A Stack Model Based Replacement Policy for a Non-Volatile Write Cache

Jehan-François Pâris
Department of Computer Science
University of Houston
Houston, TX 77204
+1 713-743-3350
FAX: +1 713-743-3335
paris@cs.uh.edu

Theodore R. Haining[†]
Darrell D. E. Long
Computer Science Department
Jack Baskin School of Engineering
University of California
Santa Cruz, CA 95064
+1 831-459-4458
FAX: +1 831-459-4829
haining@cse.ucsc.edu
darrell@cse.ucsc.edu

Abstract

The use of non-volatile write caches is an effective technique to bridge the performance gap between I/O systems and processor speed. Using such caches provides two benefits: some writes will be avoided because dirty blocks will be overwritten in the cache, and physically contiguous dirty blocks can be grouped into a single I/O operation. We present a new block replacement policy that efficiently expels only blocks which are not likely to be accessed again and coalesces writes to disk. In a series of trace-based simulation experiments, we show that a modestly sized cache managed with our replacement policy can reduce the number of writes to disk by 75 percent and often did better. We also show that our new policy is more effective than block replacement policies that take advantage of either spatial locality or temporal locality, but not both.

1 Introduction

As processors and main memory become faster and cheaper, a pressing need arises to improve the write efficiency of disk drives. Today's disk drives are larger, cheaper, and faster than they were 10 years ago, but their access times have not kept pace. The microprocessors of today have a clock rate 50 times faster than their predecessors of 10 years ago. At the same time, the average seek time of a fast hard disk is at best between one half and one third of its predecessors from the same period. Some technique must be found to bridge the performance gap if I/O systems are to keep pace with processor speed.

[†]Supported in part by the Office of Naval Research under grant N00014-92-J-1807 and by the National Science Foundation under grant PO-10152754.

The effects of this huge write latency can be reduced by delaying writes indefinitely in non-volatile memory before being sent to disk [5]. The longer that data is held in memory, the more likely it will be overwritten or deleted, reducing the necessity for a write to disk. It is also more probable that data in the cache can be grouped together to yield larger, more efficient writes to disk. Therefore, a non-volatile write cache can substantially decrease the number of reads and writes actually serviced by the disk. This substantially reduces the amount of disk latency by eliminating some of the time necessary to reposition the disk head for each write.

A cache replacement policy is required to manage such a cache. Any cache replacement policy must control two things, namely which entities to expel from the cache (the so-called *victims*) and when to expel them. The latter is very important when the processes accessing the storage cannot be delayed. In this case, any write occurring when the cache is full will *stall* and must wait while victims are being cleaned to the disk. If the selection of these victims is not performed carefully, blocks recently written into the cache will be flushed to disk. Once flushed to disk, the cleaned blocks can be reused and overwritten as additional writes are made. If overwritten, the data from victim blocks will not be present in the cache even though temporal locality dictates that they are the most likely to be accessed again.

The cost of writing from the cache to disk is an important factor in selecting victims. Each write operation incurs a penalty due to seek time and rotational delay. Single blocks in the cache are very expensive to reclaim; each requires a write operation. Groups of blocks that can be coalesced into larger contiguous segments are prime candidates because they can be written in a single I/O operation.

We propose a block replacement policy that is segment-based. To simplify the presentation of our policy, we define a segment as a set of contiguous blocks located on the same track. By using a track-based approach, cost is associated with one seek of the disk head and one rotation of a disk platter. This has the advantage of making the cost of writing an individual block inversely proportional to the size of the segment to which it belongs.

Our replacement policy is based on the following three observations:

1. Writes to blocks in the cache exhibit spatial locality: blocks with contiguous disk addresses tend to be accessed together,

2. Writes to blocks in the cache also exhibit temporal locality: the probability that a block in the cache will be accessed again is a decreasing function of the time interval elapsed since it was accessed last, and

3. The curve representing the hit ratio of the cache as a function of its size exhibits a knee: once a given minimum cache size is reached, further increases of its size lead to much smaller increases in the hit ratio (see Figure 1).

Based on these three reasonable assumptions, we develop a model of cache behavior that divides segments into *hot* and *cold* groups. Using this concept of hot and cold groups, we present a new cache replacement algorithm. The model and algorithm are described in §2. We experimentally test the algorithm using a trace-based simulation. Experimental observations and results are found in §3. Section 4 discusses related work. The final section summarizes our major results.

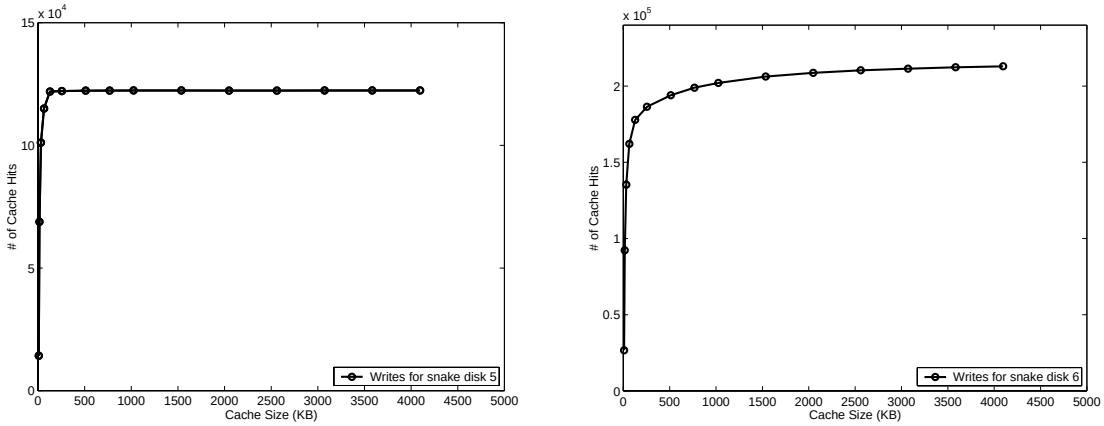


Figure 1: Simulated hit ratios as non-volatile write cache sizes increase for the two disks used in our experiments.

2 Cache Behavior

Given a cache exhibiting the properties of temporal locality, spatial locality, and a diminishing benefit to hit ratio described above, consider the m segments in S residing at any given time in a write cache. Assume that they are sorted in LRU order so that segment S_1 is the most recently referenced segment. Let n_i represent the size of segment S_i expressed in blocks. If accesses to segments in the cache follow the LRU stack model, each segment has a probability p_i of being referenced next. This probability will decrease with the rank of the segment i.e. $i < j$ implies $p_i > p_j$. Note that p_i represents the probability that keeping segment S_i in the cache will avoid a cache miss at the next disk write.

The contribution of each block in segment S_i to the expected benefit of keeping segment S_i in the cache is given by the ratio p_i/n_i. It makes sense to keep all the segments with the highest p_i/n_i ratios in the cache because this strategy makes the most efficient use of cache space. Conversely the segment with the *minimum* p_k/n_k ratio should be expelled. The blocks of that segment have the lowest probability of avoiding a cache miss during the next write.

Using these probabilities, we partition the segments residing in the cache into two groups. The first group contains segments recently written into the cache; these are the most likely to be accessed again in the near future. These *hot* segments should remain in the cache. The second group contains the segments not recently accessed and therefore much less likely to be referenced again. These *cold* segments are all potential victims for the replacement policy.

We identify these two groups of segments based on the knee in the curve representing the hit ratio of the cache as a function of its size. Let s_{knee} be the size in blocks of the cache at the knee and let C_j be the sum of the sizes of the first j segments in the LRU stack ordering of all segments in the stack:

$$C_j = \sum_{i=1}^{j} n_i.$$

The hot segments are the k most recently referenced segments that could fit in a cache of size s_{knee}, that is, all segments S_i such that $i \leq k$ where k is given by:

$$\max\{j \mid j \geq 1 \text{ and } C_j \leq s_{knee}\}.$$

All cold segments are assumed to be good candidates for replacement. We infer from the hit ratio curve that the p_i/n_i ratios for cold segments differ very little from each other. We would expect to see a greater increase in hit rate where the hit rate is nearly constant past the knee otherwise. Therefore the most efficient choice is to clean the largest segment in the cold region. This cleans the most cache blocks for the cost of one disk seek and at most one rotation of the disk platter.

A replacement policy that never expels segments until the cache is full can often cause writes to stall for lack of available space in the cache. This can be avoided by setting an upper threshold on the number of dirty blocks in the cache to force block replacement to begin. This clean space, say 10 percent of the cache, is able to absorb short-term bursts of write activity and prevent stalls. Our cache replacement policy then has two thresholds: one to determine when replacement should begin in order to keep a minimum amount of clean space, and one to determine when it ends based on the location of the knee.

The algorithm to select the segment to be expelled can thus be summarized as follows:

1. Find s_{knee} the size of the cache for x-value of the knee of the hit ratio curve.
2. Order all segments in the cache by the last time they were accessed: segment S_1 is the most recently accessed segment.
3. Compute successive $C_j = \sum_{i=1}^{j} n_i$ for all $C_j \leq s_{knee}$.
4. Let $k = \max\{j \mid j \geq 1 \text{ and } C_j \leq s_{knee}\}$.
5. When 90 percent of the cache is full of dirty pages, expel the segment S_v such that S_v has $\max\{n_i \mid i > k\}$ until $S_v = S_k$.

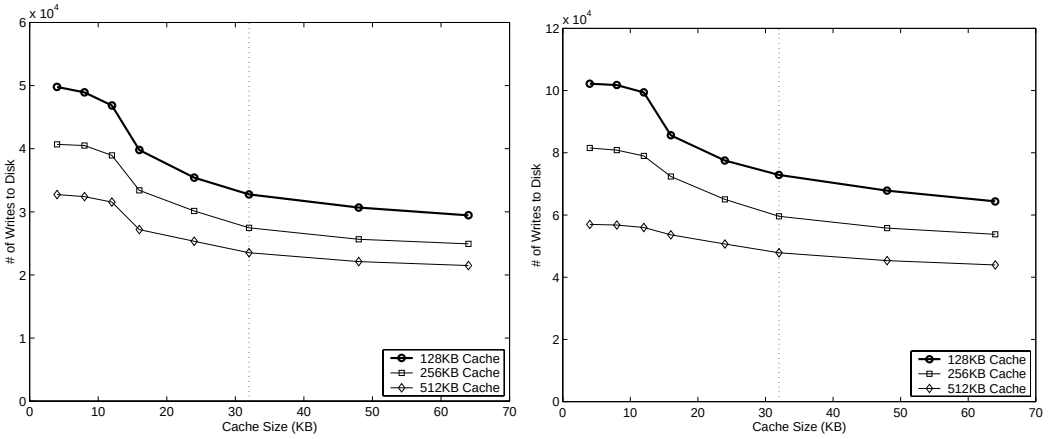
3 Results

We investigated the effects of new cache replacement policy on the utilization of the disk with a specific emphasis on the overall activity of the cache and the disk. We measured the number of times that writes were made to the disk to clean the cache and the size of each write used to clean the cache. We also recorded the number of cache hits and cache misses for the non-volatile write cache.

To run our experiments, we implemented our own model of the HP97560 disk drive. We modeled the components of the I/O subsystem of interest: the disk head, the disk controller, the data bus, and the read and write caches. We based our models on techniques described by Ruemmler and Wilkes [7] and an implementation by Kotz, Toh, and Radhakrishnan [6]. We then used the Snake 5 and Snake 6 file system traces collected by Ruemmler and Wilkes [8] to drive a detailed simulation of the disk subsystem.

To validate our approach of grouping segments into hot and cold regions, we looked at the effect of manually varying the size of the hot region of our cache. If this approach is

correct, the number of writes to disk should be high when the hot region is small because the large hot segments are frequently being replaced. As cache size increases and approaches the knee, the number of writes to disk should decrease rapidly because more hot segments fit into the hot region. The number of writes should then decrease gradually past the knee as all the hot segments are now in the hot region. The results (see Figure 2) showed precisely this type of behavior, with decreases of as much as 25 percent in the number of writes before the knee and as little as 4 percent after it.



(a) Snake disk 5 (b) Snake disk 6

Figure 2: The effects of varying the size of the hot region of the cache for three different cache sizes with snake disks 5 and 6. The dotted line indicates the location of the knee in the hit ratio curve.

We compared the performance of our replacement policy to those that use temporal locality or spatial locality but not both. We implemented two other policies for points of comparison: the *least recently used* (LRU) replacement policy and the *largest segment per track* (LST) replacement policy. The LRU policy uses temporal locality to replace segments and always purges the most stale data first. The LST policy replaces the most cost effective segments based on segment size first.

We expected the LRU and LST policies to perform worse than our policy overall. The LRU policy handles hot segments well, but makes costly small writes to disk. The LST policy makes efficient writes of large segments, but this is only useful when the segments are cold. Since our policy attempts to deal with both hot and cold segments, we expect that it performs comparably (at least) to the best metrics for LRU and LST. A comparison of the results showed this was true for the number of writes made to disk and the number of stalled writes (see Table 1). This comparison also showed that our new policy consistently overwrote data in the cache more often than the LRU or LST policies.

4 Related work

Systems using non-volatile caches have been discussed in several contexts. The Autoraid system developed by Hewlett-Packard used an NVRAM cache with a RAID disk array to

Table 1: A comparison of three metrics for snake disks 5 and 6 for a 128KB cache.

	Snake disk 5			Snake disk 6		
	LRU	LST	New	LRU	LST	New
writes to disk	33538	33895	32758	57059	54057	59592
stalled writes	418	85	97	398	0	0
cache overwrites	79678	76061	79980	143191	141814	145871

produce a high performance, high reliability storage device [9]. There has been considerable interest in non-volatile caches for use in memory based file systems [3] and in monolithic and distributed disk file systems [1], [2]. The advantages of the use of non-volatile caches with on-line transaction processing (OLTP) systems has been investigated [4].

Despite a large interest in non-volatile memory, little comparative work has been done with cache management policies in file system applications [2], [8]. Thresholds were found to significantly improve the performance of non-volatile caches which only take advantage of spatial locality [2]. Work by the authors with such policies showed that temporal locality and large write size can both be used to strongly improve overall write performance [5].

5 Conclusions

Non-volatile disk caches can reduce disk workload more effectively than conventional caches because they allow disk writes to be safely delayed. This approach has two benefits. First, some writes will be avoided because the blocks will be overwritten or deleted while they are still in the cache. Second, the remaining disk writes can be organized more efficiently by allowing contiguous blocks to be written in a single I/O operation.

We have presented a block replacement policy that organizes efficiently disk writes while keeping the blocks that are the most likely to be accessed again in the cache. Our policy partitions the blocks in the cache into two groups: the hot blocks that have been recently referenced and the remaining blocks that are said to be cold. Hot pages are guaranteed to stay in memory. Whenever space must be made in the cache, the policy expels the cold blocks that belong to the largest set of contiguous segments within a single track until enough free space has been made.

Experimental results showed that by using our replacement policy with a correctly tuned, modest sized cache, it is possible to reduce writes to disk by 75 percent on average and the policy frequently did better. The results showed the new policy to be more effective than cache replacement policies which exploited either spatial or temporal locality, but not both. In particular, data was overwritten in the cache more often using our policy than the others, saving writes to disk. The new replacement policy also gave the relative benefits of such policies without their unattractive features. It often provided the least number of writes to disk of any of the policies we used for comparison. At the same time, it often produced no stalled writes when other policies produced hundreds.

Acknowledgements

We are very grateful to John Wilkes and the Hewlett-Packard Company for making their file system traces and the libraries to read them available to us.

References

[1] Mary Baker, Satoshi Asami, Etienne Deprit, John Ousterhout, and Margo Seltzer. Non-volatile memory for fast, reliable file systems. *Operating Systems Review*, 26(Special issue):10–22, Oct 1992.

[2] Prabuddha Biswas, K. K. Ramakrishnan, and Don Towsley. Trace driven analysis of write caching policies for disks. *Performance Evaluation Review*, 21(1):12–23, Jun 1993.

[3] Peter M. Chen, Wee Teck Ng, Subhachandra Chandra, Christopher Aycock, Gurushankar Rajamani, and David Lowell. The Rio file cache: surviving operating system crashes. *SIGPLAN Notices*, 31(9):74–83, Sep 1996.

[4] G. Copeland, T. Keller, R. Krishnamurthy, and M. Smith. The case for safe RAM. In *Proceedings of the 15th International Conference on Very Large Databases*, pages 327–35. Morgan Kaufmann, Aug 1989.

[5] Theodore R. Haining and Darrell D. E. Long. Management policies for non-volatile write caches. In *1999 IEEE International Performance, Computing and Communications Conference*, pages 321–8. IEEE, Feb 1999.

[6] David Kotz, Song Bac Toh, and Sriram Radhakrishnan. A detailed simulation model of the HP 97560 disk drive. Technical Report PCS–TR94–220, Department of Computer Science, Dartmouth College, 1994.

[7] Chris Ruemmler and John Wilkes. An introduction to disk drive modeling. *IEEE Computer*, 27(3):17–28, Mar 1994.

[8] Chris Ruemmler and John Wilkes. Unix disk access patterns. In *USENIX Technical Conference Proceedings*, pages 405–20. USENIX, Winter 1993.

[9] John Wilkes, Richard Golding, Carl Staelin, and Tim Sullivan. The HP AutoRAID hierarchical storage system. *Operating Systems Review*, 29(5):96–108, Dec 1995.