PROJECT DESCRIPTION - VERSION 1

1. Description

In the project you will write an event-driven simulator for a network cache system. You will then run the simulator in several test scenarios, and write a short report to describe your test results.

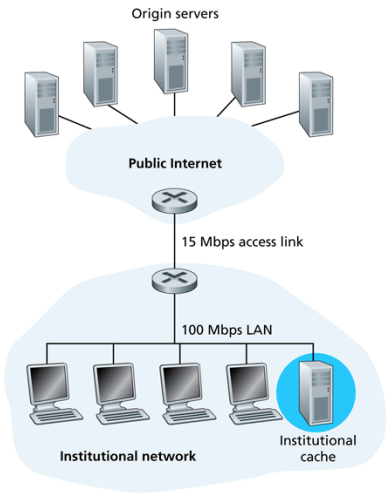

Figure 1. Network Content Cache

Consider the network system in Fig. 1. It shows an institutional network connected to the wider Internet via an access link. A simplified simulation model is shown in Fig. 2.

- There are N files (e.g., web page objects), originally residing in far-away Internet (origin) servers. N is large, e.g., $N = 10000$.
- The institutional users make requests for the files.
- For the intended objective, there is no need to distinguish the origin servers. You can assume there is a single origin server. Similarly, you can assume there is a single user (see Fig. 2).
- The user makes file requests according to a Poisson process with rate λ requests per second. Each request is for file i with probability p_i.
- We care about the response time, which is the duration from the time of the request till the time the file is received by the user.
- There is a cache server in the institution network. A user request goes to the cache first. If the cache contains the requested file, it transmits the file to the user.

- If the cache does not have the requested file, the request goes to the origin server in the Internet. After the origin server replies with the requested file, the file goes to the cache first and is cached, and then goes to the user.
- File i has a size S_i, which is a sample drawn from a Pareto distribution (heavy tail), F_S, with mean μ (e.g., $\mu = 1$ MB).
- The probability p_i reflects the popularity of file i. The p_i's are generated as follows. For $i = 1, \ldots, N$, draw q_i independently from another Pareto distribution, F_p. Let $p_i = q_i / \sum_j q_j$, so that each p_i is a probability.
- Suppose the resource limitation is at the access link in the in-bound direction. Its bandwidth is denoted R_a, which is a constant (e.g., $R_a = 15$ Mbps). We do not need to model the out-bound bandwidth, since the small request messages do not consume much bandwidth.
- The institution network bandwidth is R_c everywhere, which is a constant (e.g., $R_c = 100$ or 1000 Mbps).
- In the in-bound direction at the access link, there is a first-in-first-out (FIFO) queue with infinite capacity. The returned files enter the FIFO queue and will be transmitted by the access link in order.
- Assume the propagation time within the institution network is 0.
- The round-trip propagation time between the institution network and the origin server is D. For simplicity, assume D is a constant (e.g., $D = 400$ ms).
- If a file i is served from the cache, the response time includes only the transmission time from the cache, which is equal to S_i/R_c.
- If a file i is not in the cache at the time of a request, the following count towards the response time: the round-trip propagation time in the Internet D, the queueing delay at the FIFO queue, the transmission time at the access link, and the transmission time from the cache.

  Note that after file i is completely transmitted from the access link, it is immediately cached at the cache server (which may involve replacing some cached files). After an additional time S_i/R_c, it is received by the user.

Project Objective: Evaluate Cache Replacement Policies

- The cache storage capacity is C, with $C < \sum_{i=1}^{N} S_i$.
- Replacement Policies: Oldest First/Least-Popular First; Largest First; some combinations of the above. You are encouraged to do some quick research on other replacement policies and evaluate additional policies. You can even design and evaluate your own replacement policy.
- Performance metric: the average response time experienced by the user.

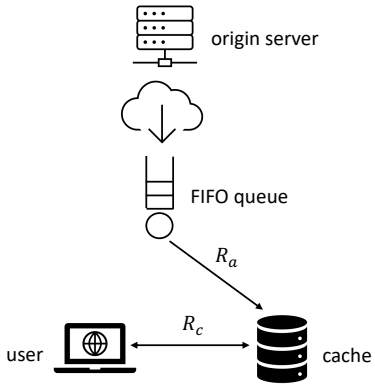

FIGURE 2. Simulation Model

Deliverable:

- Your code.
- A readme file containing: how to compile your code; how to modify the input parameters; how to run your code; how to read the output.
- A sample input file (if that is your input method) or a description in the readme file about how to set the input parameters. This allows the TA to test your code.
  Note: Your typical simulation run may take a long time, e.g., hours. However, the TA will not be able to wait for that long. Please provide a set of input parameters that takes no more than several minutes for your code to finish its execution.
- You should simulate at least two cache replacement policies. There will be bonus points for additional policies, especially if it is your own design.
- A short report (2-3 pages) in IEEE paper format (double column, 10 point font size) containing: A section describing any unique aspects about your simulator (if any); a section describing the cache replacement policies you simulated; a section containing the results of the simulation, where you use both text and figures to show your results; and a conclusion section summarizing your findings.

  You can find templates for IEEE journals or conferences, for example, here:

  https://www.ieee.org/conferences/publishing/templates.html

  If you have time to learn, it is an opportunity for you to use Latex for editing the report. Latex is great for inputing mathematical symbols and equations. Otherwise, you can use MS Office or similar editors. For generating plots on Linux, gnuplot is great.

2. Hints

Explore the Parameter Space: Although the project appears to be confined, the parameter space is quite large. It is expected that you will need to explore the parameter space. Your simulation results will depend on the parameters you use. You should show how the performance depends on the parameters for each cache replacement policy. There is not enough space to report the results for all parameters. Use your intuition about what may be realistic or interesting parameters to consider. Here are some hints.

- Think about the in-bound traffic load in response to the out-bound requests. The load is with respect to the capacity of the access link. Think about how to estimate the in-bound traffic rate at the access link based on the request rate λ (and cache miss ratio). In your simulation, you should try the light load, medium load and heavy load scenarios.
- The results may be sensitive to various constants, such as the mean file size, the ratio between R_c and R_a, and the storage capacity C relative to the total size of all the files. You should explore some of these.
- The results will be sensitive to the parameters of the distributions F_S and F_p, particularly the power. A generic Pareto distribution has the form

  $$F(x) = 1 - \left(\frac{k}{x}\right)^{\alpha}, \quad x \in [k, \infty).$$

  For $\alpha \leq 1$, the mean does not exist. For $\alpha > 1$, the mean is equal to $\frac{\alpha k}{\alpha - 1}$. Therefore, you should consider $\alpha > 1$. The smaller α is, the more heavy tailed the distribution is. Please explore a range of values of α, especially for the file size distribution F_S. For each chosen α, you want to make sure the mean file size is a reasonable value for an intended application (e.g., around 1 MB for web objects). Then, you decide the remaining parameter k.
  Side Note: Of course, a more convincing procedure is to collect real data to generate the file size distribution for the actual application, and use that distribution to drive your simulation. However, that procedure also has a drawback, which is that it lacks generalizability, or the ability to explore alternative scenarios. Our model-based study can answer 'what-if the file size distribution changes to something else'.
- Even under a fixed set of parameters, your results may be sensitive to the random samples of S_i and p_i that you draw from F_S and F_p. You will need to experiment with different sets of samples and find the average over the samples.

About Event Driven Simulation: In event-driven simulation, events are dynamically generated as the simulation proceeds. Each event has a time

of occurrence. Your simulator processes the events in the order of their occurrence times. When processing an event, one or more future events may be generated and scheduled according to their times.

The main thing needed is an efficient data structure that maintains all the outstanding events and allows you to have access to the next event based on the occurrence time. Such a data structure is known as a priority queue. Your simulator's main loop is to dequeue the next event from the priority queue (which must be the event with the smallest time among all the outstanding events), advance global time to the time of the event just dequeued, and process the event, until (i) the priority queue is empty, (ii) the total number of events reaches a limit, (iii) the global time reaches a limit, or (iv) some other stopping criterion are met.

As an illustration, the priority queue may be as simple as a sorted list that keeps the outstanding events in increasing order of their times. After processing an event on hand, your code removes the first event from the list and processes it. If during the processing, a new event needs to be scheduled to occur at a future time, you insert the new event on the list at the right place based on the time. Of course, a sorted list may not be fast enough, especially for insertion of new events. For large-scale simulation, the priority queue needs to be efficient in terms of the number of operations needed to dequeue an event or insert a new event. Generally, some types of tree structure is used for the priority queue. My sample code in C uses a splay tree. You can do some online research and read about priority queues, splay trees, and other possible implementations of a priority queue. For major languages, there should be either library functions or user implementations of typical priority queues that you can use.

Events for Our Network Cache Study: This part is intended to get you started. It may not cover all possible situations.

You keep track the global time with a variable, say, current-time. When you dequeue an event from the priority queue, you advance current-time to the event time. Then, you check the event type and process it accordingly. To illustrate, you may need the following event types.

- new-request-event: This event corresponds to a new user request for a file. When processing such an event, the following need to be done.
  - check the cache for the file
  - if there is a cached copy, generate a new file-received-event, with the event-time = current-time + S_i/R_c.
  - if there isn't a cached copy, generate a new arrive-at-queue-event, with the event-time = current-time + D.
  - generate another new-request-event with the event-time = current-time + X, where X is a sample drawn from the exponential distribution with parameter λ (i.e., mean $1/\lambda$).

- 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 (a data sample has been collected).
- arrive-at-queue-event: This event corresponds to a file arriving at the in-bound FIFO queue. When processing such an event, the following need to be done.
  -if the queue is empty, generate a new depart-queue-event, with the event-time = current-time + S_i/R_a.
  -if the queue is not empty, add the file (i.e., the info about the file) at the end of the FIFO queue.
- depart-queue-event: This event represents that the access link has finished the transmission of a file (the identity of which is recorded by the event). When processing such an event, the following need to be done.
  - 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.
  - generate a new file-received-event, with the event-time = current-time + S_i/R_c.
  - If the FIFO queue is not empty, generate a new depart-queue-event for the head-of-queue file, say j, with the event-time = current-time + S_j/R_a.

About Poisson Process: The sequence of file requests follows a Poisson process with rate λ requests per second. For this project, we only need a few facts about the Poisson process. First, it is a point process in the sense that at any time t, there can be at most one point/event happening at that time. The event in a Poisson process should not be confused with the events in our simulator. For the file request process, an event is a file request.

Now, we can talk about the inter-arrival time of the points/events, which is the interval between two consecutive requests. Let X_1 be the time of the first event/request. For $n \geq 2$, let X_n be the interval between the $n-1$-th and the n-th events. A Poisson process can be defined by the sequence of inter-arrival times, i.e., the random variables $X_1, X_2, \ldots$: The X_i's are IID exponential random variables with parameter λ. The mean of each X_i is $1/\lambda$.

The name 'Poisson process' comes from the following. For $t \geq 0$, let $N(t)$ be the number of events that occurred on the interval $[0, t]$. Then, for each t, $N(t)$ is a Poisson random variable with mean λt. This is why λ is a rate. The collection of all such random variables, $\{N(t)\}_{t \geq 0}$ is a Poisson process.

Programming Language: You can use any programming language. I will give sample code for the priority queue in C. If you use any other language, please find the code that implements a priority queue, or write your own code for it.

Random Number Generation: For major programming languages, there should be library functions or packages that you can use for drawing random samples from typical distribution functions. For instance, in C/C++, you can use the GNU Scientific Library (GSL).

Relevance of the Model: The model is not realistic enough with respect to how things work on the Internet. Some of the simplifications or omissions are obvious, such as the consolidation of the origin servers into a single one and the lack of details in the network paths. The absence of reactive control of the traffic source is a deeper issue. In reality, file transfer applications typically use TCP, which allows the traffic sources to automatically adapt to network congestion. A more realistic simulator will need to simulate the behavior of TCP, which will require packet-level simulation as apposed to file-level simulation. Having said that, it is often valuable to study a simplified, abstract model. For instance, doing so may provide a quick way for us to gain insights or understanding about the real system under study.