UNIVERSITA' DI PISA
Scuola di Ingegneria
Master Degree in Computer Engineering

THE CARREFOUR

Candidati:
Roberto MAGHERINI
Elena SCARSELLI
Alessandro SIENI

Performance Evaluation of Computer Systems and Networks

Anno Accademico 2017-2018

Contents

Introduction

Our project consist in a supermarket with two types of queue: single and multiple. Customers arrive with a IID RVs interarrival time and their service demands are IID RVs and could be exponential or lognormal. Interarrival time is exponential. The numbers of till is constant in a single simulation, but varies between different simulations. When a custumer arrives is directed to a queue (single or multiple). After the queuig time the customer is served if we are in the case of multiple queue; otherwise if we are in case of single queue, a deterministic time

$$\Delta \cdot j$$

is added at the total response time. In the end the customer is served and leave the supermarket.

1 SIMULATION ANALYSIS

1.1 Modeling

In this project we had to model two different types of systems. First, we have a fixed number of servers (during a simulation), which represents tills in a real supermarket, and a single queue, where the customers arrive and wait to be served. In this scenario the servers have the same priority and they start to serve a customer when they finished the previous one and the queue is not empty (we simulated a work conserving system). The second scenario is different, because each server has his own queue and it serves only customers in line in that queue. In this scenario, considering real life, there is a priority to reach a specific queue, because in real life the customers, in the most of cases, want to walk as less as he can, and to model that, in case of a equal lenght for more than one queue, the customer will reach the server that is more near to him. With this behavior the nearests tills are busier than the most distant tills.

1.2 Simulation Wokflow

The analyzed system consists of customers (packages) that must be sorted in the service centers (not variable value) and finally, exit from the system. There should be two types of policies for checkout:

1. single queue: the first in line is served;
2. multiple queue: each open till manage its own queue.

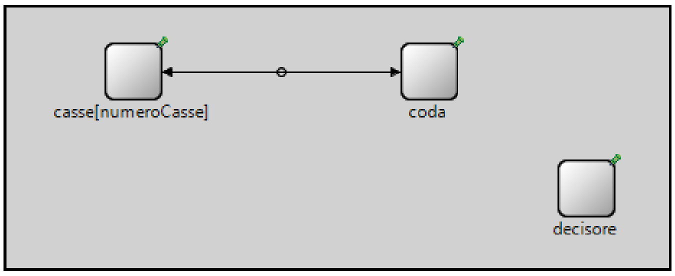


Figure 1: Generic scenario.

The model we thought generalizes both types of tails. The generic scenario is illustrated in 1: In this scheme, the customer that arrives follow the workflow, from the input it's inserted in the queue, the module D as "decisore" choose using different policies to which tills address the customer. At last, after the service time, the customer leaves the system.

1.3 Objectives

Our general objective is to evaluate the performance of the two different types of tail. It's obtained by comparing the queueing and response time of two options under a varying workload. We want to analyze the variation of:

1. the time spent in the queue,
2. the response time of the till,

assuming that there is a deterministic (predetermited) delay to reach the till from the queue that is:

$$\Delta \cdot j$$

where j is the till's number. To minimize this delay it was decided to schedule the case decision algorithm (in such a way that the cash box with the lowest ID is chosen) so that choice is the case with smaller ID (SCEGLI FRASE MIGLIORE IN INGLESE PER DIRE QUESTA COSA).

Evaluation will be done observing the behaivor of the system, varying the workload and the number of tills. We are interested in finding the scenario with better queuing time and responde time. Our objective is SMART because it's specific (comparing times), measurable (fulfill in a finite time interval), achievable (it's happen every day), repeatable (experiments can be repeated as often as necessary) and thorogh.

1.4 A priori assumption

The assumption made to have a good model to study the behavior of the system is the following:

1. The system has ideal delay = 0 s among all the network nodes and in case a) a fixed delay equal to $\Delta \cdot j$
2. Queue hasn't a maximum size, all customers that enter the system will be served
3. The system at simTime zero is empty
4. Distributions of inter-arrival time (IAT) and service time (ST) are independent from the workload
5. The pseudo-random numbers generated in the simulation by Omnet can be considered IID

In order to model the system is under the hypothesis of continuous-time Markov Chains

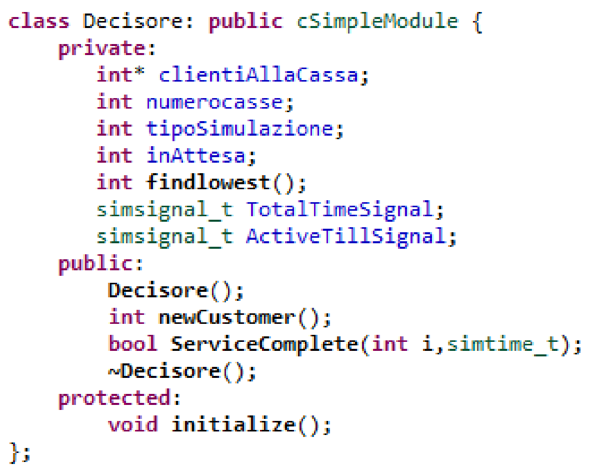
$$M/M/C/\infty/\infty$$

where:

1. C is finite because the number of tills can variate, but stay constant in a single simulation
2. The queue is infinite because the tills can serve all the people.
3. The population is infinite because before the simulation we don't know how many people will arrive at each till.

1.5 Decisor

The decisor is an entity that chooses, using a "random" definible algorithm the till in which switch the customer. Its structure is shown in figure 2. The functions that make up the decisor are the following:



```
class Decisore: public cSimpleModule {
    private:
       int* clientiAllaCassa;
       int numerocasse;
       int tipoSimulazione;
       int inAttesa;
       int findlowest();
       simsignal_t TotalTimeSignal;
       simsignal_t ActiveTillSignal;
    public:
        Decisore();
        int newCustomer();
        bool ServiceComplete(int i,simtime_t);
        ~Decisore();
    protected:
        void initialize();
};
```

Figure 2: Decisor.

findlowest This function is used to find what till has ties with the minimum number of people who are waiting to be served. At the beginning we check the parameter who indicates the type of simulation we want to execute: ties with a common queue (case 1) or ties with own queue (case 2). The function has input parameter: parametro(int), that is used to define which case we are simulating and an output: int that represents the index of the tie with the minimum number of element in queue, and if the ties have only one element, the first free tie. If two or more ties has the same elements in queue, the function chooses the till with the lowest index.

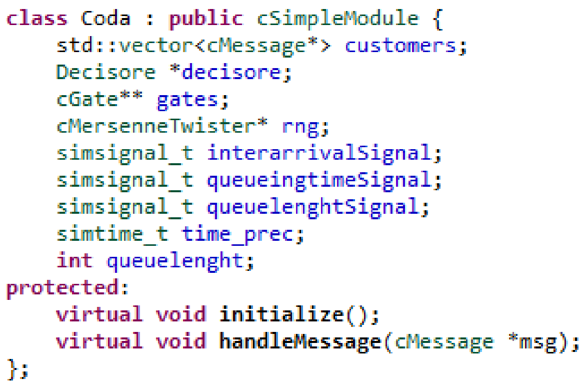
newCustomers This function finds the right till for the new customer; once it was found it must add the specific time-delay, because we want to simulate the time to reach the till, and only after that time is elapsed our new customer is added to the queue. The function has input parameter: parameter(int) that is used to define which case we are simulating and an output: int that return the position of the till.

ServiceComplete This method is used by the till to indicate that it has finished serving one customer, and in particular is usefull in case of a common queue, because the till are work-conserving, i.e. all customers in queue must be served without going in idle. This function is called by a till when it has served a customer and if there are other customers in queue it assign the fist of them immediately to the till who has called this method.

1.6 Queue

This simulation's module represent the queue, that should be one for each till or one for all. In the second case, the queue is "first in first out". Through the decisor, the custumers are sent to the tills.

initialize This function will initialize all the port gates of Coda, because it need as many outputs ports as the number of Casse in the simulation. Then the simulation will start.



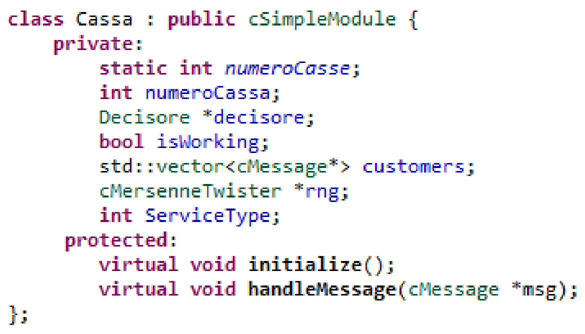
```
class Coda : public cSimpleModule {
    std::vector<cMessage*> customers;
    Decisore *decisore;
    cGate** gates;
    cMersenneTwister* rng;
    simsignal_t interarrivalSignal;
    simsignal_t queueingtimeSignal;
    simsignal_t queuelenghtSignal;
    simtime_t time_prec;
    int queuelenght;
protected:
    virtual void initialize();
    virtual void handleMessage(cMessage *msg);
};
```

Figure 3: Queue.

handleMessage This function manage the queue: it check if a new customer is arrived, if there is at least one customer in the queue and there is at least one available till, it send the customer to the till.

1.7 Till

This simulation's model represent the till. The customer does the bill and pays. This is included in service time. After this module the customer goes out of the system.



```
class Cassa : public cSimpleModule {
    private:
        static int numeroCasse;
        int numeroCassa;
        Decisore *decisore;
        bool isWorking;
        std::vector<cMessage*> customers;
        cMersenneTwister *rng;
        int ServiceType;
     protected:
        virtual void initialize();
        virtual void handleMessage(cMessage *msg);
};
```

Figure 4: Till.

initialize This function will initialize all the port gates of Coda, because it need as many outputs ports as the number of Casse in the simulation. Then the simulation will start.

handleMessage This function manage the queue: it check if a new customer is arrived, if there is at least one customer in the queue and there is at least one available till, it send the customer to the till.

1.8 Code Verification

We want to check if the code correctly implement the model which has been validated to correctly represent the system. In order ot check it, we look at the event log for a small time interval and verify that the customer's route is followed correctly: arrival, time of queue (if it's necessary), service and checkout from the system.

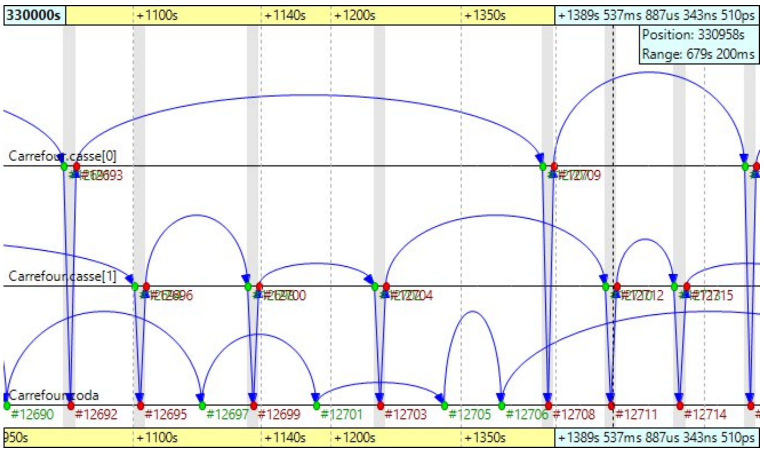


Figure 5: Single queue eventlog.

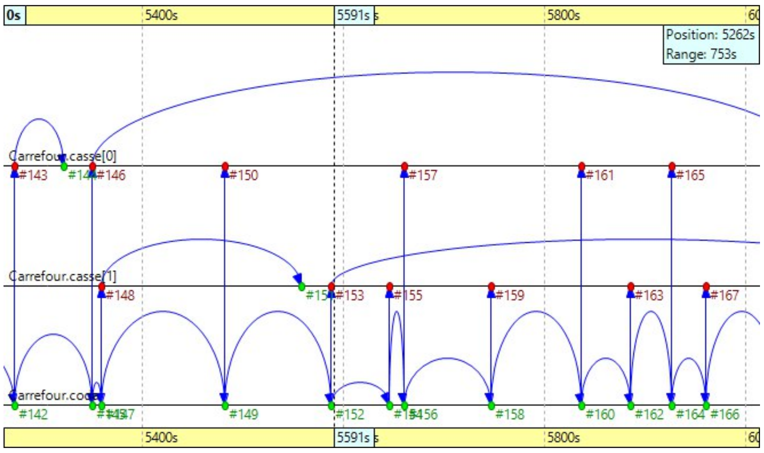


Figure 6: Multiple queue eventlog.

Since the system is the steady state, the condition are respected in both cases, single queue and multiple queues, as shown in 5 and 6.

2 Scenario Analysis

In this project, for each suitable configuration we have selected different seeds, one for each run. This action has been implemented on inter-arrival time of customers and service time, having a different seed for each run is necessary for the idea of IIDness. Seeds represent inter-arrival rate for the queue and service rate for the till. Each queue and each till have a different seed. Moreover, all statistical data are extracted from the simulator when it is in the steady state, where the parameter of interest has shown relatively constant behaivor. In general, high utilization and lowest delay and queuing time is always the case of interest, since the arrival rate and service rate, λ and μ, can lead to the overbalance of people in the system, that would cause too long waiting times. These two parameter has been the point of debate which helped us finding more suitable configurations and more accurate system limits. It is possible to have a same utilization by having different values of parameters e.g. increasing arrival time and decreasing service time, but this is not suitable for the logic of the supermarket.

2.1 Exponential Scenario

The first scenario analyzed is based on Exponential RVs, which has this PDF:

$$f(x) = \begin{cases} \lambda e^{-\lambda x} & x \geq 0 \\ 0 & x < 0 \end{cases} \tag{1}$$

and this CDF:

$$F(x) = \begin{cases} 1 - \lambda e^{-\lambda x} & x \geq 0 \\ 0 & x < 0 \end{cases} \tag{2}$$

Since in this simulation we want to study how the system works in different scenarios, we have used one seed for the arrival time of the customer and one seed for each till (a seed for each server). This choice guarantees us that the random variable present in our system are IID, because they are generated using different seed. The only fixed number present in our simulation is the time necessary for a customer to reach a till, because in a considerable amount of time, the position of the till in a supermarket are not changing.

2.1.1 Parameter analysis

In this scenario, the inter-arrival time is represent by λ [s]; that stand for how many seconds elapse on average between one customer's arrival and another. To find out how many people arrive per second, it's necessary to compute the arrival's frequency $\frac{1}{\lambda}[\frac{1}{s}]$ (inter-arrival rate). The same is true for μ [s] (service time) that stand for how many seconds elapse, on average, between one customer's service and another and for $\frac{1}{\mu}[\frac{1}{s}]$ (service rate), the frequency of service as how many people are served per second. It's realistic think that a new customer arrives each, at least every 20 seconds, at most every 300 seconds and that a customer is served at least every 10 seconds, atmost every 240 seconds. We obtain $\lambda \in [20, 300]$ and $\mu \in [10, 240]$.

Puoi parlare del tempo di ritardo, indicandolo come valore fisso, che è direttamente proporzianale al numero di cassa, che nel nostro scenario indica anche la distanza della cassa dal "punto di partenza" Oppure indicare come parametro il numero di casse, mostrando eventualmente, magari in base ad esperienze reali, quali sono i numeri di casse (C) maggiornamente rilevanti, ovvero quelli che hanno maggior probabilità di essere presneti in un supermercato Per esempio : Noi abbiamo deciso di analizzare un sistema il cui numero di casse varia da 2 a 30, ben consci però che numeri molto elevati (es sopra il 10) sono difficilmente riscontrabili, e per tanto le nostre analisi si sono focalizzate maggiormente su questo sotto-intervallo Stesso discorso lo puoi fare per lambda e mu In generale secondo me è molto importante indicare i range selezionati ed i motivi che ci hanno portato fare quelle scelte, ed in particolare le motivazioni dietro ad analisi specifiche

2.2 Lognormal Scenario

2.2.1 grafici