Improving Cache Performance of Network Intensive Workloads

U. Vallamsetty and P. Mohapatra
Computer Science and Engineering
Michigan State University
East Lansing, MI 48824
{vallamse, prasant}@msu.edu

R. Iyer and K. Kant
Enterprise Architecture Laboratory
Intel Corporation, Portland, OR
{ravishankar.iyer, krishna.kant}@intel.com

Abstract

The performance of servers for network-intensive workloads such as web services and online transaction processing applications depends on the effective utilization of the processor caches. A detailed analysis of the cache space utilization of web workloads shows us that several memory addresses are referenced only once during their lifetime in the cache. These references frequently reside in the cache for a long time contributing to the pollution of cache. The most commonly adopted least-recently-used (LRU) replacement scheme does not exploit this characteristic. In this paper, we propose an alternative block replacement policy called Single-Touch Aware Replacement (STAR) algorithm. This algorithm predicts blocks that will potentially be referenced only once and replaces them early enough to improve cache efficiency. The STAR scheme was implemented in a trace-driven cache simulator and the performance with several commercial workloads was analyzed. The use of the STAR algorithm results in up to 20% improvement in cache performance for web workloads (SPECweb96, SPECweb99) and up to 5% improvement in online transaction processing (TPC-C) workloads.

1. Introduction

As Internet usage expands rapidly, there is an increasing need to design and build more efficient and powerful Internet servers, both front-end and back-end. Recent studies have shown that the system characteristics of commercial applications such as web services [9, 12] and online transaction processing applications [3, 6] are significantly different from those of scientific applications. Our aim in this paper is to explore the cache/memory access characteristics of some of these commercial workloads and present architectural optimizations to improve system performance for the class of network-intensive applications.

With processor's speed increasing at a rapid pace, the performance of systems used for network-intensive applications depends largely on its cache/memory performance. This exacerbates the need for efficiently utilizing the cache space. In this paper, we perform a detailed analysis of the cache space utilization of web workloads (using SPECweb96 [1] and SPECweb99 [13] benchmarks) and online transaction processing workloads (using the TPC-C benchmark [14]). Our study shows that a high proportion of memory addresses are referenced only once during their lifetime in the processor cache. These references, which we term as single-touch references, are one of the primary causes of cache pollution observed in network-intensive applications. Widely used replacement policies such as Least Recently Used (LRU) and its variations fail to recognize this property. In this paper, we propose a new replacement algorithm called STAR (Single Touch Aware Replacement) that exploits the single touch property of this popular class of workload. The proposed scheme identifies blocks that will potentially be referenced only once and replaces them early enough to reduce cache pollution and thereby improve cache efficiency and performance. The design changes required to implement the STAR algorithm are minimal and can be optionally turned off in favor of the LRU replacement policy, if desired.

Related work in this area includes the annex cache [7], the LRFU replacement policy for file caches [11] and several other cache organizations/replacement policies that have been proposed to eliminate conflicts in the cache. The annex cache technique [7] uses an additional buffer to filter out the single touch accesses from entering the L2 cache. This cache acts as an auxiliary cache at the same level as the L2 cache. The main disadvantages of this scheme include the usage of extra hardware and consumption of additional cache space. Furthermore, annex cache is applicable for only direct-mapped L2 caches. Our proposed solution does not require any additional hardware. Instead we make simple modifications to the LRU policy commonly used in most current generation microprocessors. Unlike the

annex cache, our scheme is applicable for both direct-mapped and set-associative caches. In general, this increases flexibility in adapting to other cache access patterns as well as reduces the risk of increasing access time. Our approach is similar to the LRFU policy [11] proposed by Lee et al. for the hierarchy of disk caches. However, they have not exploited the presence of single touch references and the implementation of their algorithm requires linked lists that are suitable for disk caches but not for processor caches.

In order to demonstrate the benefits of our proposed approach, we have implemented and evaluated the STAR replacement policy in a cache hierarchy simulator. Using SPECweb96 and SPECweb99 traces, we show that the STAR replacement policy is capable of improving system performance significantly. In addition, we also show that for workloads that do not possess a high degree of single-touch references, the STAR replacement policy does not degrade performance but works as well as currently used policies such as LRU. The TPC-C benchmark is used as an example for this class of applications. The use of real workloads and the observed performance benefits validate the effectiveness of the STAR algorithm. We have also analyzed the impact of various cache configurations such as cache size, degree of associativity, and line size on the performance of STAR algorithm.

The rest of this paper is organized as follows. Section 2 highlights the access characteristics of web workloads that motivates our work. Section 3 describes the STAR algorithm and the implementation details. Section 4 covers the performance evaluation of STAR. Finally, section 5 summarizes our conclusions and presents a direction for future work.

2. Characterization of Single-Touch References in Benchmarks

In order to explore the existence of single-touch references in network-intensive applications, we first need to characterize these types of reference patterns. The key numbers that are of interest here are the proportion of single-touch references and the time for which they reside in the cache before getting evicted using the LRU replacement policy. In addition, we will explore the inability of the LRU cache replacement scheme in handling these single-touch references.

As mentioned earlier, not all applications exhibit single-touch referencing behavior. Single touch referencing behavior can be observed in network-related applications and is the primary cause of high cache pollution in these applications. We analyzed the memory referencing behavior of SPECweb96 [1] and SPECweb99 [13] benchmarks. To demonstrate the impact of the proposed technique for applications that do not have a significant proportion of single-touch references, we also considered the TPC-C [14] benchmark as a candidate. TPC-C is a benchmark from Transaction Processing Council, which models an Online Transaction Processing environment.

The benchmarks were processed through a trace-driven cache simulator[1], and a snapshot of every set in the cache was taken whenever there was any activity in the set. A complete snapshot of all the cache references for the benchmarks was then generated. Scripts were run to compute measures like the miss ratio, number of accesses to every cache line before it is evicted, etc. The results obtained from this study are summarized in Tables 1, 2, and 3.

S.N	Behavior	Result
1	Total number of times a line was evicted from the cache	595517
2	Average age from last touch of lines evicted by LRU	429154
3	Probability of single-touches for every eviction	0.91
4	No. of chances of removing a single-touch when LRU did not	147911
5	Avg. no. of 1-touch requests present for every possible 1-touch eviction	1.30
6	How many times LRU evicted 1-touches	398197
7	Avg. age of 1-touches when removed by LRU	502678

Table 1. Characterization of SPECweb96 benchmark

Table 1 shows a sample set of data obtained from the snapshots for a specific cache configuration (1MB, 32Byte, 4-way L2 Cache, 32KB Unified, Direct Mapped L1 cache) using the SPECweb96 benchmark. The LRU scheme was used for replacement of cache lines. In this paper, the age of cache lines is quantified in terms of the number of references to the cache. Although this quantification may not directly correspond with the timing parameter, in the absence of time measures in the simulator, this estimation is very close. Moreover we are interested in the relative values for performance comparison, for which the reference count is adequate. It can be observed from the third row of the table that there is a high probability (0.91) of finding a single-touch line for every eviction done by LRU. However,

[1] A detailed description of the cache simulator is presented in Section 4.1

the LRU scheme was not able to exploit this situation as quantified in the Table (see row 4). However, about 66% of the evictions made by the LRU scheme are actually single-touch references(rows 1/6). Another interesting observation that can be derived from the above data is that LRU waits for a fairly long time before evicting a single-touch reference. It waits for an average of about 502678 references before evicting a single-touch reference.

S.N	Behavior	Result
1	Total number of times a line was evicted from the cache	197551
2	Average age (from last touch) of lines evicted by LRU	335637
3	Probability of single-touches for every eviction	0.69
4	No. of chances of removing a single-touch when LRU did not	70146
5	Avg. no. of 1-touch requests present for every possible 1-touch eviction	1.14
6	How many times LRU evicted 1-touches	64895
7	Avg. age of 1-touches when removed by LRU	317256

Table 2. Characterization of SPECweb99 benchmark.

Table 2 shows the cache behavior for the traces of SPECweb99 benchmark workload. This data is from a smaller trace hence the number of evictions from the cache block is lower than that of Table 1. It can also be seen that the probability of single-touches for every eviction made from the cache block is 0.69, which is lower than that obtained for the SPECweb96 benchmark. This is expected since the number of single-touches is lower in case of the later benchmark. Only 32% (rows 1/6) of the evictions made by LRU in this case are single touches, whereas the percentage of single-touches is 66% in case of the SPECweb96 trace. Due to this decrease in the percentage of single-touch references we expect to see a lower improvement for this benchmark than that of SPECweb96.

Data in Table 3 is obtained by repeating the previous experiment with the TPC-C benchmark trace. This trace is similar to SPECweb96 in terms of the number of references. The percentage of single-touches in this trace (27%) is much lower than the SPECweb96 trace. We can see that the probability of obtaining a single-touch for every reference (0.59) is lower than the previous two cases. Since the TPC-C benchmark has

S.No	Behavior	Result
1	Total number of times a line was evicted from the cache	553929
2	Average age from last touch of lines evicted by LRU	228381
3	Probability of single-touches for every eviction	0.59
4	No. of chances of removing a single-touch when LRU did not	176117
5	Avg. no. of 1-touch requests present for every possible 1-touch eviction	1.12
6	How many times LRU evicted 1-touches	152802
7	Avg. age of 1-touches when removed by LRU	254297

Table 3. Characterization of TPC-C benchmark.

a very high locality of references, this workload might not suite the STAR algorithm proposed in this paper. This in fact is the reason for using this workload as the third test case. This workload would give the worst case behavior of the STAR algorithm.

From the above analysis, we found that there is definitely adequate room for improvement by exploiting single-touch reference behavior of network intensive workloads. This improvement can be obtained in two different ways. One way is by reducing the cycles wasted in keeping an invalid line in the cache; a single-touch is equivalent to an invalid line after the first touch, which might have replaced a line that is to be accessed again. The second way we can improve the cache performance is by reducing the number of inefficient evictions made by LRU. Every time a single-touch line is not evicted, another line which might be referenced again is evicted displaying the inefficiency of the LRU scheme.

3. The STAR Algorithm

The main idea of the STAR algorithm is to identify the single-touch references and evict them as early as possible making room for other cache lines that will probably be used more frequently. In addition, all the advantages of the LRU scheme are still retained by the STAR algorithm.

The STAR replacement policy tries to replace single-touch references from the cache by predicting the lines that are not likely to be referenced again. Unlike LRU and LFU, the STAR scheme uses the hit frequency information along with the age information for making predictions. In case of LFU replacement policy

only the frequency information is used, while LRU uses just the age information. By using a combination of the frequency and age information STAR identifies the single-touch references for making evictions, and is thus more effective and efficient than the LFU or LRU schemes. Furthermore, unlike other related schemes we keep track of only single-touches and one bit is enough to differentiate between a line touched once and more than once. Thus, the hardware and logic requirements of STAR are minimal.

3.1. Algorithm Description

The goal of the proposed scheme is to predict single-touch references, i.e., to detect the cache lines which are accessed once and are evicted before any additional references. There might be cases when these predictions are wrong. Thus, it may be possible to get a good percentage of correct predictions but no performance gain if the improvement gets nullified by the losses due to incorrect predictions. Detection of single-touch references can be done by having a counter for the number of hits to every line in the cache. A single bit denoting one reference or more than one reference would suffice the purpose. However, some lines that are identified as single-touch references may stay dormant for sometime before getting a bunch of hits. Ideally, these lines should not be detected as single touch references. Here we need to impose a certain time-period before considering a line as a single-touch candidate. Otherwise all lines that are fetched very recently become candidates for single-touches, which is undesirable. The waiting period after which a line is considered for a single-touch marking becomes an important tradeoff parameter. The longer we wait, the closer we get towards the LRU scheme. If we do not wait long enough, we end up making a lot of mispredictions about single-touch lines. For deciding this tradeoff factor we introduce a parameter called the maturity age. The term maturity age is defined as the age (in terms of the number of references to the cache) after which a line can be identified as a single-touch line if it has only one reference. In other words, once a line is brought in, it is not immediately checked as a possible candidate for single-touch reference during subsequent references. Once it "ages", it is then checked to detect if it is a single-touch line. Thus a line that is brought in recently is not mistaken as a single-touch line. The maturity age for cache lines is a variable that depends on the application environment and the workload. We have analyzed a wide range of parameter values for the maturity age and some inferences are derived in the next section.

The flow chart for the STAR algorithm is shown in Figure 1. In the chart M refers to the maturity age.

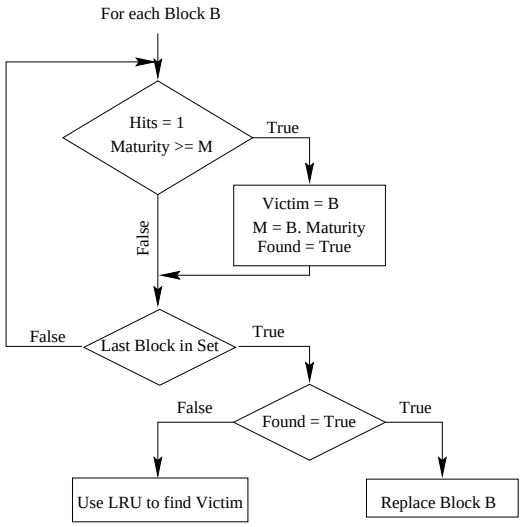

Figure 1. Flow Chart for STAR Algorithm

The algorithm looks for an older single-touch line until it reaches the last line in the set. At this point, "Found" is checked to see if there is at least one single-touch reference found in the set for eviction. If a single-touch reference does not exist, LRU algorithm is used to find a candidate for eviction. Then the oldest block in the cache will be replaced irrespective of the frequency of hits of the lines. Otherwise the oldest single-touch line is replaced by the eviction algorithm.

```
      STAR Replacement Module
maturity = δ
Found Victim = False
for each block in the set
  if no. of hits = 1 and Age  > maturity
    Victim Line = Block Number;
    maturity = Block Age;
    Found Victim = True;
if !(Found Victim )
  Victim Line = Line with Maximum Age;
```

The formal module for STAR algorithm is shown in the above box. The maturity age is initialized to δ, which sets the threshold for the age of lines that are going to be considered for eviction. The value of δ is a configurable variable and can be set on the basis of the workload characteristics. The Found Victim variable is used to determine if a block is found for eviction or not. Two criteria are used to decide whether a line is a candidate for eviction. The first one makes sure that there is only one hit to the line being considered. The second criteria used for choosing lines to be evicted is the age of the lines. It is made sure that among multiple single-touch lines, the one with the maximum age is

evicted. If no lines are found that satisfy this criteria then a line which has the maximum age is evicted. This process will ensure that the performance is always at least as good as the pseudo-LRU scheme.

3.2. Preliminary analysis of STAR

Performance of the STAR algorithm was analyzed over the three different workloads discussed earlier – SPECweb96, SPECweb99, TPC-C. The results obtained for the STAR algorithm from these experiments are discussed in this subsection.

S.No	Behavior	Result
1	Total number of times a line was evicted from the cache	595517
2	Probability of single-touches for every eviction	0.91
3	No. times STAR predictions are correct	105035
4	Avg. no. of cycles gained for prediction	119709
5	No. of times STAR predictions are wrong	3675
6	Avg. frequency of wrongly predicted lines	8.16
7	Avg. age of lines evicted after a miss-prediction	93839

Table 4. Performance of STAR over SPECweb96 benchmark

Table 4 shows data obtained for the SPECweb96 trace. One of the interesting observations from this data is that a high number of the predictions (96%) (rows 3, 5) made by STAR were correct. This means that whenever STAR is removing a line, LRU is also removing that line before it was referenced again. However, LRU takes much longer time before evicting the line. STAR evicts these lines much earlier by predicting that these lines would be removed by LRU after a single-touch only. Another inference is that the STAR replacement policy has an opportunity of removing 18% (rows 3/1) of single-touches, and for every correct prediction, LRU is waiting for a further 119K (row 4) references before removing the line from the cache. This averaged value gives an indication of the time wasted by LRU before evicting a single-touch line. From the above analysis we can see that STAR has a large potential for improvement over LRU in the Internet server environment of which SPECweb96 is a good representation.

Table 5 shows the results obtained from the analysis

S.No	Behavior	SPECweb99	TPC-C
1	Total number of times a line was evicted from the cache	197551	553929
2	No. times STAR predictions are correct	36032	89639
3	Avg. no. of cycles gained for prediction	137412	93488
4	No. of times STAR predictions are wrong	6425	18900
5	Avg. frequency of wrongly predicted lines	6.6	6.59
6	Avg. age of lines evicted after a miss-prediction	226441	71748

Table 5. Performance of STAR over SPECweb99, TPC-C benchmarks

with the SPECweb99 and TPC-C benchmark traces. In case of SPECweb96 which is a favorable workload for the STAR algorithm, we see that STAR improves performance by removing invalid lines faster than the traditional eviction schemes. We have seen that both SPECweb99 and TPC-C have almost similar behavior when tested against LRU. The percentage of single-touches is almost equal for both the workloads. However for SPECweb99 the data here shows that there are a lot lesser number of predictions made, whereas for TPC-C the data shows higher number of predictions, with equally high number of incorrect predictions. The proportion of incorrect predictions for the SPECweb99 trace is very low. This behavior indicates that, even with similar types of workloads, the eviction process is going wrong in both the cases. We can improve the performance of SPECweb99 case by increasing the number of predictions made by STAR, which can be done by decreasing the maturity age. This would increase the number of predictions made. Since the percentage of miss-predictions is very low this change in the maturity age parameter would not increase the miss-predictions by much. In case of TPC-C, we have to increase the maturity age since STAR is making a lot of incorrect predictions while evicting lines. This would allow STAR enough time before detecting the single-touch references. This would decrease the number of cycles wasted for every miss-prediction.

The loss due to miss-predictions done by STAR is also very low compared to the cycles lost due to the miss-predictions of LRU, shown in rows 5 and 3, respectively. Since the number of miss-predictions are relatively low, the average number of cycles lost due to all the miss-predictions (row 6) will not be comparable to the cycles gained by the correct predictions (row 3).

4. Performance Evaluation

In this section, we present our evaluation methodology for STAR and discuss its performance benefits based on results obtained from an extensive set of simulation runs.

4.1. Evaluation Methodology

We modeled our STAR replacement scheme using a cache hierarchy simulation framework, called CacheFlow2. This framework was originally developed in the Microprocessor Research Laboratory at Intel Corporation. Some details of the trace-driven methodology and cache hierarchy models used here are described in [5]. For comparison purposes, a pseudo-LRU [4] scheme was also modeled. The pseudo-LRU scheme is a simplified implementation of the LRU replacement policy. Pseudo-LRU uses a finite number of bits to store the recent references to a block. An optimal LRU scheme was also modeled for our study, which assumes infinite buffer memory for keeping track of all the references made to a cache block present in a set.

The simulation model takes traces in the form of load and store references. Every reference has three fields similar to the DineroIII trace format [15]. These traces were extracted from benchmark traces collected on commercial systems at Intel Corporation. The proposed scheme was evaluated for three different commercial workloads: SPECweb96, SPECweb99 and TPC-C. The simulation results and analyses are covered in the following subsection.

In the experiments, a unified, 32KB, direct-mapped L1 cache was used. The L2 cache configuration was 1MB, 8-way set-associative with 64B line size, unless otherwise specified.

4.2. Results and Analysis

Figure 2 shows the percentage of improvement in the cache miss ratio obtained by using STAR algorithm with respect to the pseudo-LRU scheme for the three types of commercial workloads. The improvement is neither uniform nor constant because of the access behavior of the different workloads. SPECweb96 shows a high degree of improvement especially for smaller

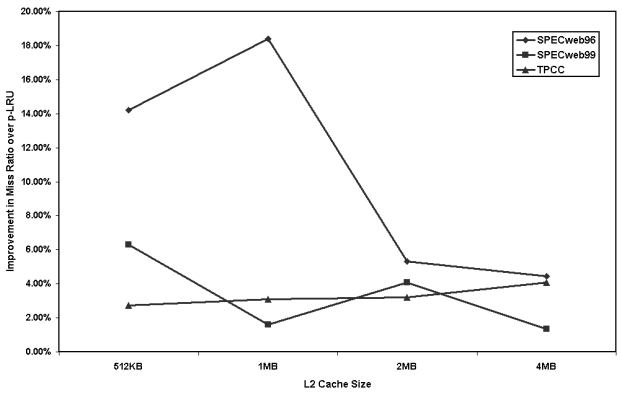

Figure 2. Improvement obtained by STAR for the different benchmarks.

caches. The improvement obtained for the TPC-C workload is not very high but is still noticeable. The small improvement factor in TPC-C is attributed to the low percentage of single touches in the benchmark due to the high locality observed in database queries. Overall it can be inferred that the STAR scheme does not under-perform for any of the workloads and gives a good improvement for workloads with low temporal locality.

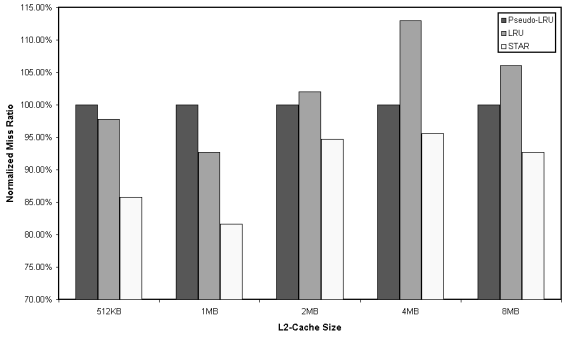

Figure 3. Improvement obtained by STAR for the SPECweb96 benchmark.

Figure 3 shows the improvement obtained over the normalized miss ratio for different cache sizes with the SPECweb96 benchmark trace. Here, the miss ratio is normalized with respect to the miss ratio of pseudo-LRU. Performance improvement of about 15%-20% was obtained for different cache sizes. An important observation is that the scheme is sensitive to the cache size. The improvement in performance increases with

increase in cache size, which follows from the fact that increasing the cache size increases the probability of a line being a single touch. This increase in probability will reduce the miss-predictions made by the STAR scheme, thereby improving the performance.

Similarly further analysis are done to find an optimal value of the maturity age for the SPECweb and TPC-C workloads. It is assumed that the scheme performs better for values close to the degree of associativity. When the maturity age is equal to the degree of associativity, the scheme will have the same effect as the pseudo-LRU algorithm used widely in most cache systems. A high value of maturity age is required to obtain better performance in SPECweb99 and TPC-C because of the low percentage of single-touches in these benchmarks. The high maturity age will allow the STAR scheme more time before making any evictions and hence reduces the number of miss-evictions.

Figure 4 shows the sensitivity of the scheme to the changes in maturity age. The figure shows the improvement obtained by STAR scheme in miss ratio with respect to the pseudo-LRU scheme for a range of maturity ages (0-7). The cache size is 1MB, 8- way associative. The maximum maturity age is 7 at which point the scheme behaves like the pseudo-LRU scheme.

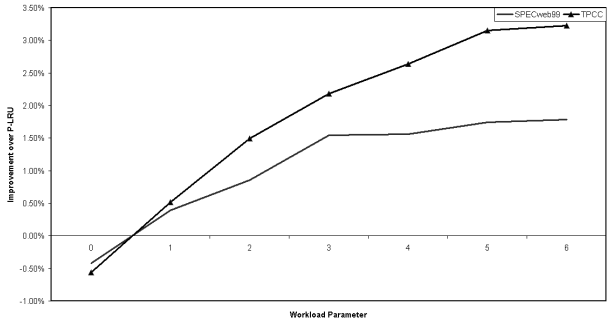

Figure 4. Sensitivity of Maturity age to the SPECweb99 and TPC-C traces.

Figures 4 and 5 indicate that the sensitivity of maturity age is different for different workloads. In case of the SPECweb99 and TPC-C benchmarks a high value is needed for the maturity. But for the SPECweb96 benchmark where the percentage of single-touches is much higher, the lowest possible value for maturity age gives the best performance. We can see that a maturity age value of 0 gives the best improvement for SPECweb96 whereas a value of 6 gives the best performance for the TPC-C and SPECweb99 workloads. Due to the high amount of single-touch requests present in the workload, low maturity age removes as many of the single-touches aspossible.

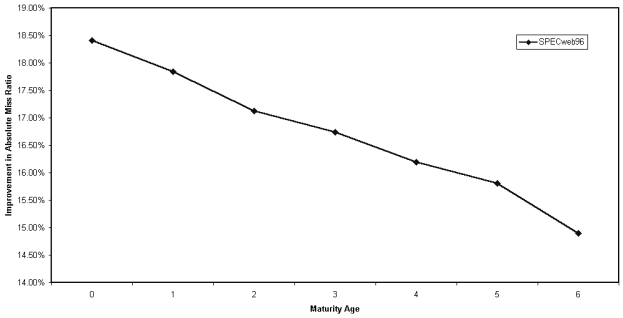

Figure 5. Sensitivity of maturity age to the SPECweb96 benchmark trace.

In case of the TPCC and SPECweb99 benchmark, the locality of reference is relatively high. Therefore there are lesser single-touch accesses. So an increased maturity value gives better performance for these workloads.

In Figure 6, we show the sensitivity of the STAR scheme with respect to the cache associativity. We increase the degree of associativity of the cache, keeping the size constant, from 4-way to 16-way. The graph shows absolute miss ratio normalized with respect to the miss ratio of Pseudo-LRU scheme for the 4-way set associative case. It is observed that the improvement obtained over the different degrees of associativity increases as we increase the associativity. This is due to the increased probability of finding a single-touch line. Thus the performance of the STAR policy is improved because of the reduction in the number of miss-predictions.

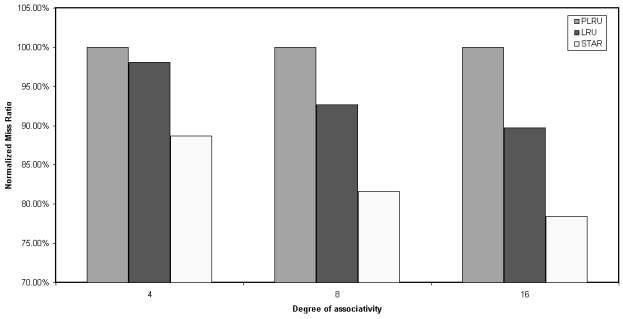

Figure 6. Sensitivity of STAR to Cache Associativity.

The STAR scheme was also evaluated to study the behavior with increasing line size. Figure 7 shows the behavior of STAR for changing line size. We can see that the improvement is almost constant with the increase in line size. This is because changing the line size changes the access behavior. However, it does not change the time for which the line stays in the cache.

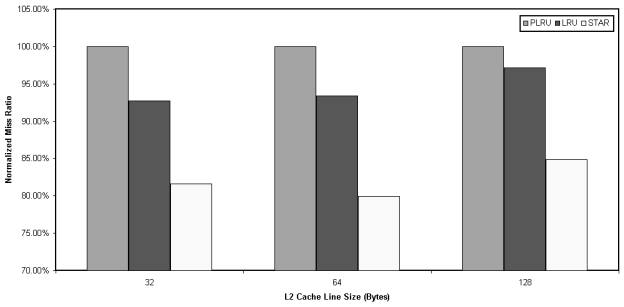

Figure 7. Sensitivity of STAR to Cache Line Size.

Overall, with the selection of an appropriate maturity age, which is based on the workload characteristics and the cache configuration, the STAR algorithm shows significant performance improvement for applications that have a high degree of single-touch references. For other types of workload, although it may not provide any significant performance improvement, it never degrades the performance.

5. Conclusions

In this paper, we have proposed a new cache replacement policy, called STAR, which improves the cache performance by about 20% for workloads with a high degree of network activity. STAR algorithm is very suitable for workloads that have a large proportion of single touch references. It tries to identify and evict these lines from the cache as early as possible, thus reducing the cache pollution. The implementation overhead of the STAR algorithm is very minimal. For applications that do not have much of single-touch references, STAR does not degrade the performance and can be turned off if desired. Overall, the STAR algorithm will be a very suitable candidate for improving cache performance of the Internet servers.

References

[1] "An explanation of the SPECweb96 benchmark," available online on the SPEC website at http://www.specbench.org/osg/web96/webpaper.html

[2] A. Agarwal, J. Hennessey, M. Horowitz, "Column-associative caches : a technique for reducing the miss-rate of direct-mapped caches," International Symposium on Computer Architecture, 1993, pp. 179-190.

[3] L. Barroso, K. Gharachorloo and E. Bugnion, "Memory System Characterization of Commercial Workloads", International Symposium on Computer Architecture, pp. 3-14, June 1998.

[4] J. Hennessey and D. Patterson, Computer Architecture: A Quantitative Approach, Morgan Kaufmann, 1996.

[5] R. Iyer, "Exploring the Cache Design Space for Web Servers", International Parallel and Distributed Processing Symposium, Apr. 2001.

[6] R. Iyer, G. Janakiraman, R. Kumar and L. Bhuyan, "A Trace-driven Analysis of Sharing Behavior in TPC-C", CAECW-2, 1999.

[7] L. John, T. Li and A. Subramanian, "Annex Cache: A Cache Assist to Implement Selective Caching," Journal of Microprocessors and Microsystems, Vol.23, Issue 8-9, pages 537-551, Elsevier Science, Dec.1999.

[8] N.P.Jouppi, "Improving direct-mapped cache permormance by the addition of a small fully associative cache and buffers," Proceedings of the 17th International Symposium on Computer Architecture. 1990 pp 364-373.

[9] K. Kant, R. Iyer, and P. Mohapatra, "Architectural Impact of Secure Socket Layer on Internet Servers," International Conference on Computer Design (ICCD 2000), Sept 2000.

[10] K. Kant and Y. Won, "Server Capacity Planning for Web Traffic Workload," IEEE trans. on knowledge and data engineering, Oct 1999, pp 731-747.

[11] D. Lee, J. Choi, S. H. Noh, S. Lyul Min, Y. Cho and C. Sang Kim, "On the Existence of a Spectrum of Policies that Subsumes the LRU and LFU Policies," Proceedings of the 1999 ACM SIGMETRICS Conference, pp. 134-143, Atlanta, GA, May 1-4, 1999.

[12] P. Mohapatra, H. Thanthry and K. Kant, "Characterization of Bus Transactions for SPECweb96 Benchmark," 2nd Workshop on Workload Characterization (WWC), Oct 1999.

[13] "SPECweb99 Design Document," available online on the SPEC website at http://www.specbench.org/osg/web99/docs/whitepaper.html

[14] Transaction Processing Performance Council, TPC BENCHMARK - C Standard Specification, Revision 3.5, http://www.tpc.org/benchmark_specifications/TPC_C/tpc-c35.pdf. Accessed: Aug. 10, 2000.

[15] Dinero Cache Simulator, http://www.cs.wisc.edu/ larus/warts.html.