Homework 4

Software Systems **Allen B. Downey**
Fall 2006

Due Monday 23 October.

The purpose of this assignment is to use simulation to verify our analysis of a simple queueing system, and to explore the behavior of the system under a workload that is harder to analyze.

4.1 Distributions

1. Download the code for this week:

   ```
   wget wb/ss/code/queue.tgz
   tar -xzf queue.tgz
   cd queue
   make avgsd
   make test_random
   ```

   As usual, the code should compile with nary a warning, and (it goes without saying) no errors.

2. Read `random.c` and `test_random.c` and then run `./test_random`. It should create five files with samples from various distributions.

3. The program `avgsd` reads a file containing a column of numbers, and prints the average, standard deviation, and coefficient of variation of the values. The coefficient of variation is the ratio of the standard deviation to the mean. Run `avgsd` on each of the following files:

   ```
   rand.exponential  rand.lognormal  rand.normal  rand.pareto  rand.uniform
   ```

   I tried to calibrate the parameters of these distributions so that they have, roughly, the same mean. Did it work?

4. The file `test.sh` contains a bash shell script that generates cdfs for the various distributions, with various transformations. Read the code, edit it by changing the path for `cdf`, and then run it like this:

   ```
   sh test.sh
   ```

   For each distribution, there should now be 4 cdf files, with names like:

   ```
   rand.exponential.cdf         rand.exponential.cdf.logx
   rand.exponential.cdf.loglog  rand.exponential.cdf.logy
   ```

   Use xgraph to plot some of these distributions and get a sense of what different distributions look like under various transformations.

(a) How can we confirm that the exponential distribution is really exponential?

(b) How can we confirm that the pareto distribution is really pareto?

(c) What do we expect the normal distribution to look like on untransformed axes? What do we expect the lognormal distribution to look like under a $\log x$ transform?

(d) How can we confirm that the uniform distribution is really uniform?

`cdf` provides one more transformation, called `norm`, that applies the inverse of the cumulative normal distribution, which has the effect of transforming a normal distribution to a straight line. Try this:

```
cdf -t norm rand.normal | xgraph
```

If the result is a straight line, that is evidence that the distribution is really normal.

5. It is not easy to get all of these distributions into the same plot in a way that makes them all visible, but the `loglog` transformation does a pretty good job. If you plot all five distributions like this:

```
xgraph *.loglog
```

you get a good picture of at least the tail behavior of the distributions. The uniform distribution has no tail at all; the normal distribution includes some large values, but they are very rare; the exponential distribution includes some larger values; the lognormal and pareto distributions have the longest tails. In general, the longer the tail, the larger the coefficient of variation. In a long-tailed distribution, most values are very small, but there are occasional very large values.

One of the best-known long-tailed distributions in the real world is the distribution of individual wealth; most people have very little wealth, but a few people have enormous amounts. One of the first people to quantify this observation was the Italian economist Vilfredo Pareto (see `http://en.wikipedia.org/wiki/Vilfredo_Pareto`).

Why should we care about this particular set of distributions?

Uniform : Easy to understand. A good place to start, but not very common in the natural world or in computer systems.

Normal : Common among natural phenomena because almost any value that is the sum of independent factors is distributed normally (quoth the Central Limit Theorem). The normal distribution has several properties that make it amenable to analysis.

Exponential : Often a good approximation of the time between random events, like atomic decay and some human activities. Because it is "memoryless," it lends itself to analysis.

Lognormal : A common distribution for many metrics in computer science, possibly because a value that is the *product* of independent factors tends to be distributed lognormally.

Pareto : Commonly observed in sociology, economics and computer science. Mandelbrot has conjectured that the Pareto distribution is common because when it is convolved with almost any distribution (including itself) the result is at least approximately Pareto.

Of course, there are lots of other distributions worth knowing about, and even more importantly, we have to remember that most phenomena in the real world do not fit neatly into any simple model. Nevertheless, these models are often a surprisingly good description of real data, and you could go a long way with just these five.

4.2 Queueing Theory

"Queue" is the British term for what we Americans call a "line" and what Canadians, God bless 'em, call a "lineup". Queueing theory is the study of systems that comprise queues, which includes many software systems.

As you read Chapter 9 from Taylor and Karlin's classic text *Stochastic Modeling*, you might find it useful to understand the following concepts:

Interarrival time : How much time elapses between arriving "customers".

Service time : How much work each customer needs done.

Process : In queueing theory literature, "process" is used as a clumsy substitute for "thing", in order to turn an unsupported adjective into an official-sounding noun phrase. It has no meaning. For example, "poisson arrival process" is just an annoying way to say that the distribution of interarrival times is exponential.

Arrival rate : The expected number of arrivals per unit of time, denoted λ.

Service rate : The expected number of customers that can be served per unit time, assuming that the server is never idle. The service rate is denoted μ, which should not be confused with the μ that is often used in other contexts to denote the mean of a distribution.

Load : Also known as "traffic intensity", load is the ratio of the total amount of work entering the system to the total amount of work the system can do. It is sometimes denoted ρ, but this notation can be confusing, because ρ is also used for...

Utilization : Utilization is the percentage of time the server is busy, which is the complement of π_0, which is the percentage of the time the server is idle. In some cases load equals utilization, but not always (for example, if load is greater than 1).

Now make several passes through Chapter 9 and try to get as much out of it as you can. Pay particular attention to the fundamental queueing formula and Equations 9.10, 9.12, and 9.15. In particular, note the assumptions that underlie these relationships, so we will know when we expect them to apply.

As always when you read dense technical material, try not to get bogged down in the details during your first pass. Make 1–2 passes now, and then get back to the homework. When you get to the experiments below, you might have to refer back to the reading.

4.3 Discrete-event simulation

If you have studied differential equations, you have probably seen simulations based on discrete approximations of continuous systems. These simulations often break time into a sequence of (usually) evenly-spaced events.

Some physical systems are better described as a series of unevenly-spaced events, with the assumption that nothing happens between events. For example, in a queueing system, the events include customer arrivals, customers beginning service, and customers completing service. Between events, we assume that servers are doing something, but nothing happens that changes the state of the system (for example, the number of customers in queue).

The usual technique for modeling systems like this is a discrete-event simulation, which is usually based on some representation of events, and some data structure that keeps track of pending future events. The most common data structure is a Heap, which is particularly efficient for implementing the two operations we need in a discrete-event simulation: adding a new event to the Heap and removing the next (most proximate) event from the Heap.

The code that you downloaded includes a program called `sim.c` that implements a discrete-event simulation of an M/M/1 queueing system. At the top of the file, you will see type definitions for the "objects" in this simulation:

Event : An arrival, departure, or observation point. The event contains a type, a time, a customer, and a queue.

Heap : The data structure that keeps track of the order of events. You don't have to look at the implementation, but if you get a chance to check it out, it is quite elegant!

Customer : Each customer has a unique ID, which is useful for debugging. The customer object also keeps track of the customer's arrival and departure times (which are timestamps), and service time (which is a duration).

Queue : There are two queues in the program: the one named `queue` keeps track of the customers waiting for service; the one named `done` is not really a queue, but just a convenient data structure for holding onto customers that have completed service.

For each data type, there is a `typedef` at the top of the program and a function named `make_whatever` that you can think of as the constructor.

"Time" in the simulation is just a floating-point number that starts at 0.0. You can think of it in units of seconds if you want to, but it doesn't matter. The variable `gtime` is a global variable that is set to the current time while an event is being processed.

The main loop of the program is in `main`:

```
while (1) {
  e = heap_remove (h);
  if (e == NULL) {
    end_queue (queue);
  }
  handle_event (e);
}
```

This is an infinite loop that pulls events from the heap and processes them. In some cases, processing an event has the effect of adding new events to the heap. When the heap is empty, it returns NULL, and the simulation ends.

Take a minute to read the event handlers and make sure you understand how the simulation works. The command line options for `sim` are

-n : The number of customers to simulate. The default value is 10, which is useful for debugging. In your experiments, you will probably want to use at least 1000. If you use more than 2000, you will want to increase the value of `QUEUE_SIZE` and recompile the program.

-r : The traffic intensity, or load. The program uses this value to compute the arrival rate, `lamda`, relative to the service rate `mu`, which is always 0.1.

-x : The seed for the random number generator (an integer). The default value is 17, which means that if you run the program repeatedly, you will get the same result, unless you provide a different seed.

-v : A flag that indicates whether or not you want verbose output, which is useful for debugging. The default value is 1, which is on; 0 is off.

You can compile and run `sim` like this:

```
make sim
./sim -v0 -n1000 -r0.5
```

The cryptic output should look something like

```
0.5     1000    18.5804 20
```

I will leave it up to you to read the program and figure out what these value are.

Running `sim` has the side effect of creating files named `interarrivals` and `service_times` that contain the interarrival times and service times for each customer. Use `avgsd` and `cdf` to confirm that these distributions are exponential with the mean and variance you expected. Note that the coefficient of variation for an exponential distribution is always 1.

4.4 Experiment 1

`loop.sh` is a bash script that runs `sim` with a range of values of `rho`. You can run it like this:

```
sh loop.sh
```

Again, review the code to make sure you understand the output.

Which equation from Chapter 9 does this experiment test? Does the output from the simulation confirm or refute the analytic result? If there are any discrepancies, can you explain them?

In `loop.sh`, increase the duration of the simulation from 1000 to 2000 customers. What effect does this have on the results?

4.5 Experiment 2

Use the simultor to test Equation 9.12. You will have to make changes in the simulator to collect some additional information.

The average number of customers in the system, L, includes the customer in service, if there is one. Also, it is a time average, which means that we have to weight the queue length with the duration of the interval. For example, if the queue is 1 for 2 seconds and 2 for 3 seconds, then the time average is $(1 + 1 + 2 + 2 + 2)/5 = 1.6$ seconds.

Fortunately, there is an easy way to estimate L without keeping track of every change in the queue length. See Experiment 5, below.

4.6 Experiment 3

Use the simulator to test Equation 9.10. Again, you will have to modify the simulator.

4.7 Experiment 4

So far our simulations have been based on the same assumptions as the analysis, exponential distributions of both service times and interarrival times. But in computer systems these assumptions are often violated, so we might wonder how different the results are under different conditions or workloads.

Modify the simulator so that the distribution of service times is lognormal with a mean of roughly 10. What effect does this have on the coefficient of variation in service times? What effect would you expect this to have on average queue length and average wait time? Run the simulation again and see if your prediction is right.

How big is the effect of this distribution on the system metrics? Is the M/M/1 queue model a reasonable approximation for a system with lognormal service times.

As a JFFE, repeat this analysis for Pareto service times.

Note: with both lognormal and Pareto distributions, there might be significant variation from run to run, so you might want to run the simulator several times with different random seeds to test the repeatability of your results. You also might want to try longer simulations to see if the results change.

Also, for a given run, there may be a significant difference between the average load you specify as an input and the actual load the system sees in the simulation. So it is probably a good idea to make the program compute the actual ratio

$$\rho = \frac{total\_work\_that\_arrives\_during\_simulation}{total\_work\_the\_system\_could\_have\_done} \tag{1}$$

4.8 Experiment 5

This one is a JFFE, but you might find it interesting. M/M/1 queues have the "PASTA property", which stands for "Poisson arrivals see time averages". What that means is that if you sample a system metric, like queue length, at random intervals with an exponential distribution of interval lengths, the average of your sample is equal to the actual value averaged over time.

Modify your simulator to see whether this property holds.