Cache Simulator using probability

*Note: Sub-titles are not captured in Xplore and should not be used

Ganesan Santhanam
Computer and Information Science Department
University of Florida,Herbet Weirthiem College of Engineering
Gainesville,Florida,United States
gsanthanam@ufl.edu

***Abstract*—The primary goal is simulating an institutional cache system using probability distributions and noting the result of varies cache replacement policy with varying distribution parameters and cache parameters. The simulations provide an insight as to how the cache replacement policy is functioning.**

Keywords—Cache Policy, probability based simulations.

I. Introduction

The cache system is about computer of institution requesting for N files that are present on the internet that are requested by users of institution. The requests made by users follow Poisson distribution that is lambda request per second.

Various cache replacement policy is explored like least probabilistically and size based used, least frequently used and least recently used. The request is first generated, then if not present in institutional cache memory, they are queued then when bandwidth allowed, they depart the queue and then is received by the institutional cache system.

A. Origin Server and File Requests

As shown in Fig 1., the institutional network bombards the requests from user that follows Poisson distribution, and then the files that are residing in the origin servers has file probability and file size following pareto distribution

B. Maintaining the Integrity of the Specifications

As shown in Fig 1., the travel to and from the origin server aka the internet is a log normal and the parameters as advised were followed those are mean 500 ms, and standard deviation 400 ms.

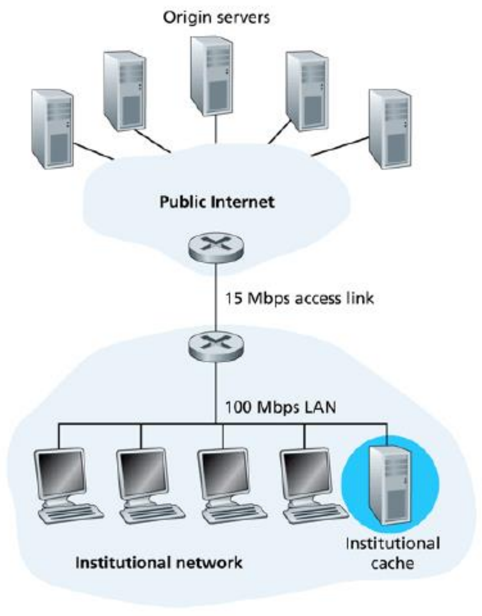

Figure 1: Network Architecture

II. Cache system architecture

Before we dive into the architecture, it's noted that the cache system is simulated 25 times and total time of 1000 units for a simulation and they code is machine independent.

A. Cache System

The Cache System class has class member memory that records the file ID, and the value of specified file ID can be a cache file object, hit count or score based on the cache policy being used.

Least Probabilistically and size based used cache policy (LPSU) uses a score, the score is calculated by the file probability and file size that follows a pareto distribution. And for subsequent hit or future request for the same will lead to score being increased by 0.0625, as this specific value provides more advantage to file that are more likely to be called in future so they would be replaced. The cache policy is further discussed in subsequent sections.

Least Frequently used follows the same implementation of LPSU cache policy but is independent of the size, unlike LPSU, as it only depends on the hits on a specified file.

Least recently used cache policy provides an edge to the file that are recently used, and the cache policy is implemented using linked list.

B. Cache Simulator

The Cache Simulator is responsible for being the communicator classes or medium for all classes to communicate with each other. As this class is a bundle of other classes like cache system that has all cache replacement policies (LPSU, LFU, LRU), file class that generates all files of the origin server, and the request queue that is being generated during the entire simulation distributed as Poisson process.

C. Events

a) New Request Event

The New Request event is implemented with two kinds of constructor, parameterized constructor that accepts instance of the cache simulator. So, the execute method for this type of constructor that has name has Generate New Request, as advertised generates New Request that uses the default constructor and has appropriate execute method that checks the cache if the asked file exists, then create file received instance, else it creates a arrive at queue instance.

b) Arrive at Queue Event

The arrive at queue event is created by its parent event that is new request event, when the request file doesn't exist in the cache memory then the new request creates a file instance of arrive at queue event. The arrive at queue then checks if the request queue to the internet is available then it computes the finish time is the summation of created time of the request and lognormal with parameters mean as 0.5 and sigma as 0.4.

c) Depart from Queue Event

The Depart from Queue is created during the execution of arrive at queue, the depart from queue request yields the file received event, after it computes the finish time by adding the result of division of request file size and bandwidth of access link and created time that is retrieved by getting New Request instance. This finish time can be affected by decreasing the bandwidth, that might affect certain cache policies.

d) File Received Event

The file received event is created if the file already exists in cache memory, in that case its parent request would be new request, but if it's not found in the cache memory then the requested file is retrieved from the origin server via the request queue. The finish time is computed by getting the New Request that is ancestor of current request, to get hold of the created time, that is then added to result of division of file size of institution bandwidth.The received file updates the cache memory as its new file that may or may not be requested in future and file to be replaced is decided by the cache policy put in place for the cache simulation

D. Files

- The file probability and file size are generated using the pareto distribution, and the files class is a later used by the Cache Simulator, so that it could be accessed by the cache policy and event classes.
- The parameters used are 1000 as number of files that is present in origin servers, for file probability pareto distribution uses parameters a = 1 and size = 1000. And, in case of file size the pareto distribution parameters are a = 2 and size = 1000.

E. Parameters

a) Distribution Paameters

As advised the parameters used for distribution are
Pareto Distribution
File Probability – a = 1, size =1000
File Size – a = 1, size = 1000
Lognormal Distribution
Mean = 0.5
Standard Deviation = 0.4
Poisson Distribution
Lam/lambda = 100

b) Cache Paameters

Number of files = 1000
Bandwidth for internet = 15
Global Time for each simulation = 100
Bandwidth for institution = 1000
Cache Capacity = 50
Number of simulations = 20

F. Cache Policy

a) Least Probabilistically and size based used (LPSU)

The LPSU cache policy is very similar to the LFU, but it assigns score to the file that are stored on cache. The score for a new that's added to memory is just the product of its probability and size but with subsequent hit adds 0.0625 to the score that creates an advantage for file with more probability of certain size to a large file that might have less probability but might have same score as the former.

This cache policy is designed to avoid storing a small file of same probability as another file with larger size of same probability should be stored in cache memory as it's the one that might take more time.

The LPSU performs little better than LFU as both assign score in a way that aid file of good probability. But LPSU takes advantage of size dependency to avoid saving smaller files. Ultimately greatly reducing the response time for most request.

b) Least Frequenly used

Like LPSU, this cache policy scores the frequently used file by incrementing the score by a scalar value of 1. Hence when new file is retrieved, the less frequently "hit" or requested file is replaced by the former.

LFU is one of simplest cache policy that can be implemented by using a dictionary and by finding score among the file stored in memory we can decide on the file to be replaced. LFU cache memory uses dictionary to store the score that is number of times it's requested.

c) Least Recently Used

This cache policy practically aims in removing the oldest file in a typical sense not being called for a while. The LRU is the only cache policy that takes advantage of file probability as event moderately requested file can be replaced at times as it might be least recently requested. The policy efficiency is much more than LPSU, as LPSU is custom designed to take advantage of minimizing the response rates of request traffic.

LRU is implemented by using a linked list structure, the implementation is somewhat risky as it never replaces the file node, instead severs the connections by changing the head and tail of the linked list. Hence the cache policy only sees the recent elements of the list that are added in the end as opposed to deleting them off the list. This implementation has a huge impact on the system for iterations of 25 and more as the list will consume the memory. And the Cache File object is created to store details of the previous, next file, size, and score. The score is again used to count the number of times it's requested while it exists in cache, ultimately improves the chances of not being deleted from cache memory.

G. Simulations

a) Heavy Parameters

a) Poisson Distribution – Lam/lambda = 150
b) Cache Capacity = 100
c) Number of simulations = 20
d) Pareto a = 3, size = 1000 for file size

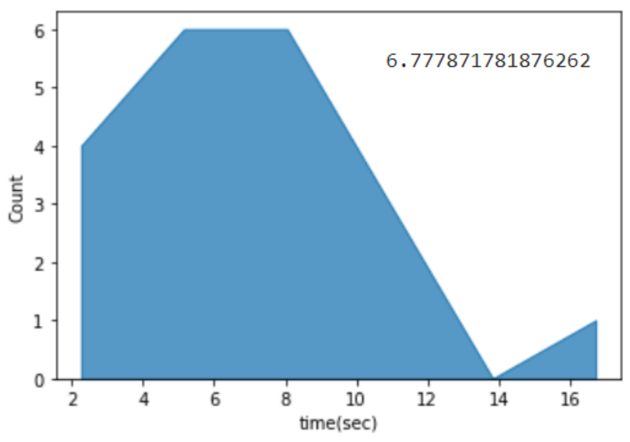

Figure 2: LPSU with heavy parameters

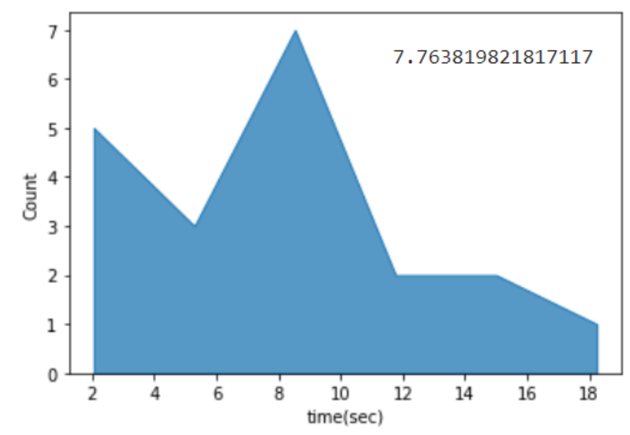

Figure 3: LFU with heavy parameters

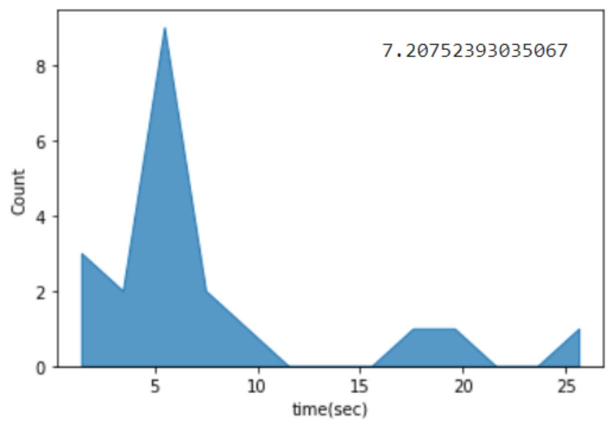

Figure 4: LRU with heavy parameters

b) *Normal Parameter*

a) *Poisson Distribution - Lam/lambda = 100*
b) *Cache Capacity = 50*
c) *Number of Simulations = 20*
d) *Pareto a= 2, size = 1000 for file size.*

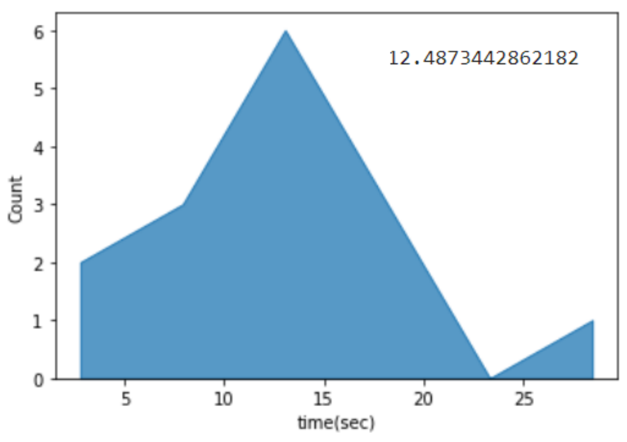

Figure 5: LPSU with Normal Parameters

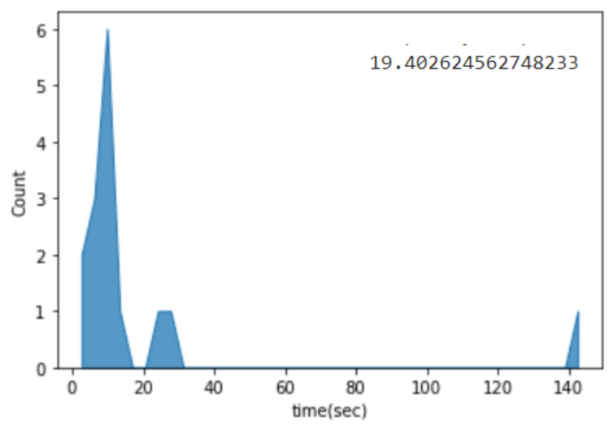

Figure 6:LFU with normal parameters

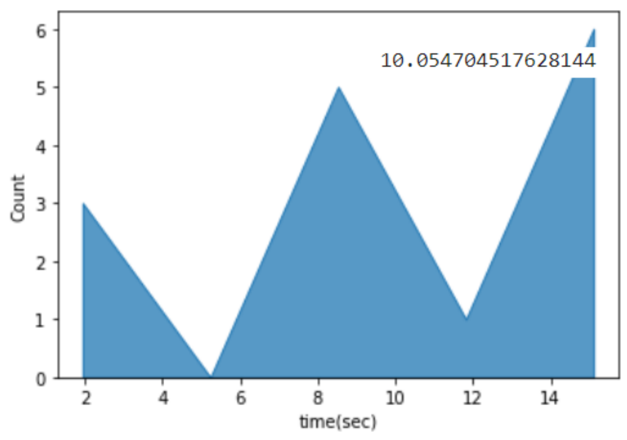

Figure 7: LRU with Normal Parameters

H. *Cache Policy and Parameters Analysis*

By changing some of the parameters like the distribution parameters for file size distribution, cache capacity and Poisson distribution of request generations in the institution we can clearly LRU improves the cache management while LPSU also performs well. But the LFU is clearly seen to perform badly even with larger cache memory. And for heavy parameters the Poisson lam/lambda is increased and the response time for a request goes down as you see in the figure 5,6,7 and indeed naturally the mean is lowers significantly.

This is evident that LPSU and LRU focuses on different aspects of file and their requests they both achieved well with increasing traffic and file size. As LRU deleted the file that least recently requested, whereas LFU also focuses on the same eliminating the file of least probability or least frequently requested. But LPSU is able to overcome that by using the file size as leverage in boosting the chances of larger file among two files of same probability and to avoid storing less frequent large file, we add 0.0625 to increase score for the more probability file or more frequent file.

I. *Conclusion*

The cache simulation has yeilded good results in simultating each cache policy for contained parameters, the assumptions of creating a custom cache policy such as LPSU has really shown the impact of overcoming the worst case of LFU in both times (normal and heavy parameters). The LPSU implemented has reduced the traffic of request response and taken advantage of levearging the expenditure of fetch the smallers file as they might take less time. And with more stricter bandwidth in access link might really show up the LPSU performance than LRU at times. As iterations are increased and size, probability are varied the LRU in the long run perform greatly than LFU. And LPSU although is custom cache policy , it acts as standard as the policy takes advantage of the known, but LRU still performs like LPSU that has been an astonishing finding .The Figure 8-10 illustrates the 75 number of simulations.

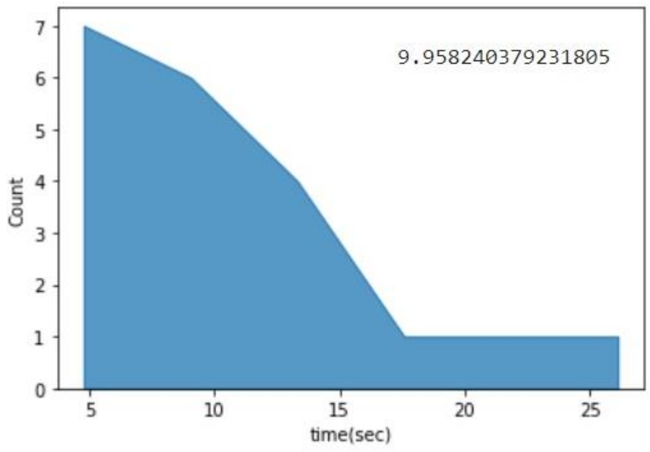

Figure 8: LPSU performance in long run

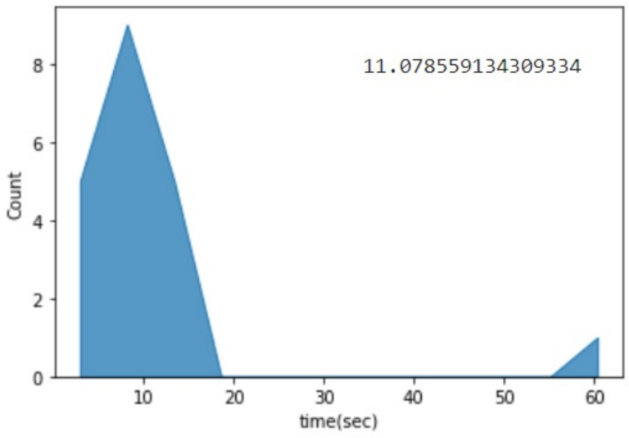

Figure 9: LFU performance in long run

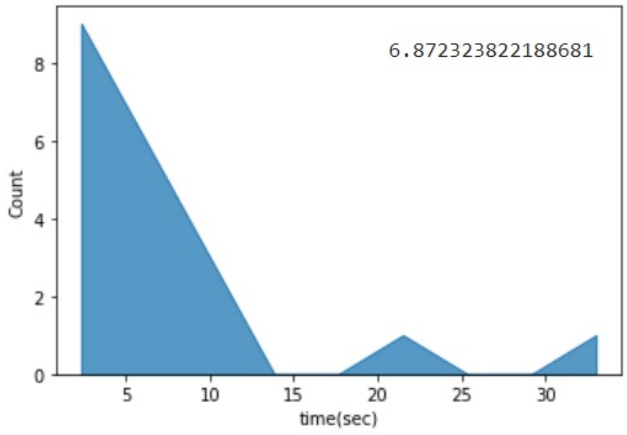

Figure 10: LRU performance in the long run