Event Driven Network Cache Simulator

CIS6930 - Probability for Computer Science
University of Florida

Anmol Lingan Gouda Patil
UFID: 1967 3150
anmol.patil@ufl.edu

Yallamandaiah Nandigam
UFID: 2945 1349
nandigamy@ufl.edu

***Abstract*— By simulating a network cache using an event driven model where in the requested files follow a Pareto distribution, with a poisson file requests distribution along with caching techniques such as LRU, Least Popular and FIFO, we try to understand the use of probability in an ever growing world of internet.**

I. INTRODUCTION

The file requests of any particular user tend to be random, and when we consider these requests as a system only then we can apply some kind of mechanisms such as caching in order to reduce the latency between a file request and response.

As a system we try to define the number of files, the idea of popularity associated with each file, the mean file size distribution and the number of requests emanating from such a system.

Our simulator which is in written in an object oriented paradigm in Java aims to consider these set of parameters to simulate an approximately real time behaviour. Through several years of research there have been implementations of caching like Least Recently Used Cache and First in First Out Cache. In this experiment we extend these techniques by providing a different flavour of a caching technique called Least Popular.

II. SIMULATOR DESCRIPTION

A. EventPriorityQueue.java

This class forms the main backbone of the entire simulation. It provides an implementation of a priority queue wherein different priorities are assigned to different event types. Basic operations include enqueue and dequeue as per the priority.

B. FIFOQueue.java

A simple first in first out queue which is responsible for buffering the requests coming from server directed to client. It has its own bandwidth of sending data. In this simulation we assume no set queue size for the same.

C. Event.java

Event data structure expressed in an object oriented paradigm. Has properties such as key, function type and encapsulates another datastructure called Packet.

D. Packet.java

Packet data structure expressed in an object oriented paradigm that denotes a simple file as a data packet in a network stream. Has properties such as identifier, popularity score, packet size, etc.

E. FileMetadata.java

This class is exclusively used to store metadata information about a packet/file.

F. FileSelection.java

This class provides method implementations to generate files based on the distribution type and parameters passed to it via exposed methods.

G. Node.java

To retrieve data from a cache store, the operations have to be in O(1) time complexity, hence a data structure to be used with a HashMap to provide all necessary details of manipulating and retrieval is a Node.

H. InputReader.java

This class provides method implementations to read input parameters passed to it via exposed methods from a file named input.txt.

I. Driver.java

The driver of the entire simulation.

J. CumulativeMeasurement.java

This class encompassed a simple data structure that facilitates the calculation of some cumulative measurements such as average queuing delay, response time, number of requests and cache hit/miss ratios.

K. LRUCache.java

An implementation of least recently used caching mechanism.

L. LPCache.java

This class provides implementation of a caching mechanism in which least popular files are evicted if more popular files are encountered when performing a replacement. Uses Double priority queue to perform the necessary operation,.

M. FIFOCache.java

An implementation of first in first out caching mechanism.

N. External Libraries for Computation of Distributions

We have used **Java's commons-math** libraries to support our simulator functionalities that require distribution types such as Pareto, Exponential and Lognormal.

III. Cache Replacement Policies

A. LRU cache

A Least Recently Used Cache organizes items in order of use, allowing you to quickly identify which item hasn't been used for the longest amount of time. The frequent items are retained and the rest are evicted upon new item arrival. It has a constant time complexity of O(1) for get operations, update operation is also constant and a space complexity of O(n).

B. Least Popular cache

A Least Popular Cache organizes items in order of popularity, allowing you to quickly identify which item is the most popular one. The most popular items are retained and the rest are evicted upon new high popularity item arrival. It has time complexity of O(1) for get operation, update operation taking O(nlogn) time and a space complexity of O(n).

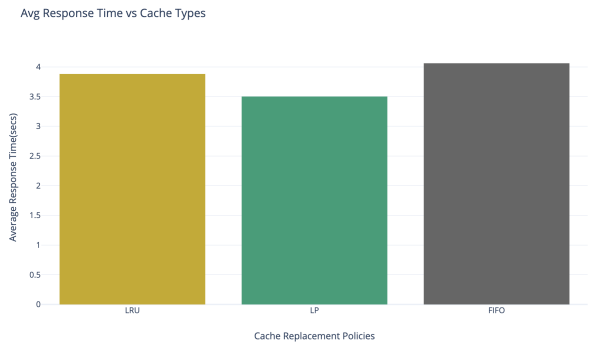
C. FIFO cache

A First In First Out cache is one that uses queuing logic for its backing store, expunging the elements at the front of the queue when a predetermined threshold, in our case the cache capacity, is exceeded.It has a constant time complexity for get operation, update operation is also constant time and a space complexity of O(n).

IV. Results

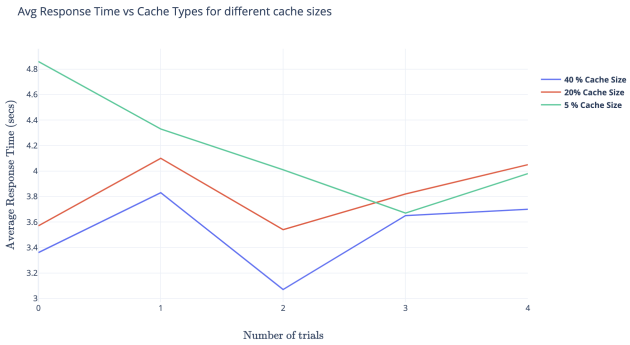
We have used our simulator to test the efficiancies of 3 different caching mechanisms namely LRU, Least Popular and FIFO. These mechanisms are tested against a variation in set of parameters namely average response time, pareto alpha for file popularity, cache size, round trip time of a request, FIFO bandwidth and total simulation time.

A. Average Response Time



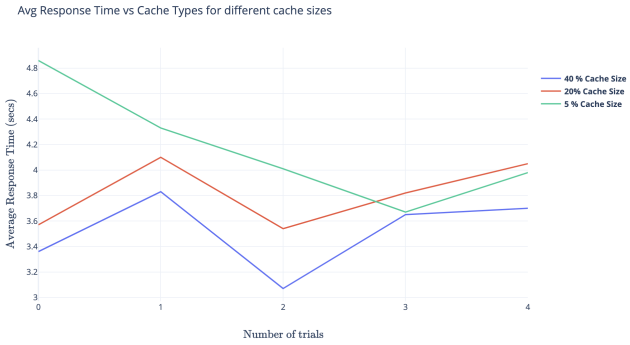
After several simulations the least popular cache replacement performed better than LRU and FIFO. The next best policy was LRU and FIFO performing the worst in all. Least popular had an average of 3.5 seconds of average response time for a mean file size of 1 MB, with file popularity alpha 1.0, file size alpha 2.0, cache size of 50 MB(approximately 5% of total file size) and a simulation time of 30 simulator minutes. Whereas as LRU and FIFO had 3.8 seconds and 4 seconds average response time respectively.

B. Cache Size



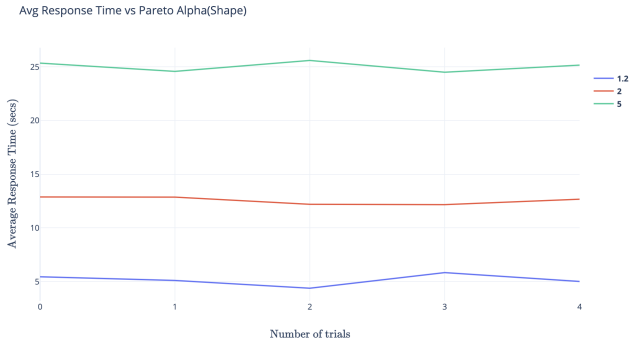
We simulated the experiment by varying the cache sizes. For a cache size of 5% of total file size there was a high average response time as compared to 20% and 40%. This clearly states that an increase in the cache size certain helps decrease the average response time by a significant factor.

C. Simulation Time



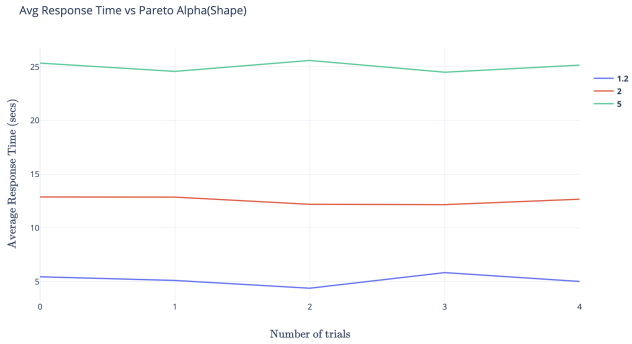
When the experiment was run for 4 different time duration. The results were quite interesting. The larger the simulation time the higher the average response time, see for example when the simulation was run for 1 simulation hour the average response time was in 14 to 17 seconds magnitude. While the average response time for simulation minutes less than 30 somewhat remained same.

D. Pareto α



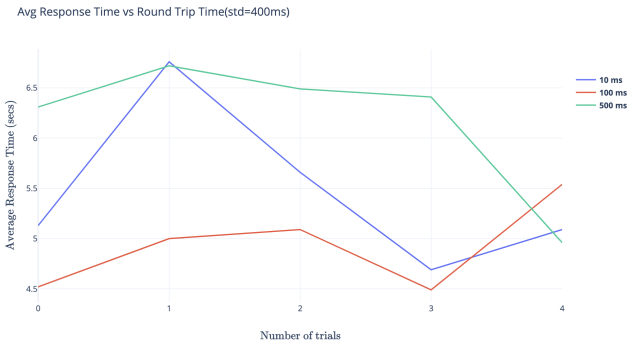
A pareto α determines the shape of a distribution. In our case a pareto α close to 1.0 means that there a few very popular files and few least popular files, making less frequent cache replacement. Whereas a higher pareto α such as 2.0 and 5.0 meant there were a lot of similar popularity files and that resulted in lot of cache misses, ultimately resulting in higher average response times.

E. FIFO Bandwidth



FIFO bandwidth decides how quickly a packet is dequeued from the FIFO queue. The higher the bandwidth the quicker and larger files delivered. This experiment with FIFO bandwidth was a very interesting one in our simulation. We noticed that increasing the FIFO bandwidth actually resulted in a rapid increase in average response time. This actually was happening because of the increased cache misses. The cache miss ratio was 0.74 at 150Mbps compared to 0.46 at 15 Mbps. This can be attributed to the fact that with an increase in FIFO bandwidth a lot of files end up at the cache forcing quicker replacement and hence leading to unfavourable conditions such as cache misses for newer requests.

F. Round trip time / Lognormal mean



Although we tried to find a correlation between the log normal mean of server response time and average response time, it was quite inconclusive in our simulation. Any extreme values would make the experiment give out unrealistic output and hence were no selected to be in the scope of our simulation.

V. Conclusion

To recapitulate out of the three cache replacement policies the Least Popular fared the best followed by LRU and FIFO. Increasing the cache sizes was directly proportionate to the inverse of average response times. And a low pareto α works best for file sizes and popularity.

The simulation results throw a good amount of light on how to design an efficient network cache system.We noticed that several aspects of the system such as file size and popularity distributions, cache size, queue and cache bandwidths, cache replacement policies and simulation time all have a significant say in determining the overall system performance.

When some aspects of the system such as file distribution and request rates cannot be compromised it is better to focus on increasing the subsystem capabilities such as cache bandwidth, cache size and server response times. Perhaps, when there is a limitation even on the capabilities of individual modules in the system, we need to focus on bettering our cache replacement policies and algorithms that support them.