Comparing File Size, File Popularity, and Least-Recently-Used Cache Replacement Policies

Trent Corey
CIS 4930 Probability for Computer Systems & Machine Learning
University of Florida
Gainesville, Florida
trentcorey@ufl.edu

Tyler Metzger
CIS 4930 Probability for Computer Systems & Machine Learning
University of Florida
Gainesville, Florida
tylermetzger@ufl.edu

I. Introduction (*Heading 1*)

This report aims to determine the difference between 3 simulated cache replacement policies. The 3 policies organize the cache based on popularity of the files, size of the files, and the time the file was requested.

The simulator preloads N files into a vector (where N is large). The files each have a unique size drawn from a Pareto distribution with a preset mean and a unique popularity drawn from a separate Pareto distribution. If a user requests a file, we want to simulate the response time. A request has two possibilities. Either the file is in the cache, in which case the cache will transmit the requested file, or the cache does not have the file in which case the request must travel further to the origin server and first cache the file then send it to the user. New requests are generated based on an exponential distribution.

In order to effectively compare cache replacement policies, the simulator has a number of presets that can be modified. Such presets include total number of files, maximum cache size, the Pareto mean for the file size distribution, the Pareto mean for the file popularity distribution, the mean and standard deviation for the random internet delay when the cache requests a file from the origin server, the lambda value for the exponential distribution, the random seed, total time and requests, institutional bandwidth, and the access link bandwidth. This report experiments specifically by adjusting the institutional bandwidth and the total number of files

II. Cache Replacement Policies

The main goal of this experiment is to compare cache replacement policies, and thus we need some sort of metric to do so. This metric will be average simulated response time experienced by the user.

Generally, a file being stored in the cache will reduce the time of that request as it does not need to go through the origin server. The cache will have some storage capacity C where C must be less than the total sum of the file sizes within it. If a new file is to be cached, we replace the files within based on some preset policy.

The three policies implemented organize the cache based on a file's popularity, size, or time of request. The cache is represented as a priority queue which will organize the objects within based on some metric. Depending on the replacement policy, these metrics are either packet size, packet popularity, or packet request time. In this case, we treat a packet as a whole file to simplify the simulation, though it's important to note that in practice files are usually broken up into separate packets that then have to be reorganized.

A replacement policy's goal should be to minimize the amount of time it takes for a user to receive a file from the time they requested. Thus, by ordering the files based on size, the cache hopes to be able to quickly transmit the biggest files to the users and just leave the smaller files to be transmitted from the internet. This is especially effective if the internet bandwidth is the bottleneck of the system though this will likely have the least number of files within the cache since each file will be large. The popularity replacement policy aims to keep the files most likely to be requested in the cache so as to maximize the probability that a given request can be responded to with a cached file. The more imbalanced the file popularity distribution is, the more effective this replacement policy. Finally, ordering the files based on most recently requested aims to take advantage of the thought that a file that was requested recently, is likely going to be requested soon again. This replacement policy is actually used in many instances in real life though in our simulation one disadvantage it will have is that the requested file is truly random. A more sophisticated simulation may increase the probability a file is requested based on when it was last requested, however determining to what extent a file will be requested again would require another study to determine those distributions. Otherwise, whatever value that may be chosen to adjust the popularity will directly affect the effectiveness of the latter replacement policy.

III. Results

The tests for the three policies were ran a total of 27 times, with different combinations of number of total possible files and three different values for institutional bandwidth. All other variables, including cache capacity, Pareto alpha and k values, average delay and delay standard deviation, lambda value, random seed, access link bandwidth, and the total number of requests to be performed, were all held as constant across all simulation trials. The performance of the policies was measured by three different metrics of success: the average number of requests in service, the average file response time—or the average time it took for the file to reach the user after being requested—and the total time simulated for each trial. Out of the 27 trials, the replacement policy based on file size outperformed the other two polices in every metric.

For trials involving 100 total possible files available to be requested, the popularity replacement policy had an average number of requests in service of 18,043, 18,012, and 18,009 for intuitional bandwidth speeds of 100, 500, and 1,000 Mbps respectively. These numbers increased dramatically with the increase of possible total files. When the number of total files was raised to 1,000, the average number of requests in service rose to 140,234, 140,209, and 140,206 for the same institutional bandwidth speeds listed previously. With the number of total files set to 10,000, the average number of requests reported were 715,771, 715,750, and 715,748. Average response times saw similar increases. With the institutional bandwidth set to 1,000 Mbps for 100, 1,000, and 10,000 files, the average response times reported were .341709, 14.8215, and 374.143 respectively.

The replacement policy based on evicting the least recently used file from the cache was the second most effective policy, outperforming the popularity policy by a significant margin. For trials involving 100, 1,000, and 10,000 total possible files available to be requested and an institutional bandwidth speed of 1,000, the least recently used replacement policy had an average number of requests in service of 13,729, 55,733, and 326,624 respectively. The average response time for the same inputs resulted in times of .206185, 2.22406, and 83.4 for file numbers of 100, 1,000 and 10,000. The total simulated time for each of these trials were recorded as 31.9663, 118.567, and 643.779.

Lastly, the replacement policy evicting the smallest file in the cache was the most effective policy implemented. For trials involving 100, 1,000, and 10,000 total possible files available to be requested and an institutional bandwidth speed of 1,000, the file size replacement policy had an average number of requests in service of 13,174, 40,419, and 240,577 respectively. The average response time for the same inputs resulted in times of .184416, 1.61613, and 48.5632 for file numbers of 100, 1,000 and 10,000. The total simulated time for each of these trials were recorded as 30.6544, 88.3666, and 477.293. Below are figures comparing the average response times, average number of requests in service, and total simulated time length of the three replacement policies using inputs of 10,000 total files and an institutional bandwidth speed of 1,000 Mbps:

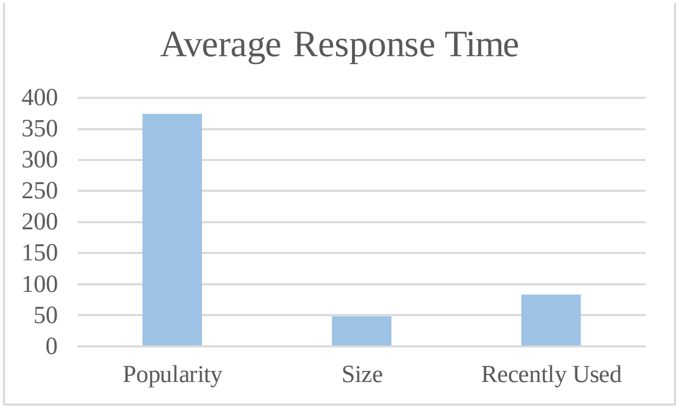

Figure 1 Average Response Times by Replacement Policy

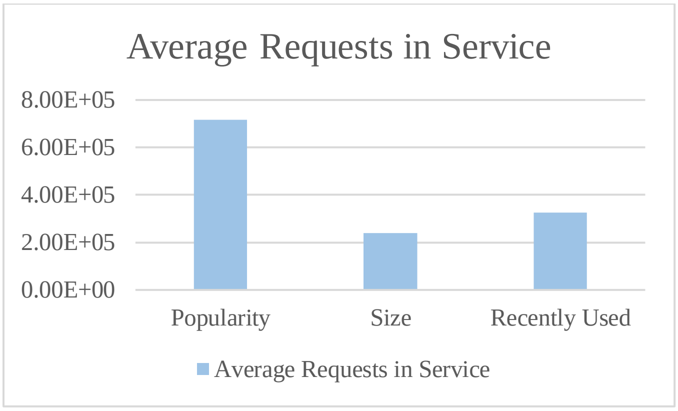

Figure 2 Average Number of Requests in Service by Replacement Policy

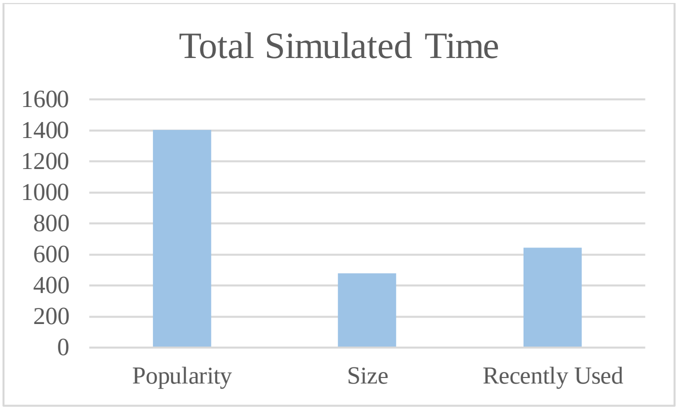

Figure 3 Total Time Simulated by Replacement Policy

IV. Conclusion

The results of our simulations point to a replacement policy based on evicting the smallest file within the cache is easily the most efficient method of cache replacement. Not only does a replacement policy based on these criteria improve the average number of outstanding requests to be completed, but it also completes them in a more timely fashion than the other two methods it was compared against. The replacement method based on file popularity was by far the least efficient method examined, especially in terms of average response time, with the average response time for a large number of files being over seven times higher than that of the size-based replacement policy. It is also clear that the number of possible files can have a dramatic effect on response time and number of outstanding requests. This makes implementing an efficient and timely replacement policy all the more critical when designing a system with a large number of files which can possibly be stored within a cache.