Combining LRU with Score based object selection

Pavan Kotak
B.Tech Information and Communication Technology
DAIICT
Ghandhinagar, India
201801120@daiict.ac.in

Rahul Khatri
B.Tech Information and Communication Technology
DAIICT
Ghandhinagar, India
201801055@daiict.ac.in

***Abstract*—The subject of caching is of utmost importance in this day and age with the increasing use of devices that are compelled to be more efficient and faster than the previous one. Even in the domain of Internet caching is of utmost importance as content delivery infrastructures are scaling their products to meet the increasing demand. To quickly satisfy users' requests for web content, the Internet Service Providers(ISPs) utilize caching, whereby frequently used files are copied and stored in a place closer to users on the network. In this paper, we'll study the performance of various implementations of a caching algorithm called the Least Recently Used(LRU) and other LRU derived caching strategies via simulations. With the help of these algorithms, the paper attempts to pinpoint how these variations of the same type of algorithm (LRU) behave under different situations.**

I. Introduction

A. Relevance

Many services like video streaming, downloads use web caching at a large scale. All the traffic is being handled via multiple channels of content delivery networks, which negates a single point of failure and is also able to provide substantial traffic savings. These also provide benefits in reducing the overall load and delays as to when caches serve requests to popular data on shorter transport paths to the users. caches are present on every scale, from big data centers to small devices. With more devices being used and for web caching more devices coming online caching is more relevant today than ever before and thus the performance of the caches is under more scrutiny as well.

B. Criteria for consideration

Numerous strategies have been designed to cache, but not all of them are good for this day and age. Then the question arises that how should one pick a caching algorithm. Well, let us look at what's being asked for the algorithm:

- First and foremost the algorithm in question should be efficient. This might seem like a given however this is the most important point, if an algorithm isn't very efficient then it isn't very useful. Now, what does it mean by being efficient? Well, simply put it means that it should have a high hit rate.
- The per-request update process in the cache should be simple and fast. This means that if after every request the cache must make a series of complicated updates then there is a chance that it will slow despite being efficient.
- The traffic and effort for loading objects into the cache should be as small as possible. In distributed caching architectures the cross-traffic between caches should also be minimized.
- The cache should be able to adapt to popularity changes over time and adapt accordingly, This is more relevant for web caching but it's also applicable for every cache. The influence of past requests on the decision about the cached content should decrease over time to enable a response to shifts in popularity and for predictive pre-fetching methods.

II. Algorithms and Implementation

A. Algorithms

While there is no consensus for whether LRU is fit to handle environments like web caching, its relatively good efficiency and lightweight implementation makes it a viable option. And thus, LRU will be used as a benchmark of sorts to compare the rest of the algorithms.

1) Score Gated LRU(SG-LRU): All four of the algorithms tested here are variants of LRU in the sense that they all use the LRU stack implementation with scores being attributed to each object. Now, a SG-LRU admits a new external object to enter a full cache only if it has a higher score than the bottom object. Otherwise, the bottom object is put on top instead of the requested object. Now, the difference between the algorithms lie in how they manage to calculate the scores of an object. The different ways to calculate scores is as follows:

- Geometric Fading: When there is a new request, all the objects present in the cache have their scores reduced by a factor of δ and the requested object has its score increased by an initially predefined value and in the case of it not being in the cache the its score is set to the initially predefined value after the object with the lowest score has been ejected.
- Sliding Window: Here the last W requested objects to the cache have been given a score based on their frequency in the same window. At every request, the object which was requested first in the window has its score reduced, and then the currently requested object has its score increased. In case of ejection, the least frequent used (and in case of multiple objects with same frequency, the least recently used object) is removed.

- LRU + LFU: The concept here is similar to Sliding Window, but every request is considered, until the requested object is not present in the cache, then the scores of all object are reduced. And the new object is added.
- Recency: An adaptation of LRU, which scores an object based on their recency, frequency and reference rate. Score of the object is calculated as following $TimeSinceLastRequest/(Frequency * TimeBetweenLast2Request)$. The object with the highest score is ejected.

Now, there are some restrictions on all these algorithms. Geometric Fading and Recency both require that all objects in the cache are updated on every request. And while, LRU+LFU and Sliding Window don't have an high update effort they do require extra storage. In Sliding Window's case there will be a scalability restriction on the size of the window, and for LRU+LFU there will be a restriction on the total size of the cache itself.

B. Implementation

The cache itself has been implemented with the help of a doubly linked list and for ease of search a hashmap has been created to improve search performance.

III. Results

A. Random requests

While some algorithms can handle pattern-based requests better, the baseline for any algorithm can only be decided based on how good they are in the face of random requests. So, to set as a baseline we have kept 5 traces, each with 10,000 page request and the varying number of unique page requests. For example in Trace 1, the total number of unique page requests is 50. A note for the graphs, while all the graphs were initially meant to have the size of the cache vary from 3 to 50, the difference between different algorithms was not noticeable, and thus now the graphs below have a much smaller window.

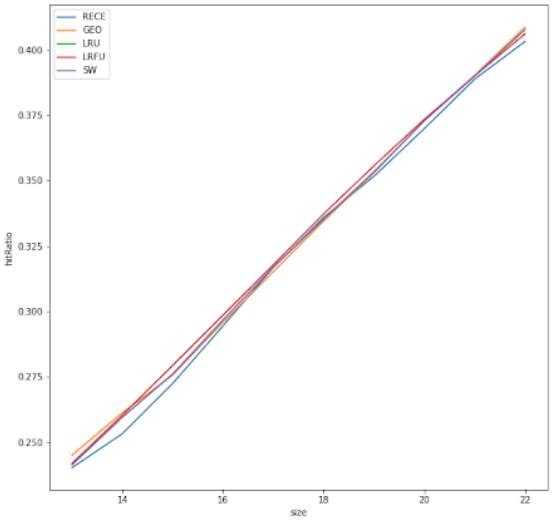

Fig. 1. HitRate vs Cache Size, Trace 1

B. Zipf request pattern

Zipf's law is an empirical law formulated using mathematical statistics that refers to the fact that for many types of data studied in the physical and social sciences, the rank-frequency distribution is an inverse relation. Now, how does this figure in? Well, if we map the popularity of high traffic websites like Wikipedia, then it resembles the theoretical results that one would get from Zipf's law. Zipf's Law governs multiple features on the Internet. Web caching strategies are formulated to account for a Zipf distribution in the number of requests for webpages, too. That is why the use of the Zipf number generator is justified as the practical and everyday use of web cache. And therefore as a way to have results that can be useful in real life scenarios as well, we'll be taking different traces in which the requests follow a Zipf distributions with different distribution parameter(α). The way α influences the pattern is that higher the α, more noticeable the drop off in frequency as compared to the most popular object.

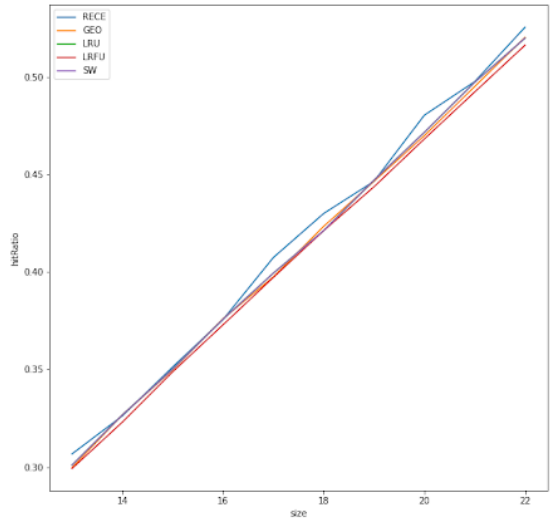

Fig. 2. HitRate vs Cache Size, Trace 3

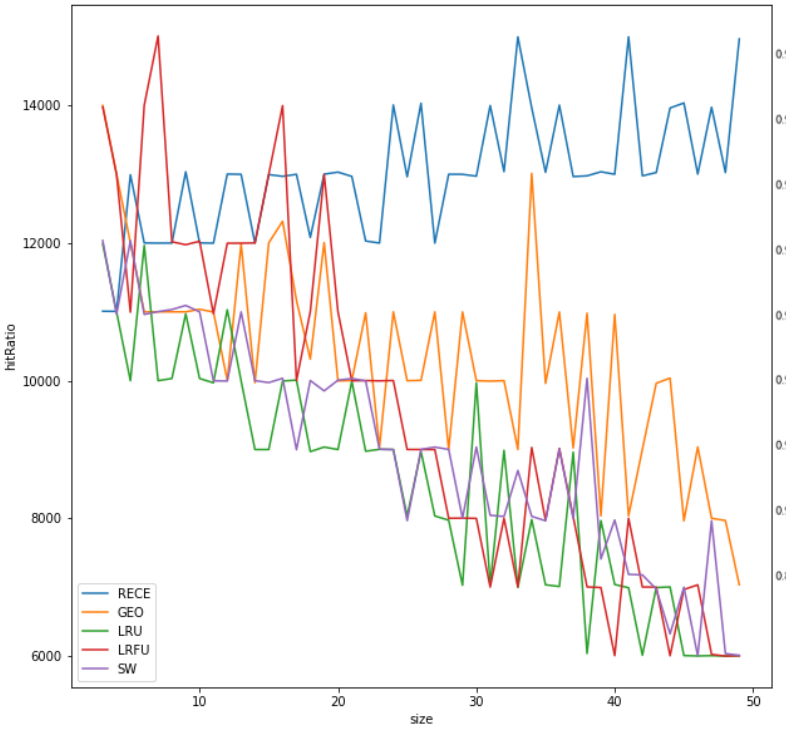

Fig. 3. Time vs Cache Size, Trace 1

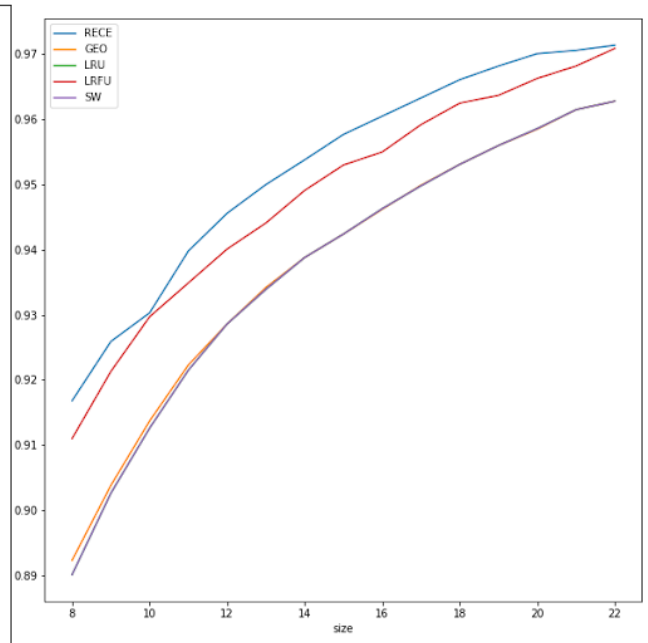

Fig. 5. HitRate vs Cache Size for α = 2

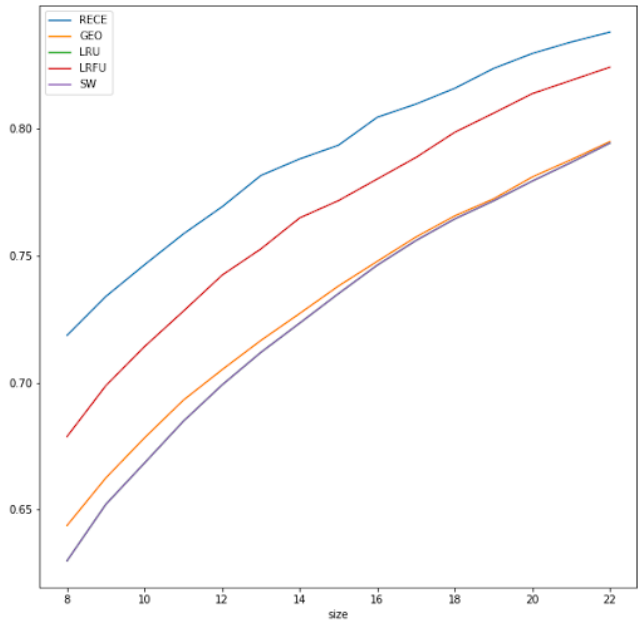

Fig. 4. HitRate vs Cache Size for α = 1.5

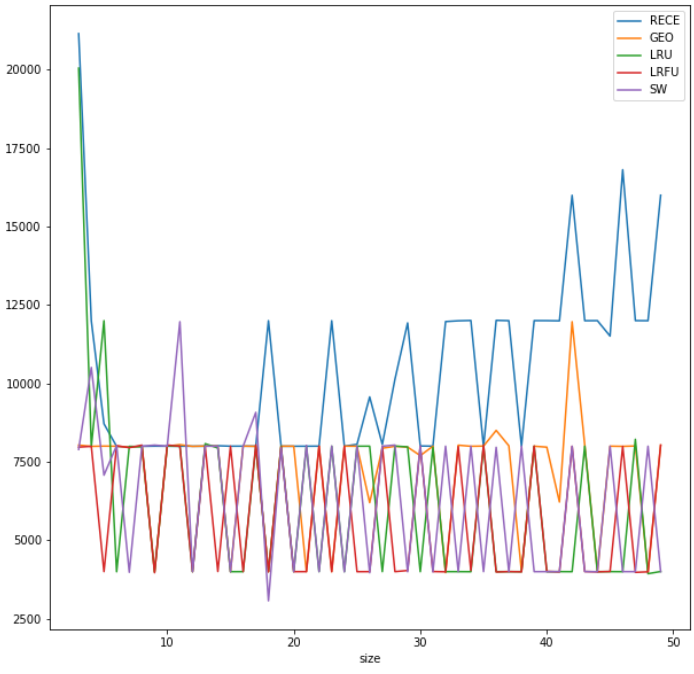

Fig. 6. Time vs Cache Size for α = 1.5

C. Inferences

Now, for the interest of not using 2 pages for all the graphs, I'll briefly summarise the inferences that I was able to get from them.

1) Random Requests:

- Overall, the efficiency of algorithms is pretty similar which to some extent is expected, as the four algorithms expect LRU are based on the bones of LRU.
- Recency is more efficient than the rest albeit not by much, still its a clear winner for these 5 traces.
- As far as time is concerned, here as well, Recency emerges at the top, that is to say that it performs the worst. While across all the Traces and Cache Sizes, there was no particular pattern forming yet, Recency was the worst in terms of time throughout.

2) Zipf Request Pattern:

- There is a similar case here, with all the algorithms performing similarly.
- A difference here is that while Recency still is the most efficient(again not by much), this time there is a constant 2nd present as well, and in the form of LRU+LFU.
- But here as well, Recency performs the worst in terms of time, but surprisingly LRU+LFU didn't stand out much in terms of being good or bad in terms of time.

IV. Conclusion

Caching algorithms can be a tricky thing, try to tune one aspect of it and another falls out of balance, there can be no such thing as a perfect balance. There are trade offs everywhere and all that matters is that what's more important to the system designer. For instance, the request rate is going to be on the lower side then you might prefer something like Recency which preforms much better. If based on past experience, the designer knows that particular type of request can be expected, for example Zipf pattern then they might want to use the LRU+LFU approach. If a simple and lightweight implementation is desired then LRU would do fine. The point here being, that there is no set answer for which algorithm is the best for caching.

References

[1] Hasslinger, G., Ntougias, K., Hasslinger, F. and Hohlfeld, O., 2017. Performance evaluation for new web caching strategies combining LRU with score based object selection. Computer Networks, 125, pp.172-186.

[2] Samiee, K., 2009. A replacement algorithm based on weighting and ranking cache objects. International Journal of Hybrid Information Technology, 2(2), pp.93-104.

[3] Glottometrics 3, 2002,143-150, Zipf's law and the Internet Lada A. Adamic and Bernardo A. Huberman, pp. 144-150