Coursework 1: M/M/m Queueing System Simulation

Kyeong Soo (Joseph) Kim
Department of Electrical and Electronic Engineering
Xi'an Jiaotong-Liverpool University
29/10/2019

I. Introduction

In this assignment, you will run a series of simulations for M/M/m queueing system and compare their average delays. For your reference, a report for a sample simulation of M/M/1 queueing system with source code, a batch file for its execution, and a plotting script is provided in the appendix.

You should take the sample code of M/M/1 queueing system, generalise it for M/M/m queueing system, carry out a series of simulations for the parameter values given below, and compare the results for the average delay (by plotting a chart). For the calculation of the average delay in the simulation, refer to the sample report and the source code.

Below are the parameter values for three cases:

- Case 1
  - Number of servers (m): 1
  - Arrival rate: 5, 10, 15, ..., 95
  - Service rate: 100 (fixed)
- Case 2
  - Number of servers (m): 2
  - Arrival rate: 5, 10, 15, ..., 95
  - Service rate: 50 (fixed)
- Case 3
  - Number of servers (m): 5
  - Arrival rate: 5, 10, 15, ..., 95
  - Service rate: 20 (fixed)

The chart should show clearly the average delays for all the cases (total 3) with the arrival rate (common for both) as x axis[1]. The sample report mentioned can give you better ideas on how to prepare for it.

Based on the results (and the chart), you should find any differences between the cases as well as between the systems, if any, and provide your own explanations for them in the report.

II. Deliverables

You need to submit the following through the ICE by 11:59 pm, Sunday, 01/12/2018.

- A coursework report.
- Source code for the M/M/m simulation program.
- Scripts for batch execution, plotting, and post-processing, if any (e.g., Python with matplotlib, gnuplot, and MATLAB)

As part of the report, you must provide detailed instructions how to run the simulations so that TA can reproduce your results on his computer (as in the slide #66 of "Basic Queueing Theory and Network Simulation").

[1]You need to set y range properly in this regard.

APPENDIX
SAMPLE REPORT: SIMULATION OF M/M/1 QUEUEING SYSTEM

In this sample simulation, we are going to compare the simulation results for the average delay of M/M/1 system with the analytical solution from the queueing theory which we have already covered during the class.

Below is the analytical solution for the average delay:

$$E[T] = \frac{E[N]}{\lambda} = \frac{\frac{1}{\mu}}{1-\rho} = \frac{1}{\mu - \lambda},$$

or

$$E[T] = \frac{1}{\mu} + \frac{E[N_Q]}{\lambda} = \frac{1}{\mu} + \frac{\rho^2}{\lambda(1-\rho)}.$$

Note that the latter expression can be applicable for both single-server and multiple-server queueing systems, while the former cannot, which will be helpful for M/M/m system.

For batch processing of lots of simulation runs, the original "mm1.py" has been updated to "new_mm1.py" (downloadable from the ICE). Now "new_mm1.py" accepts command line parameters for the number of servers ('-M' for m)[2], the arrival rate ('-A' for λ) and the service rate ('-S' for μ)—and returns the arrival rate and the average delay as results.

To run all the simulations in one batch, type the following:

```
for /L %A in (5, 5, 95) do python new_mm1.py -M 1 -A %A -S 100 >> mm1.out [RETURN]
```

The command above loops through 5 to 95 with the increment of 5 for the variable %A, which is the arrival rate for the simulation program (the service rate is fixed to 100).

The results are recorded in "mm1.out" as follows:

```
5.0000E+00    1.0521E-02
1.0000E+01    1.1094E-02
1.5000E+01    1.1734E-02
2.0000E+01    1.2464E-02
2.5000E+01    1.3283E-02
3.0000E+01    1.4218E-02
3.5000E+01    1.5298E-02
4.0000E+01    1.6571E-02
4.5000E+01    1.8075E-02
5.0000E+01    1.9893E-02
5.5000E+01    2.2097E-02
6.0000E+01    2.4819E-02
6.5000E+01    2.8262E-02
7.0000E+01    3.2894E-02
7.5000E+01    3.9330E-02
8.0000E+01    4.8813E-02
8.5000E+01    6.3937E-02
9.0000E+01    9.3027E-02
9.5000E+01    1.7819E-01
```

Note that the first and the second columns denote the utilization and the average delay, respectively.

To compare the simulation results with the analytical solutions, we use python script "plot_mm1.py", which is also avilable on the ICE:

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 fig = plt.figure()
5 ax = fig.add_subplot(1, 1, 1)
```

[2]This is for extension to M/M/m later.

```
6
7   # Major ticks every 10, minor ticks every 5 for x axis
8   x_major_ticks = np.arange(0, 101, 10)
9   x_minor_ticks = np.arange(0, 101, 5)
10
11  # Major ticks every 0.1, minor ticks every 0.1 for y axis
12  y_major_ticks = np.arange(0, 1.1, 0.2)
13  y_minor_ticks = np.arange(0, 1.1, 0.1)
14
15  ax.set_xticks(x_major_ticks)
16  ax.set_xticks(x_minor_ticks, minor=True)
17  ax.set_yticks(y_major_ticks)
18  ax.set_yticks(y_minor_ticks, minor=True)
19
20  # Or if you want different settings for the grids:
21  ax.grid(which='minor', alpha=0.2)
22  ax.grid(which='major', alpha=0.5)
23
24  # calculate and plot analytical results
25  x1 = np.arange(1, 100)
26  y1 = 1/(100 - x1)
27  plt.plot(x1, y1,'b-', label="Analysis", linewidth=1)
28
29  # load and plot simulation results
30  x2, y2 = np.loadtxt('mm1.out', delimiter='\t', unpack=True)
31  plt.plot(x2, y2, 'rx', label="Simulation")
32
33  # add labels, legend, and title
34  plt.xlabel(r'$\lambda$ [pkts/s]')
35  plt.ylabel(r'$E[T]$ [s]')
36  plt.legend()
37  plt.title(r'M/M/1 ($\mu=100$ [pkts/s])')
38
39  plt.savefig('plot_mm1.pdf')
40  plt.show()
```

Fig. 1 shows the average delay of M/M/1 queueing system based on both queueing analysis and simulations.

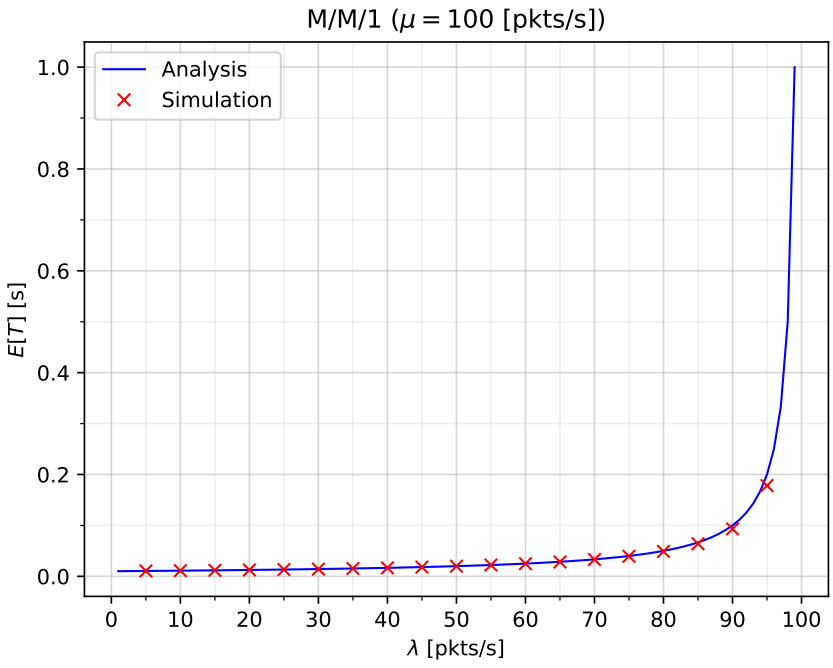

Fig. 1. Average delay of M/M/1 queueing system.

Also, we provide the source code for “new_mm1.py” as follows:

```python
#!/usr/bin/env python
# -*- coding: utf-8 -*-
##
# @file     new_mm1.py
# @author   Kyeong Soo (Joseph) Kim <kyeongsoo.kim@gmail.com>
# @date     2019-10-29
#
# @brief    Simulate M/M/1 queueing system
#
# @remarks  Copyright (C) 2019 Kyeong Soo (Joseph) Kim. All rights reserved.
#
# @remarks  This software is written and distributed under the GNU General
#           Public License Version 2 (http://www.gnu.org/licenses/gpl-2.0.html).
#           You must not remove this notice, or any other, from this software.
#

import argparse
import numpy as np
import random
import simpy
import sys


def source(env, mean_ia_time, mean_srv_time, server, delays, number, trace):
    """Generates packets with exponential interarrival time."""
    for i in range(number):
        ia_time = random.expovariate(1.0 / mean_ia_time)
        srv_time = random.expovariate(1.0 / mean_srv_time)
        pkt = packet(env, 'Packet-%d' % i, server, srv_time, delays, trace)
        env.process(pkt)
        yield env.timeout(ia_time)


def packet(env, name, server, service_time, delays, trace):
    """Requests a server, is served for a given service_time, and leaves the server."""
    arrv_time = env.now
    if trace:
        print("t={0:.4E}s: {1:s} arrived".format(arrv_time, name))

    with server.request() as request:
        yield request
        yield env.timeout(service_time)
        delay = env.now - arrv_time
        delays.append(delay)
        if trace:
            print("t={0:.4E}s: {1:s} served for {2:.4E}s".format(env.now, name, service_time))
            print("t={0:.4E}s: {1:s} delayed for {2:.4E}s".format(env.now, name, delay))


def run_simulation(mean_ia_time, mean_srv_time, num_packets=1000, random_seed=1234, trace=True):
    """Runs a simulation and returns statistics."""
    if trace:
        print('M/M/1 queue\n')
    random.seed(random_seed)
    env = simpy.Environment()

    # start processes and run
    server = simpy.Resource(env, capacity=1)
    delays = []
    env.process(source(env, mean_ia_time,
                       mean_srv_time, server, delays, number=num_packets, trace=trace))
    env.run()

    # return mean delay
    return np.mean(delays)


if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument(
        "-M",
        "--num_servers",
        help="number of servers; default is 1",
        default=1,
        type=int)               # for extension to M/M/m
    parser.add_argument(
```

```
        "-A",
        "--arrival_rate",
        help="packet arrival rate [packets/s]; default is 1.0",
        default=1.0,
        type=float)
    parser.add_argument(
        "-S",
        "--service_rate",
        help="packet service rate [packets/s]; default is 10.0",
        default=0.1,
        type=float)
    parser.add_argument(
        "-N",
        "--num_packets",
        help="number of packets to generate; default is 1000",
        default=1000,
        type=int)
    parser.add_argument(
        "-R",
        "--random_seed",
        help="seed for random number generation; default is 1234",
        default=1234,
        type=int)
    parser.add_argument('--trace', dest='trace', action='store_true')
    parser.add_argument('--no-trace', dest='trace', action='store_false')
    parser.set_defaults(trace=True)
    args = parser.parse_args()

    # set variables using command-line arguments
    num_servers = args.num_servers # for extension to M/M/m
    mean_ia_time = 1.0/args.arrival_rate
    mean_srv_time = 1.0/args.service_rate
    num_packets = args.num_packets
    random_seed = args.random_seed
    trace = args.trace

    # run a simulation
    mean_delay = run_simulation(mean_ia_time, mean_srv_time,
                                                num_packets, random_seed,
                                                trace)

    # print arrival rate and mean delay
    print("{0:.4E}\t{1:.4E}".format(args.arrival_rate, mean_delay))
```