Distributed Dependable Systems

2. Writing a Discrete-Event Simulation

Simulating A System

- We've seen an example of the type of analysis needed to evaluate how a system will behave
- To be theoretically tractable, **simplifying assumptions** must be taken
  - E.g., memoryless/exponential distributions, single server
- We want to look at what happens when we **drop** those assumptions

Discrete Event Simulation

- A queue of events, sorted by the time at which they happen
- A system state
- Iteratively:
  - Select the first event in the queue
  - Update the state
  - Add any new event triggered by this one to the queue
- Repeat until the queue is empty (or maybe a special STOP event is reached)

Our Goal

- Simulate an M/M/1 FIFO queue
- Compare our results to the theoretical ones
  - Job average time in the system
    - And let’s look at its distribution
  - Distribution of queue lengths
- Go distributed: simulate M/M/n, i.e., with n queues

Hints (1)

- Event queue:
  - Use an efficient data structure (e.g., a priority queue like a binary heap)
  - Pre-populate it with all job arrivals you want to simulate
  - Or, every time you process a job arrival, insert the new one in the queue
  - Remember that job interarrival time is exponential with mean $1/\lambda$
- State:
  - We'll need arrival and completion time for each job
  - We'll need the FIFO queue(s) on each scheduler

Hints (2)

- Event processing:
  - Job *i* arrives at time *t*
    - We mark this information, e.g., in a mapping that gives us the arrival time of each running job
    - We add the job to the server's FIFO queue
    - We generate an exponentially distributed random variable *X* (mean μ =1); if the FIFO queue is empty and add to the event queue the completion of job *i* at time *t*+*X*
  - Job *i* terminates at time *t*
    - Remove the job from the queue
    - Mark the completion time of *i*
    - If the queue is not empty, add the completion of the next job at time *t*+*X*

Hints (3)

- Getting queue lengths
  - Create an event, at regular intervals, to get how long queues are and save it for further processing
  - Results will be closer to theory if you wait a while to reach a steady state, and stop when you stop submitting jobs
- Testing
  - Try it out first with small numbers
  - See how it scales, e.g., by doubling #of jobs/simulated time each time; profile your code!

Hints (4)

- I’m used to Python; useful modules/functions are:
  - `collections.deque` for the FIFO queue
  - `heapq` for the priority queue
  - `random.expovariate` for generating exponentially distributed random variables
  - `matplotlib` for plotting
- Of course you can find similar things in your favorite language, or develop your own

Multi-Server Version

- Rather than a single server, there are n servers
- To keep a coherent load, raise the rate of new job generation to $n\lambda$
- Two strategies for selecting a queue:
  - Random queue choice
  - Considering d queues and taking the one with the shortest queue (“Supermarket model”)

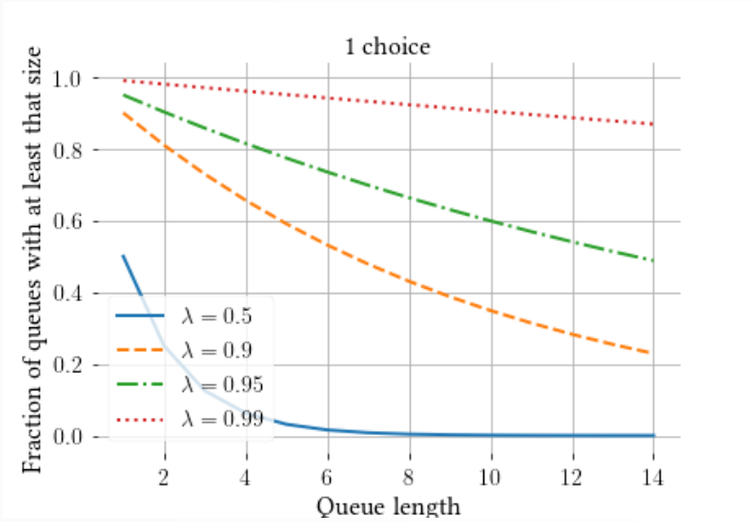
Supermarket vs. Random Queue

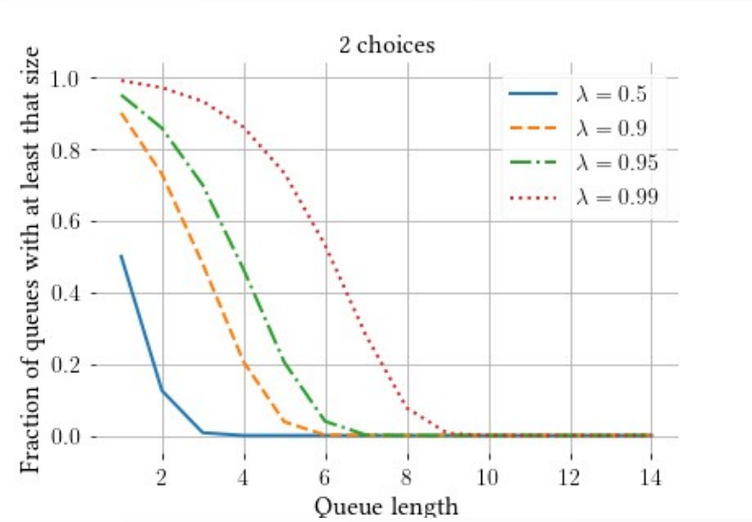
- Theory (Mitzenmacher, 2001) says that queue lengths for random queues are the same as the single-server case: a fraction λ^i queues have at least i jobs
- With the power of d choices, this number becomes

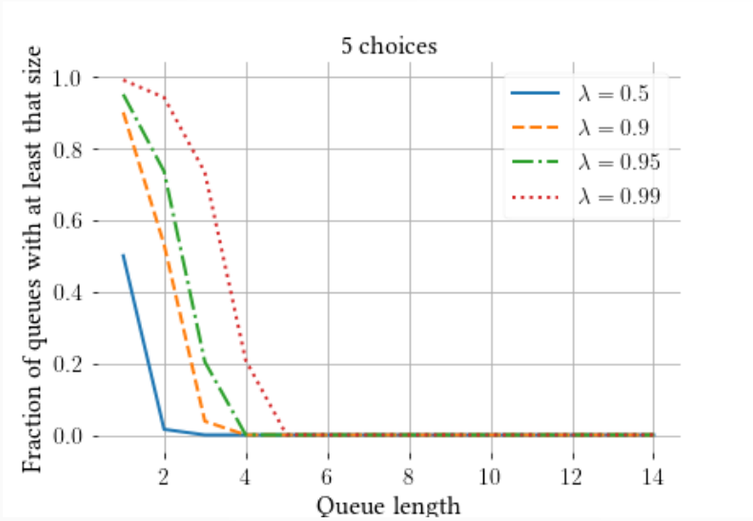
$$\lambda^{\frac{d^i-1}{d-1}}$$

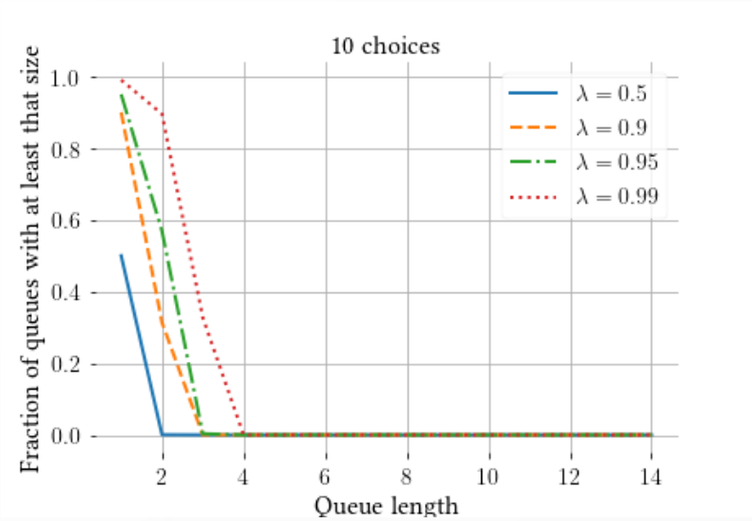
- Can we get the same results in our simulator?

Theoretical Queue Length









Exercise

- Verify whether you can reproduce the theoretical results
- Try out other distributions: job size and inter-arrival
- Try out some real-world traces (also see the paper by Amvrosiadis et al. if you're interested)
- Try out another scheduling policy: shortest job first
- Give an interpretation of these results

- There's plenty more to do if you liked this exercise: if you are interested, let's talk about this!