An Algorithm for Simulation of Waiting Systems with Different Types and Variable Number of Parallel Working Stations Each Having its Own Queue

Ion Florea, Lucian Sasu

Transilvania University of Brasov,
Department of Mathematics and Computer Science,
Faculty of Mathematics and Computer Science,
Iuliu Maniu 50, Brasov, 500091, Romania,
ilflorea@gmail.com, lmsasu@unitbv.ro

Abstract: This paper presents waiting systems with parallel working stations, for which both the clients and the working stations are grouped in classes. The working stations from the same class are identical and have their own waiting queues. The clients' arrivals, the choice of a class to whom the client belongs to and furthermore the choice of the serving station relies on a random process. Also, this approach considers a variable number of stations, influenced by the number of clients in the system. For this kind of problems there is no suitable analytical method and the support offered by specialized languages is quite poor. 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.

Keywords: Queuing System, Waiting Queue, Simulation Algorithm, Polynomial Complexity, Different Classes of Stations, Variable Number of Active Stations

1. Introduction

In this paper we approach the queuing system having a waiting queue to each station. When a new client arrives in the system, it chooses a random station based on a random process which takes into account the burden of the stations. Furthermore, we can consider that the clients themselves can be partitioned into classes and that for each class of clients there is a corresponding fixed set of deserving stations. A station from the clients' class is randomly selected to serve this client. The client's class and the station which will serve it are also randomly chosen. Inside a class, we will allow a client to migrate from a queue to another one that became empty. Here we suppose that there are two classes of clients, denoted as "privileged" and "regular", respectively. From now on we will suppose that the number of stations in each class varies, according to the agglomeration degree of the stations in that class.

In [3] we presented a simulation algorithm for waiting systems with a variable number of stations and with a single waiting queue, while in [4] we studied systems for which every working station has it own waiting queue and the probability of choosing a specific station is based on a random process. In [5] we considered the case for which the stations are clustered in two different classes, each station being devoted to a specific category and with a constant number of stations in each class.

According to the status of each station, they are partitioned in three groups: **busy**, **inactive** and **in laziness**. Busy stations are those currently serving clients. Inactive stations are the ones removed from the system, due to a low number of clients. Finally, for every class of working stations there is at most a station in laziness, when no client is in the waiting queue.

The essential difference between the model with a fixed number of parallel stations and the one that allows for a variable number is the approach of determining the laziness time of the stations. In the first case, all stations that are not currently working are in laziness while for the latter model, there is at most a single station in laziness when no other client is present in the same class and a potential arrival is expected.

Motivation: In real world there are plenty of systems that can be abstracted as described above. For example in super-markets, the pay desks set can be seen as such a waiting system. This type of simulation is potentially useful for analyzing and predicting the behavior of client-server based applications, when "organizations or resourceful individuals provide services via a set of loosely-coupled workstation nodes" [9]. We consider that simulation results are a starting point to perform load balancing for a set of resources that are accessed concurrently. Moreover, simulating different scenarios can assure decision support for making a choice between the two main types of scalability [6]: vertical scalability – adding more hardware to

the machine, which in our case is lowering the service time for a client - and horizontal scalability – which refers to adding more servers and in our case this is equivalent to adding more working stations. The main target is in this case middleware platforms that use clustered deployment not only for scalability but also for efficiently supporting multiple concurrent applications. From the same area of enterprise applications we also mention the load sensitivity which is an expression of how the response time varies with system's load [6]. The concept of different classes of clients is encountered in marketing, for which successful strategy is a classification of the firm's target markets. For example, in [10] a three tier classification system of potential and present clients is discussed. Thus our study based on different types of clients in the simulation mechanism has a strong connection with real-world applications.

2. The Model Entities and the Simulation Mechanism

Simulation of the arriv als. The entity that generates random arrivals produces both the inter-arrival time values denoted by $IntArriv$, and the service time for each client, denoted as $Stime$. 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.

Generating the category of the newly arrived client. Let p be the fixed probability that a type 1 client arrives in the system. The category corresponding to a newly arrived client is thus a Bernoulli random variable:

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

Choosing the serving sta tion. Let *nbp(i)* be the number of the active stations from category i and let *sbp(i,j)* be the index of the j^{th} active station of category i ($i \in \{1,2\}, j \in \{1,..,nbp(i)\}$). Also, let $nc(i,j)(i \in \{1,2\}, j \in \{sbp\{i,1\},..., sbp(i,nbp(i))\}$ be the number of clients that are in the waiting queue of station *sbp(i,j), i.e.* the total number of clients from the waiting queue plus the client which is currently served by the station *sbp(i,j)*. When a new client arrival is generated, that client is randomly assigned to a station belonging to its category, according to the Bernoulli distribution. Let the index of this station be *sbp(i,k)* and the the probability for the client of class i to be assigned to station k is denoted as

$p_{sbp(i,k)}$ ($i \in \{1,2\}$, $k \in \{sbp(i,1),...,sbp(i,nbp(i))\}$).

Generating $p_{sbp(i,k)}$ corresponds to generate the random variable *X(i)*:

$$X(i):\begin{pmatrix} sbp(i,1) & ... & sbp(i,nbp(i)) \\ p_{sbp(i,1)} & ... & p_{sbp(i,nbp(i))} \end{pmatrix}$$

The probabilities $p_{sbp(i,j)}$ are given by

$$p_{sbp(i,j)} = \begin{cases} \left(1 - \dfrac{nc(i,j)}{\sum_{k=sbp(i,1)}^{sbp(i.nbp(i))} nc(k)}\right) \cdot \dfrac{1}{nbp(i)-1}, \\ if\ nbp(i) > 1; \\ p_{sbp(i,j)} = 1, \\ if\ nbp(i) = 1 \end{cases}$$

Theorem 1. If $nc(i,k) > nc(i,l)$ $(i \in \{1,2\};$ $l,k \in \{1,..,nbp(i), l \neq k\})$ then $p_{sbp(i,k)} < p_{sbp(i,l)}, 0 \le p_{sbp(i,j)} \le 1$ $\forall j \in \{1,...,nbp(i)\}$ and $\sum_{k=sbp(i,1)}^{sbp(i,nbp(i))} p_i = 1, \forall i \in \{1,2\}$.

Proof. Firstly, we assume that $nbp(i) > 1, i \in \{1,2\}$. If $nc(i,k) > nc(i,l)$, then we have the inequality

$$\frac{nc(i,k)}{\sum_{j=sbp(i,1)}^{sbp(i,nbp(i))} nc(i,j)} > \frac{nc(i,l)}{\sum_{l=nbp(i,1)}^{sbp(i,nbp(i))} nc(i,l)}, \quad \text{i.e.}$$

$$1 - \frac{nc(i,k)}{\sum_{j=sbp(i,1)}^{sbp(i,nbp(i))} nc(i,j)} < 1 - \frac{nc(i,l)}{\sum_{l=sbp(i,1)}^{sbp(i,nbp(i))} nc(i,l)},$$

and thus $p_{sbp(i,k)} < p_{sbp(i,l)}$.

We have $0 \le nc(i,j) \le \sum_{p=sbp(i,1)}^{sbp(i,nbp(i))} nc(i,p)$, and thus we can write

$$0 \le \frac{nc(i,j)}{\sum_{p=sbp(i,1)}^{sbp(i,nbp(i))} nc(i,p)} \le 1, \quad \textit{i.e.}$$

$$0 \le 1 - \frac{nc(i,j)}{\sum_{p=sbp(i,1)}^{sbp(i,nbp(i))} nc(i,p)} \le 1, \quad \text{and hence}$$

$$0 \le \left(1 - \frac{nc(i,j)}{\sum_{p=sbp(i,1)}^{sbp(i,nbp(i))} nc(i,p)}\right) \frac{1}{nbp(i)-1} \le \le \frac{1}{nbp(i)-1} \le 1$$

which gives $0 \le p_{sbp(i,j)} \le 1,\ j = 1,..., nbp(i)$

$$\sum_{j=sbp(i,1)}^{sbp(i,nbp(i))} p_j = \sum_{j=sbp(i,1)}^{sbp(i,nbp(i))} \left(1 - \frac{nc(i,j)}{\sum_{j=sbp(i,1)}^{sbp(i,nbp(i))} nc(i,j)}\right).$$

$$\cdot \frac{1}{nbp(i)-1} = \sum_{j=sbp(i,1)}^{sbp(i,nbp(i))} \frac{1}{nbp(i)-1} -$$

$$- \sum_{j=sbp(i,1)}^{sbp(i,nbp(i))} \frac{1}{nbp(i)-1} \frac{nc(i,j)}{\sum_{j=sbp(i,1)}^{sbp(i,nbp(i))} nc(i,j)} =$$

$$= \frac{nbp(i)}{nbp(i)-1} -$$

$$- \frac{1}{(nbp(i)-1)\sum_{j=sbp(i,1)}^{sbp(i,nbp(i))} nc(i,j)} \sum_{j=sbp(i,1)}^{sbp(i,nbp(i))} nc(i,j) =$$

$$= \frac{nbp(i)}{nbp(i)-1} - \frac{1}{(nbp(i)-1)\sum_{j=sbp(i,1)}^{sbp(i,nbp(i))} nc(i,j)} \sum_{j=sbp(i,1)}^{sbp(i,nbp(i))} nc(i,j) =$$

$$= \frac{nbp(i)}{nbp(i)-1} - \frac{1}{nbp(i)-1} = 1$$

For $nbp(i) = 1$ the proof is straightforward.

Remark 1 . The theorem actually shows that the quantities $p_{sbp(i,k)}$ define a probability distribution.

The server entity is the $m+n$ parallel stations ensemble. We use the notation:

$$nrst(i) = \begin{cases} m, \text{if i} = 1 \\ n, \text{if i} = 2 \end{cases}$$

Any station is characterized by:

Ctime(sbp(i,j)) which is the serving finish event time of *sbp(i,j)*-server; if the server is idle, then *Ctime(sbp(i,j))*= ∞ ; *sbp(i,j)* is in laziness if:

Ctime(sbp(i,j))= ∞ *and nbp(i)=1*.

For $k \in \{sbp(i,j), i \in \{1,2\}, j \in \{1,..,nbp(i))\}$, *Tsc(k)* is the service time of the k^{th} station.

The bidimensional vector *Ts(sbp(i,j),k)* contains the service time of the k^{th} client joined to station *sbp(i,j)*, $k \in \{1,..,nc(sbp(i,j))\}$.

Remark 2 . The discrete events simulation is based on the "next event" (or "minimum time") rule. In our case, the time of the next event is given by: $\min\{Atime, Ctime(sbp(i,ip(i))/i=1,2)\}$

where *ip(i)* is the index of the first server from class *i* that finishes the service. The time of the last event is denoted by *Ltime*. If $Atime \le \min\{Ctime(sbp(i,ip(i))/i=1,2\}$

then the next event is an arrival and it will be processed as follows: we generate the random variable *B*, whose value *i* ($i \in \{1,2\}$), corresponds to the category of the newly come client. Then a value is generated for the random variable *X(i).* If there is an idle server, this one will immediately serve this client. The value *Stime* is generated and the total servers laziness time is updated. If there are inactive servers and the adding of a new server condition is fulfilled, *i.e.* if $nbp(i) < nrst(i)$ and

$$\frac{\sum_{j=1}^{nbp(i)} nc(i, sbp(i,j))}{nbp(i)} \ge l \max (i)$$

then an inactive station of the minimum index will serve the new client and the total customers waiting times per type is updated; also, we add a new column to the bidimensional array *X(i)* corresponding to the newly added station *sbp(i,nbp(i))*, having the associated probability *1/(nbp(i)-1).* If the condition corresponding to adding a new station is not fulfilled, then the client joins to the queue given by the selection value of the variable *X(i).* If

$Atime > \min\{Ctime(sbp(i,ip(i))/i=1,2\}$

then the next event is service finishing by one of the two *sbp(i,ip(i))* stations. In this case, the following actions take place:

- if $nc(sbp(i,ip(i)) > 1$ (*i.e.* there are clients in the queue of the respective station), then the first client will be served;
- if $nc(sbp(i,ip(i)) = 0$ (i.e. there are no clients in the queue of the respective station), and $nbp(i) > 1$ and $\frac{\sum_{j=1}^{nbp(i)} nc(i, sbp(i,j))}{nbp(i)} < l \min (i)$

  then this station will become inactive

– if $nc(sbp(i,ip(i))=0$ (*i.e.* there are no clients in the queue of the respective station) and $nbp(i)=1$ then the station will be in laziness.
– if $nc(sbp(i,ip(i))=0$, *i.e.* there are no clients in the queue of the respective station), and $nbp(i)>1$ and $\frac{\sum_{j=1}^{nbp(i)} nc(i,sbp(i,j))}{nbp(i)} \geq l\min(i)$

then this station will serve one client from other station. Thus we allow a client to migrate to a different station.

Determining the station from which a client migrates. For each active station *sbp(i,j) (j∈{1,..,nbp(i)})* we compute $d_{ij}=|sbp(i,ip(i))-sbp(i,j)|$, the L_1 distance between stations *sbp(i,ip(i))* and *sbp(i,j)*. Let $S_i = \sum_{j=1}^{nbp(i)} d_{ij}$. We define

$$p_{sbp(i,j)} = \frac{\left(1-\frac{d_{ij}}{S_i}\right)}{(nbp(i)-1)}$$

as the probability of choosing a client currently in the waiting queue of station *sbp(i,j)* as a migrated client.

Theorem 2. The set

$$\{p_{sbp(i,j)}\}/\ j\in\{1,...,nbp(i)\}, i\neq sbp(i,ip(i)))\}$$

defines a probability distribution. Moreover, if $d_{il} > d_{ij}$ then $p_{sbp(i,l)} < p_{sbp(i,j)}$.

Proof. As $0<d_{ij}<S_i$, we obtain $0<\frac{d_{ij}}{S_i}<1$, that is $0<1-\frac{d_{ij}}{S_i}<1$. Dividing by *nbp(i)-1*, we get $0<p_{sbp(i,j)}<1$. Clearly,

$$\sum_{j=1}^{nbp(i)} p_{sbp(i,j)} = \sum_{j=1}^{nbp(i)} \frac{1-\frac{d_{ij}}{S_i}}{nbp(i)-1} = \frac{nbp(i)-1}{nbp(i)-1} = 1.$$

For the later statement of the theorem we note that $d_{il} > d_{ij}$ yields $1-\frac{d_{il}}{S_i} < 1-\frac{d_{ij}}{S_i}$, which is to say that $p_{sbp(i,l)} < p_{sbp(i,j)}$.

The station providing the migrating client is a the value generated by the random variable *Y(i).*

For the later statement of the theorem we note that $d_{il} > d_{ij}$ yields $1-\frac{d_{il}}{S_i} < 1-\frac{d_{ij}}{S_i}$, which is to say that $p_{sbp(i,l)} < p_{sbp(i,j)}$.

The station providing the migrating client is a the value generated by the random variable *Y(i).*

Selecting the client to be served. Let $sbp(i,l)$ be the value of the random variable *Y(i), i.e.* the index of the station from whose waiting queue the client is to be migrated. The count of the clients assigned to station $sbp(i,l)$ is $nc(sbp(i,l))-1$. The index of the client to migrate, is given by the random variable $Z(i), i\in\{1,2\}$:

$$Z(i):\begin{pmatrix} 1 & .. & nc(sbp(i,l))-1 \\ \frac{1}{nc(sbp(i,l))-1} & .. & \frac{1}{nc(sbp(i,l))-1} \end{pmatrix}.$$

that is any client from the waiting queue has the same probability to migrate.

Remark 4. We define a simulation cycle as being one of the two actions: updating the arrival; finishing the service. The simulation proceeds by the looping cycles, until the number of simulated arrivals exceeds the given value denoted by *Tnra.* Because the arrival flow is less than the services flow (otherwise the length of the queue increases indefinitely) and if the value of *Tnra* is sufficiently large, almost all clients will be served, and the number of services is less than or equal with *Tnra*. So, the maximum number of cycles is no more than *2·Tnra.*

Adding a new station. Let

$$nlp(i) = \begin{cases} m-nbp(i), \text{ if } i=1 \\ n-nbp(i), \text{ if } i=2 \end{cases}$$

be the number of the inactive stations, let *slp(i,j) (i∈{1,2}, j∈{1,.., nlp(i)})* be the array containing the indexes of inactive stations. Choosing the station to be added can be performed according to any of the following two approaches:

i) randomly, by considering that any inactive station can be chosen with the same probability. In this case we use:

$$T(i):\begin{pmatrix} slp(i,1) & \dots & slp(i,nlp(i)) \\ \frac{1}{nlp(i)} & \dots & \frac{1}{nlp(i)} \end{pmatrix}$$

The index of the station to be activated is the actual value generated for the random variable *T(i)*.

ii) the first element from the vector *slp(i)* is considered.

At the end of simulation, the system efficiency factors are computed: the average of the waiting time in the queue, the average of the length of the queue, the average of the serving time of the clients, the average of the laziness coefficient of the stations.

3. The Algorithm's Description and the Complexity Study

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.

```
Procedure Sim(Tnra,lmax,lmin,…);
{block 1}{System initialization}
Read(Random generation Parameters);
sbp(1,1)←1;sbp(2,1)←m+1;
Atime←0;Nra←0;
for j=2 to m do slp(1,j)←j endfor;
for j=m+2 to m+n do slp(2,j)←j
endfor;
for i=1,2 do
 Tl(i)←0;nbp(i)←1;nc(sbp(i,1))←0;
 Ctime(sbp(i,1))←∞;busy(sbp(i,1))←0;
 Ltime(i)←0
endfor;
for j=1 to m+n
 Tw(j)←0;Tts(j)←0;Nrs(j)←0
endfor;
X(1)(1,1)←1;X(1)(2,1)←1;
X(2)(1,1)←1;X(2)(2,1)←1
{end block 1}
{block 2}
Arrival(B,X,IntArriv,Nra,Atime,Stime)
{end block 2}
{block 3}while Nra≤Tnra do
{ip(i)=the index of the first server
from class i that fin. its service}
   {block 3.1}
   for i=1,2 do     ip(i)←sbp(i,1);
     for j=2 to nbp(i) do
      if ctime(ip(i))>ctime(sbp(i,j))
        then ip(i)←sbp(i,j)
       endif
     endfor
   endfor;
    {end block 3.1}
    {block 3.2}
  if Atime≤min{Ctime(ip(i))/i=1,2}
     then {Arrival of a new client}
Arriv(B,X,IntArriv,Nra,Atime,Stime)
      else
{a station finishes serv.a client}
      If Ctime(ip(1))<=Ctime(ip(2))
       then FinishService(1)
       else FinishService(2);
      endif;
 endif; {end block 3.2}
endwhile; {end block 3}
{block 4}{Comp. eficiency factors}
Lt←max{Ltime(1),Ltime(2)};
for j=1 to m+n
 MQueue(j)←Tw(j)/Lt;{Avg. length of
the queue from the station j}
 MClen(j)←Tlen(j)/Lt;{Laziness
coefficient for station j}
 MTs(j)←TTss(j)/Tnrs(j); {Average of
serving time for station j}
 MTw(j)←Tw(j)/Nrs(j);{Average of
waiting time for station j}
endfor;
Write(Mtw, MTs, MClen,Mqueue))
{end block 4}
```

The `Arriv` procedure below simulates new arrival in the system. This procedure performs: generation of a Bernoulli random value *i* corresponding to the client's class ([2]), generation of a random value for *X(i)* giving the station index that will serve the new client, and finally generation of the delay to the next arrival and the serving time for the client. Also, the following variables are updated: the waiting time of the existing clients, the laziness time for the stations, the *Ltime(i)* and *Atime* corresponding to the time of the last event for class *i* and to the next arrival, respectively. If necessary, a new station is added by calling the `AddStation` method.

```
Procedure Arriv
(B,X,IntArriv,Nra,Atime,Stime)
{The total waiting times updating}
{block 1}
for k=1,2 do
 for j=1,nbp(k) do
Tw(sbp(k,j))←Tw(sbp(k,j))+
+nc(sbp(k,j))*(Atime-Ltime(k))
  endfor
endfor
{end block 1}
{block 2}Gen(IntArriv);Gen(B,i);
{i=client class}
Ltime(i)←Atime; Nra← Nra+1;
Atime←Atime+ IntArriv;Gen(Stime);
if (nbp(i)=1) and (Ctime(sbp(i,1)=∞))
    then {There is an idle server }
Tlen(sbp(i,1))←Tlen(sbp(i,1))+
               +(Atime-Ltime(i))
```

```
{Total laziness time of the servers
updating; the newly arrived customer
will be immediately served}
  Ctime(sbp(i,1))←Atime+Stime;
  Tsc(sbp(i,1))←Stime
   else
   AddStation(X(i));
   {update r.v. X}
   Gen(X(i),k);{k= the station that
   will serve the arr. client}
   if nc(sbp(i,k)))=0 then
      Tsc(sbp(i,k))←Stime
      {immediately served}
     else
{the client is queued at station k}
   nc(sbp(i,k))))←nc(sbp(i,k)))+1;
   Ts(sbp(i,k),nc(sbp(i,k)))←Stime
    endif;
endif.
```

The following `AddStation` procedure is called when the new arrival is simulated. It computes the number of clients that are in the station's *i* queue and it decides whether a new station should be added. The choice upon which free station should be activated is made by the procedure `Choice`.

```
Procedure AddStation(i)
Sum←0;
{The total number of clients at the
station class i}
for k=1,nbp(i) do
Sum←Sum+nc(i,sbp(i,k))
endfor;
if i=1 tts←m else tts←n endif;
if nbp(i)<tts and Sum/nbp(i)>lmax(i)
  then
 nbp(i)←nbp(i)+1;c(sbp(i,nbp(i)))←0;
 sbp(i,nbp(i)))←Choice(i);
 X(i)(nbp((i),1)← sbp(i,nbp(i)));
 X(i)(nbp(i),2)←1
endif.
Procedure Choice(i)
for k=1,nlp(i) do
    Z(i)(1,k)←slp(i,k);
    Z(i)(2,k)←(1/nlp(i));
endfor;
return(Gen(Z(i))
```

The `FinishService` procedure from below performs updates when a station finishes serving a client. The following are updated: the total waiting time values, the total serving time values for each station, the total number of served clients for that station, the time of the last event.

```
Procedure FinishService(i)
{The total waiting times is updated}
{block 1}for k=1,2 do
  for j=1,nbp(k) do
    Tw(sbp(k,j))←Tw(sbp(k,j))+
    nc(sbp(k,j))*
    *(Ctime(ip(i))- Ltime(i))
 endfor endfor;{end block 1}
{block 2}
Ltime(i)←Ctime(ip(i));
{Updat. the total working time and the
number of services}
Nrs(ip(i))←Nrs(ip(i))+1;
Ttss(ip(i))←Ttss(ip(i))+ +Tsc(ip(i));
If (nc(ip(i))>1) then {1}
{ip(i) will  serve the first  client
from its queue}{block 2.1}
     Ctime(ip(i))←Ctime(ip(i))+
     + Ts(ip(i),1)
    Tsc(ip(i))←Ts(ip(i),1)
       {the client to be served is
       removed from the queue}
    for j:=1 to nc(ip(i))-1 do
     Ts(ip(i),j)← Ts(ip(i),j+1)
    endfor
    nc(ip(i))←nc(ip(i))-1
       {end block 2.1}
       {block 2.2}{Upd.the var. X]
       if nbp(i)>1 then
{2}{Many active Stations}
        for j:=1 to nbp(i) do
          X(i)(2,j)←
((1-nc(sbp(i,j)))/ Sum(nc,sbp(i))/
(nbp(i)-1)
        endfor
       else
       {2}{a single active station}
        X(i)(2,nbp(i))←1
       endif{2}{end block 2.2}
       else
{1}{Empty queue of the station ip(i)}
{block 2.3}  if nbp(i)=1 then
{3}{the station ip(i) becomes lazy}
      sbp(i,1)←ip(i);Ctime(ip(i))←∞
              else{3}
if Sum(nc,sbp(i))/nbp(i)<lmin(i)and
(nbp(i)>1) then
 {4}{the stat.ip(i) becomes inactive}
     Ctime(ip(i))←∞;  j←1
 while(j≤nbp(i))and (sbp(j)<>ip(i))do
       j←j+1
 endwhile
{ip(i) is added to the inact. stat.}
   nlp(i)←nlp(i)+1;slp(nlp(i))←ip(i)
{ip(i) is rem.from the active stat.}
  for k=j,nbp(i)-1 do
    sbp(i,k)←sbp(i,k+1);
    nc(i,k)←nc(i,k+1)
  endfor
   nbp(i)←nbp(i)-1 {Updating X(i)}
  for j:=1 to nbp(i) do
    X(i)(1,j)←sbp(i,j);
    X(i)(2,j)←((1-nc(sbp(i,j)))
/Sum(nc,sbp(i)))/(nbp(i)-1)
   endfor
      else {4}{the station's index
from which a client is moved}
       S(i)←0;
       for j=1,nbp(i) do
```

```
        d(i,j)← abs(sbp(i,j)-
        -sbp(i,ip(i)));
         S(i)←S(i)+d(i,j)
       endfor
     for j=1,nbp(i) do
      Y(i)(1,j)←sbp(i,j);
        Y(i)(2,j)←
       (1-d(i,j)/S(i))/nbp(i)-1)
      endfor
      Gen(Y(i),sbp(i,l));
{Generates the client to be served}
       for j=1,nc(sbp(i,l))-1     do
       Z(i)(1,j) ←j;
   Z(i)(2,j)←1/(nc(sbp(i,l))-1)
      endfor
       Gen(Z(i),p);
  Ctime(ip(i))←Ctime(ip(i))+
  +Ts(sbp(i,l),p)
 Tsc(ip(i))←Ts(sbp(i,l),p)
{remove from the queue the client to
be served}
 for j=p,nc(sbp(i,l))-2 do
    Ts(sbp(i,l),j)←Ts(sbp(i,l),j+1)
 endfor
{Update the variable X}
  X(i)(2,nbp)←1
  endif{4}
 endif{3}{block 2.3}
endif{1}
```

The full computation of the simulation's complexity is given in the extended version of this paper (see http://cs.unitbv.ro/~lmsasu/publications/artflo2012.pdf). The total complexity of the simulation is $O(Tnra^2)$.

4. An OO Implementation

While the pseudocode presented in section 3 is suitable for understanding the whole process and for computing the algorithm's complexity, it is obvious that when performing a concrete implementation, an object-oriented (OO) design is a more productive approach compared to classical procedural programming. Conceiving the classes as being responsible for well-defined behavior allows one to obtain a natural design of the problem domain. In the extended version of this paper (see http://cs.unitbv.ro/~lmsasu/publications/artflo2012.pdf), we give a full description of the classes that resulted from the OO design.

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 and serving time are exponential negative distributed, with λ respectively μ parameters. The above mentioned implementation was used to obtain the statistics for various scenarios. We considered five scenarios.

i) Case 1. m=1, n=1, p=0.5, λ=0.5, μ=1;lmin(1)=lmin(2)=lmax(1)= =lmax(2)=1.

Table 1. The results for case 1

Efic. factor	Class 1	Class 2
Length Avg	0.0007	0.0006
LazCoefAvg	0.9749	0.9749
ServTimeAvg	1.0049	1.0048
WaitTimeAvg	0.0260	0.0259

ii)Case2.m=3,n=2,p=0.6,λ=0.5,μ=0.2 lmin(1)=lmin(2)=2,lmax(1)= lmax(2)=5.

Table 2. The results for case 2.

Eficiency factor	Class 1	Class 2
Length Avg	0.938	0.788
LazCoefAvg	0.018	0.092
ServTimeAvg	4.989	4.984
WaitTimeAvg	9.130	7.930

iii) Case 3. m=3,n=2,p=0.5,λ=0.5, μ=0.2,lmin(1)=lmin(2)=2,lmax(1)=lmax(2)=5.

Table 3. The results for case 3

Eficiency factor	Class 1	Class 2
Length Avg	0.745	1.196
LazCoefAvg	0.031	0.048
ServTimeAvg	4.987	4.977
WaitTimeAvg	8.337	9.544

iv) Case 4. m =3,n=2,λ=0.5, μ=0.2, p=0.5, lmin(1)=lmin(2)=2, lmax(1)=lmax(2)$\in$ {5,…,10}. The results are given in figures 1, 2 and 3 (see http://cs.unitbv.ro/~lmsasu/publications/artflo2012.pdf).

v) Case 5. m=3,n=2,p=0.5,λ=0.5, μ=0.2;lmax(1)=lmax(2)=8,lmin(1)=lmin(2)$\in$ {2,…,7}.The results are given in figures 4, 5 and 6 (see http://cs.unitbv.ro/~lmsasu/publications/artflo2012.pdf).

Remark 4.

– in *i*), the results are approximately equal with those obtained for the corresponding model with fixed number of stations considered in [5].

– by comparing the results corresponding to cases 2 and 3 for the first class of clients, it can be seen that the average length of the waiting queue and the average value of the waiting time for *iii)* are smaller than the corresponding ones

from *ii)*, while the average of the waiting time is lower for *ii)*. This is explained by the higher probability for a client to belong to the first category for *ii)*, and hence the same number of working stations can be required by a larger number of clients.

– if `Lmax(i)` decreases, then both the average length of the queue (Figure 1) and the average value of the waiting time (Figure 2) decrease. In turn, the laziness coefficient grows (see Figure 2). This is explained as follows: when `Lmax(i)` decreases the frequency of adding a new station increase and so the number of effectively working stations grows.

– If `lmin(i)` grows then generally the average length of the queue (Figure 4) and the average of waiting time (Figure 6) grows. In turn, the laziness coefficient decreases (Figure 5). This occurs because a station will become inactive more frequently.

6. Conclusions

In this paper we have presented a simulation algorithm for queuing system in which every station has its own queue and in function of the client's type, the stations are divided in two categories. Also, we have a dynamic number of the active servers. This model can be extended, by considering any number of station categories. So, besides the considered Bernoulli distribution, whose generated selection value indicates the chosen station category, any discrete random variable of the 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; so the simulation study is the only possible way. 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., C. E. LEIRSON, R. L. RIVEST, **Introductions to Algorithms** , MIT Press, Cambridge, 2001.
2. DEVROYE, L., **Non-Uniform Ran dom Variate Gen eration**, New York, Springer Verlag, 1986.
3. FLOREA, I., **One Algorithmic Appro ach of First-C ome-First-Served Queu ing Systems**, Bucharest University Annals, Informatics, ANO XLIX, 2000, pp. 41-58.
4. FLOREA, I., A. CARSTEA, **A Simulation Algorithm f or Queuing Systems with Parallel Working Stations Having one's Own Queue for Every Station**, Bulletin of the Transilvania University of Brasov, vol. 12(47), 2005, pp. 115-123.
5. FLOREA, I., A. CARSTEA, **An Alghoritmic Approach of the Queuing Systems wit h Differ ent Station Classes** , Studies in Informatics and Control, vol. 15(4), March 2006, pp. 391-402.
6. FOWLER, M., **Patterns of Enter prise Application Architecture** , Addison-Wesley, 2002.
7. GAMMA, E, R. HELM, R. JOHNSON, J. VLISSIDES, **Design Patterns: El ements of Reusable Object-Ori ented Software** , Addison-Wesley, 1994.
8. GROSS, D., C. HARRIS, **Fundamentals of Queuing Theory**, John Wiley & Sons, New York, 1998.
9. PETROU, D., K. AMIR, G. RANGER, G. GIBSON, **Easing the Management of Data-parallel Systems Via Adaptation** , Proceedings of the 9th ACM SIGOPS European Workshop, Denmark, September 17-20, 2000.
10. RADER, C., R. COMISH, D. BURCKEL, L. TURPIN, **Adapting Marketing Strategies to Custome r Buying Processes in the Context of Customer I mportance to the Firm** , Proceedings of American Society for Business and Behavioral Sciences, Volume 16, no 1, February 2009.
11. TANNER, M., **Practical Queuing Analysis**, McGraw-Hill Book Company, 1995.