Aalto University
School of Electrical Engineering

Simulation of the M/G/1 queue

Pasi Lassila
Department of Communications and Networking

Aim of the lecture

- We consider three models
  - M/G/1 FIFO, M/G/1 PS and multiclass M/G/1 PS

- Specifically, we study two Mathematica implementations:
  - M/G/1 PS
  - M/G/1 PS queue with 2 classes - Flow-level model of base station with near/far users

- Mathematica assignment 2

Contents

- M/G/1 PS queue
- M/G/1 PS queue with 2 classes
- Assignment 2

Discrete event simulation of M/G/1 – FIFO queue

- As in the simulation of birth-death processes, at any point in time there are just two possible events: arrival or departure
  - In fact, even the multiclass systems or systems with parallel queues will share the same properties!

- However, now the process is no longer Markovian so we cannot exploit that
  - This does not change the overall logic very much though

- Simulation of FIFO is very simple, since the service time only needs to be generated at the point when job enters service

Pseudo-code for simulating M/G/1 FIFO

- Simulate the development of the queue length process in the M/G/1 FIFO queue from time 0 to time T assuming the system is empty at time 0
  - The state of the system at time t is defined by the queue length X_t.
- Initialisation:
  - set $X_0 = 0$
  - draw the arrival time of the first customer from the distribution Exp(λ)
- Event handling upon the arrival of a new customer (at time t)
  - the system state, i.e. the queue length is incremented by one : $X_t = X_t + 1$
  - if the system is empty upon the arrival, generate the departure time of the customer, t + S, where S is a sample from the service time distribution
  - generate the arrival instant of the next customer, t + I, where I is the interarrival time (drawn from Exp(λ) distribution)
- Event handling upon the departure of a customer (at time t)
  - the system state, i.e. the queue length is decremented by one: $X_t = X_t - 1$
  - if there are customers left in the system, generate the departure time of the customer to be served next, t + S, where S is a sample from the service time distribution
- Stopping condition: t > T

Changing FIFO to PS

- The events remain the same, at any time there can be only either an arrival or a departure
- But event handling logic changes because of the way how PS discipline serves the flows
  - Flows are served in parallel, and each job always receives 1/n fraction of time when there are n jobs in the system
- Required changes
  - State description must be more detailed
  - Implementation of service to the flows
  - Determining the next job departure

More details on the changes (1)

- State description must be more detailed
  - To implement PS, we need to know remaining service time of each job
  - Thus, state is a list that contains this information for each job
  - This list is updated when serving the existing jobs at every event time
  - Also, every arrival/departure modifies the list

- Implementation of service to the flows
  - This requires calculating at every event time the amount of service every job received since the previous event
  - This is done before the state changes (arrival or departure)!

More details on the changes (2)

- Determining the next job departure
  - It is convenient to keep the job list sorted so that the job with smallest remaining service time is at the head of list (i.e., first)
  - Note that the list only needs to be sorted at arrival instants
  - Then the next departure event can be easily determined
  - The next departure needs to be updated at departure events, as well as arrival events

Collecting statistics

- Assume we want to also collect statistics on the delays of the jobs

- Requires to add more state per job
  - We need to store the arrival time of each job in the state of each job
  - State per job consists of the pair: {arrival time, remaining service time}

- Next, let's look at the pseudo code…

Pseudo-code for M/G/1 PS

- Simulate the queue length process in the M/G/1 PS queue from time 0 to time T assuming the system is empty at time 0
- Initialisation:
  - set joblist empty
  - draw the arrival time of the first customer from the distribution Exp(λ)
- Event handling upon the arrival of a new customer (at time t)
  - Serve all jobs in the joblist based on number of jobs and time since previous event
  - Change state: draw sample of service time S and append {current time, S} in joblist
  - Sort joblist in increasing order of remaining service times
  - Generate the arrival instant of the next customer, t + I, where I is the interarrival time (drawn from Exp(λ) distribution)
  - Generate departure time of first job in job list (shortest) based on remaining service time and number of jobs in system
- Event handling upon the departure of a customer (at time t)
  - Serve all jobs in the joblist based on number of jobs and time since previous event
  - Collect statistics on delay of first job in job list
  - Update state: delete first job from list
  - Generate departure time of first job in job list (shortest) based on remaining service time and number of jobs in system
- Stopping condition: t > T

Code for M/G/1 PS queue

- Let’s look at the code in the Mathematica notebook

- Module: simulatorPS[..]
  - Code has been written assuming exponential service times
  - Simulates starting from an empty system until stopping time
  - Collects delays of each job and calculates the mean delay (also mean delay based on queue length)
  - Includes simple initial transient control by starting delay sample collection after a given initial transient time

Examples of using the code

- Test the code

```
la = 0.8;
mu = 1;
simulatorPS[la, mu, 2000, 20000]
```

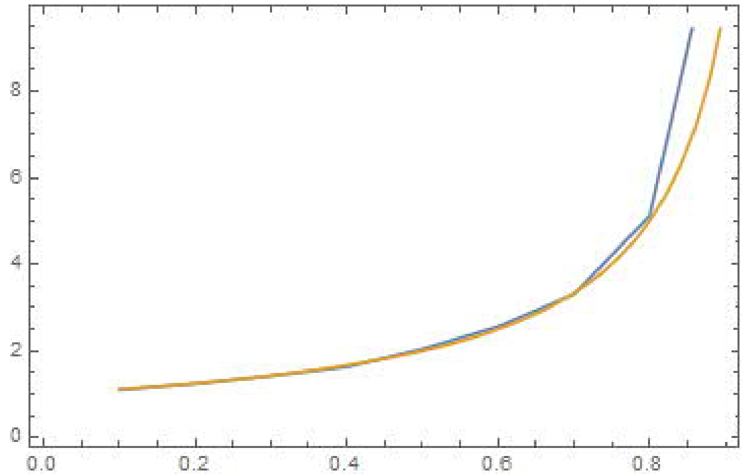
- Repeated simulations and confidence interval

```
la = 0.8;
mu = 1;
res = Table[simulatorPS[la, mu, 2000, 20000], {10}];
Needs["HypothesisTesting`"]
Mean[res[[All, 2]]]
MeanCI[res[[All, 2]]]
```

Examples of using the code

- Evaluate as a function of load and compare with exact solution

```
mu = 1;
simures = Table[{la, simulatorPS[la, mu, 10000, 80000][[2]]},
{la,0.1,0.9,0.1}];
exactres = Table[{la, 1/(mu - la)}, {la, 0.1, 0.9, 0.02}];
ListLinePlot[{simures, exactres}, Frame -> True]
```



Contents

- M/G/1 PS queue
- M/G/1 PS queue with 2 classes
- Assignment 2

Changes needed to introduce classes (1)

- Next we consider simulating the multiclass M/G/1 PS queue

- State of the system
  - Each class must have its own list representing the state of the jobs (i.e., the remaining service times and other state variables)

- Still possible to maintain the idea that at any given time the next possible event is either arrival or departure

- Arrival events
  - Generate arrival events at total aggregate rate (sum of all arrival rates in the classes), $\lambda_T = \lambda_1 + \cdots + \lambda_K$
  - Probability that the arrival is from class k is then simply

$$P\{\text{arrival is from class } k\} = \frac{\lambda_k}{\lambda_1 + \cdots + \lambda_K}$$

  - Possible to sample the class index on-the-fly at the point of the arrival!

Changes needed to introduce classes (2)

- Handling of departures from multiple classes
  - Jobs of different classes are in their own lists
  - Job lists are still ordered in an increasing order by remaining service times
  - Easy to calculate for each class when smallest job would complete and then just select the minimum from those

- Collecting statistics
  - Need to separately collect delays for each class

Code for the simulator

- Module simulator2classPS[…]
  - Code has been written assuming exponential service times
  - Simulates starting from an empty system until stopping time
  - Collects delays of each job in both classes and calculates the overall mean delay (also mean delay based on queue length) and the per class delays
  - Includes initial transient control

Example of using the code

- Run the simulator with given parameters

```
la1=0.4;
la2=0.15;
mu1=1;
mu2=0.5;
rho=la1/mu1+la2/mu2
simulator2classPS[la1,la2,mu1,mu2,200,20000]
```