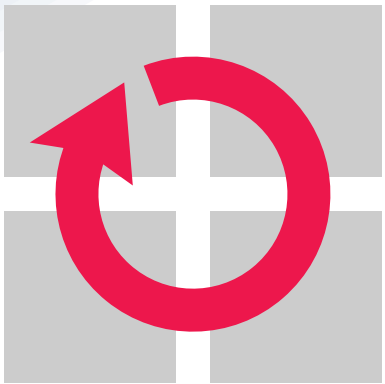
G. Bolch, J. Roszik, J. Sztrik

Modeling Finite-Source Retrial Queueing Systems with Heterogeneous Non-Reliable Servers by MOSEL

Technical Report TR-I4-2005-01 ◆ Juni 2005

Lehrstuhl für Informatik 4
Verteilte Systeme und Betriebssysteme



Friedrich-Alexander-Universität
Erlangen-Nürnberg

Universität Erlangen-Nürnberg • Informatik 4 • Martensstr. 1 • 91058 Erlangen
Tel.: 09131.85.27277 • Fax: 09131.85.28732
E-Mail: i4@informatik.uni-erlangen.de • URL: http://www4.informatik.uni-erlangen.de

TECHNISCHE FAKULTÄT

Modeling Finite-Source Retrial Queueing Systems with Heterogeneous Non-Reliable Servers by MOSEL

* G. Bolch, J. Roszik, J. Sztrik
* University of Erlangen, Erlangen, Germany
University of Debrecen, Debrecen, Hungary
jsztrik@inf.unideb.hu

Abstract

The aim of this paper is to analyze the performance of multiple server retrial queueing systems with a finite number of homogeneous sources of calls, where the heterogeneous servers are subject to random breakdowns and repairs. The requests are serviced according to Fastest Free Server (FFS) discipline.The novelty of this paper is the different service rates and non-reliability of the servers, which has a very adverse influence on the performance of the system, thus it plays an important role in practical modeling of computer and communication systems. All random variables involved in the model construction are assumed to be exponentially distributed and independent of each other.

The main performance and reliability measures are derived, and some numerical calculations are carried out by the help of the MOSEL (Modeling, Specification and Evaluation Language) tool. The numerical results are graphically displayed to illustrate the effect of the non-reliability of the servers on the mean response time, on the overall system's utilization and on the servers's utilization.

Keywords: retrial queueing systems, finite number of sources, multiple server queues, asymmetric servers, non-reliable server, performance tool, performance and reliability measures, Fastest Free Server

1 Introduction

Retrial queueing models are often used for the performance and reliability modeling of computer systems and communication networks. The reason is that the return of customers plays a special role in many of these systems as well as in other practical applications, and it has a non-neglectable negative effect on the performance measures. For some applications of retrial queues, see for example [1], [2], [3] and [4], and for some fundamental results on finite-source retrial queueing systems refer to [5], [6], [7], [8], [9].

Usually the components of the computer systems are subject to random breakdowns, which has a heavy influence on the performance measures, so it is of practical importance to investigate non-reliable retrial queueing systems, too. Non-reliable infinite-source retrial queues

[1]Research is partially supported by German-Hungarian Intergovernmental Scientific Cooperation, HAS-DFG, 2005

were studied in the works [10], [11], [12] and finite-source retrial queues with a single non-reliable server in [13].
The purpose of this paper is to generalize the model of [9], [13] and to give the main stationary performability measures of the non-reliable multi server model described in the next section. Furthermore, our aim is to illustrate graphically the effect of the non-reliability of the servers on the steady-state systems's measures.
Because of the fact that the state space of the describing Markov chain is very large, it is difficult to calculate the system measures in the traditional way of solving the system of steady-state equations. To simplify this procedure we used the software tool MOSEL (Modeling, Specification and Evaluation Language), see [14], to formulate the model and to obtain the performance measures. By the help of MOSEL we can use various performance tools (like SPNP – Stochastic Petri Net Package) to get these characteristics. The results of the tool can graphically be displayed using IGL (Intermediate Graphical Language) which belongs to MOSEL.
The organization of the paper is as follows. Section 2 contains the accurate description of the investigated retrial queueing model, the derivation of the main steady state performance measures. Section 3 is devoted to the validation of the results of the tool and some graphically displayed numerical results. Following the Comments and Conclusions, in the Appendix, MOSEL source codes are given which were generated by our shell scripts for the 4 servers and 20 sources case.

2 The $M/\vec{M}/c//K$ retrial queueing model with non-reliable servers

Consider a finite-source retrial queueing system with c servers, where the primary calls are generated by K, $c < K < \infty$ sources. Each server can be in operational (up) or non-operational (down) states, and it can be idle or busy. Each source can be in three states: generating a primary call (free), sending repeated calls and under service by one of the servers. If a source is free at time t, it can generate a primary call during the interval $(t, t + dt)$ with probability $\lambda dt + o(dt)$. If one of the servers is up and idle at the moment of the arrival of the call then the service of the call starts. At the arrival of the calls the availability and idleness of the servers are always examined according to the increasing order of the servers indices, resulting different load to the servers. The service is finished during the interval $(t, t + dt)$ with probability $\mu_i dt + o(dt)$ if the ith server is available.
Server i can fail during the interval $(t, t+dt)$ with probability $\delta_i dt+o(dt)$ if it is idle, and with probability $\gamma_i dt + o(dt)$ if it is busy. If $\delta_i = 0, \gamma_i > 0$ or $\delta_i = \gamma_i > 0$ *active or independent breakdowns* can be discussed, respectively. If the server fails in busy state, it either continues servicing the interrupted call after it has been repaired or the interrupted request returns to the orbit. In this paper we only investigate the case when the source moves into the sending repeated calls state at the moment of server's failure. The repairman follows FIFO discipline for the servers' breakdowns, and the repair time of the ith server is exponentially distributed with a finite mean $1/\tau_i$. If all the servers are failed two different cases can be treated. Namely, *blocked sources* case when all the operations are stopped expect from the repair of the servers. In the *unblocked (intelligent) sources* case only service is interrupted but all the other operations are continued.
If all the servers are busy (or failed in the unblocked case) at the moment of the arrival of a

call the source starts generation of a Poisson flow of repeated calls with rate ν until it finds an available free server. After service the source becomes free, and it can generate a new primary call, and the server becomes idle so it can serve a new call. All the times involved in the model are assumed to be mutually independent of each other.

The state of the system at time t can be described by the process $X(t) = (\alpha_1(t), ..., \alpha_c(t); N(t))$, where $N(t)$ is the number of sources of repeated calls, $\alpha_i(t)$, i=1, ..., c, denotes the state of the ith server at time t. If there is a customer under service at the ith server, $\alpha_i(t) = 1$, if it is operational and idle then $\alpha_i(t) = 0$, otherwise the server is failed and $\alpha_i(t) = -1$.
Because of the exponentiality of the involved random variables this process is a Markov chain with a finite state space. Since the state space of the process $(X(t), t \geq 0)$ is finite, the process is ergodic for all reasonable values of the rates involved in the model construction. From now on we assume that the system is in the steady state.
Let us define the stationary probabilities by:

$$P(i_1, ..., i_c, j) = \lim_{t\to\infty} P\{\alpha_1(t) = i_1, ..., \alpha_c(t) = i_c, N(t) = j\}, \;\; i_1, ...i_c = -1, 0, 1, \;\; j = 0, ..., K^*,$$

$$\text{where} \quad K^* = K - \sum_{i_k, i_k=1} i_k.$$

Furthermore, let us denote by $C(t)$ the number of busy servers, by $A(t)$ the number of available servers at time t, and denote by $p_{kj} = \lim_{t\to\infty} P\{C(t) = k, N(t) = j\}$ the joint distribution of the number of busy servers and the number of repeated calls.

Once we have obtained the above defined probabilities the main steady state system performance measures can be derived as follows:

- *The probability that at least one server is available*

$$A_S = P\{\alpha_k > -1\}, k \in \{1, ..., c\} = 1 - \sum_{j=0}^{K} P(-1, ..., -1, j).$$

- *Mean number of sources of repeated calls*

$$N = E[N(t)] = \sum_{k=0}^{c} \sum_{j=1}^{K} j p_{kj} = \sum_{i_1,...,i_c} \sum_{j=1}^{K^*} j P(i_1, ..., i_c, j).$$

- *Utilization of the k-th server*

$$U_k = \sum_{i_1,...,i_c,i_k=1} \sum_{j=0}^{K^*} P(i_1, ..., i_c, j), \quad k = 1, ..., c.$$

- *Mean number of busy servers*

$$C = E[C(t)] = \sum_{k=1}^{c} U_k.$$

- *Mean number of calls staying in the orbit or in service*

$$M = E[N(t) + C(t)] = N + C.$$

- *Utilization of the repairman*

$$U_R = \sum_{\substack{i_1,\ldots,i_c \\ -1 \in \{i_1,\ldots,i_c\}}} \sum_{j=0}^{K^*} P(i_1, ..., i_c, j).$$

- *Utilization of the sources*

$$U_{SO} = \begin{cases} \frac{E[K-C(t)-N(t);A(t)>0]}{K} & \text{for blocked case,} \\ \frac{E[K-C(t)-N(t)]}{K} & \text{for unblocked case.} \end{cases}$$

- *Overall utilization of the system*

$$U_O = C + KU_{SO} + U_R.$$

- *Mean rate of generation of primary calls*

$$\overline{\lambda} = \begin{cases} \lambda E[K - C(t) - N(t); A(t) > 0] & \text{for blocked case,} \\ \lambda E[K - C(t) - N(t)] & \text{for unblocked case.} \end{cases}$$

- *Mean waiting time*

$$E[W] = N/\overline{\lambda}.$$

- *Mean response time*

$$E[T] = M/\overline{\lambda}.$$

3 Numerical examples

In this section we consider some numerical results in the case of homogeneous and heterogeneous servers to illustrate graphically the influence of the non-reliable servers on the mean response time $E[T]$, on the overall system's utilization U_O and server's utilization.

3.1 Validation of results

The results in the reliable case were validated by the Pascal program given in [5]. In Table 1 we can see that the corresponding performance measures are very close to each other, they are the same at least up to the 8th decimal digit. In the case of non-reliable servers, the results were tested by the $M/M/1//K$ retrial model with server's breakdowns which was studied in [13].

	MOSEL	Pascal program [5]
Number of servers:	2	2
Number of sources:	5	5
Request's generation rate:	0.1	0.1
Service rate:	1	1
Retrial rate:	1.1	1.1
Server's failure rate:	1e-25	–
Server's repair rate:	1e+25	–
Mean waiting time:	0.0653833701	0.0653833729
Mean number of busy servers:	0.4518596260	0.4518596267
Mean number of sources of repeated calls:	0.0295441060	0.0295441065

Table 1: Validations

	c	K	λ	μ_1, μ_2	ν	δ,γ	τ
Figure 1	2	5	x axis	1, 1	1.1	0.001	0.01
Figure 2	2	5	0.2	1, 1	x axis	0.001	0.01
Figure 3	2	5	0.2	x axis	1.1	0.001	0.01
Figure 4, 5	2	5	0.2	1, 1	1.1	x axis	0.01
Figure 6, 7	2	5	x axis	1.5, 0.5	1.1	0.001	0.01
Figure 8, 9	2	5	0.2	1.5, 0.5	x axis	0.001	0.01
Figure 10, 11, 12	2	5	0.2	1.5, 0.5	1.1	x axis	0.01

Table 2: Input system's parameters

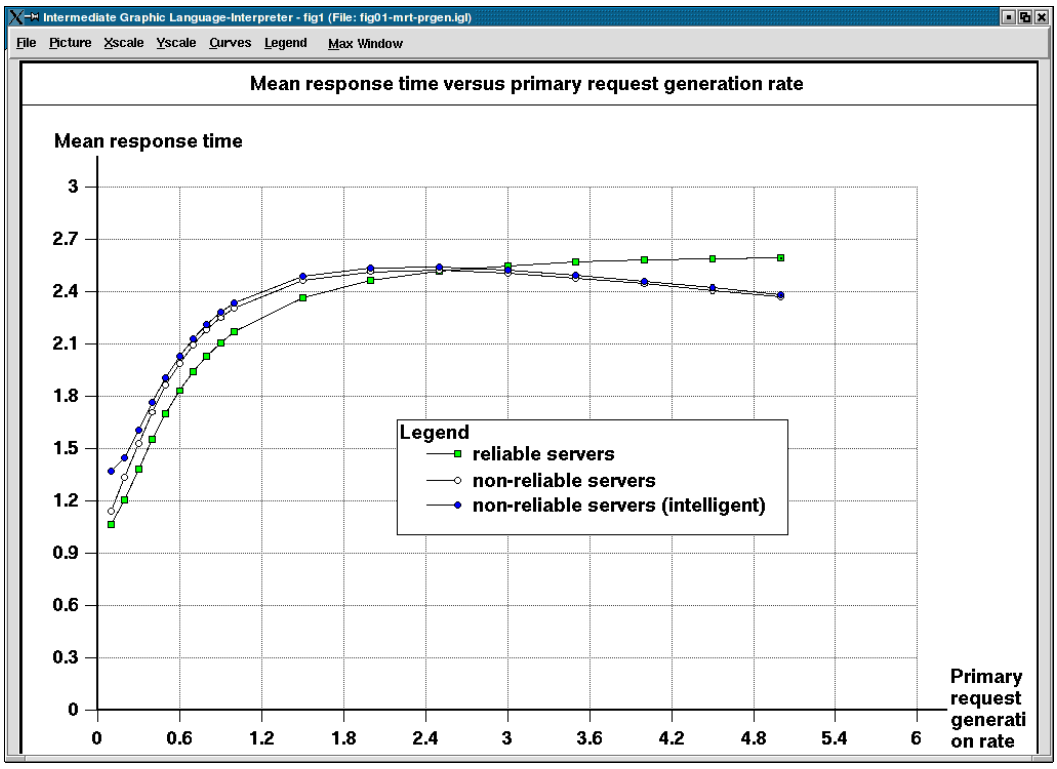


Figure 1: $E[T]$ versus primary request generation rate

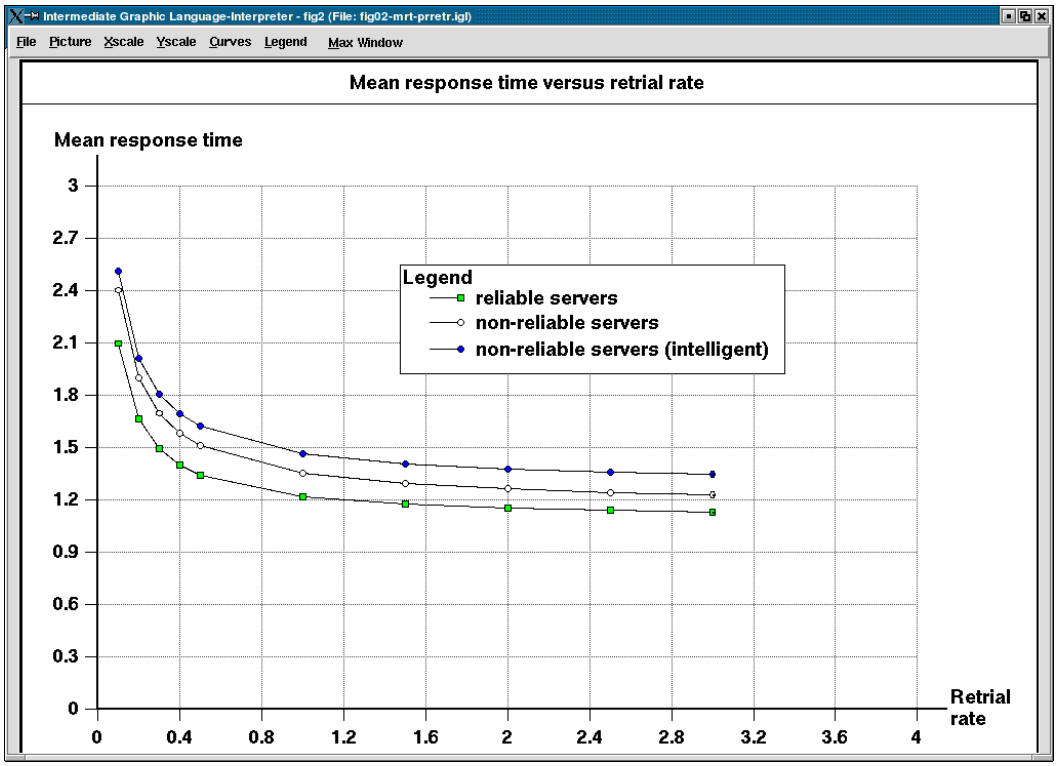


Figure 2: $E[T]$ versus retrial rate

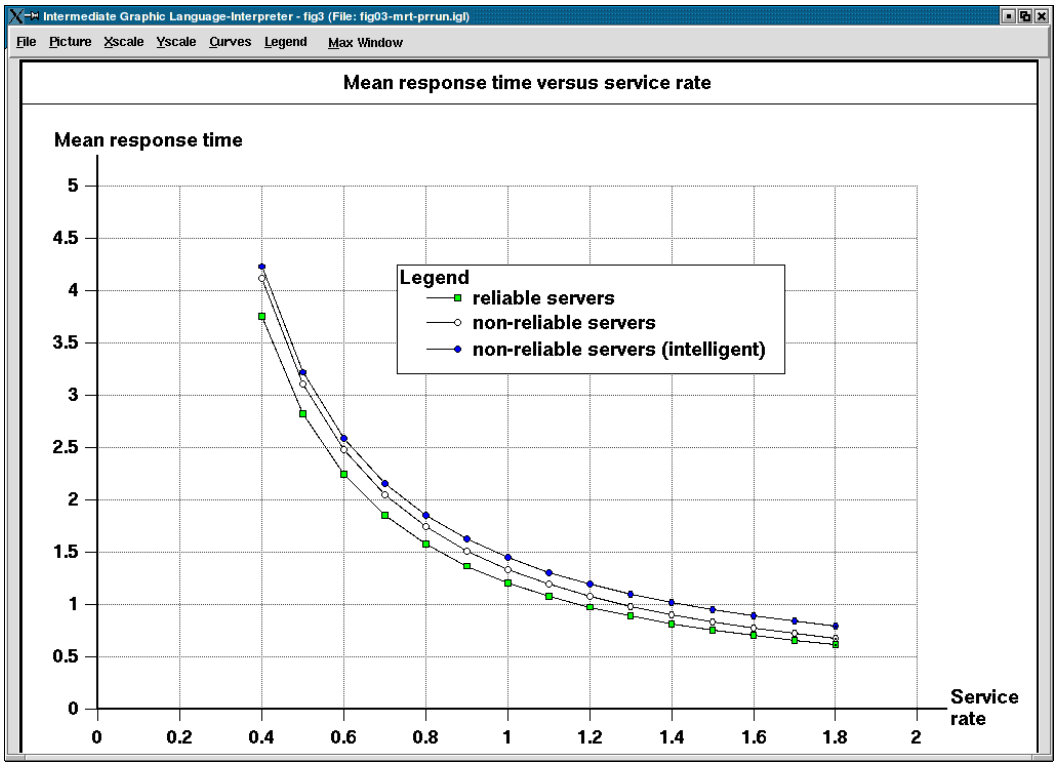


Figure 3: $E[T]$ versus service rate

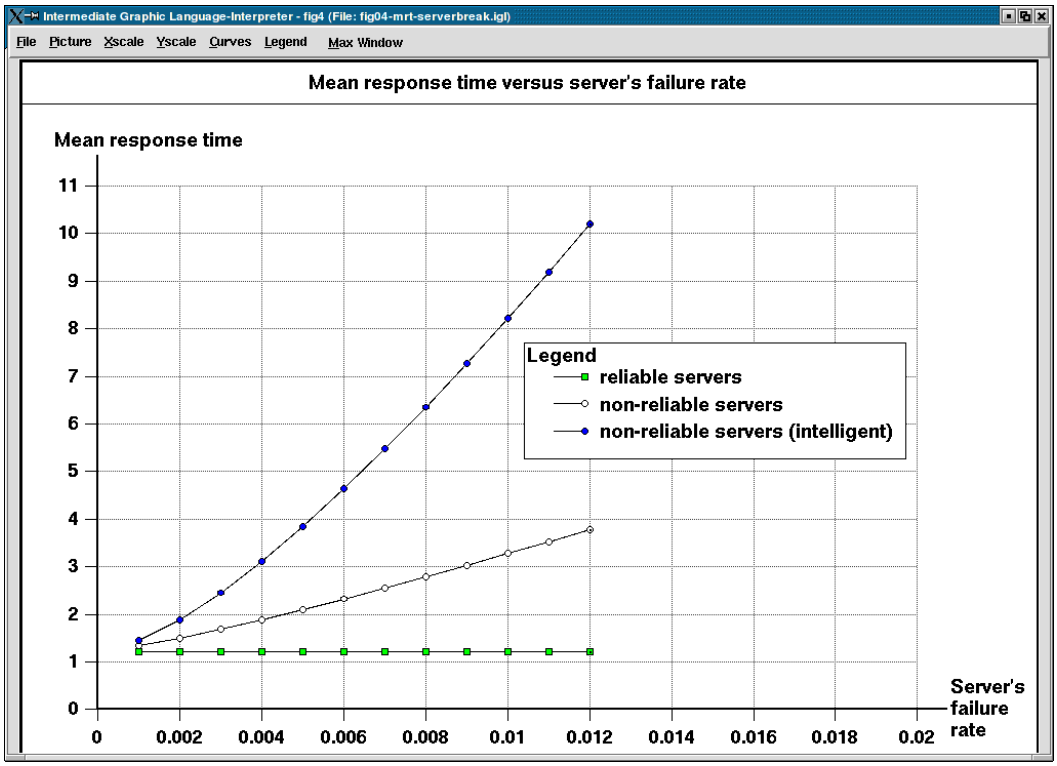


Figure 4: $E[T]$ versus server's failure rate

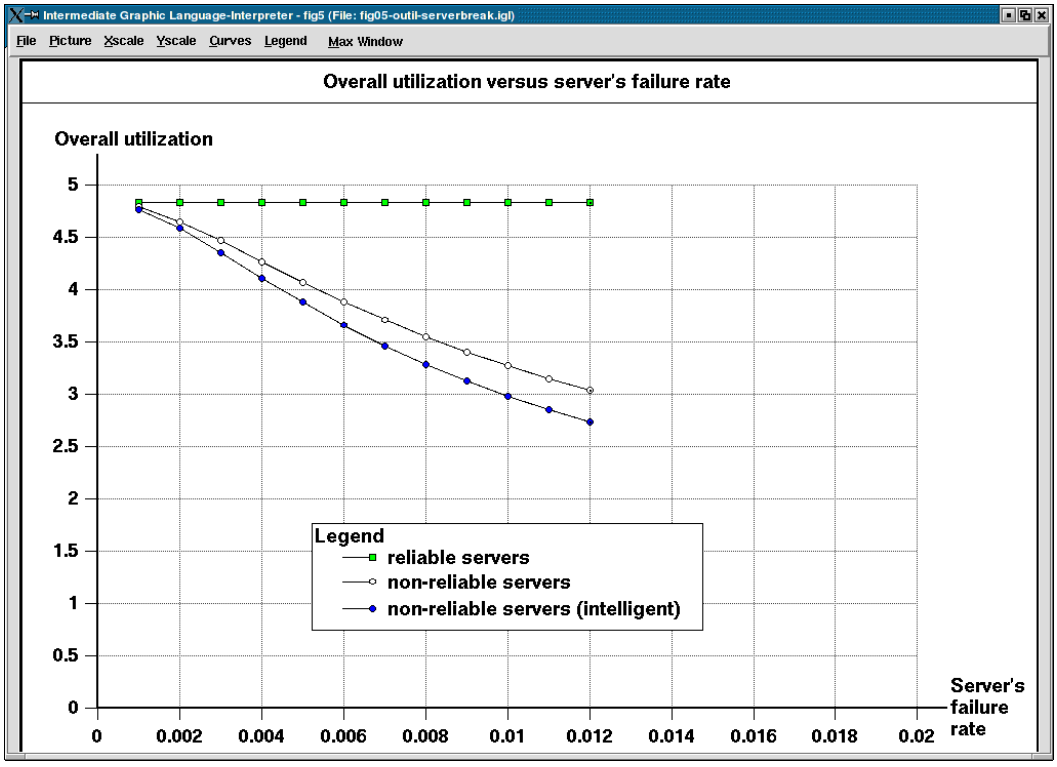


Figure 5: U_O versus server's failure rate

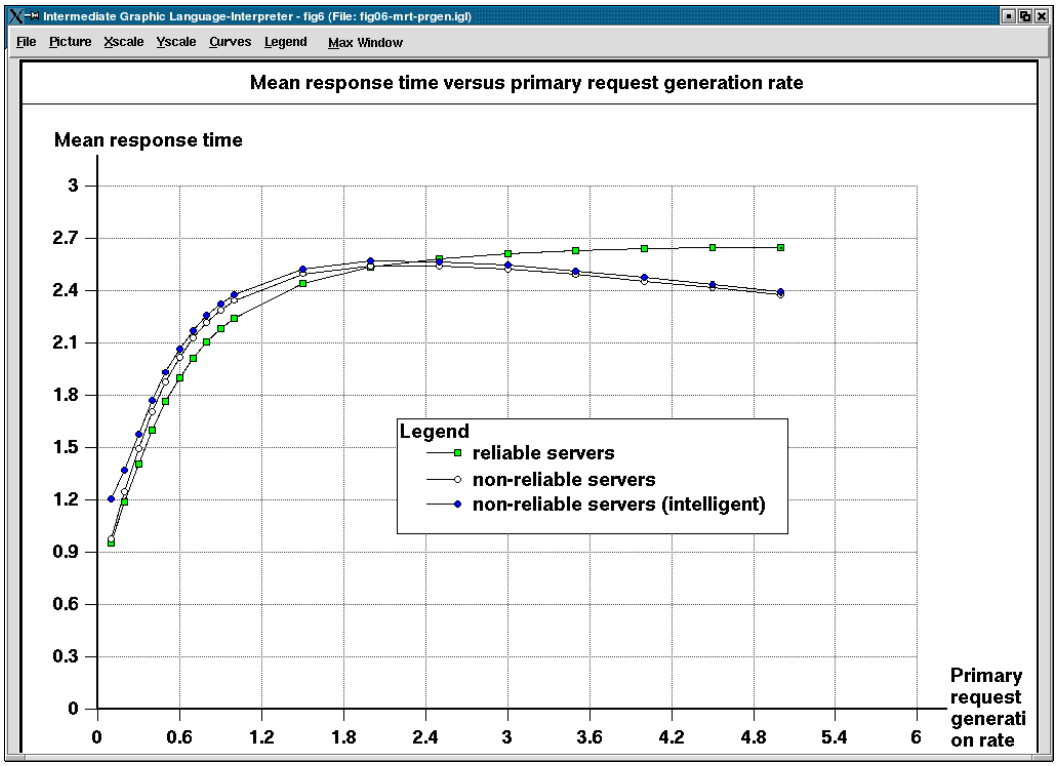


Figure 6: $E[T]$ versus primary request generation rate

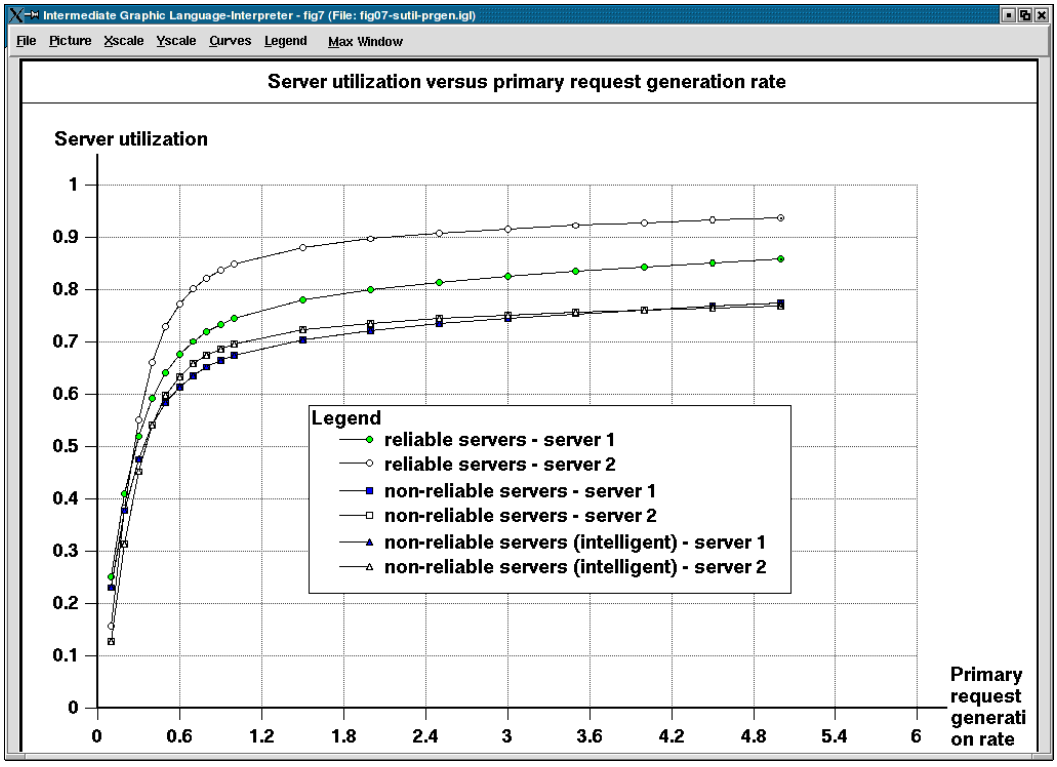


Figure 7: Server utilization versus primary request generation rate

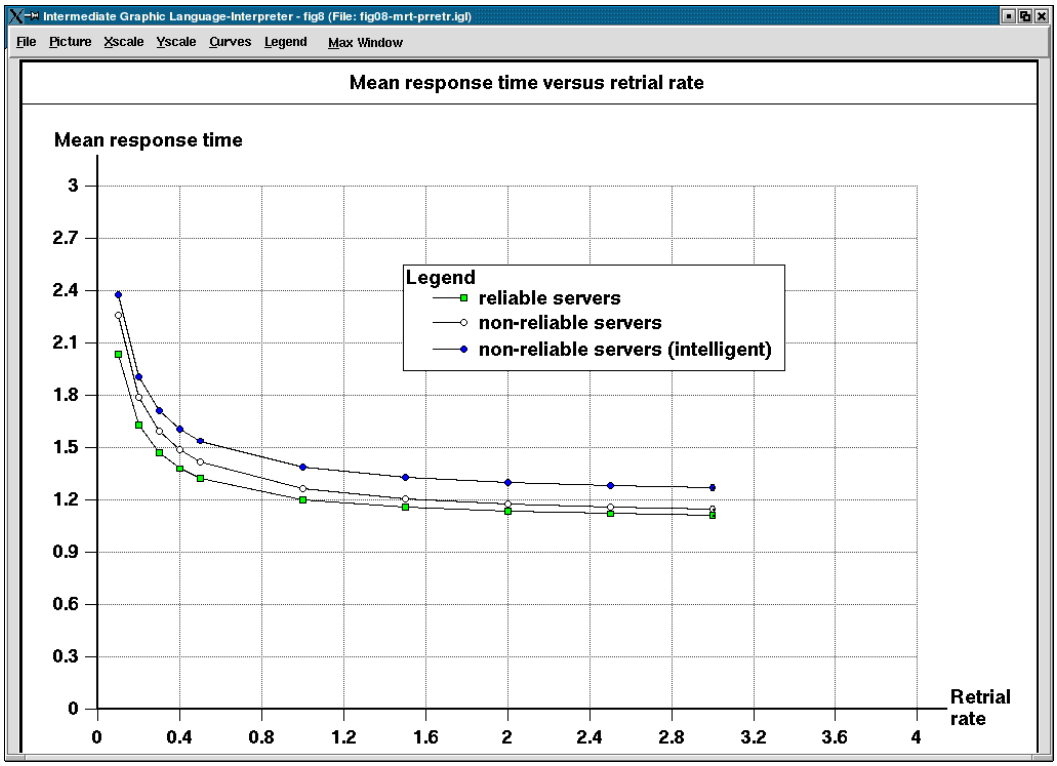


Figure 8: $E[T]$ versus retrial rate

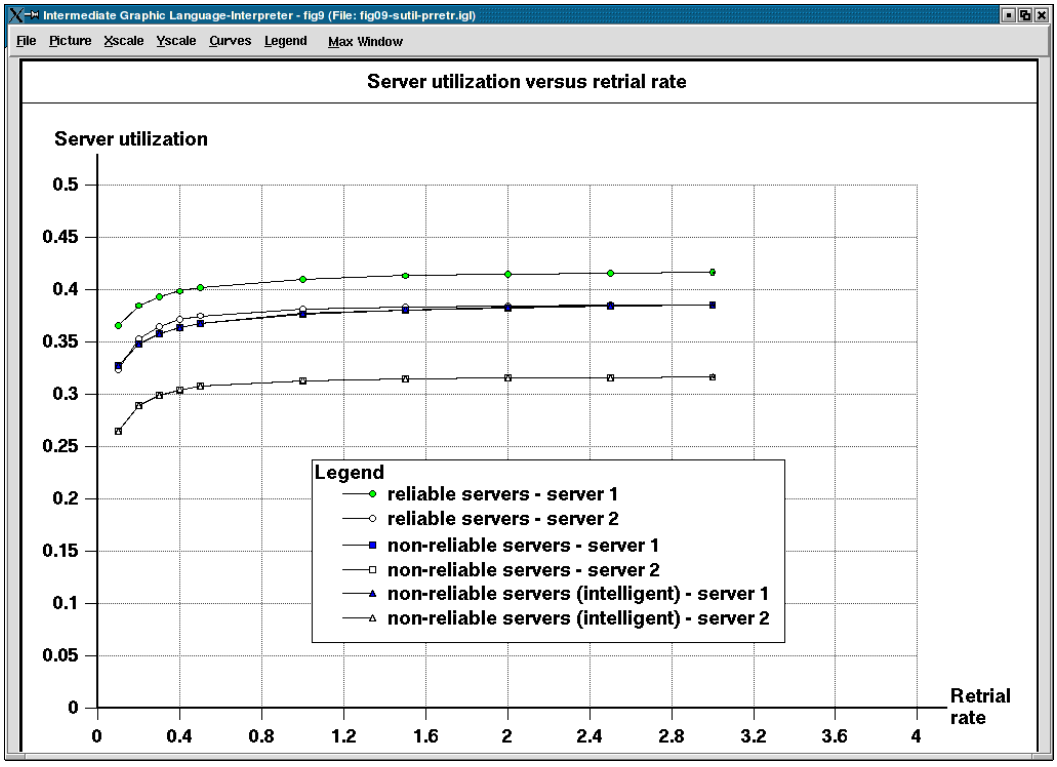


Figure 9: Server utilization versus retrial rate

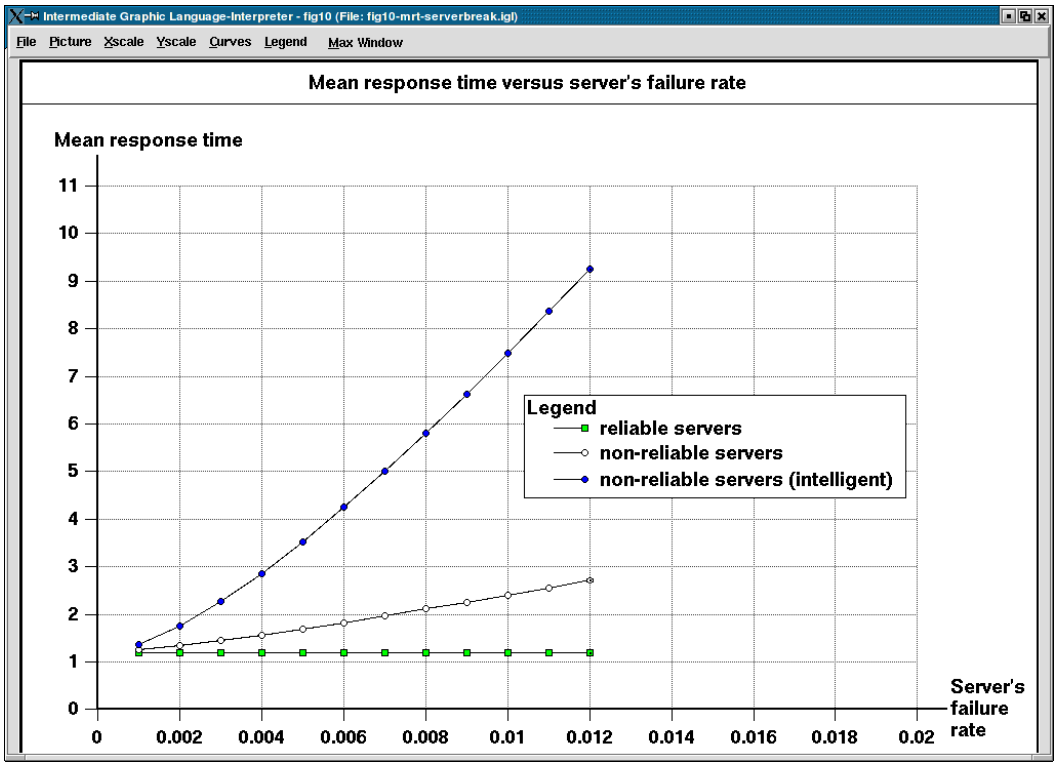


Figure 10: $E[T]$ versus server's failure rate

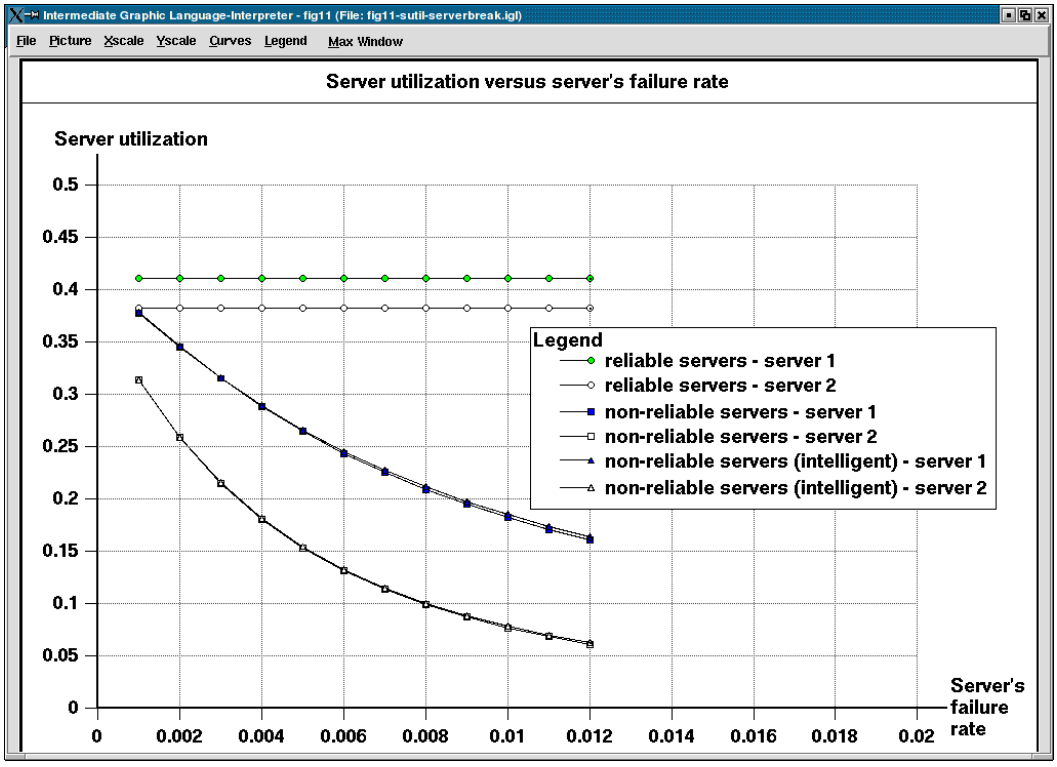


Figure 11: Server utilization versus server's failure rate

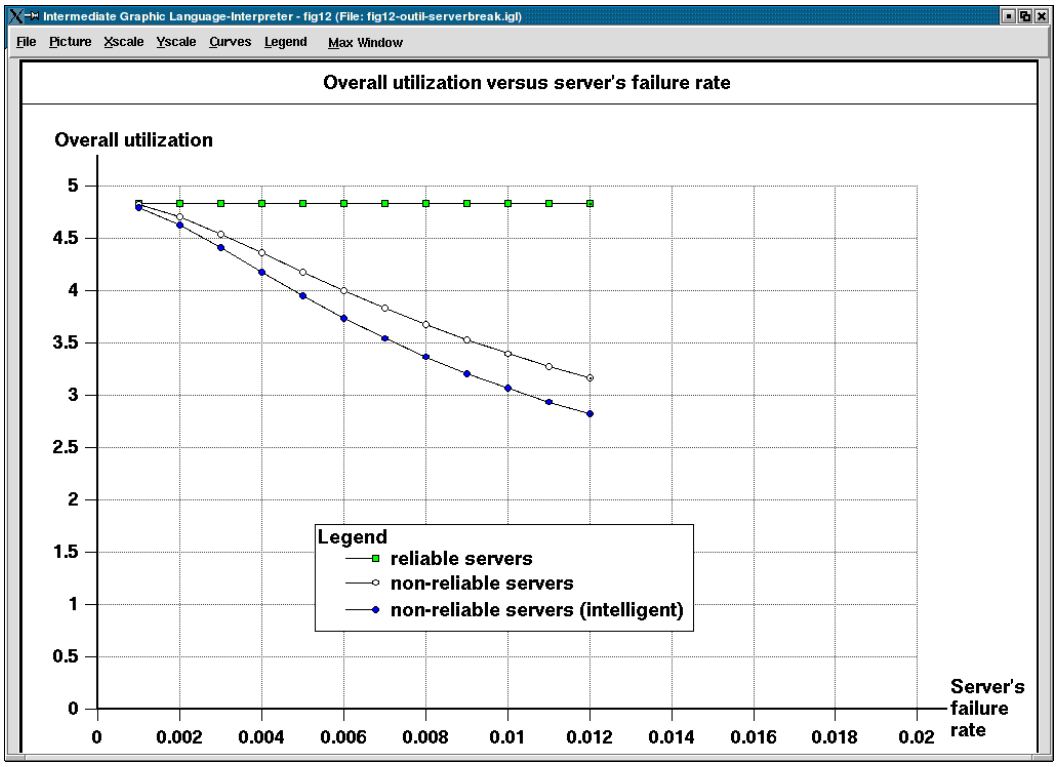


Figure 12: U_O versus server's failure rate

In the Figures the effect of some parameters is illustrated. The system parameters of the Figures are collected in Table 2.

In Figures 1-4, 6,8,10 the effects of the primary request generation rate, retrial rate, service rate and server's failure rate on the mean response time are displayed with homogeneous and different servers, respectively. In Figures 5,12 we can see the effect of server's failure rate on the overall utilization. In Figures 7,9,11 in the case of different servers the server's utilization is shown. In each Figure, the reliable, the blocked and unblocked (intelligent) cases are analyzed .

4 Comments

In Figures 1,6 we can see that with these parameter setup the difference is very small between the blocked and unblocked (intelligent) cases. The interesting phenomenon, which was mentioned in [7], too, that is retrial queues have a maximum of $E[T]$ is also noticed.
In Figures 2,8 it is demonstrated how long the retrial rate has a significant influence on the mean response time.
In Figure 3 we can see that the increase of the service rate has almost the same influence on the reliable and on the non-reliable systems.
In Figures 4,10 it can be observed that the increase of the server's failure rate can have a heavy impact on the mean response time, and as it increases the difference between the two non-reliable model increases significantly.
In Figures 5,12 it is shown that the overall utilization can be very low if the server's failure rate increases and the repair rate is not high enough.

We could see that the corresponding performance measures are better in the case of different servers, as it was expected.

5 Conclusions

In this paper, the performance of homogeneous finite-source retrial queueing systems with non-reliable heterogeneous servers is studied. The novelty of the investigation is the non-reliability and heterogeneity of the servers. The MOSEL language and software package was used to formulate the model and to calculate the system performance measures which were graphically displayed to show the effect of the non-reliability of the servers on the mean response times of the calls, on the overall system's utilization and on the servers's utilization.

References

[1] **Li H. and Yang T.** A single server retrial queue with server vacations and a finite number of input sources, *European Journal of Operational Research* 85(1995), 149-160.

[2] **Janssens G.K.** The quasi-random input queueing system with repeated attempts as a model for collision-avoidance star local area network, *IEEE Transactions on Communications* 45(1997), 360-364.

[3] **Tran-Gia P. and Mandjes M.** Modeling of customer retrial phenomenon in cellular mobile networks, *IEEE Journal of Selected Areas in Communications* 15(1997), 1406-1414.

[4] **Onur E., Delic H., Ersoy C. and Caglayan M.U.** Measurement-based replanning of cell capacities in GSM networks, *Computer Networks* 39(2002), 749-767.

[5] **Falin G.I. and Templeton J.G.C.** *Retrial queues*, Chapman and Hall, London, 1997.

[6] **Artalejo J.R.** Retrial queues with a finite number of sources, *J. Korean Math. Soc.* 35(1998), 503-525.

[7] **Falin G.I. and Artalejo J.R.** A finite source retrial queue, *European Journal of Operational Research* 108(1998), 409-424.

[8] **Artalejo J.R.** Accessible bibliography on retrial queues, *Mathematical and Computer Modelling* 30(1999), 1-6.

[9] **Falin G.I.** A multiserver retrial queue with a finite number of sources of primary calls, *Mathematical and Computer Modelling* 30(1999), 33-49.

[10] **Artalejo J.R.** New results in retrial queueing systems with breakdown of the servers, *Statistica Neerlandica* 48(1994), 23-36.

[11] **Aissani A. and Artalejo J.R.** On the single server retrial queue subject to breakdowns, *Queueing Systems* 30(1998), 309-321.

[12] **Wang J., Cao J. and Li Q.** Reliability analysis of the retrial queue with server breakdowns and repairs, *Queueing Systems* 38(2001), 363-380.

[13] **Almási B., Roszik J. and Sztrik J.** Homogeneous finite-source retrial queues with server subject to breakdowns and repairs, *Computers and Mathematics wit Applications* (submitted for publication).

[14] **Begain K., Bolch G. and Herold H.** *Practical performance modeling, application of the MOSEL language,* Kluwer Academic Publisher, Boston, 2001.

A The MOSEL program for the Fastest Free Server policy

```
// ================================== Constant definitions ====
#define NT 20
#define NS 4

// ============================ Variables (input parameters) ====
VAR double prgen;
VAR double prretr;
<1..NS> VAR double prrun#;
<1..NS> VAR double cpubreak_idle#;
<1..NS> VAR double cpubreak_busy#;
<1..NS> VAR double cpurepair#;

// ======================================= Node definitions ====
enum cpu_states {cpu_busy, cpu_idle, cpu_failed};
NODE busy_terminals[NT] = NT;
NODE retrying_terminals[NT] = 0;
NODE waiting_terminals[NS] = 0;
<1..NS> NODE cpu#[cpu_states] = cpu_idle;
        NODE freecpus[NS] = NS;
        NODE failedcpus[NS] = 0;
<1..NS> NODE sr#[NS] = 0;

// ========================================== Transitions ====
FROM cpu1[cpu_idle], busy_terminals, freecpus
        TO cpu1[cpu_busy], waiting_terminals
        W prgen*busy_terminals;
FROM cpu2[cpu_idle], busy_terminals, freecpus
        TO cpu2[cpu_busy], waiting_terminals
        IF cpu1==cpu_busy
        W prgen*busy_terminals;
FROM cpu3[cpu_idle], busy_terminals, freecpus
        TO cpu3[cpu_busy], waiting_terminals
        IF cpu1==cpu_busy
        AND cpu2==cpu_busy
        W prgen*busy_terminals;
FROM cpu4[cpu_idle], busy_terminals, freecpus
        TO cpu4[cpu_busy], waiting_terminals
        IF cpu1==cpu_busy
        AND cpu2==cpu_busy
        AND cpu3==cpu_busy
```

```
        W prgen*busy_terminals;
FROM busy_terminals
        TO retrying_terminals
        IF freecpus==0
        W prgen*busy_terminals;
FROM cpu1[cpu_idle], retrying_terminals, freecpus
        TO cpu1[cpu_busy], waiting_terminals
        W prretr*retrying_terminals;
FROM cpu2[cpu_idle], retrying_terminals, freecpus
        TO cpu2[cpu_busy], waiting_terminals
        IF cpu1==cpu_busy
        W prretr*retrying_terminals;
FROM cpu3[cpu_idle], retrying_terminals, freecpus
        TO cpu3[cpu_busy], waiting_terminals
        IF cpu1==cpu_busy
        AND cpu2==cpu_busy
        W prretr*retrying_terminals;
FROM cpu4[cpu_idle], retrying_terminals, freecpus
        TO cpu4[cpu_busy], waiting_terminals
        IF cpu1==cpu_busy
        AND cpu2==cpu_busy
        AND cpu3==cpu_busy
        W prretr*retrying_terminals;
<1..NS><NS> FROM cpu<#1>[cpu_busy], waiting_terminals{
  TO cpu<#1>[cpu_idle], busy_terminals, freecpus
  W prrun<#1>;
  TO cpu<#1>[cpu_failed],retrying_terminals,failedcpus,sr<#2>(<#1>)
  W cpubreak_busy<#1>;
  }
<1..NS><NS> FROM cpu<#1>[cpu_idle], freecpus
  TO cpu<#1>[cpu_failed], failedcpus, sr<#2>(<#1>)
  W cpubreak_idle<#1>;
<1..NS> IF sr1==# FROM sr1(#), cpu#[cpu_failed], failedcpus
           TO cpu#[cpu_idle], freecpus W cpurepair#;
<2..NS> IF sr<#-1>==0 FROM sr#(sr#) TO sr<#-1>(sr#);

// ================================================ Results ====
<1..NS> RESULT>> if(cpu#==cpu_busy) cpuutil# += PROB;
<1..NS> RESULT>> if(cpu#==cpu_busy) busycpus += PROB;
<1..NS> RESULT>> if(cpu#==cpu_idle OR cpu#==cpu_busy) goodcpus+=PROB;
<1..NS> RESULT>> if(cpu#==cpu_failed) nfailedcpus += PROB;
RESULT if(busy_terminals>0) busyterm += (PROB*busy_terminals);
RESULT>> termutil = busyterm / NT;
RESULT>> if(retrying_terminals>0) retravg+=(PROB*retrying_terminals);
RESULT>> if(failedcpus>0) repairutil += PROB;
RESULT if(waiting_terminals>0) waitall += (PROB*waiting_terminals);
RESULT>> resptime = (retravg + waitall) / NT / (prgen * termutil);
RESULT>> overallutil = busycpus + termutil*NT + repairutil;
```