LabQ: Queue simulation

Tommaso Massaglia
Politecnico di Torino ID: s292988
s292988@studenti.polito.it

I. Introduction

The lab's main goal was to simulate, and subsequently analyze, the results of a queue simulator ran using python under different circumstances. The simulated queue is a simple single server event-based priority queue (a queue in which the event scheduled for the earliest time is ran at each iteration) based on python $queue$ package.

II. Assumptions

The following theoretical assumptions are made:

- At each time *users* clients are present inside the queue
- *inter_arrival* represents the time between arrivals, sampled from an exponential distribution with $\lambda = \frac{load}{service}$
- *service_time* is time required to serve a client, sampled randomly from an exponential distribution with $\lambda = \frac{1}{service}$
- *time_to_fail* and *time_to_repair* are sampled from an exponential distribution with $\lambda = fail\_rate$ and $\lambda = repair\_rate$ respectively
- There are 4 kind of events that can happen at each time:
  1) **arrival**: a client arrives and is placed inside the queue, a departure event is placed after *service_time* time if no departure is already planned
  2) **departure**: after *service_time* has elapsed the client leaves the queue and the next departure is planned in *inter_arrival* time
  3) **failure**: the server failed, the next departure is postponed by *time_to_repair* time
  4) **repair**: the server is repaired, departures are no longer postponed and the next failure is planned after *time_to_fail* time
- The queue is initialized with an *arrival* and, given that a failure and repair rate are set, a *failure* event

III. Input Parameters

The simulator is initialized with the following parameters:

- **seed**: the seed of the random events in the simulation
- **load**: how often a client arrives with respect to the time that's required to serve them; values over or close to 1 of this parameter cause high instabilities in the system and it's thus recommended to set it to a max of 0.95 in order to have meaningful results
- **service**: the average time that a client spends in the system (excluding delays)
- **sim_time**: stopping condition of the simulation, specifies for how long to run it
- **max_delay**: a threshold, whenever a client delay goes over this we increase a counter
- **max_queue**: the maximum size of the queue, whenever an arrival would go over this parameter, the client is considered as lost
- **fail_rate**: a parameter ≈ 0 that determines the time between failures
- **repair_rate**: a parameter that determines the time required to repair the server

IV. Output Metrics

The simulation output for each set of input parameters are:

- *average service time*: the average time required to serve a client $\frac{\sum service}{tot\_users}$
- *arrival rate*: measured as $\frac{\#arrivals}{simtime}$
- *average delay*: measured as $\frac{totaldelay}{\#ofusers}$
- *p delay_below*: probability that the delay is below *max_delay*, computed as $\frac{\#delay\_below}{tot\_users}$
- *p idle*: probability that the server is idle, computed as $\frac{\sum timeidle}{simtime}$
- *p lost customer*: probability that a customer is lost due to the queue being at max capacity, computed as: $\frac{\#lostcust}{\#arr}$
- *tot_users*: number of clients served in *simtime*
- *num failures*: number of failures observed during the simulation
- *% failure time*: total time spent in a failure state with respect to the total simulation time

V. Results

Since the simulation had many input parameters and output metrics, the first order of business was to trim down to the minimum the number of parameters to analyze. From a first run, analyzing the correlations between parameters over 19k different simulations (3/4 different values per input), it became evident how *sim_time* only influenced the number of clients and failures we experienced during the simulation as can be expected.

The same could be said about the *seed*; the seed influences the random events in the simulation, but considering a large enough number of samples (events in our case, so a long enough simulation time) we are bound to fall into an average behaviour of the system, as the *Central Limit Theorem* [1] states.

The following parameters were used in all their possible 5760 combinations to generate the dataset used for the figures.

	input
seed	42
load	0.3, 0.5, 0.7, 0.9
service	10, 20, 30, 40, 50, 60
simtime	600000
maxdelay	15, 20, 30, 40
max_queue	4, 6, 8, 12, 15
failure_rate	0, 0.001, 0.005, 0.008, 0.01
repair_rate	0.005, 0.008, 0.01, 0.02

The heat map depicted below show at a high level the correlations between the input/output parameters of the simulation; at first glance, we can see that load is the parameter that independently from other input parameters more strongly influences the outputs, and as such will be mostly used in the analyses.

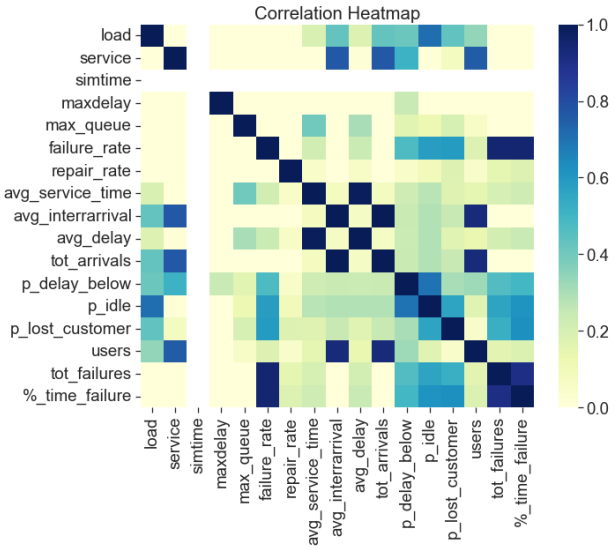

Fig. 1. Output/Input parameters absolute correlation heat map

A. Delay

Firstly we'll start by analyzing the delay in the system:

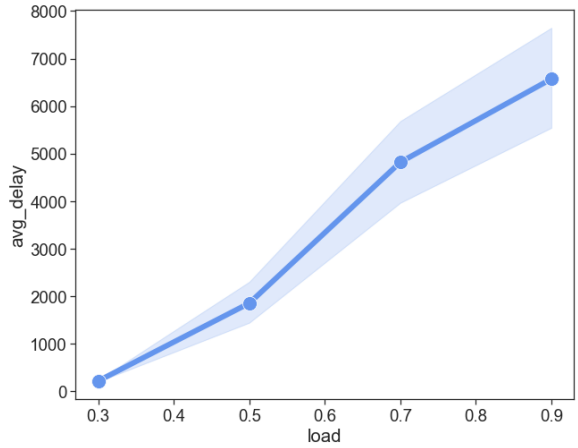

Fig. 2. Average Delay vs Load

Figure 2 shows a linear dependency between load and avg_delay; this makes sense since, as the load increases, the time a client has to wait in line before the server can get to their order increases. The graph will be further discussed in section V-C

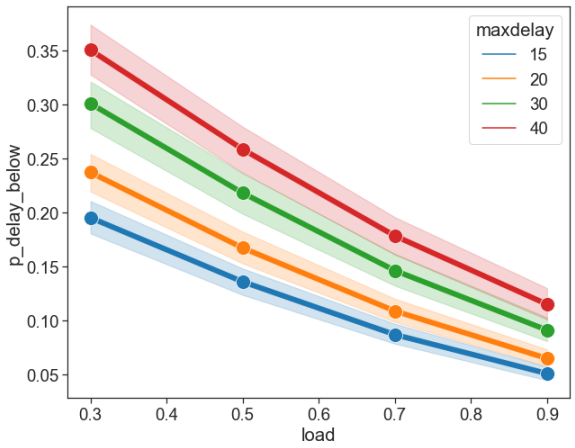

Fig. 3. Probability that delay is below *maxdelay* vs Load

Considering fig. 3, as expected, regardless of the *maxdelay* value we set, the probability for a client's delay to be under that value decreases as the load increases.

B. Idle

The server is defined as *idle* whenever the queue is empty (so $users = 0$), *p_idle* is the probability for a client to find the server as idle when entering the queue.

Fig. 4 is consistent with previous results as the higher the load, the lower the chance for the queue to be empty; it's interesting to notice how other parameters influence mildly this probability, shown by the low variance.

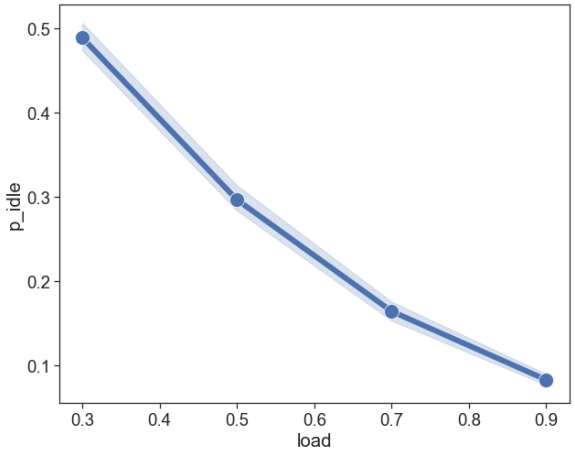

Fig. 4. Probability that the server is idle vs Load

C. Finite Waiting Line

The line is defined as finite when, given a threshold *maxqueue*, once that threshold is reached no more clients can be accepted into the system. The consequence of this parameter is that, depending on the other parameters, customers are lost during the operation; in a real case scenario, since queues cannot be infinite, it's important to size the "waiting line" correctly as to not spend too much to have a uselessly large queue and to not loose customers by having a way too small one.

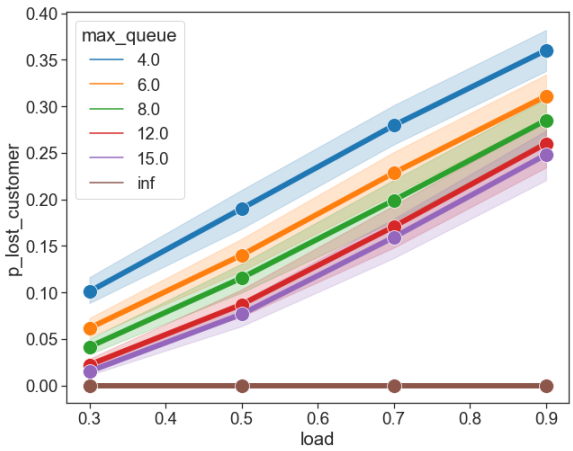

Fig. 5. Probability that customers are lost vs load

As the load becomes higher, the probability that a customer is lost increases since the average queue length with a higher load is higher. The curves have low variability and increasing the *max_queue* parameter on average is related to the same growth in probability to loose a customer independent of the load.

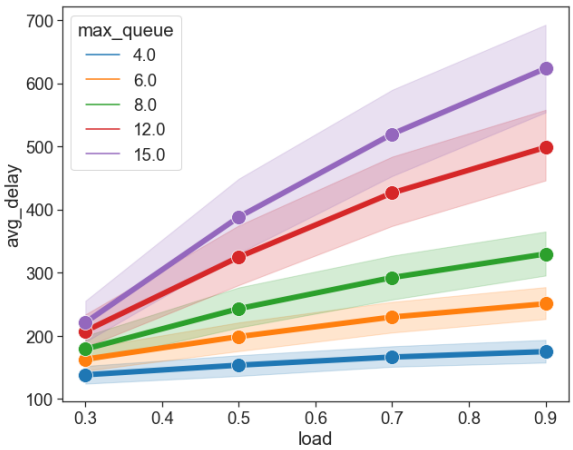

Fig. 6. Average Delay vs Load split by max_queue

Fig. 6 extends what we saw in section V-A by splitting the curve depending on the *max_queue* parameter: as stated in that section, lower values of max_queue force the delay to be lower on average, since at maximum a new client will have *max_queue*-1 users in front of them. The infinite queue curve is not displayed as the extreme growth of the avg delay had little relevance on the analysis.

D. Server with failures

Looking specifically at the influence of failures on the performance of the server, we focus firstly on their influence on average service time:

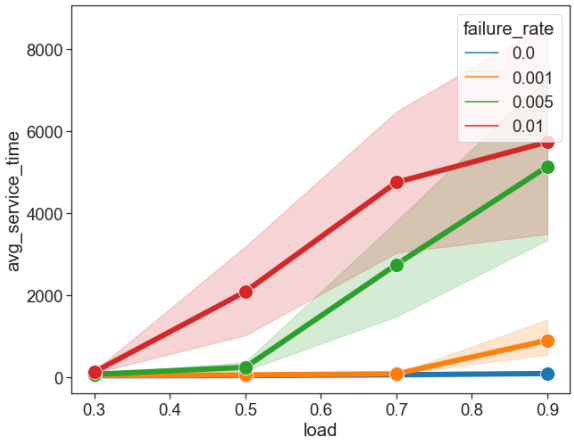

Fig. 7. Average Service Time vs Load split by failure rate

Other than the obvious different growth in service time as the failure rate increase, the most interesting aspect of this figure is the variability that a higher failure rate brings to the service time.

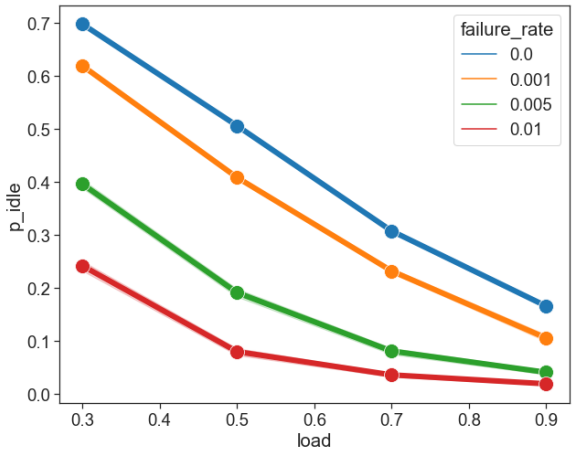

Fig. 8. p_idle vs Load split by failure rate

With no noticeable variability, the probability that the server is idle is shown to sharply decrease as the failure rate increases; the behaviour can be expected as by introducing failures, service times get larger reducing idle times.

Figure 9 and 10 show the exact opposite results; a higher repair rate leads to higher idle times and lower average service times.

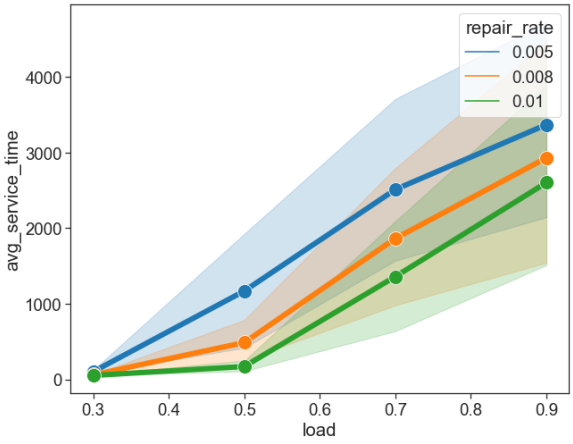

Fig. 9. Average Service Time vs Load split by repair rate

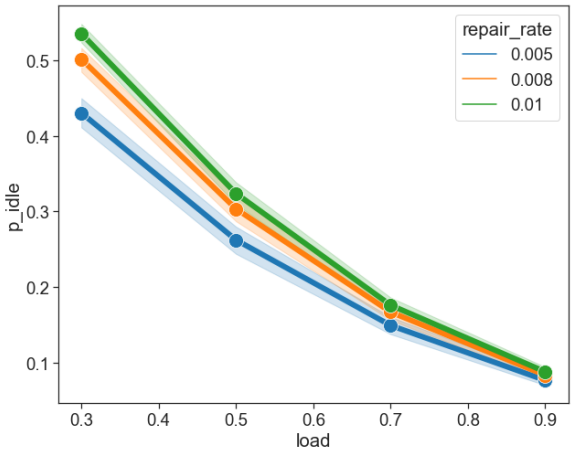

Fig. 10. p_idle vs Load split by repair rate

VI. Discussion

The results obtained in the lab represent realistic behaviours of an $M/M/*$ queue; this result is important as we can deduce that, given that our simulator has enough characterization in terms of quantity of input parameters, and given that those input parameters are sized correctly (the history of arrivals in a server can give us an idea of the arrival rate as an example), we can likely anticipate the behaviour of our queue under different scenarios in an easy manner.

The consequence is that, whenever we face a sizing problem (i.e. whenever we need to decide the number of counters in a supermarket or how powerful the servers for our website will have to be), it's possible up to a certain degree to anticipate the requirements we'll have to meet to make our system work properly and to have a good customer satisfaction.

The simulation didn't deal with other types of queues, such as multi-server or with different distributions in departure/failure/repair, but we know from previous studies such as [2], that even in a congested system the use of a correctly sized amount of servers can lead to system stability no matter the failure rate or variability of arrivals.

To conclude, the following graph represents the normalized average number of clients in the system overlapped for 81 different simulations with different parameters; the general shape we can observe is in line with the utilized distribution (exponential), with most of the density concentrated at the beginning.

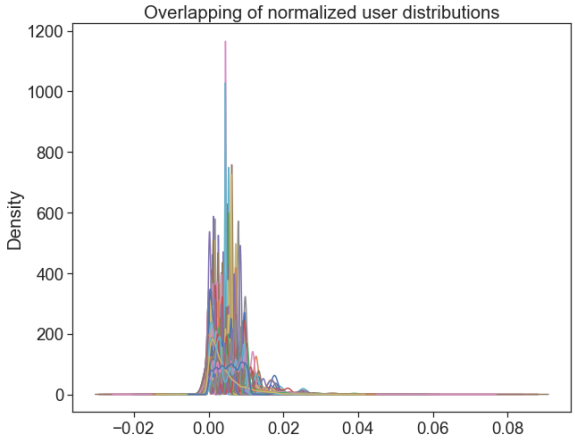

Fig. 11. Normalized user distribution

References

[1] Fischer, Hans. "A history of the central limit theorem" (PDF). Springer New York Dordrecht Heidelberg London. Retrieved 29 April 2021.

[2] Whitt, Ward. (2003). How Multiserver Queues Scale with Growing Congestion-Dependent Demand. Operations Research. 51. 531-542. 10.1287/opre.51.4.531.16093.