DOI: 10.1515/auom-2015-0007
An. Şt. Univ. Ovidius Constanţa

VERSITA
Vol. 23(1),2015, 83–98

A simulation algorithm for a single server retrial queuing system with batch arrivals

Ion FLOREA and Corina-Ştefania NĂNĂU

Abstract

Many systems of real word are modeled by retrial queuing system with batch arrivals. Analytical formulas for this class of systems are complicated and address only particular cases. The paper presents a study approach for this kind of systems, based on discrete event simulation. It is shown that the given algorithm has a polynomial complexity. Also, the object-oriented design we used for implementation is sketched.

1 Introduction

In this paper, we consider a retrial queuing system with batch arrivals having only a server. This kind of server consists of a source of customers, a serving space and an orbit (figure 1). The serving space contains a server and a possible queue with a limited number of places.

Customers arrive into the system in batches; a batch is a set (vector) of customers. Batch size is a given discrete random variable, which will be described in the next section. If in the arrival time of a batch of customers, the server is free, one of the customers in the batch is served immediately. The other clients are being introduced in the server queue, or they are placed in the orbit, or they are giving up the service. If the server is busy, the customers are

Key Words: Retrial Queuing system, Batch Arrivals, Simulation Algorithm, Polynomial Complexity.
2010 Mathematics Subject Classification: Primary 46G05, 46L05; Secondary 47A30, 47B47.
Received: 8 May, 2014.
Revised: 10 June, 2014.
Accepted: 30 June, 2014.

being placed in the queue (if the queue is not full), or they are being placed in the orbit, or they are giving up to be served.

The customers that will come back to be served, after certain intervals of time randomly generated, are placed in the orbit. This kind of customer can't see the server state. If the server is free, it will receive the service, otherwise it will be re-inserted into the orbit or it will leave the system. Such an event is called a retrial. We can also assume that the number of attempts of a client to get the server service can not exceed a maximum value.

In some regards, the orbit is like a queue, in that customer spend time waiting to be served. At the finish of service the client exits the system (figure 1).

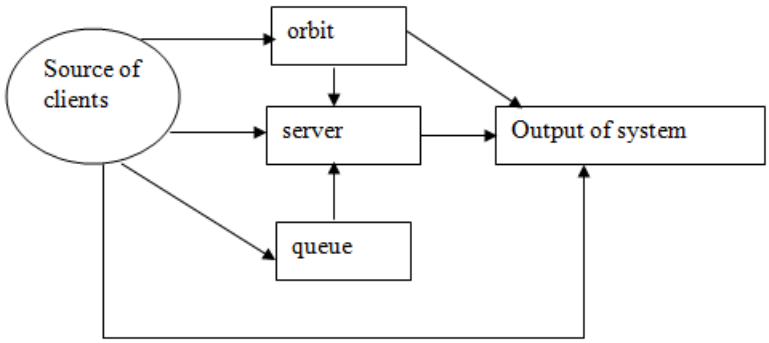

Figure 1: Retrial queuing system structure

Remarks 1.

i) Queuing systems can be studied analytically (see[11], [12]) or by simulating algorithms. Using mathematical methods, formulas for the system efficiency factor are obtained, available in particular cases. Through simulation, such systems can be studied in situations where does not exist analytical formulas.

ii) In classical queuing theory, when a client arrives in the system, if the server is idle it is immediately served otherwise it is queued. When the server becomes free, if the queue is not empty, a customer is picked up from the queue and it will be served. Otherwise, the server becomes lazy. In this kind of systems, the phenomenon of returning the customer does not appears. Such a system can be considered a returning system, considering zero probability that a client to be placed in the orbit.

iii) A client from the orbit can not monitor the server; while the server is free, it could be customers in the orbit that do not ask to be served. So, there is a delay in time until a customer in orbit realizes that the server is free and

begins the service.

iv) The order of serving customers in orbit is random; it depends on the random order of returning customers to be served.

The characteristics of this system which we will be interested in calculating are quantities like the mean number of customers in the system, the fraction of the time that the server is idle, the mean length of time a customer can expect spend in the system to receive service, and so on (the efficiency factors of the system).

An example of such a system is an Ethernet network. In the context of the CSMA/CD method, the clients are the messages to be transmitted over the network communications environment, arriving in batches, and the server is the communication environment itself. When a host wants to send a message over the communications, it can do it only if the communications are free; otherwise the collision phenomenon will appear. If the communication environment is busy, the host will return after a random period of time. All the messages to be sent by the host are placed into the orbit.

Staging systems can be studied analytically; such mathematical formulas exist for certain cases, even for the traditional approach. Except for a few simple models (see [1], [2], [3], [4]) retrial queues are generally difficult to analyze analytically. The formulas obtained describe only certain efficiency factors. Because there are ways to generate any type of random size (see [6]), by a simulation algorithm it can be studied any kind of system.

In [9] we present a simulation algorithm for a staging system in the traditional approach, and in [10] we present such an algorithm for systems with returns, with individual arrivals of the clients. Because a system with clients return extends a classical one (without return), and a batch with the size equals with 1 can be considered one client, simulation algorithm presented in this paper extends those in the works we mentioned above, but considering the arrival in batches and the return of the clients.

2 The model entities and the simulation mechanism

Simulation of the arrivals. The interval between two consecutive batches of customers is a continuous random variable denoted by *IntArriv*. The global variable *Atime* contains the event time of the next arrival. Initially, *Atime* is set to 0 and after any generation the *IntArriv* value is added to it.

Also, the batch size denoted by *DimBatch* is a discrete random variable of the form:

$$\begin{pmatrix} 1 & 2...mb \\ a_1 & a_2...a_{mb} \end{pmatrix}$$

where mb represents the maximum number of clients in a batch and a_i *(i = 1, mb)* represents the probability that the arrived batch to have the size equals with i.

When a batch arrives in the system can occur one of the situations:

2.1) The server is free; we denote by nb the number of the customers in the batch, $nb \leq mb$

- a client is selected from the batch for being served; the selection can be done using two strategies: the first client in the batch or a random one is selected.
- because the server is free, the queue is empty; if $nb \leq nqmax$, where $nqmax$ is the maximum number of the clients in the queue, all the clients in the batch are inserted into the queue, otherwise $nqmax$ clients are inserted in the queue, and the other clients are inserted into the orbit or they leave the system. The mechanism for this situation will be presented later.

2.2) The server is busy; we denote by nq the number of the customers in the queue

- if $nqmax = nq$ means that the queue is full and all the clients in the arriving batch are placed in the orbit.
- otherwise, if $nq < nqmax$, $nqmax$ - nq clients from the batch are introduced in the queue, and the others will be introduced in the orbit or they will leave the system.

For all the customers in the batch who don't leave the system, it generates service time denoted by *Stime*.

Placing customer in the orbit. We denote by *1* the event of the remaining in the system of the customer and his placement in the orbit (with the probability p) and with *2* the event of his exiting the system (with probability *1-p*).

The category corresponding to a newly arrived client of this type is thus a random Bernoulli variable:

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

When a batch arrives, for each client that is not served immediately or it is not placed in the queue, the system generates a value of the variable selection, denoted by B. If $B = 1$, then the customer is placed in the orbit. Otherwise, the customer leaves the system.

In addition, it is generated the time interval after which the customers will return to be served, denoted by *IntRet*, and the maximum number of returns, denoted by *NoRet*, for those who remain in the system and are placed in the orbit.

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

Ctime represents the event time for termination of service of a customer (meaning the server clock). If the server is free and there are not clients to be served, then $Ctime = \infty$.

Tsc variable contains the length of service for the current customer.

In addition to the variables *nq* and *nqmax*, the **queue** entity is characterized by *Ts* vector which contains the values for the serving times of the customers in the queue. The queue is organized on the principle of FIFO, meaning that *Ts(1)* corresponds to the first element, and so on, *Ts(nq)* corresponds to the last element.

The **orbit** entity is characterized by:

- *no* variable, which indicates the number of the clients in the orbit at a time;
- *To* vector; *To(i), with (i = 1,...,no)*, corresponds to the i-th customer in orbit and it consists of three fields: the number of remaining returns, the value of time for the next return and the time for the client service. If we denote:

  $$To(i) = (To(i).Rev\_Ram, To(i).Time\_Rev, To(i).Time\_Serv),$$

  then

  $$To(i).Time\_Rev < To(i+1).Time\_Rev, \text{ with } (i = 1, \ldots, no\text{-}1).$$

  That means that the customers in the orbit are sorted ascending by the time for the next return.

The algorithm which we present is based on the 'next event' rule or 'minimum time' rule. There are three possible types of events:

- a system arrival, if $min\{Atime, Ctime, To(1).Time\_Rev\} = Atime$;
- a service finishing, 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 time value for the next event in the system, the *Ltime* variable will contain the time value for the last event in the system. Processing of an arrival consists of:

- If the station is lazy, one of the customers in the batch is served immediately; the client selection is conform the presented things in 2.1; in this case the total laziness time, denoted by *Tlen*, is updated.
- If in the arrival time the station is busy and the queue is not full, the customers in the batch are placed in the waiting queue or in the orbit (according to 2.2); the total time of waiting in the queue, denoted by *Twq* is updated.
- For the customers who remain in the system and are placed in the orbit it is generated and maintained *IntRet* and *NoRet*.

Finishing a client service consists of:

- total waiting time in the queue, total working time (*Tserv*) and number of served clients (*Tnrserv*) are updated;
- if the queue is not empty, then a new client is served and the queue length is decremented. Otherwise, the station becomes lazy, that *Ctime* $= \infty$;

Returning of the first client in the orbit consists of:

- if the station is lazy (*Ctime* $= \infty$), he will be served. The number of the clients in the orbit (*Tnrorb*), the total time spent in the orbit by the clients (*Torb*), the station laziness time will be updated and the current client will be removed from the orbit;
- if the station is busy and the queue is not full, then the client is inserted in the waiting queue (*nq* will be incremented and *Ts(nq)* will get $To(1).Time\_Serv$ value); The number of the clients in the orbit, the total time spent in the orbit by the clients, the total waiting time in the queue is updated and the current client will be removed from the orbit.
- if the station is busy and the queue is full, then $To(1).Rev\_Ram$ value is decremented; if this value becomes 0, the client is removed from the system and the total waiting time in the orbit and the number of clients in the orbit are updated; otherwise another return time will be generated and the client will be put again in the orbit.

Remarks 2.

i) The simulation runs until the number of arrivals generated reaches a given value, denoted by *Tnra*.

ii) If we define processes like arrivals, services or returns of the clients like cycle, all the simulation consists of repeating these cycles.

iii) If we assume that the inflow is less than the serving flow, the serving number will not exceed the value of *Tnra*. Also, if the number of simulated arrivals has a large enough value, then almost all customers will be served. In this way, the customers that are served immediately upon arrival are also taken into consideration.

iv) Because for each customer who returns for being served, the number of returns has a maximum value denoted by *NoMaxRet*, we consider that the total number of returns does not exceed the value *NoMaxRet*mb*Tnra*.

v) We can admit that the number of cycles of the simulation do not exceed a value of the form *c * Tnra*, where *c* is a constant value.

At the end of the simulation we determine the efficiency factors of the system:

- *MTwq* represents the average waiting time in the queue of customers: *MTwq =Twq/Tnrserv* (Remarks 1);
- *MTOrb* represents the average waiting time in the orbit of customers: *MTOrb =Torb/Tnrorb* (Remarks 1);
- *Mts* represents the average serving time of a customer: *Mts=Tserv/Tnrserv*;
- *Clen* represents the workstation laziness factor: *Clen=Tlen/Ltime*;
- *Mqueue* is the average length of the queue: *Mqueue=Twq/Ltime*.

3 The algorithm's description

The following procedure describes in pseudocode the main part of the simulation algorithm. The fine-grain actions are grouped as procedures called from the main procedure. Due to lack of space, describing the procedures pseudocode is not included in the article; this is presented in detail at: *http://www.unitbv.ro/Portals/19/departament/colectiv/florea_ion/PseudocodProcArtMACOS_FloreaNanau*.

The main procedure named *RetrQueuingSystOneStat* is presented below:

```
1: procedure RetrQueuingSystOneStat(Tnra)
2:     //generated parameters for the interval between two consecutive
3:     //arrivals, for service time, for time to spend in the orbit, the
```

```
4:  //probability to enter in the orbit and the size of the batch
5:  Read();
6:  //Initial state
7:  Nra ← 0; //the number of the arrivals
8:  nq ← 0; //there is no client in the queue
9:  Ltime ← 0; //time value for the last event is 0
10: Ctime ← ∞; //the server is free
11: Tnrserv ← 0; //total number of services is 0
12: Tnrorb ← 0; //number of clients in the orbit is 0
13: Torb ← 0; //total time spent in orbit by clients is 0
14: Tserv ← 0; //total service time for the clients is 0
15: Tlen ← 0; //total server laziness time is 0
16: Twq ← 0; //total waiting time in the queue for the clients is 0
17: no ← 0; //number of clients in the orbit is 0
18: To(1).Time_Rev ← ∞; //time for the first client return is ∞
19: //first batch arrival is generated
20: Gen(IntArriv, DimBatch); Atime ← IntArriv;
21: //simulation lasts while number of processed arrivals don't exceed a
22: //given value
23: while Nra <=Tnra do (1)
24:     if min{Atime, Ctime, To(1).Time_Rev} = Atime then (1)
25:         Update_Arriv();
26:         //next event is an arrival
27:     else(1)
28:         if min{Atime, Ctime, To(1).Time_Rev} = Ctime then (2)
29:             //next event represents the end of a service
30:             Update_Fin_Serv();
31:             //next event is a return
32:         else(2)
33:             Update_Retrial();
34:         end if; (2)
35:     end if; (1)
36: end while(1)
37: //Calculation of efficiency
38: MTwq ← Twq/Tnrserv; //average waiting time in the queue
39: MTOrb ← Torb/Tnrorb; //average waiting time in the orbit
40: Mts ← Tserv/Tnrserv; //average serving time
41: Clen ← Tlen/Ltime; //workstation laziness factor
42: Mqueue ← Twq/Ltime; //the average length of the queue
43: Write(MTwq, MTOrb, Mts, Clen, Mqueue);
44: end procedure
```

The *Update_Arriv* procedure simulates a new arrival in the system. We take into consideration the following cases:

1) The station is lazy; in this case, the values for overall laziness of the station and the total time spent in orbit by the clients are updated. Also, a customer is selected from the batch to be served. The other clients are placed into the queue or into the orbit, considering various possible situations.

2) The station is busy; in this case, total waiting time and total time spent into the orbit by the customers is updated. The customers from the batch are placed into the orbit or into the queue, depending on the size of batch and the number of available places into the queue.

The *PutInQueue* procedure generates the service time of a customer and inserts it in the queue

The procedure *InsertInOrbit* achieve the orbit placement of a client. We generate the time after that it will take place the first return and the total number of possible returns. The client is introduced into the data structure corresponding to the orbit, with the expectation of meeting the increasing order of laziness times of the customers.

The selection of a customer from the batch for being served is realized by *Select* procedure in two ways:

1) the first client in the batch is selected $(ClServ = 1)$;

2) the number of the client to be selected from the batch is randomly generated $ClServ = RND(DimBatch), ClServ = 1, ..., DimBatch$

The *Update_Fin_Serv* procedure processes finishing of a service; there are two possible situations: the queue is not empty and the first client in the queue will be served or the queue is empty and the server becomes lazy.

The procedure updates: the total station serving time, the total number of served clients, total waiting time of customers into the queue. We treat two situations:

1) the first client from the queue will be served and it will be removed from the queue

2) the queue is free and the station becomes lazy

The *Update_Retrial* procedure processes the return of the first customer in the orbit to be served. If he finds the server free, then he will be served. In other situations, if the initial return number is not exceeded, then another return time will be generated. Otherwise, the client leaves the system without being served.

The procedure addresses two situations:

1) The station is free. The total laziness time of the station is updated and the first client from the orbit is served and removed from there.

2) The station is busy. If the total number of returns doesn't exceed the value generated when the client arrives, a new time for the client return is

generated and the client is placed into the orbit again. Otherwise, the client will leave the system.

Algorithm complexity

We denoted by *Tnra* the number of the batches arrived into the system. In this case, the maximum number of the clients arrived is *mb*Tnra*, where *mb* represents the maximum size for a batch.

For estimating the complexity, we consider that the clients in the system reach this maximum number.

Remarks 3. The execution of the procedures that generate the varieties is linear (see [5]). They have a polynomial complexity, and in this case it is *O(1)*.

Lemma 1.

*The complexity for the InsertInOrbit procedure is O(mb*Tnra).*

Proof.

We denoted by *no* the number of the clients in the orbit at a certain moment, with $no \leq mb * Tnra$.

In the worst case (see *http://www.unitbv.ro/Portals/19/departament /colectiv/florea_ion/PseudocodProcArtMACOS_FloreaNanau*), the content of the structures while (1) and for (1) is executing *mb*Tnra* times each one (see *http://www.unitbv.ro/Portals/19/departament/colectiv/florea_ion/ PseudocodProcArtMACOS_FloreaNanau*). It results that the complexity is *O(mb*Tnra)*.

Lemma 2.

The complexity for the procedure PutInQueue is O(1).

Proof.

Because its execution is linear, these procedure complexity is O(1) (see *http://www.unitbv.ro/Portals/19/departament/colectiv/florea_ion/ PseudocodProcArtMACOS_FloreaNanau*).

Lemma 3.

*The procedure Update_Arriv has the complexity max(O(mb), O(2*nqmax-mb)).*

Proof.

We have $DimBatch \leq mb$ *and* $DimBatch - nqmax + nq - 1 \leq mb$. In the worst case, the maximum number of appearances is:

On 'Then' branch of the structure (1) the complexity is *O(1)+max(O(mb), O(mb))=O(mb)* (see *http://www.unitbv.ro/Portals/19/departament/colectiv/ florea_ion/PseudocodProcArtMACOS_FloreaNanau*).

On 'Else' branch of the structure (1) the complexity is *O(1)+max(O(mb), O(nqmax)+O(nqmax-mb))=max(O(mb),O(2*nqmax-mb))*
(see *http://www.unitbv.ro/Portals/19/departament/colectiv/florea_ion/ PseudocodProcArtMACOS_FloreaNanau*).

In this case, The complexity of the procedure *Update_Arriv* is calculated *max(O(mb), max(O(mb), O(2*nqmax-mb)))=max(O(mb),O(2*nqmax-mb))*.

Lemma 4.

The procedure Update_Fin_Serv has the complexity O(nqmax-1).

Proof.

We have $nq \leq nqmax$ and in this case, the complexity of the procedure is *O(1)+O(nqmax-1)=O(nqmax-1)* (see *http://www.unitbv.ro/Portals/19/departament/colectiv/florea_ion/PseudocodProcArtMACOS_FloreaNanau*).

Lemma 5.

*The procedure Update_Retrial has the complexity O(mb*Tnra).*

Proof.

We have $no \leq mb * Tnra$. In this case, the complexity is *O(1)+max(O(1)+O(mb*Tnra),max(O(1)+max(O(mb*Tnra,O(mb*Tnra)))))= O(mb*tnra)* (see *http://www.unitbv.ro/Portals/19/departament/colectiv/florea_ion/PseudocodProcArtMACOS_FloreaNanau*).

Theorem 1.

*The complexity for all the algorithm is mb*Tnra*(O(max(O(mb),O(2* nqmax-mb))+O(nqmax-1)+NoMaXRet*O(mb*Tnra)))*

Proof.

For demonstrating the complexity for all the algorithm, we analyze the main procedure named *RetrQueuingSystOneStat*. The two linear structures called inside this procedure have the complexity O(1). In the iterative structure while, we have:

On 'Then' branch labeled by (1), the *Update_Arriv* procedure is called *mb*Tnra* times. Using **Lemma 3**, the maximum number of operations performed in this branch is *mb*Tnra*(O(max(O(mb),O(2*nqmax-mb))))*

On 'Then' branch labeled by (2), the *Update_Fin_Serv* procedure is called *mb*Tnra* times (see **Remarks 2**). Using **Lemma 4**, the maximum number of operations performed in this branch is *mb*Tnra*O(nqmax-1)*

On 'Else' branch labeled by (2), the *Update_Retrial* procedure is called *NoMaxRet*mb*Tnra* times. Using **Lemma 5**, the maximum number of operations performed in this branch is *NoMaXRet*mb*Tnra*O(mb*Tnra).*

Summing all of these, it follows that the maximum number of operations performed by *while* cycle and giving the algorithm complexity is *mb*Tnra* *(O(max(O(mb),O(2*nqmax-mb))+O(nqmax-1)+NoMaXRet*O(mb*Tnra)))*

4 An object-oriented implementation

For implementing the algorithm described in pseudo-code in the previous section, we choose the object oriented programming language C#.

In this way we define a class named *Client* with the following properties: the number of returns from the orbit (called *Rev_Ram*), the time for the next return from the orbit (called *Time_Rev*) and the service time (called *Time_Serv*). This class implements the **IClient** interface that permits the insertion into the orbit of a customer.

Another class, named *RandomGenerator*, that implements the interface *IRandomGenerator* defines a set of methods used for generating random varieties mentioned in the description of the algorithm.

The central class is named *ServerActivity*; its interface contains the following methods: *Update_Arriv, Update_Fin_Serv, Update_Retrial and RetrQueuingSisytOneStat*, which implement the procedures mentioned in the pseudocode description. This class contains instances for the Client class, these instances being grouped in lists. A list of clients represents actually a batch of customers arrived into the system.

The *UserInterface* class represents the user form that displays the charts for the efficiency factors.

In this way a friendly interface is provided as the charts for the system efficiency factors obtained by the execution of the program.

5 Validity of the algorithm and practical considerations

In the following we consider 30000 arrivals simulated. Also, we use the denotations specified around the paper. The inter-arrival time, serving time and retrial time are exponential negative distributed, with λ, μ respectively ν parameters.

The above mentioned implementation was used to obtain the statistics for various scenarios. The resulted statistics are thus reported. The simulation results are averaged over several runs, such that the final results can be considered as representative. We considered five scenarios.

i) Case 1. We consider that: $\lambda = 1, \mu = 2$ and $nqmax = \infty$ (every arrived customer enters the queue if the server is busy); *mb = 1;* B = $\begin{pmatrix} 1 & 2 \\ 0 & 1 \end{pmatrix}$ (the probability for a client that enters the system when the server is busy to be placed into the orbit is 0).

This model is equivalent with a staging system without the client return $exp(\lambda)/exp(\mu)/1 : (\infty, FIFO)$. This model is studied by simulation in [9]. In *Table 1* the obtained results are presented comparative, executing the simulation programs corresponding to this two algorithms.

Efficiency factor	Results obtained with the simulation program with no return	Results obtained with the simulation program with return
The average waiting time in the queue	1.00185	0.99388
The average length of the queue	0.50223	0.49857
The average serving time	0.99547	0.99103
Traffic intensity	0.49629	0.49778

Table 1

Note that the results obtained for a factor of efficiency using the two simulation programs are approximately equal.

ii) Case 2. We consider that: $\lambda = 1, \mu = 2$ and $nqmax = 0$ (no customer enters the queue); $B = \begin{pmatrix} 1 & 2 \\ 0.5 & 0.5 \end{pmatrix}$ (the customers who arrive when the server is busy enters into the orbit or leave the system with the same probability), $mb = 1$. This model is similar to the individual arrivals and customer return model addressed in [10].

In this case, for the average stationary time in the orbit of clients, by executing the two programs, the results obtained are approximately equal.

iii) Case 3. We consider that: $\lambda = 1, \mu = 2, \nu = 1$, $nqmax = 0$; $B = \begin{pmatrix} 1 & 2 \\ 0.5 & 0.5 \end{pmatrix}$; $mb > 1$. This model is analytical studied in [3, 8, 14, 15]; the comparative results obtained by simulation and those analytical obtained are presented in *Table 2*.

Efficiency factor	Results obtained analytical	Results obtained by simulation
The average waiting time in the queue	1.00185	0.99388
The average length of the queue	0.49725	0.49857
The average serving time	1.00742	0.99103
Traffic intensity	0.50149	0.49778
Average stationary time in the orbit	1.51001	1.49751

Table 2

iv) Case 4. We consider that: $\lambda = 1, \mu = 2, \nu = 1$, $qmax > 0$; $B = \begin{pmatrix} 1 & 2 \\ 0.5 & 0.5 \end{pmatrix}$; $mb > 1$. This model is not studied analytically. In *Table 3* the

results obtained by simulation are compared, considering two different values for nqmqx.

Efficiency factor	Values for nqmax = 100	Values for nqmax = 200
The average waiting time in the queue	1.10185	0.99388
The average length of the queue	0.49725	0.49857
The average serving time	1.00742	0.99103
Traffic intensity	0.50149	0.49778
Average stationary time in the orbit	1.23158	0.74609

Table 3

Note that if nqmax increases, the average waiting time in the orbit decreases.

6 Conclusions

In this article we present a simulation algorithm for queuing system with a single service station, with customers arrivals in batches and with unserved client return. This class of waiting systems models many systems in the real-world.

Also, these systems are studied analytic only for certain distributions of the time between two consecutive arrivals, for the service time and the return time, to obtain the service of the station. There is no analytical approach where the server has a queue where are introduced some of the clients that are going to be served immediately after a service finishes.

Thus, we can say that the system studied by simulation extends the analytical studied system. Therefore, it results the need and the utility of such a simulation algorithm.

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, vol. **15** (2002), no. 1, 101-129.

[3] Artalejo, J.R. a.o., *On the Single Server Retrial Queue with Batch Arrivals*, The Indian Journal of Statistics, Vol. **66** (2004), no. 1, 140-158.

[4] Choi, B,D. a.o., *Retrial queues with collision arising from unslotted CSMA/CD protocol*, Queueing Systems Theory Appl. (1992), 335-356.

[5] Cormen, T. H. a.o., *Introduction to algorithms*, MIT Press, Cambridge, 1992.

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

[7] Falin, G.I. a.o, *Retrial Queues*, Chapman and Hall, London, 1997.

[8] Falin, G.I. a.o, *A single-server batch arrival queue with returning customers*, European Journal of Operational Research (2010), 186-179.

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

[10] 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)** (2013), no. 2, 95-106.

[11] Gross, D. a.o., *Fundamentals of queuing theory*, fourth edition, John Wiley & Sons, 2008.

[12] Leahu, A., *Statistical Inference on the Traffic Intensity for the M/M/s Queuing System*, Analele Stiintifice ale Universitatii. Ovidius Constanţa, Vol. **11(1)** (2003), no. 2, 101-104.

[13] 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.

[14] Nawel K. A. a.o., *On the asymptotic behaviour of M/G/1 retrial queues with batch arrivals and impatience phenomenon*, Mathematical and Computer Modelling (2012), 654665.

[15] Yamamuro K., *The queue length in an M/G/1 batch arrival retrial queue*, Queueing Syst (2012), 187205.

[16] Yang, T. a.o. J.G.C, *A survey on retrial queues*, Queueing Systems Theory Appl (1987), 203-233.

Ion FLOREA,
Department of Mathematics and Computer Science,
Transilvania University of Braşov,
Str. Iuliu Maniu 50, 500091, Braşov, Romania.
Email: ilflorea@gmail.com

Corina-Ştefania NĂNĂU,
Department of Mathematics and Computer Science,
Transilvania University of Braşov,
Str. Iuliu Maniu 50, 500091, Braşov, Romania.
Email: cory2512@yahoo.com