Bulletin of the *Transilvania* University of Braşov • Vol 6(55), No. 2 - 2013
Series III: Mathematics, Informatics, Physics, 95-106

AN ALGORITHMIC APPROACH OF RETRIAL QUEUING SYSTEM WITH ONE SERVING STATION PART I: THE DESCRIPTION OF THE SIMULATION ALGORITHM

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

Abstract

Many systems of real word 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. It is shown that the given algorithm has a polynomial complexity.

2000 *Mathematics Subject Classification:* 60K25, 65C20, 68U20
Key words: retrial queuing system, simulation algorithm, polynomial complexity.

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 maximum value. In some regards, the orbit is like a queue,

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

in that customer spends time waiting to be served. At the finish of service the client exits the system (Figure 1).

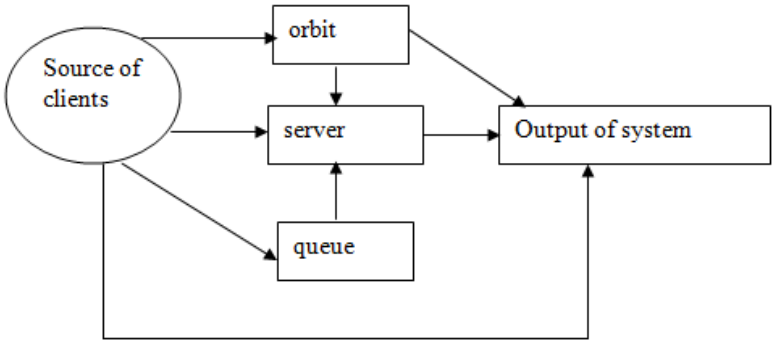

Figure 1: Retrial queuing system structure

Remarks 1.

i) A customer can't monitor server orbit; the server may be free and, at the same time, customers may be in the orbit and they may not require to be served.

ii) There is a delay in time until a customer in orbit realizes that the server is free and begins the service.

iii) The order serving customers in orbit is random, it depends on the random order of returning the customers for being served.

iv) It is considered that the orbit is similar to a queue. When times spent in orbit by customers are null, it means they will return instantly to check server status and eventually be served. In this case, we say that the orbit behaves like a queue with random service discipline, each client having the same probability of being served.

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

Motivation: In real world there are plenty of systems that can be abstracted as described above:

i) Telephone systems - a caller to a busy phone will repeat the call with a certain probability or will give up with a complementary probability. More callers of the same telephone can find it busy and they can be placed in orbit, planning to return with a new attempt.

ii) A store with one cash register - a customer who finds the queue at the cash register is large enough will choose between returning after a random time (placed in orbit) with a certain probability or will give up shopping and will leave the store.

iii) An Ethernet network - in the context of the method CSMA/CD, each host on the network can be considered as a client and the server is the communication medium. When a host wants to send a message on the communication medium it can do it only if it is free, otherwise it will return after a random time. More hosts may be in this situation, all of them being placed in orbit. Thus, we can say that all hosts (clients) who find communication environment (server) being busy will be placed in orbit with probability 1, and the probability of leaving the system is 1 too.

In classical queuing theory, when a client arrives in the system, if the server is idle it is immediately served or it is queued. When the server becomes free, if the queue is non-empty, a customer is selected from the queue for being served. Otherwise, the system become lazy. Otherwise, the server becomes lazy. In these systems, does not occur customer return. In [5] we show a simulation algorithm for a waiting system without customer return. Also, a return system extends a classical one, which will be explained later in this article.

Except for a few simple models (see [1], [2], [6], [7]), retrial queues are generally difficult to analyze analytically. In this paper we will present a simulation algorithm for the waiting system with client return described before. Simulation study has the advantage that does not impose restrictions on some random descriptive variables of the system; so we can study for those systems without analytical formulas for efficiency factors.

2 The model entities and the simulation mechanism

Simulation of the arrivals means that customers who arrive in the system are divided into three categories:

i) customers arriving when the server is free or busy, but there is a vacancy in the queue;

ii) customers arriving when the server is busy, the queue is full and they want to come back for being served (they are placed in orbit);

iii) customers arriving when the server is busy, the queue is full and they will not return to be served (they leave the system).

For all customers, it generates time value between two consecutive arrivals, denoted by *IntArriv*. It also generates service time denoted by *Stime* for the clients in categories i) and ii).

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

Customers who arrive in the system and find the server busy and full queue can be divided into two categories. We note with 1the event of remaining in the system of the client and his placement on the orbit (with probability p) and with 2, the outputs of the system event(with probability 1-p).

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

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

For customers who remain in the system and are placed in the orbit, the time after which they return to be served, denoted by *IntRet* and the maximum return, denoted by *NoRet*, are generated too.

IntArriv, *Stime* and *IntRet* are selection values of given random variables that could be generated using the computer and *NoRet* is a random number less than or equal to a given number, *NoMaxRet* .

Ctime represents the event time for ending a client service, that means server clock. If the server is free and there are no clients to be served, then $Ctime = \infty$.

Tsc variable contains the serving time for the current client.

Queue entity is characterized by:

- *nc* variable indicating the number of customers queued at any given time; it can not exceed a given maximum value.
- *Ts* vector containing serving times values for customers in the queue.

The queue is organized on the FIFO principle, that means TS(1) coresponds to the first element, and so on, Ts(nc) coresponds to the last element.

Orbit entity is characterized by:

- *no* variable indicating number of customers in the orbit at any given time;
- *To* vector; one element To(i) (i=1,..,no) coresponds to the i client in the orbit and it is composed by three fields: number of remaining returns, time value for the next return and the client service time.

If we denote $To(i) = (To(i).Rev\_Ram, To(i).Time\_Rev, To(i).Time\_Serv)$, then $To(i).Time\_Rev < To(i+1).Time\_Rev, (i = 1, .., no-1)$, that means the clients in the orbit are sorted by the time value 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, then the client is served immediately and the total laziness time, denoted by *Tlen* is updated;
- if the station is busy when the client arrives and the queue is not full, then the client is added to the queue and the total waiting time in the queue (denoted by *Twq*) is updated. The *nc* variable is incremented and *Ts(nc)* will receive the *Stime* value (generated service time);
- if the station is busy when the client arrives and the queue is full, then the client is placed in the orbit (a new node will be inserted in the list); total waiting time in the queue is updated;
- for remaining customers that are placed in orbit, besides serving time (*Stime*), two other variables are generated: the time after which they return to be served, denoted by *IntRet* and maximum number of returns, denoted by *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 (*nc* will be incremented and *Ts(nc)* 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. So we can say that the number of cycles of simulation does not exceed *3*Tnra*.

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

- *MTwq* represents the average waiting time in the queue of customers: *MTwq =Twq/Tnrserv* (Remarks 2);
- *MTOrb* represents the average waiting time in the orbit of customers: *MTOrb =Torb/Tnrorb* (Remarks 2);
- *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 pseudo-code the main part of the simulation algorithm. The fine-grain actions are grouped as procedures called from the main procedure.

1: **procedure** RetrQueuingSiytOneStat(Tnra)
2: *Read*(); //generated parameters for the interval between two consecutive arrivals, for service time, for time to spend in the orbit and the probability to enter in the orbit
3: //Initial state
4: $nc \leftarrow 0$; //there is no client in the queue
5: $Ltime \leftarrow 0$; //time value for the last event is 0
6: $Ctime \leftarrow \infty$; //the server is free
7: $Tnrserv \leftarrow 0$; //total number of services is 0
8: $Tnrorb \leftarrow 0$; //number of clients in the orbit is 0
9: $Torb \leftarrow 0$; //total time spent in orbit by clients is 0
10: $Tserv \leftarrow 0$; //total service time for the clients is 0
11: $Tlen \leftarrow 0$; //total server laziness time is 0

```
12:     Twq ← 0; //total waiting time in the queue for the clients is 0
13:     no ← 0; //number of clients in the orbit is 0
14:     To(1).Time_Rev ← ∞; //time for the first client return is ∞
15:     //first arrival is generated
16:     Gen(IntArriv, Stime); Atime:=IntArriv; Nra:=1;
17:     //simulation lasts while number of arrivals not exceed a given value
18:     while Nra <=Tnra do (1)
19:         if min{Atime, Ctime, To(1).Time_Rev} = Atime then (2)
20:             Update_Arriv();
21:             //next event is an arrival
22:         else(2)
23:             if min{Atime, Ctime, To(1).Time_Rev} = Ctime then (3)
24:                 //next event represents the end of a service
25:                 Update_Fin_Serv();
26:                 //next event is a return
27:             else(3)
28:                 Update_Retrial();
29:             end if; (3)
30:         end if; (2)
31:     end while(1)
32:     //Calculation of efficiency
33:     MTwq ← Twq/Tnrserv; //average waiting time in the queue
34:     MTOrb ← Torb/Tnrorb; //average waiting time in the orbit
35:     Mts ← Tserv/Tnrserv; //average serving time
36:     Clen ← Tlen/Ltime; //workstation laziness factor
37:     Mqueue ← Twq/Ltime; //the average length of the queue
38:     Write(MTwq, MTOrb, Mts, Clen, Mqueue);
39: end procedure
```

The *Update_Arriv* procedure below simulates new arrival in the system. Three possible cases are dealt with:

- the server is free ($Ctime = \infty$) and the client is served immediately;
- the server is busy and the queue isn't full and the client is inserted in the queue;
- the server is busy and the queue is full; if the generated value of B variable is 1, then the client is inserted into the orbit, otherwise he leaves the system.

```
1: procedure UPDATE_ARRIV
2:     if Ctime = ∞ then (1)
3:         //total workstation laziness time is updated
4:         Tlen ← Tlen + Atime − Ltime;
5:         Tsc ← Stime; //the client is served immediately
6:         Ctime ← Atime + Stime; //finishing service time is updated
7:         //the server is serving a client
```

```
    else (1)
        Tw ← Tw + nc * (Atime − Ltime); //update waiting time in the queue
        if nc <d then (2)
            //there are free places in the queue
            nc++; Ts(nc) ← Stime;
            //the client is introduced in the queue
        else (2)
            //overall resting time in orbit is update
            if no > 0 then (3)
                Torb ← Torb + nc * (Atime − Ltime); (3)
            end if; (3)
            //it generates a selection value of variable B
            if Gen(B) = 1 then (4)
                //the client is inserted into the orbit
                Gen(IntRet); Gen(NoRet); i ← 1;
                while IntRet >To(i).Time_Rev do (5)
                    i++;
                end while; (5)
                no++;
                //insertion into the orbit
                for k=no,i downto do (6)
                    To(k + 1).Time_Rev ← To(k).Time_Rev;
                    To(k + 1).Time_Serv ← To(k + 1).Time_Serv;
                    To(k + 1).Rev_Ram ← To(k).Rev_Ram;
                end for; (6)
                To(i).Rev_Ram ← NoRet;
                To(i).Time_Rev ← IntRet;
                To(i).Time_Serv ← Stime;
            end if; (4)
        end if; (2)
    end if; (1)
    Ltime ← Atime; //the time for the last event is updating
    Gen(IntArriv, Stime); //a new arrival is generating
    Atime ← IntArriv; Nra ← Nra + 1;
end procedure
```

The following procedure may serve the finishing of a service; there are two possible situations: when the queue is non-empty and the first customer in the queue will be served or the queue is empty and the server enters laziness.

```
procedure UPDATE_FIN_SERV
    Tserv ← Tserv + Tsc; //update total working time
    Tnrserv++; //update total number of services
    Torb ← Torb + no * (Ctime − Ltime); //update overall resting time in orbit
    Tw ← Tw + nc * (Ctime − Ltime); //update overall waiting time in queue
    Ltime ← Ctime; //update time for the last event
```

```
7:    if nc > 1 then (1)
8:        //there are clients in the queue
9:        Tsc ←Ts(1); //first client in the queue will be served
10:       Ctime ← Ctime+Ts(1); //update the time for finishing the service
11:       for i=1,nc-1 do (2)
12:           //the client is removed from the queue
13:           Ts(i) ←Ts(i+1);
14:       end for; (2)
15:       nc- -;
16:   else(1)
17:       Ctime ← ∞;
18:   end if; (1)
19: end procedure
```

The following 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.

```
1: procedure UPDATE_RETRIAL
2:    //update overall resting time in orbit for the clients
3:    Torb ← Torb + no ∗ (To(1).Time_Rev-Ltime);
4:    if Ctime = ∞ then (1)
5:        //update total laziness time for the station
6:        Tlen ← Tlen+To(1).Time_Rev-Ltime; //update time for the last event
7:        Ltime ←To(1).Time_Rev; //first client in orbit will be served
8:        Ctime ← To(1).Time_Rev + To(1).Time_Serv;
9:        Tsc ← To(1).Time_Serv;
10:       //the client is removed from the orbit
11:       for k=1,no do (2)
12:           To(k).Time_Rev ← To(k + 1).Time_Rev;
13:           To(k).Time_Serv ← To(k + 1).Time_Serv;
14:           To(k).Rev_Ram ← To(k + 1).Rev_Ram;
15:       end for(2)
16:       no–; Tnrorb++;
17:   else(1)
18:       //update overall resting time in queue for the clients
19:       Tw ← Tw + nc ∗ (To(1).Time_Rev−Ltime);
20:       To(1).Rev_Ram − −; //decrement of returns number
21:       if To(1).Rev_Ram > 0 then (3)
22:           //it generates a new time for return
23:           To(1).Time_Rev ← Gen(IntRet);
24:           //orbit reordering
25:           i ← 2;
26:           while To(1).Time_Rev > To(i).Time_Rev do (4)
27:               i++;
```

```
28:             end while; (4)
29:             x ← To(1).Time_Rev;
30:             y ← To(1).Time_Serv;
31:             z ← To(1).Rev_Ram;
32:             To(1).Time_Rev ← To(i).Time_Rev;
33:             To(1).Time_Serv ← To(i).Time_Serv;
34:             To(1).Rev_Ram ← To(i).Rev_Ram;
35:             To(i).Time_Rev ← x;
36:             To(i).Time_Serv ← y;
37:             To(i).Rev_Ram ← z;
38:         else(3)
39:             //the client quits the service and it is removed from the orbit
40:             for k=1,no do (5)
41:                 To(k).Time_Rev ← To(k + 1).Time_Rev;
42:                 To(k).Time_Serv ← To(k + 1).Time_Serv;
43:                 To(k).Rev_Ram ← To(k + 1).Rev_Ram;
44:             end for; (5)
45:             no–;
46:         end if; (3)
47:     end if; (1)
48: end procedure
```

4 The study of the complexity of the algorithm

The complexity of this simulation algorithm is calculated for each procedure separately. The procedure *RetrQueuingSiytOneStat(Tnra)* calls the following other procedures:

- *Read* is the procedure that treats the reading of the values for the interval between two consecutive arrivals, the serving time, the time to spend in the orbit for the clients, with their probability of being introduced in the orbit.
- The procedure generically named *Gen* handles the generation of random values that has been presented; they are generated like selection values for any random variables. Regardless of the random variable chosen, the generation algorithm has a polynomial complexity, that does not depend on *Tnra* (the total number of clients arrival in the system). In this case we assume that the complexity is *O(1)* [4].
- *Update_Arriv* is the procedure that simulates a new arrival in the system, treating the three possible cases.
- *Update_Fin_Serv* is the procedure that simulates the finish of a service.
- *Update_Fin_Serv* is the procedure that processes the return from orbit of the first customer to be served.

- *Write* is the procedure that displays the efficiency of the system.

Read and *Write* procedures are executed only once and we can assume that they have the complexity *O(1)* [3].

Update_Arriv procedure contains four calls for *Gen* procedure that has complexity *O(1)*. It also contains a *while* loop with which the first client in the system that must return to be served is selected; for this client $IntRet \leq To(i).Time\_Rev$. This *while* will not cycle more times than the number of the clients in the orbit at that time. The procedure also contains a *for* cycle dealing with clients insertion in the orbit. We can conclude that *Update_Arriv* procedure has complexity *O(no)* [3]. How the maximum number of returns of a client is *NoRet* (the given constant) and the maximum number of clients in the orbit can't exceed *Tnra*, it results that the complexity is *O(NoRet*Tnra)*.

Update_Fin_Serv procedure updates the values at the end of serving a client. With a *for* instruction it removes the served client from the waiting queue by shifting to the left the other clients in the queue. This operation will not run more times than the number *nc* (the given constant). In this way we deduce the value *O(nc)* for *Update_Fin_Serv* procedure complexity [3]. How the maximum number of returns of a client is *NoRet* (the given constant) and the maximum number of clients in the orbit can't exceed *Tnra*, it results that the complexity is *O(NoRet*Tnra)*.

Update_Retrial procedure brings out the client from the orbit by moving to the left the other clients. After this operation a new return time is generated and the orbit is reordered. The operation requires a maximum number of repetitions equal with the size of the orbit. The case when the client quits the service, leaving the orbit is also treated. Thus, the complexity in this case is *O(no)*.

Complexity of the procedure *RetrQueuingSiytOneStat(Tnra)* is determined by the complexity of procedures called inside it that will run as long as it does not exceed the number of arrivals denoted by *Tnra*. Thus, we have the complexity of *O(Tnra)+O(max(nc,NoRet) *Tnra)*.

5 Algorithm's implementation

The algorithm presented has been implemented using the Java object-oriented language facilities. In a future article we will present details of the implementation, case studies and validate the simulator. We consider that:

- system's input stream is 1 and the output stream is 2;
- distribution of time between two consecutive arrivals respectively serving time is an exponential variable with parameter λ=1 respectively μ=2;
- the queue length is large enough (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 that a customer entering the system for being placed in the orbit when

the server is busy is 0.

This model is equivalent to a model without calling customer return *exp(λ)/exp(μ)/1:(∞,FIFO)*. This model is studied by simulation in [5]. The results obtained by the simulator execution model presented in the article are: Average waiting time in the queue = 0.99388, Average length of the queue = 0.49857, Average serving time = 0.99103 and Traffic intensity = 0.49778. They are approximately equal to those obtained for the simulated system in [5].

6 Conclusions

In this article we present 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 service station. In many analytical studies, mathematical formulas are complicated and difficult to use in practice. For these reasons, such a simulation algorithm is booth 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] Cormen, T. H. a.o., *Introduction to algorithms*, MIT Press, Cambridge, 1992.

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

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

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

[7] 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.