An Algorithmic Approach of the Queuing Systems with Different Station Classes

Ion Florea

"Transilvania" University of Brasov

Department of Computer Science, Faculty of Mathematics and Informatics

Iuliu Maniu 50, Brasov, Romania

florea@yahoo.com

Alexandru Cârstea

"Transilvania" University of Brasov

Department of Computer Science, Faculty of Mathematics and Informatics

Iuliu Maniu 50, Brasov, Romania

Abstract: A system with parallel serving stations where every station has its own queue can be abstracted to a system that has identical stations. In such a system a client can choose a serving station following a random mechanism. Often, such systems divide the clients in classes and allocate serving station to a specific class of clients. Such models imply that every client chooses to be served by a working station within its own class. Systems with multiple client classes have no corresponding analytical models that study them. This paper presents a study method using a discrete simulation approach. It is proved in the paper that the proposed algorithm has a polynomial complexity. The implementation of the algorithm is achieved by an object oriented approach.

Keywords: Queuing System, Waiting Queue for Every Station, Simulation Algorithm, Polynomial Complexity, Different Classes of Stations

Ion Florea is an assistant of professor in the Computer Science Department at the "Transilvania" University of Brasov, Romania. He received the Msc, respectively the Ph. D degree from the Bucharest University, Faculty of Mathematics and Informatics. His current research interests include: Discrete Events Computer Simulation, Algorithm Theory, Operating Systems Theory. He is authoring over 30 papers, published in journals from Romania or other countries and proceedings of international conference too.

Alexandru Cârstea born 23/11/1980, Brasov, Romania graduated Mathematics and Informatics Faculty, "Transilvania" University of Brasov in 2003. He is master in computer science "Matematica si Informatica" Faculty Brasov in 2004 and Ph. D student starting with 1/10/2005 in Distributed Computing field. He is authoring 4 papers. His current research interests include: Distributed Programming/grid computing, Computer Simulation.

1. Introduction

The queuing models deal with systems study that has agglomerations. Such models must consider generation mechanisms for the arrivals in the system. A client that enters the system has a corresponding serving class and amount of serving time that he requests. Using the client's details and the status of the system, a serving station for the arrived client must be generated. In the traditional approach, where the stations topology is parallel and there is only one queue to all stations, if the system has an idle station, the arrived client is immediately served. If not, the client joins to the waiting queue. When a station finishes a service and the queue is not empty, based on a serving discipline (FIFO, HOL etc.), the next client is served.

As an alternative approach we consider a system that has a waiting queue for every station in the system. That means that an arrived client chooses the station that he wants to be served by. If a multiple class is considered, the arrived client can only choose from the stations available for its client class. If a station finishes a service and the queue is empty we consider two scenarios: one scenario in which the station becomes idle and one that lets the clients to migrate from one station to another. In the last case, a station that has no client in its own queue starts serving a client from another stations' queue, if such a client is available. In order to determine the station from witch the client will migrate we use a random based algorithm that will be described later on in the paper.

In the paper [2] we presented an algorithm for simulating this class of systems with a single queue and in the paper [3] we studied the systems where every station has its own queue and a client chooses a certain serving station using a random mechanism. In the following we consider that every arrived client belongs to a class of clients and it will be served by a station from its corresponding class. Determining the class

of the client and the station within the class that the client is assigned to is implemented using random based mechanisms. In this paper we consider two classes of clients: privileged and ordinary clients.

In every day life many systems can be abstracted using this model. As an example, in a super-market, the way the counters are organized follows such a model. In the same way, the operating system of a computer manages resources divided in same type components. Every such a resource has a queue that contains the processes that are waiting to gain access to the resource.

2. Simulation Entities

In the following we consider two classes of clients and two corresponding serving station classes. We consider that the number of serving station to be $m+n$, where *m* denotes the number of first class stations(that will serve privileged clients) and *n* denotes the number of second class stations(that will serve ordinary clients). Every station is uniquely identified by an integer $i(i=1,..,m+n)$

The arriving mechanism. The *Atime* variable denotes the time of the next arriving event in the system. Initially, *Atime* holds the value 0 and after every arrival in the system, its value is incremented by *IntAriv*, the interval between two consecutive arrivals, that is random generated. At the same moment we generate the time requested by the new client to be served.

Selecting the station that will serve the client. Denoting *p* the probability that a privileged client arrives into the system and $q=1-p$ the probability that an ordinary client arrives, we implement the arrival using a Bernoulli random variable:

$$B:\begin{pmatrix}0 & 1\\ p & q\end{pmatrix}$$

If the generated value is 0, then the client will pick a serving station from the first class. Let p_i be the probability of choosing the station *i.* The selection of a station will use a discrete random variable *X* given by

$$X:\begin{pmatrix}1 & \dots & m\\ p_1 & \dots & p_m\end{pmatrix}$$

Thus, choosing the station to serve the client is equivalent to generating the *X* variable. If we have a client from the second class of clients, choosing a station will be done by generating the random variable *Y,* given by

$$Y:\begin{pmatrix}m+1 & \dots & m+n\\ q_{m+1} & \dots & q_{m+n}\end{pmatrix}$$

where $q_j(j=m+1,...,m+n)$ is the probability of choosing serving station *j*.

The p_i and q_j probabilities are given by

$$p_i=\begin{cases}\left(1-\dfrac{nc(i)}{\sum_{j=1}^{m}nc(j)}\right)*\dfrac{1}{m-1}, & \text{if } m>1\\ 1, & \text{if } m=1\end{cases} \qquad (1)$$

$$q_j = \begin{cases} \left(1 - \dfrac{nc(j)}{\sum_{k=m+1}^{m+n} nc(k)}\right) * \dfrac{1}{n-1}, \text{ if n} > 1 \\ \text{q}_\text{j} = 1, \text{if n} = 1 \end{cases} \quad (2)$$

Theorem 1. i) If $nc(k) > nc(l)(l,k \in \{1,..,m\}, l \neq k)$ then $p_k < p_l$, $0 \le p_i \le 1, i = 1,...,m$ and $\sum_{i=1}^{m} p_i = 1$.

ii) If $nc(k) > nc(l)(l,k \in \{m+1,..,m+n\}, l \neq k)$ then $q_k < q_l$, $0 \le q_i \le 1, i = m+1,...,m+n$ and $\sum_{i=m+1}^{m+n} q_i = 1$.

Proof. i) Firstly we assume that $m > 1$. If $nc(k) > nc(l)$ then we have the inequality $\dfrac{nc(k)}{\sum_{j=1}^{m} nc(j)} > \dfrac{nc(l)}{\sum_{j=1}^{m} nc(i)}$, i.e. $1 - \dfrac{nc(k)}{\sum_{j=1}^{m} nc(j)} < 1 - \dfrac{nc(l)}{\sum_{j=1}^{m} nc(i)}$, i.e. $p_k < p_l$. In the same mode we prove that $q_k < q_l$

We have $0 \le nc(i) \le \sum_{j=1}^{m} nc(j)$, that we can write $0 \le \dfrac{nc(i)}{\sum_{j=1}^{m} nc(j)} \le 1$, i.e. $0 \le 1 - \dfrac{nc(i)}{\sum_{j=1}^{m} nc(j)} \le 1$,

thus $0 \le \left(1 - \dfrac{nc(i)}{\sum_{j=1}^{m} nc(j)}\right) \dfrac{1}{m-1} \le \dfrac{1}{m-1} < 1$

$$\sum_{i=1}^{m} p_i = \sum_{i=1}^{m} \left(1 - \frac{nc(i)}{\sum_{j=1}^{m} nc(j)}\right) * \frac{1}{m-1} = \sum_{i=1}^{m} \frac{1}{m-1} - \sum_{i=1}^{m} \frac{1}{m-1} \frac{nc(i)}{\sum_{j=1}^{m} nc(j)} =$$

$$= \frac{m}{m-1} - \frac{1}{(m-1)\sum_{i=1}^{m} nc(i)} \sum_{i=1}^{m} nc(i) = \frac{m}{m-1} - \frac{1}{m-1} = 1.$$

For $m = 1$ the proof is obvious. Also, in the same mode we can prove ii).

Remark 1.

i) This theorem states that $\{p_i / i = 1,..,m\}$ and respectively $\{q_i / i = m+1,..,m+n\}$ forms a complete probability system.

ii) The smaller the number of clients queued to a station is, the higher the probability of an arrived client to choose that station is.

Every station is characterized by the following variables:

- *Ctime(i)* denotes, for the *i* station, the end of the serving event time. If the station is free and there aren't clients queued for the station, the *Ctime(i)* variable is $Ctime(i) = \infty$ *;*
- the *nc* array; *nc(i)* holds the number of queued clients for the station *i*, at a given time;

the bi-dimensional vector *Ts* holds the serving time values needed for the corresponding clients queued at the station *i*. Here the *Ts(i,j)* value represents the amount of time needed to serve the client *j* queued at the station *i.*

We define the *ip* index by the formula:

$$ip = \min\{j / Ctime(j) = \min\{Ctime(i) / i = 1,..,n\}\} \quad (3)$$

i.e. the station index that finishes its service first.

The algorithm that we are going to present follows the "next event"("minimum time") rule. In every moment it's possible to have one of the two events:

- handling of an arrival (arrival event - Aevent); this event arises when $Atime \le Ctime(ip)$ *;* a selection value for the variable X is generated, denoting the working station the client is queued at;
- handling of an end of service event(Cevent), if $Atime \le Ctime(ip)$.

Remark 2. After such an event, the system efficiency factors are refreshed. The simulation ends when a certain number of arrivals (*Tnra),* representing the total number of arrivals permitted is reached. If we define as a simulation cycle handling a *Aevent* or a *Cevent,* we can consider that the simulation is done by executing such cycles. If we consider that the number of arrival events is smaller that those of end of service events, and the number of arrivals to be *Tnra* than we can say that the number of executed simulation cycles is smaller that *2*Tnra.* If we consider the *Tnra* value to be big enough, at the end of the execution of the algorithm almost all the clients get to be served. This assumption is true in a FIFO serving discipline system.

If $Atime \le Ctime(ip)$ an arrival of a client is handled:

- a value of the B variable is generated to determine the class of the client. Using this value, a selection value for X or Y variable is generated, depending on the class of the client.
- if the selected station is free, it will start serving the client. Otherwise the client will join the waiting queue.
- the efficiency factors are refreshed.

If $Atime \le Ctime(ip)$, we have end of service event. In this situation, the efficiency factors are refreshed and if the queue is not empty, the station starts serving the next client. If the queue is empty we consider the following scenarios:

i) the station becomes idle.
ii) the station tries to retrieve and serve a client queued to another station, following a mechanism that will be described onward.

Every handled event involves an update of the following values:

- $Tw = (Tw(i) / i = 1,..,m+n)$, where *Tw(i)* represent the total time of wait for the clients that were queued at the working station i*,*;
- $Tlen = (Tlen(i)/i = 1,..,m+n))$, where *Tlen(i)* represents the total idleness amount of time for the working station *i* .

In addition, after every end of service event, the following are updated:

- $Tts = (Tts(i) / i = 1,..,m+n)$, where *Tts(i)* represents the total amount of time spent by the working station *i* to serve clients*;*
- $Tw = (Nrs(i) / i = 1,..,m+n)$, where *Nrs(i)* represents the total number of clients served by the working station *i* ;

At the end of the simulation we determine the following global efficiency factors:

• *Mtw(i)* represents the average amount of time spent by the queued clients waiting to be served by the working station *i.* This value is given by the formula:

$$MTw(i) = \frac{Tw(i)}{Nrs(i)} \tag{4}$$

• *Mtw1* represents the average amount of time spent by the queued clients waiting to be served by the working stations in the class 1. This value is given by the formula:

$$Mtw1 = \frac{\sum_{i=1}^{m} Tw(i)}{\sum_{i=1}^{m} Nrs(i)} \tag{5}$$

• *Mtw2* represents the average amount of time spent by the queued clients waiting to be served by the working stations in the class 2. This value is given by the formula:

$$Mtw2 = \frac{\sum_{i=1}^{m+n} Tw(i)}{\sum_{i=m+1}^{m+n} Nrs(i)} \tag{6}$$

• *Mts(i)* represents the average amount of time spent by the working station *i* to serve clients. This value is given by the formula:

$$MTs(i) = \frac{Tts(i)}{Nrs(i)} \tag{7}$$

• *Clen(i)* represents the idleness coefficient for the working station *i.* This value is given by the formula:

$$Clen(i) = \frac{Tlen(i)}{Ltime} \tag{8}$$

• *Clen1* represents the idleness coefficient for the working stations in the class 1. This value is given by the formula:

$$Clen1 = \frac{\sum_{i=1}^{m} Clen(i)}{m} \tag{9}$$

• *Clen2* represents the idleness coefficient for the working stations in the class 2. This value is given by the formula:

$$Clen2 = \frac{\sum_{i=m+1}^{m+n} Clen(i)}{n} \tag{10}$$

• If *Ltime* is the number of time units of the entire simulation and *Mqueue(i)* represents the average length of the queue for the working station *i*, *Mqueue(i)* is given by :

$$MQueue(i) = \frac{TW(i)}{Ltime} \tag{11}$$

• *Mqueue1* represents the medium size of the queue for the 1 class and is given by:

$$MQueue1 = \frac{\sum_{i=1}^{m} MQueue(i)}{m} \tag{12}$$

• *Mqueue2* represents the medium size of the queue for the 2 class and is given by:

$$Mqeue2 = \frac{\sum_{i=m+1}^{m+n} MQueue(i)}{n} \tag{13}$$

3. Random Selection of the Client to be Served by the Empty Queue Station that Ends a Service

Determine the station that a client will migrate from. Let $\{i_1,..,i_k\}$ be the set of stations from a certain class of stations that are busy and that have at least one client in the waiting queue at a given time. Thus, $\{i_1,..,i_k\} \subset \{1,..,m\}$ or $\{i_1,..,i_k\} \subset \{m+1,..,m+n\}$. For every station *i,* $i \in \{1,..,m\}$ or $i \in \{m+1,..,m+n\}, i \notin \{i_1,...,i_k\}, j \in \{i_1,...,i_k\}$ we define $d_{ij} = | i - j |$, the "distance" from station *i* to the station *j*. We calculate the sum $S_i = \sum_{j=i_1}^{i_k} d_{ij}$. Considering *i* the station that ends a service, we define:

$$p_j = \begin{cases} 1 - \dfrac{d_{ij}}{S_i} \\ \dfrac{}{k-1}, \text{if } k > 1, j \in \{i_1,...,i_k\} \\ 1, \text{if } k = 1 \end{cases}$$

the probability that the station *i* will take a client from the station *j*.

Theorem 2. The set $\{p_j / j \in \{i_1,..,i_k\}\}$ forms a complete probability system. If $d_{il} > d_{ij}$, than we have $p_l < p_j$ ($l, j \in \{i_1,..,i_k\}, i \notin \{i_1,..,i_k\}$).

Proof. Having $0 < d_{ij} < S_i$, implies that $0 < \dfrac{d_{ij}}{S_i} < 1$, that is $0 < 1 - \dfrac{d_{ij}}{S_i} < 1$. If we divide the formula by k-1 we obtain $0 < p_j < 1$.

We also have $\sum_{j=i_1}^{i_k} p_j = \sum_{j=i_1}^{i_k} \dfrac{1 - \dfrac{d_{ij}}{S_i}}{k-1} = \dfrac{k-1}{k-1} = 1$.

In order to demonstrate the last state of the theorem we observe that $d_{il} > d_{ij}$ which leads to

$1 - \dfrac{d_{il}}{S_i} < 1 - \dfrac{d_{ij}}{S_i}$, equivalent to $p_l < p_j$.

Remark 3. The condition $d_l > d_j$ denoted the fact that station *l* is situated at a bigger distance from *i* station *i* than the station *j.* The theorem justifies that the selection of a client from a station situated at a bigger distance is done with a smaller probability.

We consider the following random variable $Y:\begin{pmatrix} i_1 & \dots & i_k \\ p_{i_1} & \dots & p_{i_k} \end{pmatrix}$. The station that will provide the client can be selected using a selection variable on Y.

Selecting the client to migrate. Let l be the generated selection value of the variable Y, i.e. the identification number of the station that will provide the migrating client. The number of clients in queued for the station l is $nc(l)-1$. In order to select the client that will migrate we use a selection variable of

$$T:\begin{pmatrix} 1 & \dots & nc(l)-1 \\ \dfrac{1}{nc(l)-1} & \dots & \dfrac{1}{nc(l)-1} \end{pmatrix}$$

i.e. all clients can migrate with the same probability.

4. Simulation Algorithm, Complexity Determination and OO Approach

The following procedure describes in pseudocode the main part of the simulation algorithm. The fine-grain actions are grouped as well in procedures used within this main procedure.

```
Procedure QueSystTwoCatSt(m,n,Tnra);
Read(Parameters Generation of Stime, IntArriv,ClientCat );
Atime←0;Ltime←0;Nra←0;
for i=1,m+n do nc(i)←0; Tts(i)←0 Nrs(i)←0;Ctime(i) ←∞;Tw(i)←0;
GenArrival(Atime,Stime,NrSt,Nra);
   While Nra≤Tnra do
                ip←1;
                 for i=2,m+n do
                 if Ctime(i)>Ctime(ip) then ip←i endif
                  endfor;
                 if Atime≤Ctime(ip) then {Aevent}
                          tipev←Aev;timpev←Atime;
                          PrelVectEfFact (tipev,timpev,Tw,timpev,Ltime,nc,Tlen) ;
                          JoinStation(Stime,NrSt);
                          Ltime←Atime;
                          GenArrival(Atime,Stime,NrSt,Nra) ;
                                        else{ Cevent}
                          tipev←Cev;timpev← Ctime(ip);
                          PrelVectEfFact(tipev,timpev,Tw,timpev,Ltime,nc,Tlen)
                          if nc(ip)>1 then PrelFinServ(ip, Ctime, Ts, Tss, nc)
                                        else PrelIdlSt(ip,Ctime)
                          endif
                 endif
        endwhile;
 EfFactComp() ;
Write( MTw, Mqueue,Mclen, Mts )
end.
```

Depending on the type of the next event. the procedure *PrelVectEfFact* will refresh the values used to calculate the efficiency factors at the end of the execution.

```
Procedure PrelVectEfFact (tipev,timpev,Ltime,nc, Tw ,Tlen,Nrs,Tts)
                for i=1,m+n do
                        Tw(i)←Tw(i)+nc(i)*(timpev-Ltime)
                endfor;
               for i=1,m+n do
                    if Ctime (i)= ∞ then Tlen(i)←Tlen(i)+timpev-Ltime
                    endif;
               endfor;
                Ltime←timpev;
                if tipev=Cev then
                        Nrs(ip)← Nrs(ip)+1; Tts(ip)←Tts(ip)+Tss(ip)
                Endif;
End.
```

Procedure *GenArriv* will generate the time needed for an arriving client to be served, the amount of time between two consecutive arrivals, the station that will receive the client. It will also refresh the values of the variable *Atime* and *Nra* . In order to generate *IntAriv* and *Stime* we generically described the generating procedure that will generate values with the distribution requested by context. *Discr* represents the procedure that generates the discrete random variable([])

```
Procedure GenArriv(p,nc,Stime,Atime,NrSt);
    Gen(Stime); Gen(IntAriv);Atime← Atime+ IntAriv;Nra←Nra+1;Discr(p,B);
    if B=0 then Discr(nc,X,NrSt)
            elseDiscr(nc,Y,NrSt)
    endif
end;
```

The procedure *JoinStation* will send the client to be served by a certain station. If that station is free it will immediately begin to serve de client, otherwise it will put the station in the waiting queue using the procedure *GenArriv*.

```
Procedure JoinStation(Stime,NrSt);
    if Ctime(NrSt)= ∞ then{Free Station}
            Ctime(NrSt)←Atime+Stime;Tss(NrSt)←Stime
              else {The arrived client joins to queue of  NrSt station }
                    nc(NrSt)←nc(NrSt)+1;Ts(NrSt,nc(NrSt)←Stime
     endif;
end;
```

The procedure *FinServ* is executed when a station finishes a service and has a non empty waiting queue. The procedure takes the first client from the queue and starts to serve it.

```
Procedure FinServ(ip, Ctime, Ts, Tss, nc);
    Ctime(ip)←Ctime(ip)+Ts(ip,1); Tss(ip) ←Ts(ip,1);
    for i=1,nc(ip) do
        Ts(ip,i)← Ts(ip,i+1)
      endfor;
    nc(ip)←nc(ip)-1
end ;
```

The procedure *PrelIdlSt()* deals with the situation when a serving station finishes to serve a client and it

has no clients waiting in queue. This procedure deals with two approaches: the one that sets the station to "idle" and the one that tries to get a client from another station.

```
Procedure PrelIdlSt(ip, Ctime);{first approach1}
                    {The station becomes  iddle}
      Ctime(ip)←∞
End;
Procedure PrelIdlSt(ip, Ctime);{second approach}
If ip<=m then
          li←1;     ls←m
          else
          li←m+1;ls←m+n
endif;
k←0;
for i=li to ls do
          if nc(i)>1then
                    k←k+1; ind(k)←i;d(i) ← abs(ip-i)
          endif
endfor;
S←0;
for j=1 to k do S←S+d(ind(j));
for j=1 to k do
          X(1, ind(j)) ← ind(j); X(2,ind(j)) ←(1-d(ind(j))/S))/(k-1);
endfor;
Discr(X,NrSt);{Number station generation}
For j=2 to nc(NrSt) do
          Y(1,j) ←j;Y(2,j) ←1/( nc(NrSt)-1)
endfor;
Discr(Y,NrCl);{Number Client generation}
{Station  ip serves the client received from the selected station}
 Ctime(ip)←Ctime(ip)+Ts(NrSt,NrCl);
Tss(ip) ← Ts(NrSt,NrCl);
{The client is taken out of the queue}
 for i=NrCl,nc(NrSt)-1 do
          Ts(NrSt,i)← Ts(NrSt,i+1)
endfor;
nc(NrSt)←nc(NrSt)-1;
end ;
```

At the end of the simulation the procedure *EfFactComp* that implements the formula (4)-(10), determines the system efficiency factors.

Complexity. As stated above (remark 2), the simulation algorithm will execute at most *2*Tnra* cycles that handle either an arrival in the system or an end of service. The numbers of arrivals is greater but approximately equal to the number of end-of-service cycles.

Every simulation cycle has to determine de station with the smallest Ctime moment and this is done in *O(m+n)*. Since the number of station is small we can consider this as having a complexity of O(1).

If we have an Aevent we execute:

- *PrelVectEfFac - 2*O(m+n)*
- *JoinStation – O(1)*
- *GenArrival –with known polynomial complexity*

If we have a *Cevent* we execute:

- *PrelVectEfFact - 2*O(m+n)*

- in the worst scenario we can consider to execute only the *PrelIdlSt* procedure. This procedure has to determine the distance between the station that ends the service an the other stations, elect a station to get a client from its queue and pull the client from its queue. This in done in *O(3*(m+n)). Since m+n* is a small value, this value is again very small. To sum up, we have at the end a polynomial complexity of *O(2*Tnra)*

Description of OOP solution. The use of objects in programming is a way of shorten the distance between the real life and the way it has to be modeled for the computer world. Bearing in mind with this approach we structured the algorithm to fit an OOP programming language. In order to develop this simulation environment, we have chosen the JAVA programming language. As a natural way of structure things, we created classes to map every entity in the system.

The algorithm simulates a SYSTEM. This System has one ore more SERVING STATIONS that serve CLIENTS. In addition, we had to generate the interval between two arrivals, we had to choose the station that a client will be served by and the time a client requires to be served. For this purpose we constructed several classes that implement the behavior of the entities stated above and a class that handles the generating process.

We used a similar system to simulate a system that has one or more working stations with owns queue in [3]. An addition to that model, to simulate the presented algorithm we added functionality to generate the Bernoulli selection variable and means to select and move clients from one serving station to another within the same station class.

5. Validity of the Algorithm and Practical Considerations

In the following we consider 10,000 arrivals simulated.

i) Case 1. n=1,m=1,p=0,5**.** This model is the same with FCFS and one station model, with only one queue. We will consider the model exp(λ)/exp(μ)/1:(∞,FIFO), that is analytically studied [5]**.** If we take λ=2, μ=2, we obtain the results given in table 1**.**

Table 1. The results for case 1

	First station	Second station	Analytical approach
Average time of wait	0,488	0,507	0,5
Idleness factor	0,491	0,499	0,5
Average serving time	0,505	0,501	0,5
Average queue length	0,492	0,507	0,5

As we can see from the above table, our simulation tends to equal the values obtained using the analytical approach. This validates our simulation model.

ii) Case 2. *n=2,m=1 λ=5, μ=2,p=0.5* In this case we have an equal probability that the clients belong to one of the two classes but for the privileged class we have two serving stations. This case treats the case with non migrating clients. The results are given in the following table:

Table 2. The results for case 2

	Privileged class	Ordinary class
Average time of wait	0,17	0,45
Idleness factor	0,579	0,36
Average serving time	0,504	0,517
Average queue length	0,142	0,556

In this case we have smaller values for the Average Time of Wait and Average Queue Length for the Privileged Class. Also, the value for the Idleness factor is greater for the Privileged class. As expected the value of the Average Serving time for both classes is approximately equal.

iii) Case 3. n=2,m=1 $\lambda=5, \mu=2, p=0.66$. In this case we have a greater probability for the system to generate privileged clients. The probability p is proportional to the number of stations in each class. The clients don't migrate from their initial stations.

Table 3. The results for case 3

	Privileged class	Ordinary class
Average time of wait	0,229	0,19
Idleness factor	0,505	0,534
Average serving time	0,493	0,481
Average queue length	0,23	0,184

In this case we have proportional values for both classes. The Average Serving Time will remain equal for both classes.

iv) Case 4. $n=2,m=1\ \lambda=5,\ \mu=2,p=0.5$ In this case we have an equal probability that the clients belong to one of the two classes but for the privileged class we have two serving stations. This case treats the case with migrating clients. The results are given in the following table

Table 4. The results for case 4

	Privileged class	Ordinary class
Average time of wait	0,08	0,243
Idleness factor	0,576	0,388
Average serving time	0,499	0,491
Average queue length	0,068	0,303

In this case we have much smaller values for the Average Time of Wait and Average Queue Length for the Privileged Class. This can be explained by the ability of clients to migrate. Also, the values for the Idleness factor are much greater for the Privileged class. As expected the value of the Average Serving time for both classes is approximately equal.

v) Case 5. $n=2,m=1\ \lambda=5,\ \mu=2,p=0.66.$ In this case we have a greater probability for the system to generate privileged clients. The probability p is proportional to the number of stations in each class. The clients can migrate from their initial stations.

Table 5. The results for case 5

	Privileged class	Ordinary class
Average time of wait	0,124	0,165
Idleness factor	0,493	0,5
Average serving time	0,499	0,504
Average queue length	0,126	0,164

The Average Time of Wait and The Queue length tends to increase a little due to the greater amount of clients that are served but, comparing to **case 2** these values are much smaller. This shows the increased efficiency of the model that permits migrating clients.

6. Conclusions

In this paper we have presented a simulation algorithm for queuing system in which every station has its own queue and the clients and the stations are divided in two categories corresponding categories. This model can be extended, by considering any number of categories. So, besides the considered Bernoulli distribution, whose generated selection value indicates the chosen station category, any discrete random variable of form $\begin{pmatrix} 0 & ...k-1 \\ p_0 .. & p_{k-1} \end{pmatrix}(k \geq 2)$, in which $p_i (i = 0,..,k-1)$ represents the probability of choosing of the *i* station category and, obviously the choosing category will be a generated value of this variable.

For this queuing system type there aren't analytical solutions then the simulation study is the only possible solution. Also, two possible ways have been taken in consideration in the finishing of the serving station: the station becomes idle or it will serve a different client from other station's queues, the station and the client being selected based on a random mechanism. The simulation results show that the efficiency factors of the system have better values in the second case.

REFERENCES

1. CORMEN, T.H., LEIRSON, C.E, RIVEST, R.L. **Introductions to Algorithms**. MIT Press, Cambridge, 1992.
2. FLOREA, I., **One Algorithmic Approach of First-Come-First-Served Queuing Systems**. Bucharest University Annals, Informatics, ANO XLIX, 2000, pp. 41-58.
3. FLOREA, I., CÂRSTEA A. **A Simulation Algorithm for Queuing Systems with Parallel Working Stations Having one's Own Queue for Every Station**, Bulletin of the Transilvania University of Braşov, Vol. 12(47), (to appear), 2005.
4. DEVROYE, L., **Non-Uniforme Random Variate Generation**, New York:springer Verlag, 1986.
5. GROSS, D., HARRIS C., **Foundamentals of Queuing Theory**, John Wiley & Sons, NewYork, 1998.
6. KLEINROCK, L., **Queuing Systems, Volume I: Theory**, John Wiley & Sons, NewYork, 1975.
7. KLEINROCK, L., **Queuing Systems, Volume II: Computer Applications**. John Wiley & Sons, 1976.
8. TANNER, M., **Practical Queuing Analysis**. McGraw-Hill Book Company, 1995.
9. VĂDUVA, I., **Computer Simulation Model**, Technical Publishing House, Bucharest, 1977.
10. VĂDUVA, I., STOICA, M., ODĂGESCU, I. **Economy Processes Simulation,** Technical Publishing House, Bucharest, 1983.