CSE 6010 HW2 Queuing Network Simulation Report

Hongyang Wang, Yunhe Song

Division of work:

Yunhe Song:
- Implementation of Queue, Priority Queue and random generator
- Priority queue execution time measurement

Hongyang Wang:
- Development of the simulation engine and event handlers
- Simulation of the assembly line

Worked together on this report

Q1 – QUICK DEMO OF OUR SIMULATION

At first we generate a short simulation to demonstrate that program is functioning correctly. Here we have a sample : “print out a series indicating what events are processed in the simulation”.

```
memory error detector
==7639== Copyright (C) 2002-2012, and GNU GPL'd, by Julian Seward et al.
==7639== Using Valgrind-3.8.1 and LibVEX; rerun with -h for copyright info
==7639== Command: ../bin/simulate 10 30 3 50 1
==7639==
Parameters: avg_service_time=10.000000, avg_interval=30.000000, num_stations=3, to-
tal_simulation_time=50.000000
-------------------------------------------------------------------------------------
Event_Type  Timestamp   Station_ID  Part_ID     Serv_Times(for Source) OR Wait_Times(for Sink)
-------------------------------------------------------------------------------------
Create_Part 0.000000    Source      1           10.058634   10.058634   10.058634
Enqueue     0.000000    A           1
Start_Serv  0.000000    A           1
Create_Part 4.564579    Source      2           0.246098    0.246098    0.246098
Enqueue     4.564579    A           2
Dequeue     10.058634   A           1
End_Serv    10.058634   A           1
Enqueue     10.058634   B           1
Start_Serv  10.058634   B           1
Start_Serv  10.058634   A           2
Dequeue     10.304733   A           2
End_Serv    10.304733   A           2
Enqueue     10.304733   B           2
Create_Part 13.643580   Source      3           43.041294   43.041294   43.041294
Enqueue     13.643580   A           3
Start_Serv  13.643580   A           3
Dequeue     20.117268   B           1
End_Serv    20.117268   B           1
Enqueue     20.117268   C           1
Start_Serv  20.117268   C           1
Start_Serv  20.117268   B           2
Dequeue     20.363367   B           2
End_Serv    20.363367   B           2
Enqueue     20.363367   C           2
Create_Part 25.696049   Source      4           39.736314   39.736314   39.736314
Enqueue     25.696049   A           4
Dequeue     30.175902   C           1
End_Serv    30.175902   C           1
All_Done    30.175902   Sink        1           0.000000    0.000000    0.000000
Start_Serv  30.175902   C           2
Dequeue     30.422001   C           2
End_Serv    30.422001   C           2
All_Done    30.422001   Sink        2           5.494055    9.812536    9.812536
Create_Part 36.372431   Source      5           8.044114    8.044114    8.044114
Enqueue     36.372431   A           5
Create_Part 42.336313   Source      6           24.993178   24.993178   24.993178
Enqueue     42.336313   A           6
Create_Part 44.018245   Source      7           3.943815    3.943815    3.943815
Enqueue     44.018245   A           7
Create_Part 48.847247   Source      8           2.610243    2.610243    2.610243
Enqueue     48.847247   A           8
-------------------------------------------------------------------------------------
```

```
Result: avg_total_time=28.016662, avg_waiting_time=12.559563, total_part_finished=2

==7639==
==7639== HEAP SUMMARY:
==7639==     in use at exit: 0 bytes in 0 blocks
==7639==   total heap usage: 92 allocs, 92 frees, 2,128 bytes allocated
==7639==
==7639== All heap blocks were freed -- no leaks are possible
==7639==
==7639== For counts of detected and suppressed errors, rerun with: -v
==7639== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 6 from 6)
```

In the output above, this simulation is a simple M/M/3 queue simulation. We generate 3 stations and set the total simulation time to 50 time units. This is how the simulation works(let's take the first one as an instance):

1. Create part (customer) at "Create_Part", the "station_ID" will be "Source", and it will have a timestamp also we generate a random service time as "Serv_Times" .
2. Then the customer will enter the first queue for station A, the "Event_Type" will be "Enqueue" and the "station_ID" will be "A", also it will have a timestamp about when this entrance happened.
3. After waiting in the queue, the customer enter the station and has a "Event_Type" ,which will be "Start_Serv" and the "station_ID" will be "A" because the service is done in station A.
4. Also at the same timestamp, a "Event_Type" called "Dequeue" will be written.
5. Then the customer will leave the station A after his service, the "Event_Type" will be "End_Serv" and the "station_ID" will be "A", also it will have a timestamp about when this departure happened.
6. Finally a "All_done" type will create, which means this customer has done everything and leave for "Sink", in the end, the simulation system will tell us how long this customer waits. Luckily, in this simulation, the queuing delays for customer 1 at the 3 stations are all 0 because he is the first one to get served so does not need to wait. For customer 2 however, the queuing delays are 5.49, 9.81 and 9.81 respectively.

The simulation has attributes :

- interval/service times are exponentially distributed.
- FIFO queue for every station.
- Priority queue that tell what is the next event (as shown above).

- Event-oriented simulation : each step is a event.

Result for general simulation will like this:

Parameters: avg_service_time=10.0, avg_interval=30.0, num_stations=1, total_simulation_time=50.0
Result: avg_total_time=28.02, avg_waiting_time=12.56, total_part_finished=2

"avg_total_time" means average total time for one customer from the source to the sink. "avg_waiting_time" means average waiting time for one customer from the source to the sink. "total_part_finished" means in total how many customers have been served.

In the next part of this report, which shows the result for a large simulation, we can discover a pattern: avg_total_time = 30 + avg_waiting_time, which is what we expected. Since avg_total_time = avg_service_time + avg_waiting_time, and the service time obeys to an exponential distribution with a mean of 10, and with 3 stations in the system the avg_service_time is expected to be 30. This also proves the correctness of our simulation.

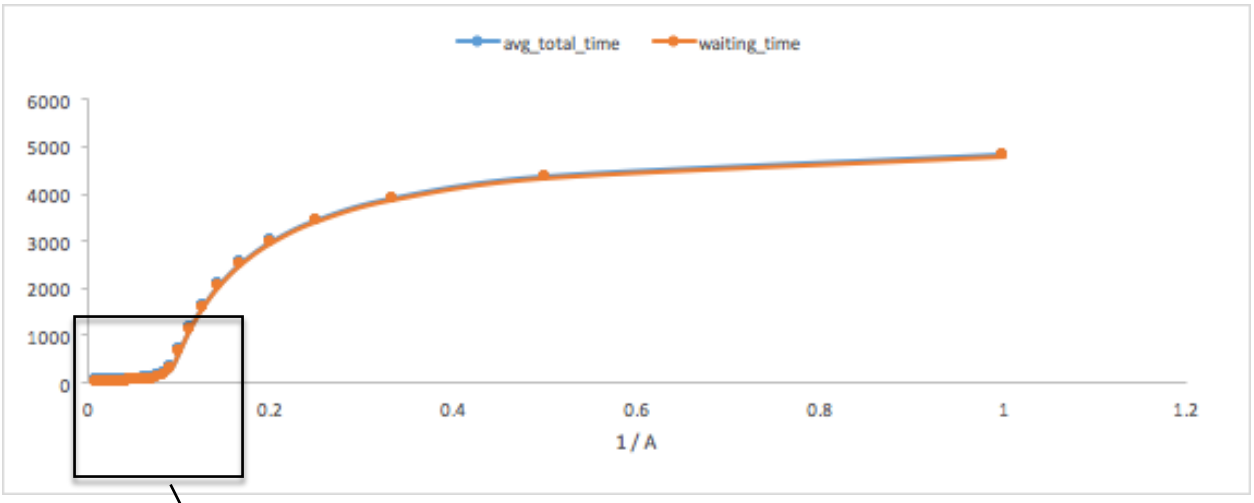
We also used Valgrind for memory leakage check to make sure all the malloc'ed heap space are freed before the exit of the program, as shown in the printed log above.

Q2 – SIMULATION RESULT AS ARRIVAL RATE CHANGES

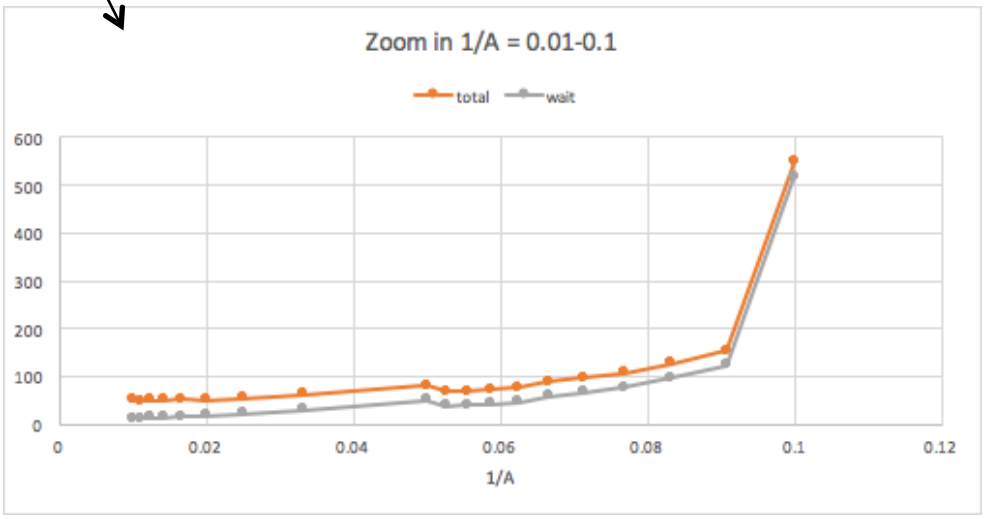
In this part, we use a fixed service time which is 10 units, and vary the arrival time from 1 to 100, which means 1/A varies from 0.01 to 1.

interval time	1/A	avg_total_time for one customer	total_waiting_time for one customer
1	1	4818.817191	4788.63875
2	0.5	4360.156605	4329.978164
3	0.333333333	3901.496019	3871.317578
4	0.25	3442.835433	3412.656992
5	0.2	2984.174847	2953.996406
6	0.166666667	2525.514261	2495.33582
7	0.142857143	2066.853675	2036.675234
8	0.125	1608.193089	1578.014648
9	0.111111111	1149.532503	1119.354062
10	0.1	690.871917	660.693476
100	0.01	31.808811	3.024276

As can be seen from the graph below, when average rate becomes large ($1/A \rightarrow 1$), which means interval time is small (A = 1 unit), average total time(including service time and waiting time) is extremely large. This is expected because the service time is 10 units, which means almost everybody in the queue has to wait for a long time. We can also notice that the turning point of the curve is at somewhere around 0.1 meaning that A = 10, which is exactly the average service time at each station.



Zoom in: 1/A = 0.01 - 0.1



Literature survey on queueing networks

In queueing theory, a model is constructed so that queue lengths and waiting time can be predicted.[see 1] And also in our experiment, the queue is a FIFO queue, which means where the oldest (first) entry, or 'head' of the queue, is processed first.

In the experiment, the service time should be selected from an exponential distribution with mean value of *S*. Assume that the time between the generation of successive parts at the source, i.e., the inter-arrival time, is also exponentially distributed with mean *A*.

Random variables $\tau j = tj - tj\text{-}1$ are inter-arrival times, for Poisson arrival, the inter-arrival times are:

– IID (independent and identically distributed)

– exponentially distributed (i.e., CDF $F(x) = 1 - e^{-x/a}$)

This means this queue is a M/M/1 queue[see 2] which represents the queue length in a system having a single server, where arrivals are determined by a Poisson process and job service times have an exponential distribution.(And the assignment has a 3-servers model, M/M/1 model with more than one server is the M/M/c queue.

A M/M/1model has attributes: [see 3]

– Exponentially distributed interarrival times

– Exponentially distributed service times

– One server

– Infinite number of buffers

– Infinite population size

– First-come-first-serve service discipline

The model is considered stable only if $\lambda < \mu$. If, on average, arrivals happen faster than service completions the queue will grow indefinitely long and the system will not have a stationary distribution. The stationary distribution is the limiting distribution for large values of *t*.

Various performance measures can be computed explicitly for the M/M/1 queue. We write $\rho = \lambda/\mu$ for the utilization of the buffer and require $\rho < 1$ for the queue to be stable. ρ represents the average proportion of time which the server is occupied.

The average response time or sojourn time (total time a customer spends in the system) does not depend on scheduling discipline and can be computed using Little's law [see 4] as $1/(\mu - \lambda)$.

The average time spent waiting is $1/(\mu - \lambda) - 1/\mu = \rho/(\mu - \lambda)$. [see 5]The distribution of response times experienced does depend on scheduling discipline.

Experiment & compare results with the theoretical results

theoretical results:

Consider the following queue :

- *$\lambda = 0.3$ (avg_interval=3.33 unit)*
- *$\mu = 0.5$ (avg_servicetime=2.00 unit)*

mean response time r:

- *$r= 1/(\mu-\lambda) =1/0.2 = 5.0$*

average waiting time:

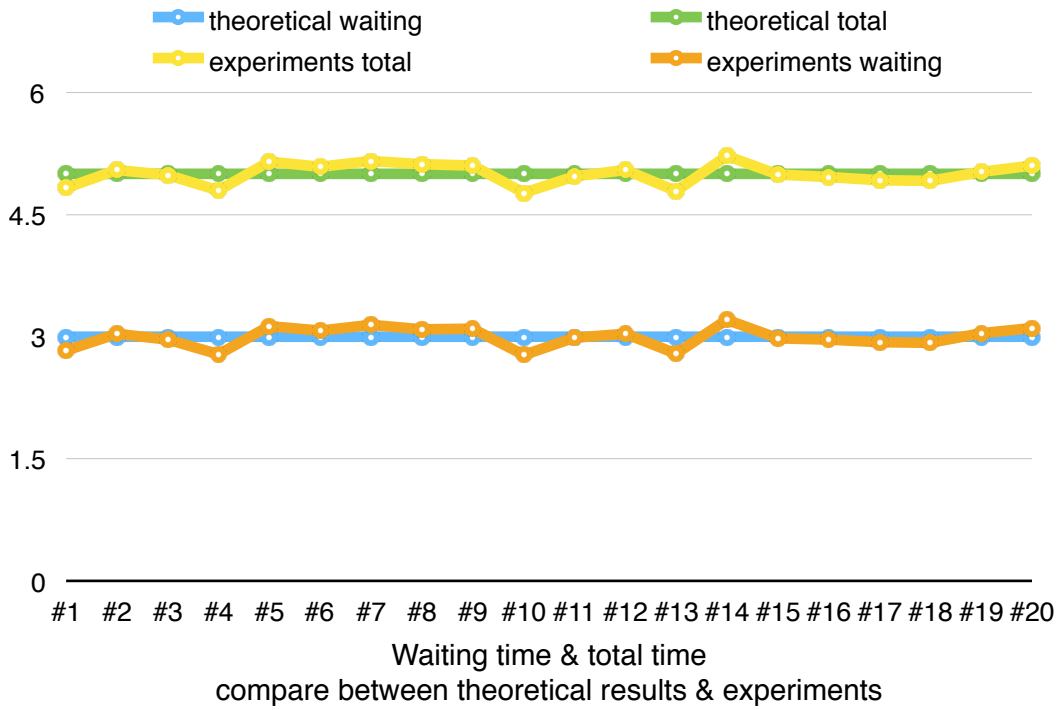
- *$r - 1/\mu =5.0 - 2.0 = 3.0$*

experiments results:

	avg_total_time	avg_waiting_time	total_part_finished
experiment#1	4.828088	2.827892	14877
experiment#2	5.04781	3.038786	14879
experiment#3	4.987849	2.96247	14856
experiment#4	4.797885	2.784073	14737
experiment#5	5.148744	3.132438	14967
experiment#6	5.090198	3.07556	15119
experiment#7	5.158284	3.148938	15048
experiment#8	5.123122	3.090085	14980
experiment#9	5.100552	3.095214	15006
experiment#10	4.767832	2.774759	14913
experiment#11	4.970071	2.991747	15232
experiment#12	5.045273	3.033029	15004

	avg_total_time	avg_waiting_time	total_part_finished
experiment#13	4.788731	2.790659	14954
experiment#14	5.22055	3.211216	15032
experiment#15	4.99378	2.979739	14996
experiment#16	4.96318	2.966155	15157
experiment#17	4.921651	2.932356	14828
experiment#18	4.915215	2.925678	15055
experiment#19	5.026819	3.041003	15055
experiment#20	5.10735	3.106729	15018

graph:



Compare theoretical results with our experimental results, although we only choose one pair of parameters, from the graph we can tell that our experiment has consistency with the M/M/1 queue theory. The experiments do have some vibrance, but generally, near - perfect theoretical agreement with experiment was achieved.

Reference:

[1] Sundarapandian, V. (2009). "7. Queueing Theory". Probability, Statistics and Queueing Theory. PHI Learning. ISBN 8120338448.

[2] Abate, J.; Whitt, W. (1987). Transient behavior of the M/M/l queue: Starting at the origin . Queueing Systems. 2: 41.

[3] Guillemin, F.; Boyer, J. (2001). Analysis of the M/M/1 Queue with Processor Sharing via Spectral Theory. Queueing Systems. 39 (4): 377.

[4] Adan, I.; Resing, J. (1996). Simple analysis of a fluid queue driven by an M/M/1 queue. Queueing Systems. 22: 171.

[5] Harrison, P. G. (1993). Response time distributions in queueing network models. Prefermance Evaluation of Computer and Communication Systems. Lecture Notes in Computer Science. 729. pp. 147–164.

Performance measurement of our priority queue

Write a program to measure the average time to perform a single insert followed by a single remove operation in the priority queue containing N events where each event has a timestamp uniformly distributed between 0.0 and 1.0.

This graph shows our simulation result. As we can see, the execution time linearly increases as N goes up. In our implementation, for each insertion operation, the node to be inserted must be compared with every node already in the priority queue until finding its position. So the amortized number of comparison will be N/2, thus the time complexity for each insertion operation will be O(N). For the deletion operation, it will just remove the head of the queue since the queue is already sorted. So the time complexity is O(1). So for every insertion and deletion, the complexity will be O(N), and that is exactly what we got.