Bulletin of the *Transilvania* University of Braşov • Vol 7(56), No. 2 - 2014
Series III: Mathematics, Informatics, Physics, 183-192

AN ALGORITHMIC APPROACH OF RETRIAL QUEUING SYSTEMS WITH ONE SERVING STATION PART II: THE IMPLEMENTATION OF THE SIMULATION ALGORITHM

Ion FLOREA[1] **and Corina-Ştefania NĂNĂU**[2]

Abstract

Many real world systems are modeled by retrial queuing system. Analytical formulas for this class of systems are complicated and address only particular cases. By algorithmic approach, one studies through simulation the cases that are not studied analytically. In this paper we present the implementation of the simulation algorithm.

2000 *AMS Classification:* 60K25, 65C20, 68U20.
Key words: retrial queuing system, simulation algorithm, implementation.

1 Introduction

In this paper, we consider a retrial queuing system having only a server. Such a system consists of a source of customers, a serving space and an orbit. The serving space contains a server and a queue, with a limited number of places. Arriving customers in the system require server service. If an arriving client finds the server come free, he is served immediately. If the server is busy and there are more free places in the queue, the client is placed in the queue. Otherwise, with a certain probability the customer leaves the system permanently (without being served) or he is transferred on the orbit with a complementary probability. Customers that will return for service, from time to time, are placed in the orbit that is randomly generated. Such a client can't see the server state. If the server is free, he will receive the service. Otherwise it will be reintroduced in the orbit or it will leave the system. Such an event is called a retrial. We can also assume that the number of attempts by a client to obtain the server service can not exceed a

[1]Faculty of Mathematics and Informatics, *Transilvania* University of Braşov, Romania, e-mail: ilflorea@gmail.com
[2]Faculty of Mathematics and Informatics, *Transilvania* University of Braşov, Romania, e-mail: cory2512@yahoo.com

maximum value. In some regards, the orbit is like a queue, in that customer spends time waiting to be served. At the end of service the client exits the system (**Figure 1**).

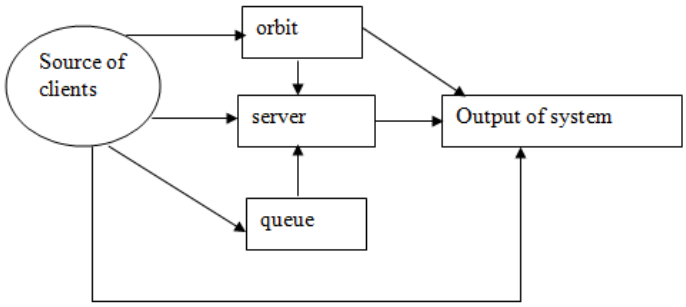

Figure 1: Retrial queuing system structure

In [5] a simulation algorithm for this system is presented. Below we present the simulator corresponding to the algorithm. The simulator was developed based on object-oriented programming techniques, and the C# programming language was chosen (see [7]).

The application can be divided into two parts: the Graphic User Interface part (GUI) and the business logic part (the implementation of the algorithm)

2 The description of the algorithm implementation

In [5], the entities of the simulation are detailed using the pseudocode. In the following, we briefly describe the model entities and the simulation mechanism.

The customers arriving in the system are divided into three categories: the customers arrive when the server is free or busy, but there is a vacancy into the queue; the customers arrive when the server is busy, the queue is full and they will come back to be served (they are placed in the orbit in this case); the customers arrive when the server is busy, the queue is full and they will not return to be served (leave the system in this case).

For all customers we generate the time value between two consecutive arrivals, denoted by *IntArriv*. We also generate the service time, denoted by *Stime*, for the clients in the first and in the second category.

The global variable *Atime* contains the event time of the next arrival. Initially, *Atime* is set to 0 and after any generations, the *IntArriv* value is added to it.

The customers who arrive in the system and find the server busy and the tail full, can be divided into two categories. We denote by 1 the event of lagging system by the customer and his placement on the orbit (with probability p) and we denote by 2 the event of his output from the system (with probability *1-p*). The category corresponding to a newly arrived client of this type is thus a random Bernoulli variable:

$$\text{B:} \begin{pmatrix} 1 & 2 \\ p & 1-p \end{pmatrix}$$

The time interval after which the clients will return to be served, denoted by *IntRet*, and the maximum return number, denoted by *NoRet*, are generated in addition for the clients who remain in the system and they are placed on the orbit.

IntArriv, *Stime* and *IntRet* are selection values for some given random variables, that can be generated by computer, and *NoRet* is a random number, less or equal with a given number *NoMaxRet*.

Ctime represents the event time for finishing service of a client; it means that it represents the server clock. If the server is free and there is no customer to be served, then *Ctime*=∞.

Tsc variable contains the service time for the current client.

The **queue** entity is characterized by: *nc* variable and *Ts* vector. *nc* variable indicates the number of customers in the queue at a time; it can not exceed a maximum given value. *Ts* vector contains the values for serving times of the clients in the queue.

The queue is held on the FIFO principle. In this way *Ts(1)* corresponds to the first element, *Ts(2)* corresponds to the second element, and so on, *Ts(nc)* corresponds to the last element.

The *orbit* entity is characterized by: *no* variable, which indicates the number of clients in the orbit at a time and *To* vector.

An element $To(i)(i = 1..no)$ corresponds to the i-th client from the orbit and it consists of three fields: number of remaining, next return time value and the next service time for the customer.

If we denote $To(i) = (To(i).Rev\_ram, To(i).Time\_Rev, To(i).Time\_serv)$, then $To(i).Time\_Rev < To(i+1).Time\_Rev, (i = 1, .., no-1)$. This means that the clients in the orbit are sorted based on the value of the next return.

The algorithm presented in [5] is based on the rule of "the next event" or "the minimum time". There are three possible types of events:

- An arrival in the system, if $min(Atime, Ctime, To(1).Time\_Rev) = Atime$;
- The finish of a service, if $min(Atime, Ctime, To(1).Time\_Rev) = Ctime$;
- A return of the first client in the orbit, if $min(Atime, Ctime,$ $To(1).Time\_Rev) = To(1).Time\_Rev$.

If at some point, one of the three variables will contain the value of the time for the following event in the system, the variable *Ltime* will contain the value of the time for the last event in the system.

The processing of an arrival consists of:

- if the station is lazy, the client is served immediately and in this case, the total laziness time of the station, denoted by *Tlen*, is updated;
- if at the moment of the arrival the station is busy and the queue is not full, the client is placed into the queue (incrementing of the *nc* and *Ts(nc)* get *Stime* value) and the total waiting time in the queue of the customers, denoted by *Twq*, is updated too;

- if at the arrival moment, the station is busy and the queue is full, the client is placed in the orbit; the total waiting time into the queue of customers is updated;

- for customers who remain in the system and are placed in the orbit, besides service time (denoted by *Stime*), the time after which they return to be served (denoted by *IntRet*) and the maximum number of returns (denoted by *NoRet*) will be generated.

Ending serving a client consists of:

- the total waiting time in the queue of customers, the total working time (*Tserv*) and the total number of customers served by the station (*Tnrserv*) are updated;

- if the queue is nonempty, a new client is served and the queue length is decremented. Otherwise the station enters into laziness and *Ctime* becomes ∞.

Returning the first customer from the orbit consists of:

- if the station is lazy ($Ctime = \infty$), the first client on the orbit will be served; total number of the clients placed in the orbit (*Tnrorb*), total time spent in orbit by the clients (*Torb*) and station laziness time are updated. After that, the client is deleted from the orbit.

- if the station is lazy and the queue is not full, the client is placed into the waiting queue (the incrementing of *nc* and *Ts(nc)* get $To(1).Time\_serv$ value); total number of clients placed in the orbit, total time spent in the orbit by the clients and total waiting time into the queue of the customers are updated; the client is removed from the orbit.

- if the station is lazy and the queue is full, $To(1).Rev\_ram$ is decremented; if it becomes 0, the client is removed from the system and total spent time in the orbit by the clients and total number of the clients placed into the system orbit are updated. Otherwise, a new return time is generated and the client is re-inserted on the orbit.

The implementation of the algorithm is composed of the following classes: **Client**, **ServerActivity**, **Utils** and **Program**. For better flexibility the **Client** class implements the **IClient** interface.

The **Client** class contains three properties and a method. The properties of the client are: the number of remaining returns, denoted by Rev_Ram, which represents the number of customer from orbit trials to get the station service; the return time, denoted by $Time\_Rev$, which represents the time interval after which the customer will come back to be served; the service time, denoted by $Time\_Serv$, which represents the time it takes to serve the customer.

The method in **Client** class is called **InsertInOrbit** and it returns a list of **Client** objects. This is the list of the clients on the orbit, after adding the current client. This list is ascending sorted by the return time of customers.

Because during the algorithms, generating random numbers and variables is used repeatedly, we created a class named **Utils** which contains different methods for generating entities like these. This class already contains a method which calculates the minimum between three real numbers.

The methods in **Utils** class are: **readDouble** that returns a selection value of negative exponential variable; **readInt** that returns a random integer number, less than a number given as parameter; **readBernoulliVariable** that generates a selection value for the Bernoulli variable and it has as an argument a given probability; **min** that returns the minimum of the three real numbers given as parameters.

The class **ServerActivity** contains all the logic of the application and implements all the algorithms presented in detail in [5]. In this way, it simulates all the server activity through the system. **ServerActivity** class consists of 23 properties and 5 methods.

The properties of this class are:

- the current number of clients in the queue (nc);
- the time for the last event that happened into the system ($Ltime$);
- the time of customer service termination ($Ctime$);
- total number of served clients ($Tnrserv$);
- total number of the clients from the orbit ($Torb$);
- total working time of the server ($Tserv$);
- total laziness time of the server($Tlen$);
- total waiting time into the queue of the clients (Twq);
- the current client service time (Tsc);
- Ts vector that contains service time values for customers from the queue;
- the number of clients from the orbit at a time (no);
- the size of the server queue (d);
- the vector of clients from the orbit (To);
- the interval between two consecutive arrivals of the clients into the system ($IntArriv$);
- the service time of a client ($Stime$);
- the arrival time of a client ($Atime$);

- the number of the clients who arrived into the system (*Nra*);
- the time interval during which a customer that entered the orbit returns (*IntRet*);
- the number of clients that canceled the system without being served (*NrLeaveSyst*);
- the parameter for random generation of *IntArriv* (*lambda*);
- the parameter for random generation of *Stime* (*miu*);
- the parameter for random generation of *IntRet* (*teta*).

The methods in **ServerActivity** class are the following:

Init is the method that initializes the system state, providing initial values for all 23 properties. **UpdateArriv** is the method that processes an incoming new client into the server, treating three cases presented above, in this section. Also, the laziness time of the system, the service time interval between two arrivals and the number of customers arriving into the system are updated. **UpdateFinServ** is the method that processes the serving of a client. Two cases described above in this section are treated. **UpdateRetrial** is the method that processes a customer return from the orbit to be served. In this method, the following situations are treated : if when the client returns from orbit, the server is free, he will be served immediately; if when the client returns from orbit, the server is busy, then he can return to the orbit with a new value for his return time or he can leave the system if his number of returns is 0; after each return of the client from the orbit, the number of his returns is decremented with one unit. **RetrQueuingSystOneStat** is the main method that calls the other four methods discussed above. At the beginning, the state variables of the system are initialized. While the current number of clients in the system is less than the maximum number of arrivals, we can call one of the three methods: **UpdateArriv**, **UpdateFinServ** or **UpdateRetrial**, according to the algorithm presented in [5].

3 Graphical interface description

The application offers a graphical interface that is user friendly. The graphical interface part uses a "windows form" that looks as in the following image. (**Figure 2**)

Using GUI we select the type of the random values which selection variables are *IntArriv*, *Stime* and *IntRet*. We will use the exponential negative variable. The generation parameters for these variables are introduced through the user interface. The user will also introduce the generation parameter for Bernoulli variable. After the introduction of these values, the user will press **LoadAlgorithm** button. The action of this button is to execute the **RetrQueuingSystOneStat(Tnra)** method from the class **ServerActivity** and to return in a dialog box the number of seconds in which this method executed. In fact, at that moment,

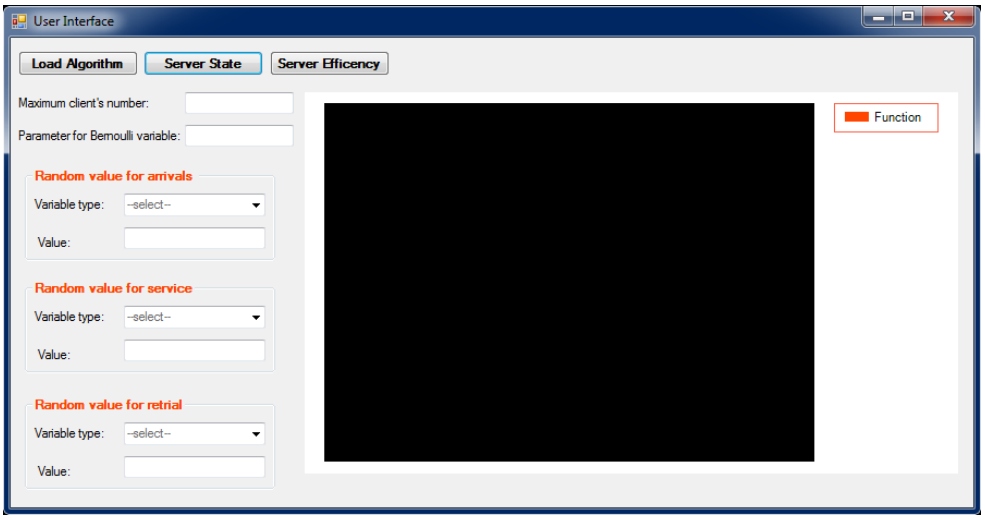

Figure 2: Graphical interface of the application

the number of clients that we have introduced have arrived into the system and they have tried to access the server to be served. At this point, the efficiency factors of server are also calculated: the average waiting time into the queue of the customers ($MTwq$), the average waiting time in the orbit of the customers ($MTOrb$), the average serving time of the customers (Mts), the laziness time of the server ($Clen$) and the average queue length ($Mqueue$). To view these efficiency factors of the system, the user will click on the **Server Efficiency** button and a dialog box which displays these values will appear, and in the black section in the central part of the "windows form" these values will appear as a *Bar Diagram*. We can see this in **Figure 3**.

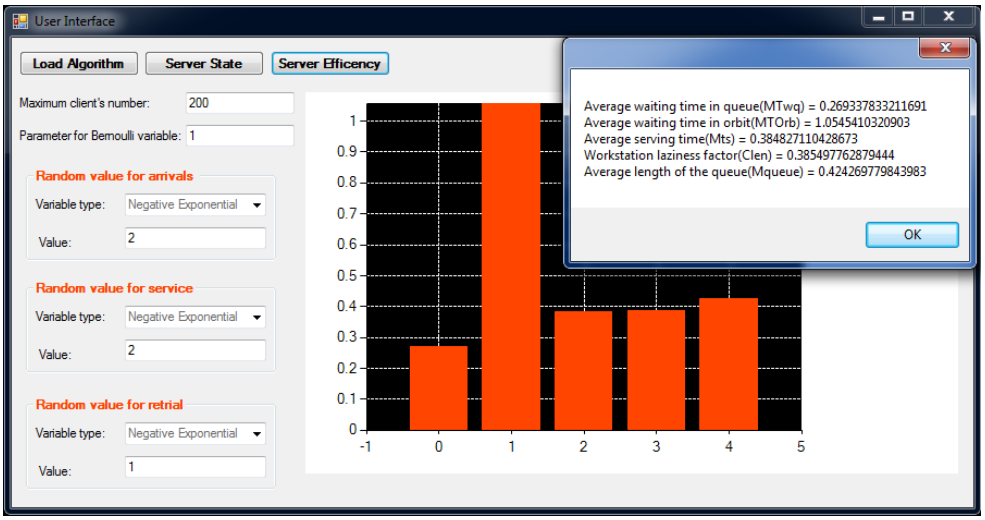

Figure 3: Efficiency factors of the system

The vertical axis represents the time and on the horizontal axis we have the 5 efficiency factors of the system. If the user clicks the **Server State** button, he can see the state of other variables of the system, at the ending of algorithm execution. These values can be seen in **Figure 4**.

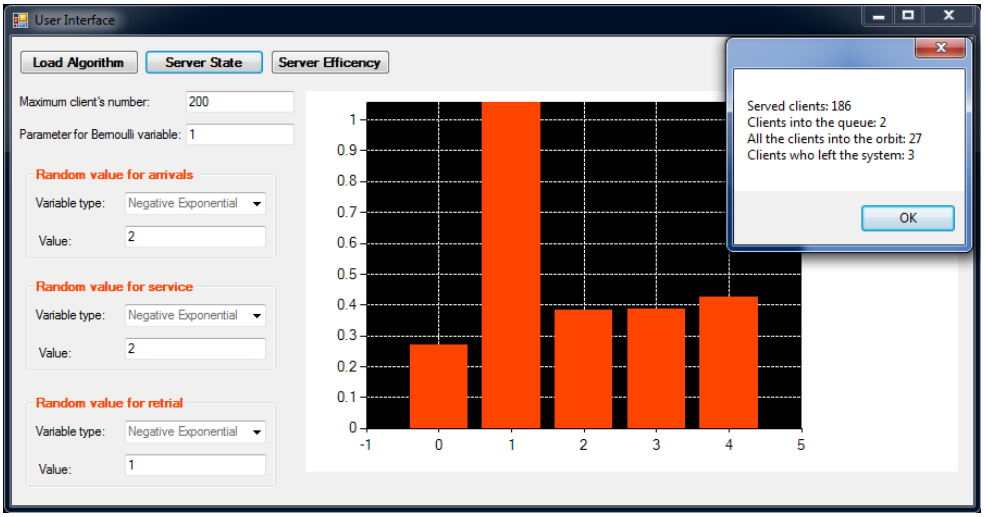

Figure 4: Server state at the end of the execution

All the code of the application can be found in Annexe 1, at the following address `http://www.unitbv.ro/fmi/DepartamentulMI/Colectiv/FloreaIon.aspx/Annex1ArtBullTranUnivN02_2014`

4 Validity of the algorithm and practical considerations

In the following lines we consider 30000 arrivals simulated. The inter-arrival time, serving time and the placement time on the orbit are exponentially negatively distributed with the parameters lambda = 1, miu = 2, teta = 2. The above mentioned implementation was used to obtain the statistics for four scenarios

Case i) We consider that:

- the queue length is ∞; (any arrived client enters the queue if the server is busy)
- random variable B described above is B: $\begin{pmatrix} 1 & 2 \\ 0 & 1 \end{pmatrix}$, which means that the probability of a customer entering the system being placed in the orbit when the server is busy is 0.

This model is equivalent to a model without calling customer return, exp(lambda)/exp(miu)/1:(1,FIFO). This model is studied by simulation in [4].

The results obtained by the simulator execution model presented in the article are: number of served clients is 29998, number of clients who left the system is 0 and the number of clients into the queue at the end of the execution is 2. Because the queue size is ∞ and the probability of entering the orbit is 0, every new client entered the queue and no client entered the orbit. In this case, the average waiting time in queue($MTwq$) is 0.80239, the average serving time (Mts) is 0.74834, the workstation laziness factor ($Clen$) is 0.53699 and the average length of the queue ($Mqueue$) is 0.77647. They are approximately equal to those obtained for the simulated system in [4].

Case ii) B: $\begin{pmatrix} 1 & 2 \\ 0 & 1 \end{pmatrix}$, d = 0 (no customer can enter the queue).

The system is studied analytically in [6]. The results obtained by simulation are: number of served clients is 23468, number of clients who left the system is 6532. Because the queue size is 0 and the probability of entering the orbit is also 0, no client entered the queue or the orbit. In this case, the average waiting time in queue($MTwq$) is 0, the average serving time (Mts) is 0.45437, the workstation laziness factor ($Clen$) is 0.62532 and the average length of the queue ($Mqueue$) is 0. They are approximately equal to those presented in [6].

Case iii) B: $\begin{pmatrix} 1 & 2 \\ 0.5 & 0.5 \end{pmatrix}$, d = 100 (case iii.1), d=200 (case iii.2).

The results obtained by simulation for the number of customers in orbit are:

In **case iii.1**, the number of served clients is 29955, number of clients who left the system is 43, the number of clients who entered the orbit is 75, the average waiting time in queue($MTwq$) is 0.38322, the average waiting time in orbit($MTOrb$) is 1.29604, the average serving time (Mts) is 0.50485, the workstation laziness factor ($Clen$) is 0.49381 and the average length of the queue ($Mqueue$) is 0.38422. In **case iii.2**, the number of served clients is 29986, number of clients who left the system is 14, the number of clients who entered the orbit is 49, the average waiting time in queue($MTwq$) is 0.34879, the average waiting time in orbit($MTOrb$) is 1.92451, the average serving time (Mts) is 0.45688, the workstation laziness factor ($Clen$) is 0.57996 and the average length of the queue ($Mqueue$) is 0.32053. That respects the logic of the model: if more clients can be placed in the queue, then fewer clients will reach the orbit.

Case iv) B: $\begin{pmatrix} 1 & 2 \\ 0.5 & 0.5 \end{pmatrix}$ (case iv.1), B: $\begin{pmatrix} 1 & 2 \\ 0.75 & 0.25 \end{pmatrix}$ (case iv.2) and d = 100.

The results obtained by simulation for the number of customers in orbit are:

In **case iv.1**, the number of served clients is 29955, number of clients who left the system is 43, the number of clients who entered the orbit is 75, the average waiting time in queue($MTwq$) is 0.38322, the average waiting time in orbit($MTOrb$) is 1.29604, the average serving time (Mts) is 0.50485, the workstation laziness factor ($Clen$) is 0.49381 and the average length of the queue ($Mqueue$) is 0.38422. In **case iv.2**, the number of served clients is 29973, the number of clients who left the system is 27, the number of clients who entered the orbit is 82, the average waiting time in queue($MTwq$) is 0.28975, the average waiting time in orbit($MTOrb$) is 0.59761, the average serving time (Mts) is 0.49266, the

workstation laziness factor (*Clen*) is 0.52517 and the average length of the queue (*Mqueue*) is 0.27926. That respects the logic of the model: if the probability of a customer caming into the system being placed in the orbit increases, then the total number of customers in the orbit will increase.

5 Conclusions

In this article we present the object oriented for a simulation algorithm for queuing systems with a single service station and returning customer service. This class of waiting systems models many real-world systems, presented in the introduction. Also, these systems are analytically studied only for some distributions of the time between two consecutive arrivals of the service time and the return time to get service for the service station. In many analytical studies, mathematical formulas are complicated and difficult to use in practice. For these reasons, such a simulator is both needed and useful. Also, the system studied by simulation extends the analytic system by introducing a queue at the service station and considering that each client placed in orbit can have a number of randomly generated returns.

References

[1] Artalejo, J.R. a.o., *Retrial queuing systems: A computational approach*, Springer, 2008.

[2] Artalejo, J.R. a.o., *Standard and retrial queuing systems: A comparative analysis*, Matematica Complutense **15** (2002), no. 1, 101-129.

[3] Devroye, L., *Non-uniforme random variate generation*, Springer Verlag, New York, 1986.

[4] Florea, I., *One algorithmic approach of first-come-first-served queuing systems*, Bucharest University Annals, Informatics, **49**, 41-58, 2000.

[5] Florea, I., a.o., *An algorithmic approach of retrial queuing system with one serving station Part I: The description of the simulation algorithm*, Bulletin of the Transilvania University of Braşov, Vol. 6(55), no. 2, 95-106, 2013.

[6] Krishna, K. B. a.o., *The M/G/1 retrial queue with Bernoulli schedules and general retrial times*, Computers and Mathematics with Applications **43** (2002), 15-30.

[7] Skeet, J., *C# in Depth, Second Edition*, Manning, Stamford, 2010.