DAP**)
  The purpose of the data server (**DS**) is to be a caching tier between the front-end server and the payload server (PS). A front-end server makes an advertisement request with a user id to the **DS**. The **DS** looks up the user id in the cache, appends any information available for that user to the request, and passes the request to the PS. The PS is responsible for ad selection. The traces from the DS and PS cover a 24-hour period and are broken into 30-minute intervals.
- MSN storage metadata and file servers (**MSN**)
  The CFS server stores metadata information and blobs correlating users to files stored on the back-end file server(**MSN**). The **MSN** provides the files requested by CFS. The servers are used by serval Live data services. The traces from the CRS and **MSN** cover a 6-hour period and are borken into 10-minute intervals.
- RADIUS authentication and back-end server (**RAD**)
  The RADIUS authentication server (AS) is responsible for worldwide corporate remote access and wireless authentication. It runs the IPSec NAP scenario. Data comes in via SQL replication on the back-end SQL server (**ES**). The traces from the AS and **ES** cover an 18-hour period and are broken into 1-hour intervals.

3) Traces 3: FIU traces

- Virtual machine running 2 web-servers (**WEB-VM**)
  The **WEB-VM** workload is collected from a virtualized system that hosts two CS department web-servers, one hosting the department's online course management system and the other hosting the department's web-based email access portal. the local virtual disks which were traced only hosted root partitions containing the OS distribution, while the http data for these web-servers reside on a network-attached storage.
- Mail-Server from FIU (**MAIL**)
  The **MAIL** workload serves user INBOXes for the entire Computer Science department at FIU(Florida International University).
- NFS server from FIU (**HOMES**)
  The **HOMES** workload is that of a NFS server that serves the home directories of FIU's small-sized research group;

	EXCH	**DAP-DS**	**MSN**	**RAD-AS**	**RAD-ES**	**HOMES**	**WEB-VM**	**MAIL**
$p = 0$	12.71	3.59	16.71	22.19	19.43	59.28	73.16	54.61
$p = \frac{c}{2}$	12.88	3.78	16.53	22.42	20.64	58.81	70.71	53.65
$p = c$	12.65	3.69	15.77	20.34	19.57	59.39	73.13	54.60

TABLE II: Hit ratio as varying initial value p

	EXCH	**DAP-DS**	**MSN**	**RAD-AS**	**RAD-ES**	**HOMES**	**WEB-VM**	**MAIL**
$p_{max} = c$	12.71	3.59	16.71	22.19	19.43	59.28	73.16	54.61
$p_{max} = \frac{1}{3}c$	12.86	2.84	16.26	19.59	12.62	56.18	67.59	54.26
$p_{max} = \frac{2}{3}c$	12.88	3.45	18.07	21.58	20.62	59.05	74.60	55.81
$p_{max} = \frac{4}{3}c$	12.85	3.56	9.84	19.09	20.91	49.87	79.47	49.87
$p_{max} = \frac{5}{3}c$	12.48	3.37	7.01	15.80	19.77	42.87	78.90	42.87

TABLE III: Hit ratio as varying limit value of p

	EXCH	**DAP-DS**	**MSN**	**RAD-AS**	**RAD-ES**	**HOMES**	**WEB-VM**	**MAIL**
ORIGIN	12.71	3.59	16.71	22.19	19.43	59.28	73.16	54.61
CONST	12.79	3.64	15.81	21.32	20.60	58.82	71.94	52.87
LOG	12.71	3.57	16.26	21.61	19.44	59.22	73.45	53.91
EXP	12.88	3.72	16.37	22.47	20.60	58.72	70.63	53.59

TABLE IV: Hit ratio as varying the value of learning rate δ

activities represent those of a typical researcher consisting of software development, testing, and experimentation, the use of graph-plotting software, and technical document preparation.

To simulate the cache behaviours, we fix the disk cache size as 256MB. From fixing the disk cache size and page size, and collecting disk I/O traces, we run total 72 simulation by varying some tunable parameters as we mentioned in Section IV.

B. Experimental Result

1) Experiment 1: Initial Value of p
As we mentioned in Section IV, we vary initial p value as 0, $\frac{c}{2}$ and c. Table II shows the result hit ratio of each trace. The first column of the table represents the initial value of p. In case of $p = c$ which means the initial value of p is c, only HOMES derives higher hit ratio compared with the other 2 cases. In case of $p = c2$, EXCH,DAP-DS,RAD-AS and RAD-ES derives higher hit ratio compared with the other 2 cases. Maximum difference value among $p = 0$ and $p = c$ appears as approx 3% in WEB-VM. Normally, the gap among all 3 cases comes with the degree of 10^{-1}. Conclusionally, there is no tendency with the initial value p and the amount of standard deviation in Experiment 1 is normally negligible.

2) Experiment 2: Limit Value of p
We now vary the limit value of p which means the maximum length of LRU portion as $\frac{1}{3}c$, $\frac{2}{3}c$, c, $\frac{4}{3}c$, $\frac{5}{3}c$. This intuitively indicates that the rate between LRU and LFU becomes 1:5, 1:2, 1:1, 2:1 and 5:1 since ARC uses ghost cache size as $2c$. Table III shows the result hit ratio of each trace. p_{max} means that we fix the maximum size of p as p_{max}. In case of $p_{max} = c$, DAP-DS, RAD-AS and HOMES derives the most highest hit ratio among them. In case of $p_{max} = \frac{2}{3}c$, EXCH, MSN and MAIL derives the most highest hit ratio. In case of $p_{max} = \frac{4}{3}c$, RAD-ES and WEB-VM derives the most highest hit ratio. We can expect that these EXCH, MSN and MAIL traces are more LFU-biased workloads since ARC can expand its maximum size for LRU portion maximum to $\frac{2}{3}c$. i.e. L_1. On the contrary to this, RAD-RES and WEB-VM traces are more LRU-biased workloads since ARC can expand to its maximum size for LRU portion maximum to $\frac{4}{3}c$. i.e. L_1. Lastly, DAP-DS,RAD-AS, and HOMES are balanced workload between LRU and LFU since they derive the most highest ratio in case, $p_{max} = c$. Note that we are dealing with the cache directory size i.e. ghost cache size not the actual size of the cache. Interesting finding is that for all of them, the worst-case hit ratio configurations always appear when p_{max} is either $\frac{1}{3}c$ or $\frac{4}{3}c$. This intuitively indicates that since we forcibly suppress the size of p into $\frac{1}{3}c$ or $\frac{4}{3}c$, it cannot freely enjoy the freedom of adaptive replacement. At first, we anticipated that more LRU/LFU-intesive workload can get more higher ratio on these configurations. However, this tendency tells us that real-world workloads are not purely 100% biased to LRU or LFU, in which, is the mixture type of LRU and LFU. Also, we can see original-ARC always performs above the average with all simulation cases. In summary, ARC performs well with respect to the adaptive replacement. Also, there exists some cases that performs better than ARC when we assign the maximum value p with the bounded range from c.

3) Experiment 3: Learning Rate
In this simulation experiment, We now vary the learning rate value with the ORIGIN, CONST, LOG and EXP. ORIGIN denotes that the value of learning rate is same as ARC [4]. CONST denotes that in contrary to Eq. (1), if we encountered the phantom-hit, then we always increase/decrease the size of T_1 as 2. LOG denotes that the value of learning rate is $\log_2 \delta$. Note that δ denotes the original learning rate value of ARC. Lastly, EXP denotes that the value of learning rate is 2^{δ}. Table IV shows the hit ratio of each traces. For MSN, HOMES, and MAIL, **ORIGIN** derives the most highest hit ratio among 4 configurations. For WEB-VM, **LOG** derives the most highest. For EXCH, DAP-DS, RAD-AS, and RAD-ES, **EXP** derives the most highest. **LOG** indicates that it smoothly does re-sizing compared to **ORIGIN**. Similarly, **EXP** indicates that it roughly does re-sizing compared to **ORIGIN**. We denote these trace sets as 1) Group 1 : EXCH, DAP-PS, RAD-AS, RAD-ES 2) Group 2: MSN, HOMES, MAIL, 3) Group 3 : WEB-VM. From Table IV, We can speculate the traces of Group 3 smoothly changes its I/O access pattern among three groups. Contrary to Group 3, we can speculate the traces of Group 1 roughly changes its I/O access pattern among three groups. Conclusionally, ARC(i.e. ORIGIN) performs well above the average for the all simulation cases. Also, in case of EXP, it performs better than ARC(i.e. ORIGIN) in 4 traces up to maximum 0.6%. For the other 4 cases, EXP performs worse than ARC, but the difference between EXP and ARC(i.e. ORIGIN) is up to maximum 0.4% except MAIL traces. We suggest that the exponential learning rate value is quite good-option based on the results in Table IV.

Finally, we present our contributions. 1)We confirm that the effectiveness of ARC is still valid through real-life benchmark traces which is widely used in research-area nowadays. In Table III, ARC performs better than $p_{max} = \frac{5}{3}c$ up to maximum 17% except RAD-ES and WEB-VM. $p_{max} = \frac{5}{3}c$ indicates that it behaves like LRU(not exactly LRU) since the portion of LRU is 5 times larger than the portion of LFU. With this result, we can say ARC performs better than LRU for the most of general workloads. 2)We investigate the authentic ”self-tunablity” by performing Experiment 1 to 3. From Experiment 1, we confirmed that the change of initial value p affects negligibly small amount on the hit ratio. In terms of initial value p, we can say ARC is self-tunable. From Experiment 2, we confirmed that the changes of limit value p affect quite larger portion of the hit ratio. However, it is workload-dependent. Also, we verified when the limit value of p gets smaller or bigger from c, it performs much worse on every workload. This indicates that if we balance the portion of LRU(i.e. L_1) and LFU(i.e. L_2) closer to 1:1, we can ensure that it will perform not the highest but the above average on a certain workload compare to other configurations. In short, the choice of p to c was best-choice. From Experiment 3, we confirmed that the value of learning rate in ARC performs above the average on all traces. However, we also found that assigning the value of learning rate like **EXP** is also quite good-option. We insist that selecting the learning rate value like **EXP** gives much higher ratio on some workloads which dynamically changes its I/O access. And it also ensures that even if it does not perform than ARC, the gap between EXP and ARC are quite tolerable for that kind of workloads.

VI. Discussion

About practicality, we cannot say it is practical since we does not consider the trend of the second-storages. e.g. Flash-Memory, SSD. Espeically for flash-memory, it involves the write amplification and wear-leveling due to P/E cycle which affects the life-cycle of flash memory. We only measured the hit ratio of the disk traces, However, we expect that there is some amplifications for disk operations due to garbage collection or wear-leveling. Despite of that, we investigate the validity of ARC in terms of 3 tunable parameters and find that 1) EXP is quite good-option, 2) ARC is still useful replacement policy in current-days workloads. If interested in second storage properties, refer these following research papers [18], [19], [12], [20].
About experiment, we collected about 30 traces. e.g. traces from SYSTOR17 Lee et al traces [21], and other traces from [16] e.g. TPC-C, TPC-E, LM-TFE, WBS, DTRS, and etc. we spent about 4 days for parsing the disk I/O patterns of each trace. There was a lack of time for simulating all 30 traces within short days. So we only showed the simulation results for 8 traces. we spent about 8 hours for simulating those 8 traces. It would be more nice to measure the other 22 traces with the same configurations in Section V. Also, we fixed the size of disk cache as 256MB, but the size of disk cache in currently released servers varies from 128MB to even 2GB. We think that it would be also useful to simulate those traces with a much higher disk cache size. All sources used in this paper are freely available on our GitHub repository [22]. Note that even each parsed data was so big to upload at web-site, we omitted the actual/parsed traces. However, the code for parsing disk traces are available on web-site [22] and all actual traces are freely available on SNIA/IOTTA [17]. For researchers who are interested in this paper, we anticipate that anyone can easily use our open-source code. Hopefully, it would be nice that our source could be used for the basis of the further research or implementation for ARC or variants of ARC.

VII. Conclusion

This paper precisely evaluates the validity of ARC in terms of some tunable-parameters. For this, we suggest three points for configurations. First, We vary the inital value of p which affects the beginning behaviour of disk cache. Second, We vary the limit value of p which affects quite large portion of the entire cache behaviour. Lastly, We vary the value of the learning rate which affects the re-sizing of the cache when phantom-hit occurs. From this three points, we show the evaluation results through cache simulation with page size as 4KB, cache size as 256MB. Experimental results say that the initial value of p and the limit value of p was a best-choice for ARC. Additionally, we show that changing the value of learning rate as EXP is quite good-option for some workloads which dynamically changes the I/O disk access of it. Furthermore, from comparing original ARC with $p_{max} = \frac{5}{3}c$, we show the effectiveness of ARC is still valid compared to LRU(not exactly LRU) for the real-world workloads that is currently and widely used in research-area.

References

[1] E. G. Coffman and P. J. Denning, *Operating systems theory*. prentice-Hall Englewood Cliffs, NJ, 1973, vol. 973.

[2] D. Lee, J. Choi, J.-H. Kim, S. H. Noh, S. L. Min, Y. Cho, and C. S. Kim, "Lrfu: A spectrum of policies that subsumes the least recently used and least frequently used policies," *IEEE transactions on Computers*, no. 12, pp. 1352–1361, 2001.

[3] Y. Zhou, J. Philbin, and K. Li, "The multi-queue replacement algorithm for second level buffer caches." in *USENIX Annual Technical Conference, General Track*, 2001, pp. 91–104.

[4] N. Megiddo and D. S. Modha, "Arc: A self-tuning, low overhead replacement cache." in *FAST*, vol. 3, no. 2003, 2003, pp. 115–130.

[5] S. Watanabe, *Solaris 10 ZFS Essentials*. Pearson Education, 2009.

[6] G. Smith, *PostgreSQL 9.0: High Performance*. Packt Publishing Ltd, 2010.

[7] C. Hogan and D. Epping, *Essential Virtual SAN (VSAN): Administrator's Guide to VMware Virtual SAN*. VMware Press, 2016.

[8] L. A. Belady, "A study of replacement algorithms for a virtual-storage computer," *IBM Systems journal*, vol. 5, no. 2, pp. 78–101, 1966.

[9] J. T. Robinson and M. V. Devarakonda, *Data cache management using frequency-based replacement*. ACM, 1990, vol. 18, no. 1.

[10] S. Huang, Q. Wei, D. Feng, J. Chen, and C. Chen, "Improving flash-based disk cache with lazy adaptive replacement," *ACM Transactions on Storage (TOS)*, vol. 12, no. 2, p. 8, 2016.

[11] B. S. Gill and D. S. Modha, "Sarc: Sequential prefetching in adaptive replacement cache." in *USENIX Annual Technical Conference, General Track*, 2005, pp. 293–308.

[12] S.-y. Park, D. Jung, J.-u. Kang, J.-s. Kim, and J. Lee, "Cflru: a replacement algorithm for flash memory," in *Proceedings of the 2006 international conference on Compilers, architecture and synthesis for embedded systems*. ACM, 2006, pp. 234–241.

[13] M. K. Qureshi, A. Jaleel, Y. N. Patt, S. C. Steely, and J. Emer, "Adaptive insertion policies for high performance caching," *ACM SIGARCH Computer Architecture News*, vol. 35, no. 2, pp. 381–391, 2007.

[14] A. Jaleel, K. B. Theobald, S. C. Steely Jr, and J. Emer, "High performance cache replacement using re-reference interval prediction (rrip)," in *ACM SIGARCH Computer Architecture News*, vol. 38, no. 3. ACM, 2010, pp. 60–71.

[15] R. Koller and R. Rangaswami, "I/o deduplication: Utilizing content similarity to improve i/o performance," *ACM Transactions on Storage (TOS)*, vol. 6, no. 3, p. 13, 2010.

[16] S. Kavalanekar, B. Worthington, Q. Zhang, and V. Sharda, "Characterization of storage workload traces from production windows servers," in *2008 IEEE International Symposium on Workload Characterization*. IEEE, 2008, pp. 119–128.

[17] S. IOTTA, "Storage networking industry associations input/output traces, tools, and analysis."

[18] F. Ye, J. Chen, X. Fang, J. Li, and D. Feng, "A regional popularity-aware cache replacement algorithm to improve the performance and lifetime of ssd-based disk cache," in *2015 IEEE international conference on networking, architecture and storage (NAS)*. IEEE, 2015, pp. 45–53.

[19] Z. Chen, N. Xiao, and F. Liu, "Sac: Rethinking the cache replacement policy for ssd-based storage systems," in *Proceedings of the 5th Annual International Systems and Storage Conference*. ACM, 2012, p. 13.

[20] T. Kgil and T. Mudge, "Flashcache: a nand flash memory file cache for low power web servers," in *Proceedings of the 2006 international conference on Compilers, architecture and synthesis for embedded systems*. ACM, 2006, pp. 103–112.

[21] C. Lee, T. Kumano, T. Matsuki, H. Endo, N. Fukumoto, and M. Sugawara, "Understanding storage traffic characteristics on enterprise virtual desktop infrastructure," in *Proceedings of the 10th ACM International Systems and Storage Conference*. ACM, 2017, p. 13.

[22] J. Jeong and T. Lee, "After arc : Advanced operating system project," https://github.com/koreanTonyStark/AdvOS_ARC, 2019.