Network Cache Event-Driven Simulator

Venkata Shiva Sai Mallikarjun Devasani
Electrical and Computer Engineering
University of Florida
Gainesville, FL, USA
vdevasani@ufl.edu

Ram Sundeep Koduru
Electrical and Computer Engineering
University of Florida
Gainesville, FL, USA
ramsundeepkoduru@ufl.edu

***Abstract*— In this project we created an event driven simulator for network cache system and run that simulator for several test scenarios and compare the efficiency of LRU, FIFO and Largest First cache system implementations. For every simulation, a new file request events from user according to poisson process which are generated from a Pareto distribution, as well as in the remote server the popularity size of files are generated from a Pareto distribution.**

***Keywords*— FIFO, LRU, Largest First, Cache replacement policies, probability distributions**

INTRODUCTION

The NCS (Network and Cache Simulator) is a discrete event network and caching simulator based on HTTP traces. It has a lot of parameters to ensure that it works with prior caching and network simulations. It consists of multiple number of files originally residing in far-away Internet servers where a multiple number of users make requests to access those files the major problem we face is that it is difficult to know exactly which file has been requested by which user. Because of this it is challenging to create a cache-replacement policies which works under optimal conditions. Different types of cache-replacement policies have been developed in the past years whose moto is to predict correctly which files need to be accessed by the user. In this project we developed a Network Cache Event Driven simulator in python and compared the efficiency of LRU, FIFO and Largest First cache replacement policies.

The Network cache simulator we wrote generally represents the process that happens when a user requests a file from the remote server. This simulator uses Poisson and Pareto distributions for generating file sizes, file popularities and new file request event times. We used GNU scientific library in implementing these events.

ARCHITECTURE

We developed our simulator using python. The description for the files used are explained in short.

A. Cache

For implementing the FIFO, LRU, and Largest First caches, we wrote Cache as a main class and LRUCache FIFOCache, and Largest First as sub classes. Even though these classes have different internal operations and structures, their type of implementations are the same. These caches are called one after the other using an if else statement which are implemented in sequence.

B. Event

In the event class which is a main classes where all the events that are used in this simulator as sub classes those are New Request Event which handles all the requests that are generated by users, File Received Event This event represents that a file has been received by the user. When processing such an event, the following need to be done calculate the response time associated with that file and record the response time where all the files that are received from server are placed in FIFO queue ArriveAtQueueEvent it checks if the queue, if the queue is not empty, add the file at the end of the FIFO queue if the queue is empty "if the queue is empty, generate a new depart-queue-event and DepartQueueEvent store the new file in the cache if there is enough space. If the cache is full, remove enough files based on your cache replacement policy and store the new file, If the FIFO queue is not empty, generate a new depart-queue-event for the head-of-queue-file.

C. Input Parameters

The list of input parameters that are need to be given by the user are Total_Requests (i.e the number of requests for the files),Num_Files (i.e the number of files in the server) Request_Rate (i.e the rate at which the requests are made), Network_Bandwidth, Access_Link_Bandwidth, Round_Trip, Pareto_Alpha,Cache_Size,Cache_Type.

CACHE REPLACEMENT POLICIES

In this project we generated a Network Cache Simulator that implements LRU, FIFO, and Largest First cache replacement policies.

A. LRU

LRU stands for Least Recently Used, and it signifies that the file that has been accessed the least recently will be removed first. The files are added to the cache through a doubly-linked list, and they are placed at the top of the list when they are added. If a file that already exists in the cache is requested, it will be moved to the top of the list. If the cache fills up and a new file arrives, it will be placed to the front of the linked list, and the file at the end will be deleted. The file size at each file ID is also saved using a map.

B. FIFO

First In First Out (FIFO) is a queueing system. This implies that once the cache is full, it will begin eliminating the oldest files and putting the fresh ones to the back of the queue. To do this, we utilized a deque and map object to store the file ID and size, allowing us to retrieve the file information and determine which file should be erased next.

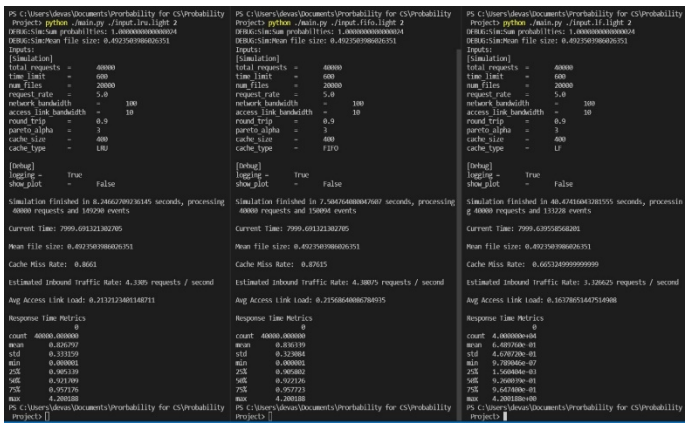
C. LF

LF stands for Largest First, as the name implies among the multiple number of files present in the network the request which is for the large size gets implemented first by keeping other requests on hold. As once the Largest file gets processed it creates ease for the network to process rest of the files with in a small amount of time.

Results

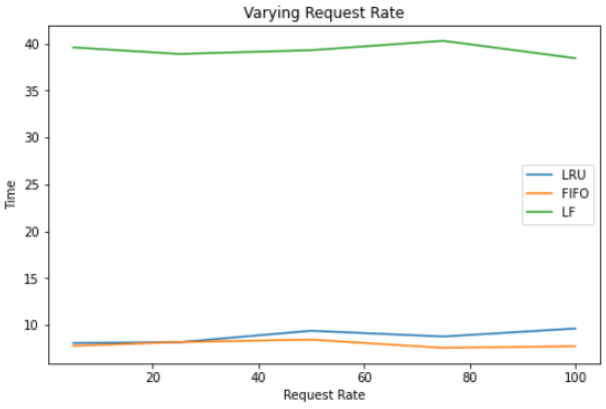
In this testing, we compared the efficiency of LRU,FIFO and Large First cache replacement policies. Also we test the effect of changes in the following parametres.

Firstly with the same input parameters we check the outputs for the three caches we are simulating. We could see the difference in the cache miss rate where LF out performs LRU and FIFO.



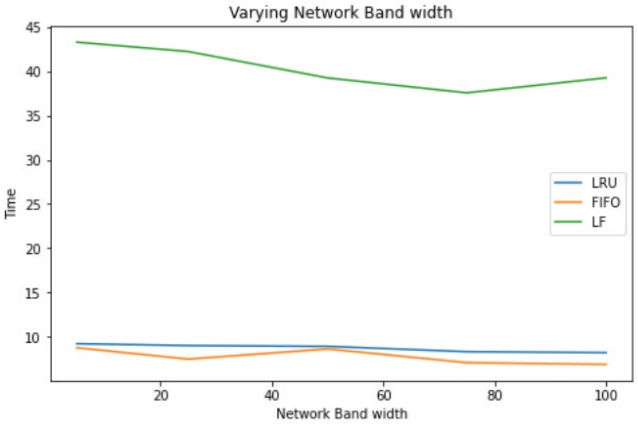
1) Varying Request Rate:

Here we are varying the request rate with equal intervals from 5 to 100 and test the response of LRU, FIFO and LF cache replacement policies. Request rate has a huge impact on the response time explicitly when the size is large. From the below graph we can see that LRU and FIFO are far better than LF. LF is taking 5times the time taken by the LRU and FIFO cache policies.



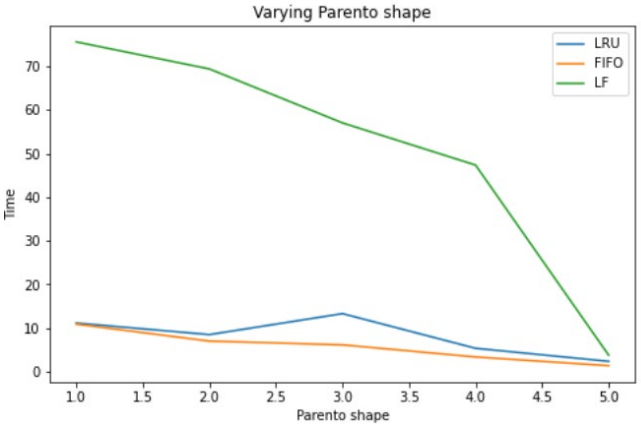
2) Network Band width:

Here we varied the Network Band width with equal intervals from 5 to 100.By varying network band with the response time varies inversely proportional. As we can see from the below graph with increase in the network bandwidth the response time of all the cache replacement policies are decreasing. Even here the performance of LF is far worse when compared to LRU and FIFO



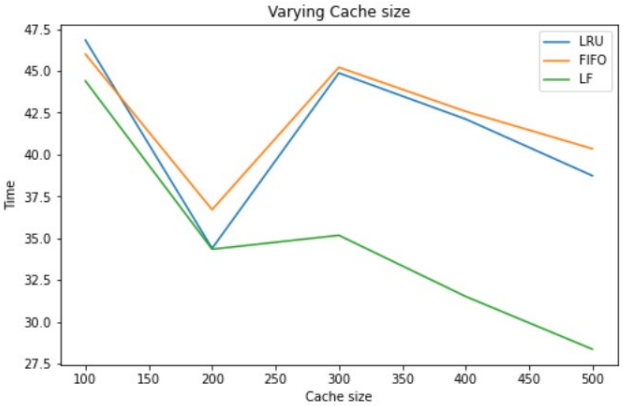
3) Parento shape:

The average reaction time was significantly influenced by the Pareto form. A low Pareto shape value (around 1) indicated that the Pareto distribution will have a bigger tail, with a few very large and very popular files. With a high Pareto shape value, all of the files will be of comparable size and popularity, making it difficult for the cache to predict which files will be requested next. As a result, the simulator does not produce any useful information when the Pareto shape is high, as all cache-replacement policies are equally worthless.



4) Cache Size:

As expected all the three cache replacement policies behave similarly with increase in the cache size. They all show a similar trend with respect to response time. We increased cache size with the intervals from 100 to 500. The response time for each cache replacement policy is shown below. With increase in the cache size the response time decreases for all the three cache replacement policies.



Conclusion

From the results section we can confirm some things that are, firstly system specs such as cache size, number of requests, Network band width, Request rate and parento shape have a huge impact on the response time experienced by the users than the cache replacement policy used in the simulator. Some parametres values prove that a change in these parametres create a huge performance difference in the cache replacement policies. The most important factors that determine the performance of cache replacement policies are Request rate, cache size, parent shape, network bandwidth and the number of requests. From the graph we can see that LRU performs the best followed by FIFO and on the other hand LF was the worst performing network cache replacement policies. LRU cache was to use for a cache out of the algorithms. From these results we can say that, in order to increase the file response time for the users in a network. The prompt solution would be to increase on system specs ie. Cache size, bandwidth etc. Finally to conclude from the three Network cache replacement policies we implemented LRU provides the best resuts.