Computer Caching and Human Memory

Avi Press and **Michael Pacer** and **Tom Griffiths**
{avipress, mpacer, tom_griffiths}@berkeley.edu
Department of Psychology
University of California, Berkeley

Abstract

The abstract should be one paragraph, indented 1/8 inch on both sides, in 9 point font with single spacing. The heading **Abstract** should be 10 point, bold, centered, with one line space below it. This one-paragraph abstract section is required only for standard spoken papers and standard posters (i.e., those presentations that will be represented by six page papers in the Proceedings).

Keywords: memory, caching

Introduction

[Insert intro here]

Anderson & Schooler's Model

Anderson & Schooler (1991) designed a model of human memory to explain phenomena they witnessed in human data. There were three observed phenomena they wished to explain:

• Recency - The more recently an item was used, the more likely it is to be used again.

• Practice - The more times an item has been used, the more likely it is to be used again.

• Spacing - The pattern of prior uses of an item is also a predictor of furutre usage probability. This pattern can be described by the amount of time passing in between uses, and how much time passed since this interval occured.

A rational model of memory needs to explain both how a mind learns and forgets. Recency corresponds to Anderson's power law of forgetting; the probability that a person can recall a piece of data decays as a power law with time since the data was seen. Similarly, practice corresponds to learning a piece of data. Anderson's power law of practice showed that recall probabily increases as a power law with the number of times a piece data has been seen.

The reasoning behind spacing may not be intuitive, but it desribes the idea that acesses of an item may follow a pattern. For instance, the if the average space between accesses of a memory has been decreasing with time, it will probably continue to be used more and more frequently, and gets stronger as a result. Inversely, if an item's use is becoming more and more spaced with time, it will most likely be used less with time, and will be less strong of a memory. For the library example, books about taxes are probably accessed far more frequenly during tax season. Once tax season is over, these books can probably be brought back to reserves despite their frequency and recency.

Anderson & Schooler decided to assume that memories have both a strength and a cost to retrieve them. If the memory strength is higher than the cost of recall, it will be remembered and not if otherwise. They noted that in the brain, the amount of time it takes to recall the memory is a good metric for this cost, but suggested that metabolic expenditures are likely be at factors as well.

For human memory, the strength funtion the suggested for a single item was:

$$S_{And} = A\sum_i s(t_i)$$

$$s(t_i) = t_i^{-d_i}$$

$$d_i = max[d_1, b(t_i - t_{i-1})^{-d_1}]$$

where t_i is the time of the ith acesses of the item. d_1 and A are tunable parameters. d_i represents how the strength from the ith presentation decays with time; it can be different for each presentation. A is a scaling factor.

If one could have access to a log of every time they used a particular memory, and hypothetical costs for recalling that memory this model could attempt to predict whether or not that memory could be successfully retrieved. An important aspect of this model is that any item with a strength above its cost threshold can be remembered. This implies that the set of things a person can remember is not explicity limited in size. However, this model does impose an implicit upper bound, due to the recency effect. As more time passes, strength of memories are decaying, and eventually their strengths fall below their cost if they are not used. If one wanted to remember as many things as possible, one will eventually hit the point where they are forgetting things at a similar rate to practicing new ones, as it will necessarily be used less because there are more other items being used.

In order to relate this human model to computer caching we will have to make a slightly different assumption. Instead of being able to recall any memory with strength above a threshold, we will assume memory has a strict maximum capacity, and the items with the highest strengths will be kept. Indeed, caches can be formally described in this exact way. In the next section, we will formally introduce the problem of caching and then go on to mathematically describe the algorithms that manage real caches.

To adapt the Anderson memory model to a caching algorithm, the only adaption is the change in assumption about what gets remembered. Originally, all memories with strength greater than recal cost gets remembered. Here, as with a normal computer cache, we will assume that only the C stongest items are remembered. We are not trying to claim that human memory size is fixed. We only to see how far the

model will get under this assumption that allows us to directly compare the human model to caching algorithms.

Throughout this paper, it is important to keep in mind that we have changed the framing of the model from a cost-benefit model to a rational model trying to maximize the total strength of things it can remember, given a fixed memory size. We would like to argue that the assumptions beneath the two different types of models are actually not radically different. Because the recency effect puts a soft upper bound on memory, having a strict upper limit on memory size is in general not much more restrictive. One intersting difference between the two assumptions is that remembering the strongest N items means that you always can recall those top N items, regardless of what the absolute strengths are.

Problem Formulation

In this section, we will talk about the formal problem of caching. Note that we will abstract from actual implementation of caching in computer hardware, and simply describe the abstracted behavior mathematically.

Let C represent our cache, a small but very fast block of memory in a computer. D represents disk storage, which has far more storage volume but is many orders of magnitude slower. This is the case because in modern computers, memory used for caches is very expensive with respect to storage volume, while disk storage is cheap. The challenge computer scientists face is to maximize the use of the fast memory, and use the slow memory as little as possible.

Now we will formalize this problem. Suppose all the data a person or agent has seen and used is the set D. We will have a finite amount of memory C, a set of items $c_0, c_1, ..., c_k$. We also have H, an access history of each item. This can be described as a set of sets of times each item was accessed, ie. $t_0, t_1, ..., t_n$ for each $c_i \in C$. Whenever an item needs to be accessed, it must be retrieved from memory. If the item is in C, we can retrieve it for a small cost of $cost_c$ (referred to as a cache hit). If the item is not in cache (a cache miss), we must retrieve it from D, for $cost_d$ put it in C and finally recall the item for $cost_c$, totalling $cost_c + cost_d$. If the item is not in D either, ie. it is a previously unseen item, then the item must be both written to D and the placed in C. For our purposes, we will ignore costs of writing new memories, and treat them the same as a cache miss (ie. we wont see any new items).

Our goal then, is to minimize our cost the retrieving items we need from memory. This means that given C and H, we want to minimize our cost of retrieving sequences of items we need. To do this, we must manage C in order to maximize retrevals from C, and minimize retrevals from D, where the cost to retrieve is much higher.

A cache policy will rate every item in its cache according to a strength function. The strength function represents how stong of a memory the item is. Whenever a new item needs to be placed in cache, it will evict the item with the lowest strength, and put the new item in the evicted item's place.

The Algorithms

We will now discuss a set of algorithms that computer science has developed to solve the problem of memory caching. Some of the cache policies encode recency, others frequency, and some try to balance both. We are interested in how these policies compare to the Anderson model with respect to the way in which recency, practice, and spacing effect probabilty of recall. We are also interested in overall miss rate of the Anderson model. How good is this model of human memory at caching for a computer?

Least Recently Used (LRU) Perhaps the most simple of caching algorithms, LRU evicts the item that was used least recently. Because the only information needed to implement LRU is order of uses, it can easly be implemented with just a list. LRU's strength function can be described by

$$S_{LRU} = \frac{1}{t_{current} - t_n}$$

Quite simply, the strength of a memory is proportional to how much time has passed since its last access.

We also used a random replacement algorithm to approximate LRU. For this algorithm, a random item is selected for eviction.

Least Frequently Used (LFU) Another straightforward approach is to evict the item that has been used least. While often a good approach, it is less feasible to implement, as LFU requires an ubounded amount of additional memory to store counts of item uses. While this may or may not necessarily be problem on the for the brain's computational architechture LFU's strength function is

$$S_{LFU} = n$$

since an item's strength grows the more times it is used. LFU can be very effective when items are used often but not necessarily in temporal proximity. A major downside to LFU is that the cache can become littered with items that were once extremely popular, but might never be used again. If human memory worked just like LFU, aging would become far more cumbersome, as heavily enforced memories from childhood would never decay, they would only be overtaken by memories that were practiced even more.

LRU-2 LRU-2 evicts the item with the least recent penultimate use. [Insert example if space allows]. Both LRU and LRU-2 can be described as being part of the LRU-K family of algorithms, where the item with the least recent kth use is evicted (LRU being $K = 1$). LRU does not account for frequency, so LRU-K is a way to approach an LFU approximation without the unbounded additional memory requirement that LFU suffers from. As K increases, behavior becomes closer to LFU, and closer to LRU as K approaches 1. LRU-K's strength function is

$$S_{LRU_2} = \frac{1}{t_{current} - t_{n-k+1}}$$

2Q 2Q is an algorithm that tries to find a balance between accounting for recency and frequency by splitting the cache into two queues (and technichally a third queue, but that will be mentioned later). The first queue is managed as an LRU queue. If a hit occurs in this queue, the item is promoted to the second queue, which is managed as a LFU queue. The LFU queue has a predefined maximum size, so items will be evicted from the LFU queue if it queue is larger than the predefined size, and evicted from the LRU queue otherwise. Thus, the strength of an item in the cache will depend on how large the LFU queue is, and what queue the item is in. If an item is in the queue that will be evicted from, then its strength will be equivalent to the LRU or LFU functions (depending on which queue it is). If the item is not in this queue, its strength is essentially infinite; it will never be evicted until the size of the LFU queue changes.

2Q also keeps track of the items that were evicted from the LFU queue, by storing pointers to their address in memory in a third (negligably small) queue. If one of these evicted items should be used again, the item can be brought back into memory much faster, making the miss far less costly. Because storing a pointer takes a very small amount of space, it can reduce the cost of the LFU misses with a small addition to the memory requirement.

This aspect of 2Q can help the analogy between a caching algorithm and human memory by introcing this third queue. This queue acts as an intermediate between working memory (cache) and long term memory (disk), ie short term memory.

An issue with 2Q is the fact that where LFU queue size has a strict set maximum size, and this maximum size may be hard or impossible to estimate effectively ahead of time.

Adaptive Replacement Cache (ARC) ARC is very similar to 2Q but with added complexity that is generally beneficial. The maximum size of the LFU queue can adapt to the data as it comes in. ARC keeps track of items that have been evicted from each queue. Upon a cache miss of an item that was recently evicted from one of the queues, ARC will make that queue larger, as it got rid of an item that it should have kept. ARC's strength function is identical to that of 2Q.

LRFU LRFU subsumes both LRU and LFU. Each item has a combined recency-frequency count. Intuitively, an item strength continually climbs the more it is used, but that strength decays with time. The exact function and parameter that compute this strength can vary. As suggested in Lee et. al. (2001), we calculated the strength of an item as:

$$S_{LRFU} = \sum_i \frac{1}{2}^{\lambda(t_{now}-t_i)}$$

λ is a tunable parameter. For our purposes, .001 worked well.

Belady's Algorithm Belady's algorithm is the optimal caching algorithm, but it is a hypothetical cache policy because it requires knowing the sequence of item accesses ahead of time. It works simply by evicting the item that will be used in the most distant future. This is not a real caching policy, because we will never know the sequence of accesses ahead of time. However, we will use it in this paper just to compare each algorithm to the optimal caching policy.

Methods

A dataset of headlines from The New York Times from January 1986 to December 1987 was used to test the various caching algorithms. This is one of the data sets used by Anderson & Schooler. Words from these headlines were sequentially cached, each word being its own item. Words were sanitized of punctuation, and made all lowercase. For a range of cache sizes $2^2 <= |C| <= 2^{11}$, we calculated miss/hit rates for each cache policy. We then calculated chances of an item being in cache after 100 day intervals based on its recency, practice, and spacing metrics described in the paper (for a fixed cache size, $|C| = 2^8$). Anderson looked at 100 day intervals and plotted chances of being used on the 101st day based on these metrics. Instead of looking at what actually was used on that next day, we looked at what was in the cache. These two calculations are very related. What is in the cache on the 101st day are the items that the cache estimates will be used that day; how good of an estimate this is is dependent on how good the caching algorithm is.

Results

Recency

A recency effect was shown for all cache policies. The Anderson model actually had an increase in hit probability once an item was in cache an amount of time (proportional to cache size).

Pracice

An effect of practice similar to those seen by Anderson was observed for all cache policies. Effects were, as expected strongest for LFU and weakest for LRU.

Spacing

In short, no cache policies were observed to have comparable spacing effects to the Anderson model. LFU was notably closer to the human model than all other cache management algorithms for this phenomena. Other policies did indeed have effects of spacing on hit rate, but these effects largely decayed with recency.

Overall Miss Rate

The miss to hit ratio dropped at a rate fairly similarly across cache policies. In general, the human model performed worse than all computer caching policies. LFU and LRFU were the best performing algorithms for this data set. LFU performed worse than LRFU for small cache sizes, but then overtook it as cache size approached and exceeded 2^8 items.

Discussion

Conclusion