SF2567 - Queuing Theory Project

Queuing Systems with Multi-types Servers

Rémi Lacombe
Advisor: Pr. Per Enqvist[1]

[1]Department of Mathematics
KTH Royal Institute of Technology

13^{th} December 2017

Outline

Context and General Overview

- Queues are a huge hindrance to productivity.
- 37×10^9 hours are wasted waiting in queues every year in the US alone.

Extract from Murphy's Law on queues:

- If you change queue, the one that you left will start to move faster than the one you are in now.
- Your queue always goes the slowest.
- Whatever queue you join, no matter how short it looks, it will always take the longest to you to get served.

- Only one service class.
- s servers.
- Inter-arrival time between two customers has an exponential distribution of parameter λ.
- Inter-leaving time between two customers for a single server has an exponential distribution of parameter μ.

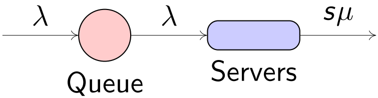


Figure: M/M/s queue

M/M/s queues are a Markov Processes. The steady-state probabilities are:

$$\begin{cases} P_n = \dfrac{(\lambda/\mu)^n}{n!} P_0 & \text{if } 0 \leq n \leq s, \\ P_n = \dfrac{(\lambda/\mu)^n}{s! s^{n-s}} P_0 & \text{if } n \geq s, \end{cases} \qquad [1]$$

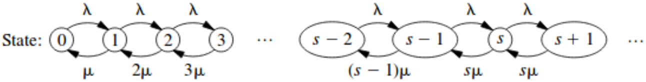
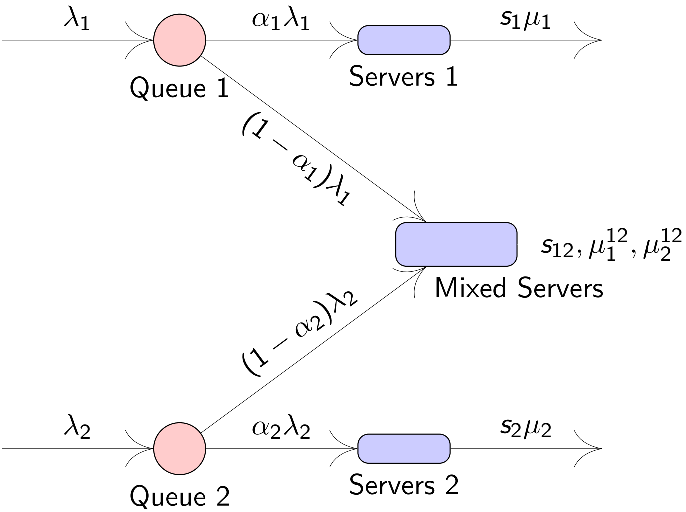


Figure: Representation of a M/M/s queue as a Markov Process [1]



λ_1
$\alpha_1\lambda_1$
$s_1\mu_1$
Queue 1
Servers 1
$(1-\alpha_1)\lambda_1$
$s_{12}, \mu_1^{12}, \mu_2^{12}$
Mixed Servers
$(1-\alpha_2)\lambda_2$
λ_2
$\alpha_2\lambda_2$
$s_2\mu_2$
Queue 2
Servers 2

How can to compute the steady-state parameters of this type of queueing system?

1. Compute a M/M/s queue and compare the steady-state probabilities with the analytical results.
2. Do the same for the multi-types servers QS and check some simple cases.

- s is the number of servers.
- There are only $s + 1$ possible events:
  - Either a customer leaves one of the s servers.
  - Either a customer joins the queuing system.
- *next_event_index* is an *int* in $[0, s]$ which indicates what the last event was.
- *next_event* is an *array* of *double* of size $s + 1$ which contains the delays before each next event.
- *busy* is an *array* of *bool* of size s which indicates which servers are currently busy.

II - Modelisation

Main Loop of the Program

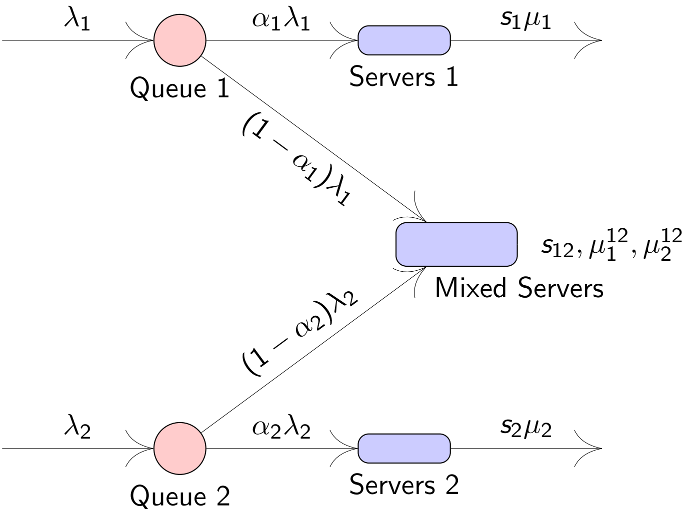
- if *next_event_index* $< s$ the delay until next customer leaves this same server is computed and entered in *next_event*
- else:
  - if the last event was an arriving customer, the delay before another joins the QS is computed and entered in *next_event*
- find the minimum value in *next_event*: its index indicates the next event
- if a new customer joins the QS:
  - if there is an available server the customer joins it and *next_event_index* becomes the index of that server
  - else the customer joins the queue and *next_event_index* becomes s
- else:
  - if at least one customer was waiting in the queue, one joins this server and its index is the new value of *next_event_index*
  - else *next_event_index* becomes s
- update values in *next_event*

What changes when adding multi-types servers?

- Three queues (*queue*_1, *queue*_2, *queue*_12) instead of one.
- *queue*_12 has its *array next*_*event* of size $s_{12} + 2$.
- The next event is decided with the global minimum value in *next*_*event* for the three queues.
- *current*_*order* is an *array* of *bool* of varying size to keep track of the priority order in *queue*_12.
- *last*_*event* is an *int* which can take its value in $\{1, 2, 12\}$ depending on what the last event was.

II - Modelisation

General Idea of the Simulations



II - Modelisation

Code Implementation

- Base *class Queue* and derived *class DoubleQueue*.
- Definition of constructors/getters/setters for both.

```
class Queue {

private:

    double lambda, mu, high_value;
    unsigned server_number, current_queue, new_server_time_index;
    vector<double> next_event;
    vector<bool> busy;
...

class DoubleQueue : public Queue {

private:

    double mu_2;
    vector<bool> current_order;
...
```

- Overloading of some member functions of the base class.

```
// returns a boolean value to assess if a new time should be
//computed for the next arrival
bool Queue::nextArrival() {
    if (next_event[server_number] == high_value) {return true;}
    return false;
}
...

bool DoubleQueue::nextArrival() {
    if ((getEvent(getServer()) != getHigh()) and
        (getEvent(getServer() + 1) != getHigh())) {
        return false;
    }
    return true;
}
```

```
// if this condition is verified it means that the arrival time of
// the next customer is not yet known so the following computes it
if (queue_12.nextArrival()) {

    if (queue_12.getEvent(queue_12.getServer()) ==
        queue_12.getHigh()) {

        double new_event_time = entry_distrib_queue_11(generator);
        queue_12.setEvent(new_event_time, queue_12.getServer());

    }

    else {

        double new_event_time = entry_distrib_queue_22(generator);
        queue_12.setEvent(new_event_time, queue_12.getServer() + 1);

    }

}
```

III - Results

Goal: compute the steady-state probabilities.

```
// updates time trackers
if ((queue_1.getQueue() + queue_1.getServer()) <
    10*queue_1.getServer()) {

    steady_state_probabilities[queue_1.busyNumber() +
        queue_1.getQueue()] += next_event_time;
    total_time += next_event_time;

}
```

- *array* of *double* *steady_state_probabilities* that stores the amount of time spent in each state by the QS.
- *total_time* is a *double* that keeps track of the total amount of time.
- Exports results to a text file.

III - Results

$$\begin{cases} P_n = \dfrac{(\lambda/\mu)^n}{n!} P_0 & \text{if } 0 \le n \le s, \\ \\ P_n = \dfrac{(\lambda/\mu)^n}{s! s^{n-s}} P_0 & \text{if } n \ge s, \qquad [1] \end{cases}$$

- The mean square error and the mean waiting time are computed.
- For the following computations we chose: $s = 10$, $\lambda = 0.02$, $\mu = 0.003$.
- Divergence if $\rho = \dfrac{\lambda}{s\mu} \ge 1$ can be another criterion.

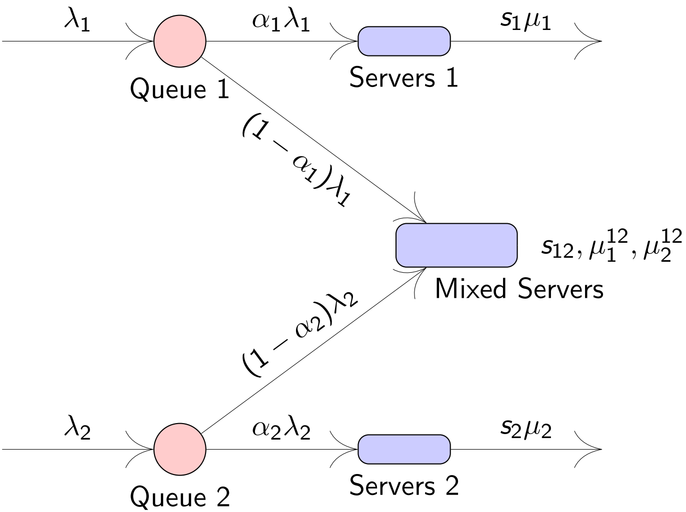
III - Results

For a simple M/M/s queue with parameters $\{0.02, 0.003, 10\}$ we have the following results:

Number of iterations ($\times 10^6$)	MSE ($\times 10^{-8}$)	Computation Time (s)
10	37	13
20	22.4	28
50	10.1	65
100	5.6	135
200	2.3	281
500	1.3	740

- The individual precision of each P_n is at least 1.2×10^{-4} for 500×10^6 iterations.

III - Results



λ_1
$\alpha_1\lambda_1$
$s_1\mu_1$
Queue 1
Servers 1
$(1-\alpha_1)\lambda_1$
$s_{12}, \mu_1^{12}, \mu_2^{12}$
Mixed Servers
$(1-\alpha_2)\lambda_2$
λ_2
$\alpha_2\lambda_2$
$s_2\mu_2$
Queue 2
Servers 2

What about a QS with multi-types servers?

- No general analytic results yet, but there are two particular cases for which we have explicit formulas:

1. Check that the steady-state probabilities are the same as for the M/M/s queue with $\mu_1^{12} = \mu_2^{12}$.
2. Compare the observed probabilities with the theoretic results for the case $s_{12} = 1$ (results from Pr. Enqvist):

$$\begin{cases} \alpha_{k+1} = (\alpha_k - \lambda \boldsymbol{\mu}^T \mathbf{A} \mathbf{\Pi}_k)/(\boldsymbol{\mu}^T \mathbf{A} \mathbf{p}) \\ \mathbf{\Pi}_{k+1} = \mathbf{A}(\alpha_{k+1}\mathbf{p} + \lambda \mathbf{\Pi}_k) \\ \mathbf{\Pi}_k = \begin{bmatrix} \pi_{1k} \\ \pi_{2k} \end{bmatrix} \quad \text{steady-state probability vector in state k} \end{cases}$$

III - Results

For a multi-types servers QS with parameters $\lambda_1 = 0.01$, $\lambda_2 = 0.02$, $\mu_{11} = 0.04$, $\mu_{22} = 0.03$, $\alpha_1 = 0.75$, $\alpha_2 = 0.5$ and of course $s_{12} = 1$, we have the following results:

Number of iterations ($\times 10^6$)	MSE ($\times 10^{-8}$)	Computation Time (s)
10	35.3	23
20	4.2	46
50	3.5	117
100	1.1	236
200	1.2	511
500	1.3	1294

- The simulation correctly models a QS with multi-types servers for the two particular cases we have analytic formulas for.

Conclusion

- Working real-time simulation of a queuing system with two service classes for the customers.
- Reliable tool to check validity of further analytic results.
- I will now focus on the theoretical part of the problem with Pr. Enqvist during the second semester.

[1] F. Hillier. G. Lieberman
Introduction to Operations Research.
McGraw-Hill, 1967.