An improved adaptive policy Based on Recency and Frequecy

Hongliang YANG

School of Informacs, Linyi University, Linyi, 276009, China

email: lytuyhl@163.com

Keywords: Cache; Replacement Policy; LRFU; Adaptive Method; Improved LRFU

Abstract. Cache replacement policy is one of the caching techniques, including the recency based algorithms and the freqency based algorithms etc. Although combining the recency and frequency, the LRFU algorithm could not dynamically adjust itself to adapt to the practical circumstance. This paper proposes an improved LRFU algorithm, which can dynamically modify the λ value of the LRFU to choice the appropriate replacement policy according to the practical case. Adopted the trace simulations with three common access patterns, the experiment shows that the improved LRFU algorithm can improve the hit ratio compared the LRFU, LRU and LFU algorithms.

Introduction

Caching techniques are fundamental for bridging the performance gap between components in a computer system, they are widely used in storage system, databases, virtual memory management in OS, etc. Performance of caching techniques has great influence over memory latency, processor performance and energy consumption. Cache replacement policy is one of the techniques, managing the contents of the cache so as to improve the overall performance. In this paper, we study the cache replacement policy focusing on the processor and memory. We assume that both main memory and cache are managed in discrete, uniformed-sized units called block. An efficient cache replacement algorithm must replace the block that is used farthest in the future. When the block is present in the cache, then it can be served quickly resulting in a “cache hit”. On the other hand, if a requested page is not present in the cache, then it must be fetched from the main memory resulting in a “cache miss”. Usually, latency on a cache miss is significantly higher than that on a cache hit. The main performance index of the cache replacement policy is the rate of cache hit or miss. Hence, the cache strategies focus on improving the hit ratio.

There are several kinds of existed cache replacement policies, which have been divided into the access pattern based algorithms and the analysis model based algorithms [2]. The policies based on the access pattern are made decision by the intuitive inspiration and experience summary, such as ARC, MQ, FBR, 2Q etc. The policies based on the analysis model are proposed the reasonable policy by access features, such as LRU, EELRU, LIRS, LFU etc. In this model, the locality and the frequency are the features of the most common access, so the polices based on the balance strategy algorithm of the recency and frequency have the better serviceability, and there are many kinds of existed replacement policies combined recency and frequency such as ARC, MQ, FBR, LRFU, etc.

Although, LRFU policy compromised the LFU and LRU two replacement policies, which transforms from the LFU to the LRU with the varying of the parameter λ from 0 to 1. In a particular LRFU algorithm, because the value λ is fixed, the recency and frequency can't be adjusted even using the appropriate valve of CRF, when the access patterns and access features changed. So the LFRU cannot be applied well to the practical environment. In this paper, we proposed an adaptive mechanism to improve LRFU combining the existed ideas of the dynamic adjustment.

Prior Work

The advantages of LRU are that it is extremely simple to implement and captures recency that is common to many workloads. While LRU captures the “recency” features of a workload, it does not capture and exploit the frequency features of a workload. More generally, if some blocks are often

re-requested, but the temporal distance between consecutive requests is larger than the cache size, then LRU cannot take advantage of such pages with long-term utility. LRU can be easily polluted by a scan, that is, by a sequence of one-time use only page requests leading to lower performance.

LFU replaces the least frequently used block and is optimal under the IRM [2], but it has several drawbacks [5]: (1) Its running time per request is logarithmic in the cache size. (2) It is oblivious to recent history. (3) It does not adapt well to variable access patterns; it accumulates stale pages with past high frequency counts, which may no longer be useful.

LRFU [5] is the novel caching algorithms that have attempted to combine recency and frequency (CRF) with the intent of removing one or more disadvantages of LRU. The police always choose a block with minimum value of CRF to evict. The CRF value is calculated by weight function $C_{t_{present}}(b) = \sum_{i=1}^{k} F(x)$, $F(x) = (1/p)^{\lambda x}$, among them, $C_{t_{present}}(b)$x is the CRF value of b block in the current time, λ is the time interval from the last access to the current, λ is the balance parameter of locality and frequency. For example, the accessed time of some block is at 2, 5 and the current time is at 7, then the value of the CRF of the block is F(7-2)+F(7-5). It is obvious that if the parameter is the greater, the policy is more inclined to focus on the access locality, if the λ value gets smaller, the policy is more inclined to focus on access frequency features. But the λ value is fixed, so the LRFU is not a self-tuning policy.

Improved LFRU

The improved LFRU dynamically adjusts the λ value in the whole access, which can adapt the algorithm to the changes of access pattern, so the system can get the best performance.

Only using FIFO queues, 2Q, MQ, ARC, CAR [4]and other algorithms saved the information of the evicted block recently, when the blocks accessed again, according to analyzing the information in queues, the policies could be adjusted dynamically. We used this strategy for reference into the improved LFRU policy.

In order to count the block information with a longer time access, we also use a FIFO queue called Lout to record the number and the CRF value of the evicted block, and the queue length is set to C. Lout. In this queue, the blocks are arranged according to the evicted order, which evicted recently is placed at the tail of the queue. When the queue is full, the block on the head will be eliminated. When accessed miss hit, the record will be searched in the Lout firstly, if the corresponding record is found, the block is to be loaded into the cache, at the same time, the present value of CRF is calculated according to the last access time and CRF value of the recorded block in the Lout, and then, the relevant record is deleted. If there is no record in Lout, the block will be loaded directly from the memory and the CRF value is set to F (0), besides, the information of the replaced block will be placed in the tail of the Lout queue. If Lout queue is full, the head record in the queue will be deleted.

when accessing the new block, if the corresponding recorders in the Lout queue are found successively, or found successively and the times of the block in the queue are more than out of the queue, these cases indicate that the block being accessed more times is more likely to be accessed again, so we reduce the λ value to make the replacement strategy more intend to LFU algorithm. If the new a block is not in Lout accessed successively, or not in the Lout for many times and the times of a new block not in Lout are more than in the Lout times, which indicates that the block being accessed at the earliest is the least to be accessed again, so we increase the λ value to make the replacement strategy more intend to LRU algorithm. Besides, if the requested block is found in the cache, which indicates that the block accessed more times is more likely to access again, the pattern is dealt as the times found in Lout of the new block.

In conclusion, this algorithm can be expressed as following:

(1) Following the basic LRFU algorithm, and count the times of the successional access.

(2) Using the FIFO queue to save the record number and CRF value of the block evicted the cache and the queue length is set to C.Lout. When missed hit and searched the corresponding record in Lout, the block can be re calculated CRF value according to the record of the last access time and

λ value.

(3) Adjusting dynamically the λ value as the following rules:

```
Set and initialize parameters;
If (new block is recorded in Lout) or (hit in cache)
   {
Count times with a parameter InTimes;
If (this block is in Lout or hit last time)
{
Then count the successional times with SuccTimes;
   Else clear SuccTimes;
 }
Else
Count times with another parameter OutTimes;
}
If ((InTimes > M_Times AND InTimes/OutTimes > M_Ratio) OR (SuccTimes> M_Times AND
 successional in the Lout))
{
 Modify λ to λ/M_Value;
 Clear all parameters;
}
If ((OutTimes >M_Times AND OutTimes/InTimes > M_Ratio) OR (SuccTimes>M_Times AND
successional out the Lout))
{
 Modify λ to λ* M_Value;
 If the value is more than 1, regard as 1;
 Clear all parameters;
}
```

Here, M_times expresses the change opportunity, M_ratio expresses the change possibility. In the practical application, we can adjust the M_Times and M_Ratio according to the practical circumstance. Usually, we set the initial λ=0.001, and increases or reduce λ value according to the rules in each phase. From the test, we find that when we set M_Ratio=1, M_Times=5 and M_Value =5, the dynamical effect is the best.Test results

Experiment and evaluation

In this section, we use the trace simulation to evaluate the performances of the improved LRFU algorithm. The experiment environment is the Memory Buddies Trace of Linux server [1]. Memory Buddies Trace is the memory access trace of the data virtualization center server combined the resources of the multiple devices and provided users with a unified logical interface, the users can access the different resources by the different patterns [7]. In practical application, the different replacement algorithm has different performance in application environment, according to the feature of the improved LRFU, we choose three kinds of trace simulation, representing respectively three kinds of access pattern: linear access pattern, probability access pattern, locality access pattern. Although the whole hit ratio is not high, Fig1~Fig2 shows that the improved LRFU can improve the hit ration in different circumstance compared with LRFU, LRU and LFU, which illustrates that adjustment dynamically of the λ value plays the positive role. And Fig3 shows that the improved LURF policy do not reflect the good performance compared the LRFU policy, because when accessed flow with a strong local characteristics and access successively, the algorithm reduce λ value to tend to LFU policy, which caused the negative effect, especially in the large capacity cache. Because the adjustment of λ depends on the change of access feature, and the improvements of hit rate in cache need not extra expenditure, so it may be adopted in wide fields.

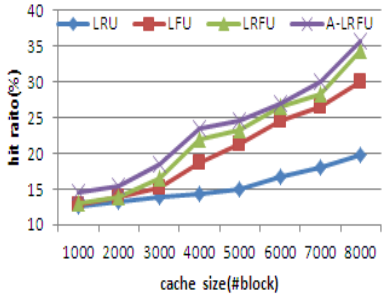

Fig.1. Compare hit rate in linear access pattern pattern

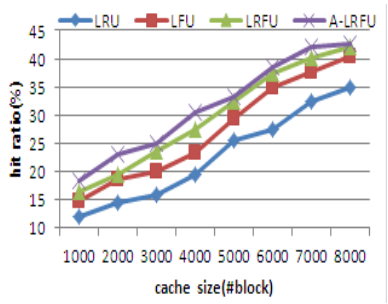

Fig.2. Compare hit rate in probability access pattern

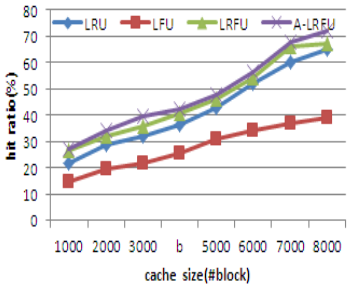

Fig.3. Compare hit rate in local access pattern

Conclusion

This paper improved the LRFU algorithm with adaptive combined the strategy of dynamical adjustment, in this strategy, according to the recorded history information from access recency (or frequency) to access frequency (or recency), the CRF value can be adjusted with the change of the λ value, and so, the different strategies can be adjust dynamically to adapt the different circumstance. Thus, the hit rate is improved by adopting this strategy, and the test results show that this policy did not bring any negative effect. Of course, this strategy depends on the CRF value, and what the CRF recorded is the history information, which perhaps is no use for the present replacement strategy, so, the evicted block is not the suitable. This is our late task to improve.

Acknowledgement

In this paper, the research was sponsored by the Natural Science Foundation of Shandong Province (Project No. ZR2014FL012), Achievements Transformation Major Projects of Shandong Province (Project No. 2014ZZCX02702) .

References

[1] Wood T, Tarasuk G L, Shenoy P. “Memory buddies: Exploiting page sharing for smart colocation in virtualized data centers,” Proceedings of the ACM SIGPLAN/SIGOPS Conference on Virtual Execution Environments. New York, USA, 2009:31-34.

[2] Courtois P. vantilborgh H. “A decomposable model of program Paging behavior,” Aeta Information, 1976, 6(3):251-275.

[3] Glass G, Cao P. “adaptive page replacement based on memory reference behavior,” Proceedings of the ACM SIGMETRICS Conference on Measurement and Modeling of Computer System. Seattle, USA, 1997:115-126.

[4] Johnson T. Shasha D. “2Q: A low overhead high performance buffer management replacement algorithm,” Proceedings of the International Conference on Very Large Data Bases. Santiago, Chile, 1994:439-450.

[5] Lee D, Choi J, Kim J H. “LRFU: A Spectrum of policies that subsumes the least recently used and least frequently used policies,” IEEE Transactions on Computers, 2001, 50(12):1352-1361

[6] B. Fields, “Focusing Processor Policies via Critical-Path Prediction,” In Proceedings of the 28th International Symposium on Computer Architecture, Sweden, June 2001:74–85.

[7] G. Glass and P. Cao, “Adaptive page replacement based on memory reference behavior,” In ACM SIGMETRICS Conference on Measurement and Modeling of computer systems, pages 115-126, 1997.

[8] Robinson J T, Devarakonda M V, “Data cache management using frequency based replacement,” Proceeding of the ACM SIGMETRICS Conference on Measurement and Modeling of Computer System. Atlanta, USA, 1990:134-142.

[9] Yannis Smaragdakis, “General adaptive replacement polices,” ISSM’04, October 24-25, 2004, Vancouver, British Columbia, Canada.

[10] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, “A Free, Commercially Representative Embedded Benchmark Suite,” In Proceedings of the 4th Workshop on Workload Characterization, USA, December 2001:83–94.

[11] Y. Smaragdakis, S. F. Kaplan, and P. R. Wilson, “EELRU: Simple and efficient adaptive page replacement,” ACM SIGMETRICS Conference on Measurement and Modeling of computer systems, 1999:122-133.

[12] Sun Guozhong, Yuan Qingbo, Chen Mingyu, “An improved adaptive buffer replacement algorithm used for second level buffer,” Journal of Computer Research and Development, 2007, 44(8):1331-1338. (in chinese)

[13] Song J, Zhang X. “Making LRU friendly to weak locality workloads: A novel replacement algorithm to improve buffer cache performance,” IEEE Transactions on Computers, 2005, 54(8):939-952.