2009 11th IEEE International Conference on High Performance Computing and Communications

SSARC: the Short-Sighted Adaptive Replacement Cache*

Zhiguang Chen, Nong Xiao, Fang Liu, Yingjie Zhao
Department of Computer Science, National University of Defense Technology, Changsha, China
chenzhiguanghit@gmail.com,{nongxiao,liufang}@nudt.edu.cn,hyperseymour@163.com

Abstract

As the performance gap between disks and processors continues to increase, dozens of cache replacement policies come up to handle the problem. Unfortunately, most of the policies are static. Nimrod Megiddo etc put forward a low overhead adaptive policy called ARC. It outperforms most of the static policies in most situations. But, ARC adapts itself to the workloads by the feedback of the missed pages. It hasn't carried out the adaption before missed pages are discovered. We propose a high performance adaptive replacement policy. It adapts itself to the workloads by the feedback of the hit pages, so, it is more sensitive to the changes of the workloads than ARC. As the policy stares at the tails of the queues regardless of other pages, we name the policy as short-sighted adaptive replacement policy.

The ARC usually regrets for the missed pages and wishes to rescue the neighborhood of them. However, SSARC endeavors to protect the would-be-reused pages from being replaced aggressively. So, it outperforms ARC in most situations. We compared SSARC with LRU、2Q and ARC. The trace-driven experiments represent that SSARC gains higher performance.

1. Introduction

1.1. Background

The communication between processors and disks is deteriorating. Fortunately, the data requested by processor has localities, which encourages us to introduce into the cache. As the demanding page at the very moment will be re-accessed in the near future, we keep a copy of the page which is fetched from disk in the cache, whose speed is comparable with processors. The next demand will be satisfied by the cache directly, avoiding the disk latency. However, the cache can't be as enormous as we expect, it will be cram-full sooner or later. From then on, before a new requested page is inserted into the cache, a free slot must be supplied. The policy determining which page to be replaced is a hot-spot of cache research.

1.2 The short-sighted policy

Our contribution is that we propose a high performance adaptive replacement policy, which is abbreviated as SSARC. SSARC divides the pages into two groups: pages only have been accessed once (which are called once-accessed pages) and pages have been accessed multiple times (which are called multi-ply accessed pages). Correspondingly, SSARC builds up two LRU queues: once-accessed queue and multi-ply accessed queue. As the victim of the replacement is either the LRU page of the once-accessed queue or the LRU page of the multi-ply accessed queue, SSARC compares the LRU-ends of the queues (we call the LRU-ends once-accessed tail and multi-ply accessed tail respectively), the victim is taken from the useless tail. That's to say: if the once-accessed tail is more useful, SSARC replaces the LRU page in the multi-ply accessed queue; otherwise, replaces the LRU page in the once-accessed queue.

To compare the utility of the tails, we introduce into the concept "Emergency". When a page of the once-accessed tail is hit, SSARC calculates the "Emergency" for the page, and adds the result to the total "Emergency" of the once-accessed tail. The same work is carried out for multi-ply accessed tail. When a free slot is required, SSARC selects the victim from the very tail that has less total "Emergency".

SSARC balances between once-accessed tail and multi-ply accessed tail with the changes of workloads. It is an adaptive policy. SSARC emphasizes on the tails of the two queues. It protects the maybe-reused pages in the tails from being replaced, but ignores other pages. It is short-sighted, so, we call it short-sighted policy. To compensate for the short-sighted decision, the poli-

* This work is partially supported by National Natural Science Foundation of China grants NSFC60736013 and 863－2006AA01A106.

978-0-7695-3738-2/09 $25.00 © 2009 IEEE
DOI 10.1109/HPCC.2009.82

IEEE computer society

cy maintains a ghost cache to capture some of the missed pages.

SSARC maintains three queues: once-accessed queue、multi-ply accessed queue and ghost cache. They are all LRU queues, so the policy has constant-time complexity per request. We simulated the policy with traces and compared it with LRU [1]、2Q [2]、ARC[3]. The simulation shows that SSARC outperforms them with most of the traces.

The rest of the paper is arranged as follows: in section 2, we describe the policy elaborately; section 3 represents the experiments and comparisons; in section 4, we briefly introduce the related work; and conclude the paper in section 5.

2. SSARC

2.1. Queues

A high performance replacement policy should be scan resistant. An effective way to achieve scan resistance is dividing the pages into once-accessed pages and multi-ply accessed pages. When the scan pattern comes up, only the once-accessed pages can be replaced, the repeated pages are protected. 2Q、ARC etc achieve scan-resistance this way. 2Q replaces the tail of A1in, Am protects the multi-ply accessed pages; ARC replaces the tail of L1, L2 protects multi-ply accessed pages. SSARC adopts the same method, divides the pages into once-accessed pages and multi-ply accessed pages.

We analyze traces from PCs and servers for store-online, and present a characteristic of them on table 1. As you can see, the only twice accessed pages take half the sum of the multi-ply accessed pages, or more. The characteristic complies with the fact that the distribution of the number of references to a page tends toward that of a power law. When an only-twice accessed page comes its second reference, ARC takes it as a multi-ply accessed page and inserts it into L2, but, the page will never be reused again. He will hire the slot without any payment for a long time. SSARC solves the problem this way: when a once-accessed page is requested, it becomes a twice accessed page, but SSARC inserts it into the once-accessed queue all the same. If it hasn't been requested before it comes to the tail of the once-accessed queue, it will be taken as a once-accessed page and be replaced ultimately. If it did be reused, SSARC inserts it into multi-ply accessed queue. Some correlated references also can be masked this way, which is similar with the method adopted by 2Q. Now, the once-accessed queue contains some pages that had been accessed twice, but we still call it once-accessed queue.

Maybe, the policy is not scan-resistant anymore because of the twice-accessed pages in the once-accessed queue. Fortunately, the ghost cache can compensate for it more or less. When a miss page was captured by the ghost cache, SSARC inserts it into the multi-ply accessed queue directly. The size of the ghost cache is the same with the cache. Its overhead is estimated by [3].

2.2. Short-sighted Policy

SSARC pays attention to once-accessed queue and multi-ply accessed queue. As the LRU policy does, replacement occurs at the tails of the two queues. As the figure 1 depicts, SSARC compares the utilities of once-accessed tail and multi-ply accessed tail. If the once-accessed tail is more valuable, the victim is taken from multi-ply accessed tail, or otherwise.

The concept "Emergency" is introduced into our policy to explain the utility of each tail. We illustrate the use of "Emergency" as follows. When a page belonging to once-accessed tail is hit, SSARC calculates the "Emergency" for it, then, adds the "Emergency" to the total "Emergency" of the once-accessed tail. The total "Emergency" is the utility of the once-accessed tail. Replacement occurs at the tail with less utility. "Emergency" is explained elaborately in the next sub-section.

Table 1: **the distributive characteristic of traces**

Traces	source	Unique pages	Multi-ply accessed pages	Twice accessed pages
Trace1	PC	2727253	713346	400472
Trace2	PC	7194783	1691458	842358
Trace3	PC	7570361	1124877	598411
Trace4	PC	3703387	857921	487562
Trace5	PC	7876198	1513692	705481
Trace6	Sever for store-online	4661422	984419	476611
Trace7	Sever for store-online	5397134	1024357	478241

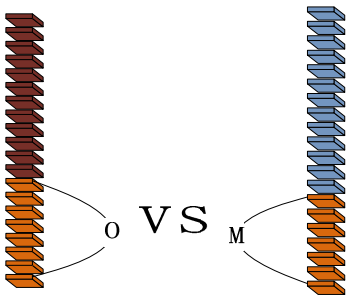

Figure 1: The comparison of the tails, when the once-accessed tail is more valuable, the victim is taken from multi-ply accessed tail, or otherwise. "O" denotes the once-accessed tail; "M" denotes the multi-ply accessed tail.

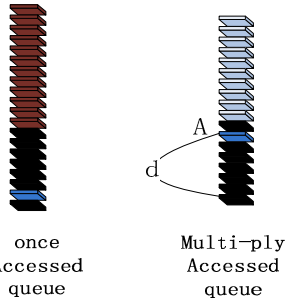

Figure 2: Two hit pages(colored blue) belong to once-accessed tail and multi-ply accessed tail respectively, page A has little "Emergency" because of the longer distance between A and the tail of the queue.

2.3. Emergency

Case 1: figure 2 represents two hit-pages (blue pages), they belong to once-accessed tail and multi-ply accessed tail respectively. The neighbor pages of them are assumed to be requested soon after, and should be protected from being replaced. Compared to the hit-page in the multi-ply accessed tail, the one belonging to the once-accessed tail is more dangerous to be replaced due to the shorter distance between the page and the tail of the queue. That is to say, the hit-page in the once-accessed tail has great "Emergency".

Case 2: if the once-accessed queue has fewer pages than the multi-ply accessed queue, the multi-ply accessed tail is more likely to be replaced; or otherwise.

Beyond the analysis above, we define the two components of "Emergency" correspondingly.

Case 1: as the figure 2 depicts, we denote the hit-page in the multi-ply accessed tail A, the distance between page A and the tail of the queue is d, the size of cache is C. The first component is defined as below.

$E_1=\log_m(C/d)$ （m is a parameter）

Case 2: for the same hit-page A, we denote $SIZE_m$ and $SIZE_o$ as the sum of pages in the multi-ply accessed queue and once-accessed queue respectively. Parameter Q is defined as $SIZE_o/SIZE_m$ (of course, if the hit-page belongs to the once-accessed tail, $Q=SIZE_m/SIZE_o$). The second component is defined as below.

$E_2=\log_m Q$ （m is a parameter）

The total "Emergency" of A is: $E=E_1+E_2$.

2.4. Replacement

We denote the utilities of the tails U_m and U_o respectively. When a page A in the multi-ply accessed tail is hit, SSARC calculates the "Emergency" E_A, update the U_m , $U_m=U_m+E_A$; the same work is carried out for a hit page in the once-accessed tail. When a replacement is on the way, update the U_m and U_o:

$U_m=U_m\times C/(U_m+U_o)$

$U_o=U_o\times C/(U_m+U_o)$

If $SIZE_o\geq U_o$ or $SIZE_m \leq U_m$, replace the LRU page of the once-accessed queue; otherwise, replace the LRU page of the multi-ply accessed queue.

2.5. Three parameters

Only the hit-page belongs to the tail that could make contribution to the utility of its queue. But how to determine whether a hit-page is in the tail or not? We define the size of tail as $|tail|=\min(C/m, SIZE_o, SIZE_m)$.

If the distance between a hit-page A and the tail of the queue is less than |tail|, the page A is in the tail, SSARC calculates the "Emergency" for it.

The distance between the hit-page A and the LRU-page of the queue is necessary to calculate the "Emergency". SSARC maintains stamp for once-accessed queue and multi-ply accessed queue respectively, $stamp_o$ and $stamp_m$. When a page is added to the MRU end of the queue, SSARC labels the page with the stamp of the queue. SSARC calculates the distance for a hit-page in the tail as below: SIZE denotes the sum of pages belonging to the queue, $stamp_{head}$ and $stamp_{tail}$ denote the stamp of the MRU page and the LRU page respectively. The stamp of hit-page A is $stamp_A$. The distance between page A and the LRU page of the queue is approximated as

$d=SIZE\times(stamp_A-stamp_{tail})/(stamp_{head}-stamp_{tail})$

The parameter m is necessary to calculate the "Emergency" and |tail|. It is a tunable parameter depending on the cache size. Dozens of experiments show that m maps the cache size as table 2 .

Table 2: selection of m

Cache size(Pages)	64K	128K	256K	512K	1M	2M
m	2	4	8	16	32	64

```
Block A is accessed

If(A is in the once-accessed queue) {
    If(A is in the once-accessed tail) {
        If(E1>=1)
            Uo= Uo +E1;
        If(E2>=1)
            Uo = Uo +E2;
    }
    If(A is a twice accessed page)
        Stamp A with stampm and add it to the MRU end of the multi-ply accessed queue.
    Else
        Label A as a twice accessed page, Stamp it with stampo and add it to the MRU end of the once-accessed queue.
}Else if(A is in the multi-ply accessed queue) {
    if(A is in the multi-ply accessed tail) {
        If(E1>=1)
            Um = Um +E1;
        If(E2>=1)
            Um = Um +E2;
     }
        Stamp A with stampm and add it to the MRU end of the multi-ply accessed queue.
}Else {
    If(there is no free slot)
        Replace();
    If(A is in the ghost cache)
        Stamp A with stampm and add it to the MRU end of the multi-ply accessed queue.
    else
        Stamp A with stampo and add it to the MRU end of the once-accessed queue.
}
```

```
Subroutine Replace()
{
    Um=Um×C/(Um+Uo)     //C is the size of cache
    Uo=Uo×C/(Um+Uo)
    If((|once-accessed queue|>=(int)Uo)||(|multi-ply accessed-queue|<=(int)Um)) {
        While(true)   {
            If(the tail of once-accessed queue is a once-accessed page or a dated multi-ply accessed page) {
                Remove the tail of once-accessed queue;
                Break;
            }Else
                Shift the tail of once-accessed queue to its head, Label it as a dated multi-ply accessed page ;
        }
    }Else
        Remove the tail of multi-ply accessed-queue;
}
```

Figure 3: pseudo-code of the short-sighted policy

3. Experiment

3.1. Experiment results

Some traces used in our simulations have been used in [10]. We chose four of them to represent different kinds of workloads. Besides that, we collected traces from severs for store on-line for the simulations.

We compare the short-sighted policy with LRU、2Q and ARC. As the results show, the short-sighted policy outperforms LRU、2Q and ARC on most workloads.

The preliminary principles of the short-sighted policy and ARC are alike, so they react analogously on the same workloads. As Figure 4 and Figure 5 represent, the hit ratio curves achieved by the two policies are approximate parallel. But, the short-sighted policy takes action earlier than ARC dose, that's why it outperforms ARC substantially.

Most requests of trace 4 are scan-requests, so only a scan-resist policy could be competent with it. SSARC、ARC、2Q are scan-resist except LRU. As the fourth picture in Figure 4 shows, LRU gains the worst hit ratio obviously.

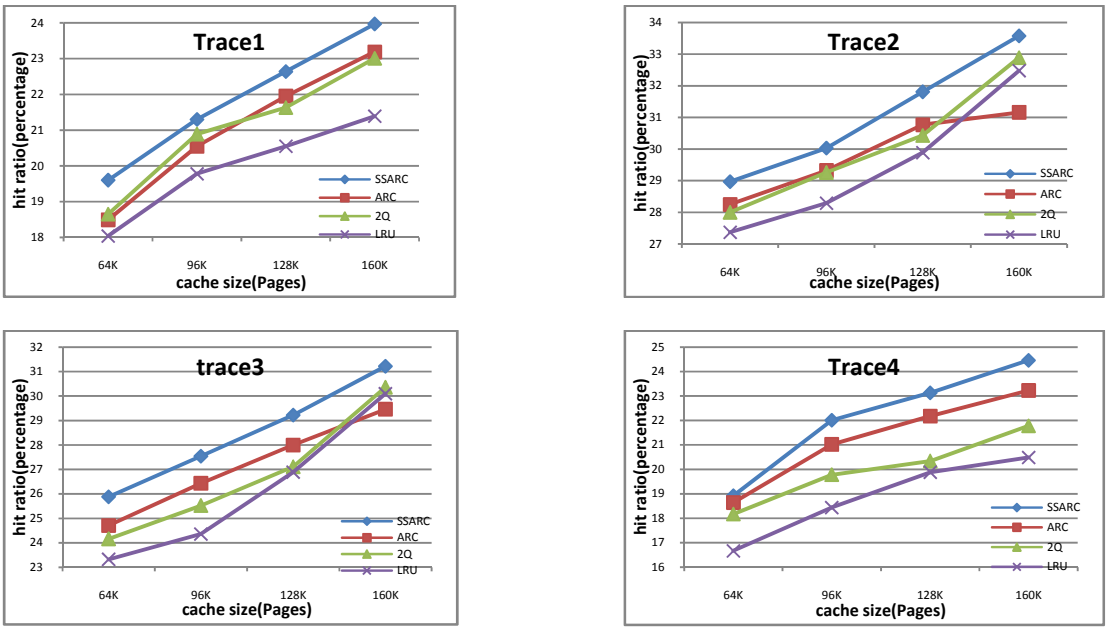

Figure 4: Hit ratio achieved by SSARC、ARC、2Q and LRU with traces of PCs

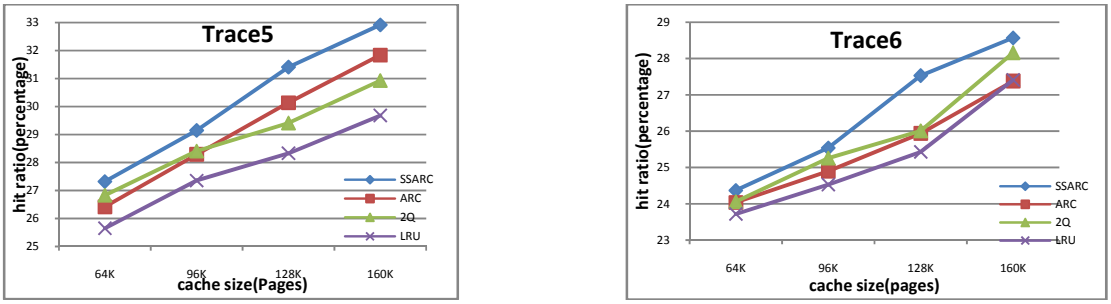

Figure 5: Hit ratio achieved by SSARC、ARC、2Q and LRU with traces of severs

3.2. Comparison and analysis

We test the parameter |K1in| for 2Q, select the best one |K1in|=0.4, and assign |Kout|=0.5 as the paper suggests. However, SSARC outperforms 2Q in most circumstances. 2Q isn't competent for the fluctuating workloads.

ARC is an adaptive policy. But it hasn't taken action before miss pages are discovered. SSARC scouts the changes of the workloads aggressively, takes action timely, and prevents unnecessary misses. In addition, SSARC masks some correlated references. So, it's unsurprised that SSARC outperforms ARC.

4. Related work

All the block-level replacement policies are based on locality and/or frequency. Such as, LRU [1] depends on locality; LFU [5] depends on frequency; LRFU [6]、LRU/K [7]、MQ [8] take both locality and frequency into account. To design a replacement policy using both locality and frequency, you confront with two problems: (1) How to describe the locality? (2) How to conjunct the locality and frequency together?

4.1. Description of the locality

MIN provides the most accurate description of locality. It uses the period of logical time between the current reference and the next reference to a page to describe the locality. ULC [9] calls it ND. MIN selects the page that has the largest ND as the victim. Unfortunately, MIN is an offline policy, it can't be put into practice, but gives the best performance.

LRU uses the period of logical time between the last reference and current logical time (ULC calls it R) to simulate MIN's ND. It selects the LRU page as the victim. In fact, R simulates ND so perfectly that most of the following complicated policies are its mutations.

ULC defines LLD, which denotes the R when the page was referenced last time. Then the larger of LLD and R is considered as the simulation of ND. So, the simulation is more accurate than R. LIRS defines IRR, which is similar with LLD. The difference is that: LLD statics the references between the two consecutive requests of a page, but IRR statics the different pages referenced between the two consecutive requests of a page.

LRU/K was designed for masking correlated references in database. LRU/K remembers the period of time of the last k references. When the value of k is 2 or 3, the correlated references are masked. When the value of k is 1, LRU/K collapses to be LRU.

4.2. Conjunction of the locality and frequency

LRU and LFU are accused of their prejudice against frequency or locality. A high performance policy should take both locality and frequency into account. The typical attempt to conjunct locality with frequency is carried out by LRFU. It assigns a weight for each page as follows:

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

Where λ is a tunable parameter. LRFU always selects the page with least weight as the victim. When the λ approaches 0, LRFU collapses to LFU; when the λ approaches 1, LRFU collapses to LRU.

Multi-queue policy records the sum of references for each page, and dispatches all the pages into eight priority queues. The more frequently a page was referenced, the higher priority queue it belongs to. Multi-queue considers frequency to be more important than locality.

2Q and ARC dispatch all the pages to once-accessed queues or multi-ply accessed queues based on the frequency. The LRU queues represent the locality. In fact, ARC improves 2Q. The short-sighted policy divides all the pages into three classes: once-accessed pages、twice accessed pages and multi-ply accessed pages. We make use of frequency a little more.

5. Conclusion

We propose a high performance tunable cache replacement policy. It balances the replacement between once-accessed pages and multi-ply accessed pages continually. In fact, the principle of SSARC is similar with ARC, but ARC hasn't taken actions before missed pages are caught; on the other hand, SSARC protects the would-be referenced pages aggressively, so it achieves high performance.

Acknowledgement

We appreciate Lipin Chang, Po Liang and Joshon for traces T1 to T4, and anonymous reviewers for valuable suggestions that greatly improved this paper. This work is partially supported by National Natural Science Foundation of China grants NSFC60736013 and 863－2006AA01A106.

References

[1] L. Belady, "A Study of Replacement Algorithms for a Virtual-Storage Computer", IBM Systems J., 5(2):78-101, 1966.

[2] T. Johnson and D. Shasha, "2Q: A low overhead high performance buffer management replacement algorithm", in Proc. VLDB Conf., pp. 297–306, 1994.

[3] N. Megiddo and D.S. Modha, "ARC: A Self-Tuning, Low Overhead Replacement Cache", in Proc. Second USENIX Conf. File and Storage Technologies, pp.115-130, Mar. 2003.

[4] S. Jiang and X. Zhang, "LIRS: An efficient low inter-reference recency set replacement policy to improve buffer cache performance", in Proc. ACM SIGMETRICS Conf., pp.31-42, 2002.

[5] D. Lee, J. Choi, J.-H. Kim, S. H. Noh, S. L. Min, Y. Cho,and C. S. Kim, "On the existence of a spectrum of policies thatsubsumes the least recently used (LRU) and least frequently used(LFU) policies," in Proc. ACM SIGMETRICS Conf., pp. 134–143, 1999.

[6] 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 Trans. Computers, vol. 50, no. 12, pp. 1352–1360, 2001.

[7] E. J. O'Neil, P. E. O'Neil, and G. Weikum, "The LRU-K page replacement algorithm for database disk buffering", in Proc. ACM SIGMOD Conf., pp. 297–306, 1993.

[8] Y. Zhou and J. F. Philbin, "The multi-queue replacement algorithm for second level buffer caches", in Proc. USENIX Annual Tech. Conf. (USENIX 2001), Boston, MA, pp. 91–104, June 2001.

[9] S. Jiang and X. Zhang, "ULC: A file block placement and replacement protocol to effectively exploit hierarchical locality in multi-level buffer caches",in Proc. ICDCS Conf., pp.168–177, 2004.

[10] Li-Pin Chang and Tei-Wei Kuo, "An Adaptive Stripping Architecture for Flash Memory Storage Systems of Embedded Systems", IEEE Eighth Real-Time and Embedded Technology and Applications Symposium (RTAS), San Jose, USA, Sept 2002.