1 Simulator

The simulator occupied a significant place in our research. It allowed us to simulate the different systems we are interested in evaluating.
One of our goals is to check how the choice of system characteristics influences WT prediction. Hence, the simulator is a valuable tool to study the different target systems.
Moreover, to evaluate our simulator, we compared the event logs obtained from the simulation (based on parameters extracted from the real system) and properties of the real system itself.

1.1 Simulator parameters

The purpose of the simulator is to simulate realistic queueing systems. Starting with the basic components and parameters of a queueing model, we considered extensions that make the system more compatible with real systems, such as:

- Realistic parameters
  We enabled periodic modification of system parameters (arrival rate, service rate, number of servers and patience distribution). The purpose of this option is to reflect potential changes that may, or may not, occur within an interval time (ranging from several hours to several days) (aka: dynamic context).

- Customer diversity
  An additional source of variation that is to be taken into consideration for the simulator to be realistic is that due to the diversity among the customers themselves. It is reasonable to expect, for example, that demographic information concerning the customer (age, gender, and more) may be correlated to the WT, and generally speaking, may therefore influence the behaviour of the system. As we are working with a dataset that contains several types of customers, the simulator allows for specific arrival rates, service rates and patience for each customer type.

- Dependent service time
  We wanted the simulator to mimic systems in which the service time is correlated to the current load. In such systems, the agents may increase

(or even perversely decrease) their service rate in case of excessive load (i.e. high demand) in order to decrease the load (or to express frustration). As a way to incorporate this phenomena, we defined a speed-up mode under which a service time is generated based on the queueing load. This mode affects the service time realization of each customer.

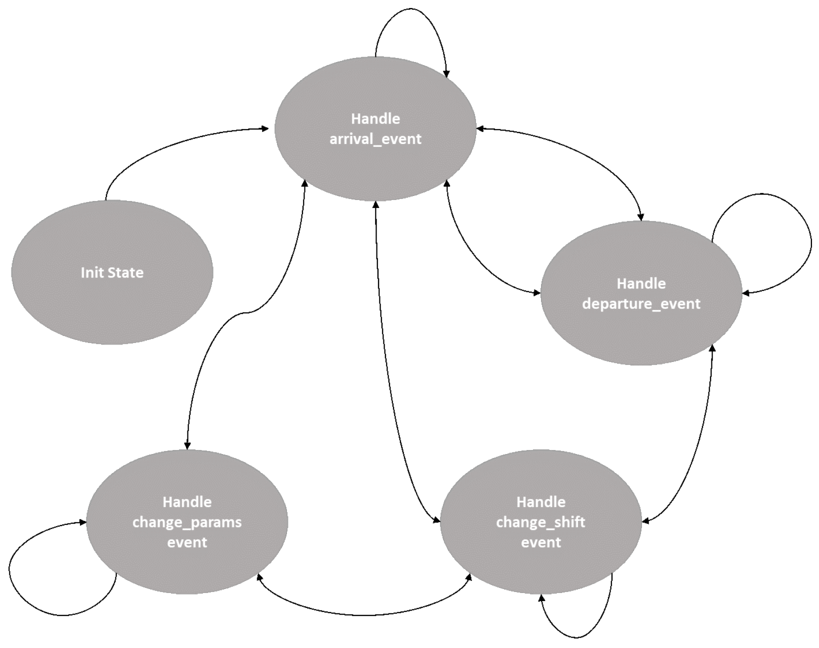
The simulator may use different service time distribution as well. In this way, we may be able to compare the influence of the service time distribution on the WT prediction.
These options enable to design experiments with an increasing level of complexity by including marginal modifications into the simulated systems.

1.2 Technical definition

In order build a simulator that deals with all the options described above, a simulation class has been defined.
The simulator has been implemented in the *python* programming language, which supports Object-Oriented-Programming (OOP). Briefly, OOP enables the generation of objects, containing attributes and methods, corresponding to a unified pattern, denoted class. The object is initialized using a particular function, denoted the constructor, which is in charge of setting the initial values of the different attributes. An important advantage of OOP is that objects may be modified through methods defined beforehand, so that the modification process is supervised.
The *Simulator* class is initialized for each day and for each week with intrinsic data (such as the week number of the simulation) and the parameters. One of the first steps of the initialization process is the generation of arrival events, i.e. the arrival timestamps. Since the arrival rate is derived from an exponential distribution, we sampled a Poisson number of events for each hour with the appropriate mean, and using a uniform distribution, we sampled the times of those arrival events for each hour. We finally sorted those events in order to draw a list of successive arrival events. We sampled a specific patience for each customer, defining the maximal WT each customer is willing to wait in the queue.

The data we used to feed the simulator can be viewed as a list of successive arrival events that reflect customer arrivals. For each arrival, we defined the type of customer we are dealing with and his patience. With that, we provided the mean service time for each customer type and for each hour of the day and the number of servers available at each hour of the day.

Once the initialization is completed, the simulator runs as a loop, handling the different events which may occur during the simulation, using the different methods in the class: *handle_arrival*, *handle_departure*, *change_parameters*, and more. The simulation ends at a time specified by the user at the initial stage. Figure 1 presents the simulator class as a state machine. It illustrates the simulator's state for a given specific time and the transition options for each state. The edges represent the possible transitions from a state to another. For example, the edge drawn from "Handle arrival event" to "Handle departure event" represents the next operation to be taken care of after one's arrival: handle a departure and modify the class parameters accordingly. A method *advance_time* has been designed to perform the right transition between two states. This function verifies which event should be handled first and calls the proper method.



States.png

Figure 1: Deterministic State machine diagram of the simulator given a specific seed.

1.3 Input and output

As we explain above, the simulator contains the option of periodic changes in the parameters values, when the periods are defined for each hour and for each day of the week. The input of the simulator is defined based on this selection.
Four different matrices are fed into the simulator for each kind of customer:

- arrival rates
- service rates
- number of servers
- patience rates

In each matrix, $24 \times 7 = 168$ elements are reported: 24 parameters for each day of the week (one for each hour of the day) and this across 7 days a week. In the scope of this study, we did not let any flexibility regarding the input structure.
The different distributions and the remaining options defined above (such as the speed-up mode) are defined into the simulator (initialization).
The simulator returns a ”*.csv*” file containing an event log. The event log reports the successive events of the system for a 6-week-long period.