RAIRO. RECHERCHE OPÉRATIONNELLE

M. BECKER
A.-L. BEYLOT
G. DAMM
W.-Y. THANG

Automatic run-time choice for simulation length in mimesis

RAIRO. Recherche opérationnelle, tome 33, n° 1 (1999), p. 93-115

<http://www.numdam.org/item?id=RO_1999__33_1_93_0>

RAIRO Rech. Opér.
(vol. 33, n° 1, 1999, pp. 93-115)

AUTOMATIC RUN-TIME CHOICE FOR SIMULATION LENGTH IN MIMESIS (*)

by M. BECKER ([1, 2]), A.-L. BEYLOT ([3]),
G. DAMM ([1, 2, 4]) and W.-Y. THANG ([4])

Communicated by Bernard LEMAIRE

Abstract. – *This paper presents an algorithm which prevents a simulation user from choosing a simulation length. This choice is always tricky and often leads to CPU-time waste, not to mention user-time waste.*

Too often, simulation users forget to compute confidence intervals: they only guess a simulation length and ignore the confidence on the simulation results. Those who do compute them generally try several lengths (and thus run several simulations) so as to obtain small enough confidence intervals.

The algorithm aims at optimizing this length choice by running only one simulation and by stopping it nearly as soon as possible, i.e. *when some predefined relative confidence intervals on each of the performance criteria are reached. For this purpose, the confidence intervals are periodically computed, at run-time, with the batch mean method. According to these intermediate results and to estimators properties, a mobile simulation length is (also periodically) predicted. The algorithm automatically determines batch size and batches number. This process goes on until all confidence intervals are smaller than the predefined thresholds.*

This algorithm is implemented in MIMESIS, a computer architecture performance evaluation tool.

Keywords: Simulation, stopping procedure, confidence intervals.

Résumé. – *Cet article présente un algorithme qui évite à l'utilisateur de choisir une durée de simulation. En effet, ce choix est toujours délicat et aboutit souvent à une consommation inutile de temps machine, sans compter la perte de temps pour l'utilisateur. Trop souvent, les résultats de simulation sont donnés sans intervalle de confiance. Les utilisateurs choisissent arbitrairement une durée de simulation et ignorent la confiance qu'ils peuvent accorder aux résultats. Ceux qui prennent la peine de calculer les intervalles de confiance essaient en général plusieurs durées (et lancent ainsi plusieurs simulations) jusqu'à obtenir des intervalles suffisamment petits.*

L'algorithme présenté ici vise à optimiser ce choix de durée en n'exécutant qu'une seule simulation qui est interrompue dès que (ou presque) certains seuils sur les intervalles de confiance sont atteints. Pour cela, ces derniers sont calculés plusieurs fois pendant la simulation à l'aide de la méthode des blocs. Grâce à ces résultats intermédiaires ainsi qu'aux propriétés des estimateurs, une durée de simulation est périodiquement estimée. Le nombre et la taille des blocs sont déterminés par

(*) Received May 1996.

([1]) INT/INF, 9, rue Charles Fourier, 91011 Évry Cedex, France.

([2]) MASI Laboratory Associate Members, 4, place Jussieu, 75252 Paris Cedex, France.

([3]) PRiSM Laboratory, University of Versailles-St-Quentin, 45, avenue des États-Unis, 78035 Versailles Cedex, France.

([4]) EDF/DER, 1, avenue du Général de Gaulle, 92141 Clamart Cedex, France.

E-mails: mbecker@int-evry.fr, Andre-Luc.Beylot@prism.uvsq.fr, Gerard.Damm@int-evry.fr, Wei-Ying.Thang@edfgdf.fr

Recherche opérationnelle/Operations Research, 0399-0559/99/01/$ 7.00

l'algorithme en cours d'exécution. L'algorithme s'arrête lorsque tous les intervalles de confiance sont plus petits que les seuils.

Cet algorithme a été intégré dans MIMESIS, un atelier d'évaluation des performances des architectures informatiques.

Mots clés : Simulation, procédure d'arrêt, intervalles de confiance.

1. INTRODUCTION

Simulation users are interested in obtaining performance criteria estimations of their modelled systems, such as a response time, an occupation rate or a throughput. It is very important to get these results with right confidence intervals. The latter depend on simulation length (*i.e.* the elapsed simulated time), the choice of which is therefore crucial and difficult, even for simulation specialists. If simulation length is chosen too short, the confidence in the results may be poor, especially if the simulation is dealing with quite rare events. If it is chosen too long CPU time is wasted. Moreover, if several simulations are run until results are reliable, CPU time is also wasted.

This paper presents an algorithm which is able to choose, at run-time, a satisfactory simulation length (*i.e.* with good confidence intervals on results). The user only has to launch the simulation without a time limit and results are obtained automatically. Should a burst of events happen, the algorithm would notice it and have the simulation run longer.

Confidence intervals are computed for each variable in the simulation. Several methods exist [1] among which the most generally implemented are:

– replication/deletion methods;

– sequential methods.

We did not use replication/deletion methods [2] since independence between runs has to be proven and also because the transient part at the beginning of each replication is expensive [3].

As for the sequential methods, many nice papers have been written when theoretical assumptions are made about the processes. For each variable, the confidence interval depends upon the estimator variance. When an assumption of strong invariance holds (*i.e.* when the centered partial sum process is close to a Brownian motion with probability one), "consistency of the variance estimator can be achieved even in the nonstationary context" [4].

The number of batches and the size of batches that give an optimal confidence interval can be computed when some theoretical assumptions hold [5].

Some recent methods consist of two stages:

– in the first stage, the confidence intervals are estimated for a given batch size and a given batch number [6]. Then, using theoretical results, the batch size or the batches number are computed in order to get a predefined confidence interval,

– in the second stage, the simulation is run and the confidence interval is not computed again.

Our work does not make assumptions about the laws of the estimators variances. We only assume that the variances exist. So, we do not try to determine the batch size as a function of these laws.

Besides, we prefer to deal with a small enough number of batches (a few tens) and to choose an adequate size for a batch since it is preferable to have batches that are not highly correlated: the variance estimator is good enough. Our algorithm uses a multistage method and at each stage, the confidence interval is estimated again.

Since confidence intervals are considered for several criteria, no assumption is made concerning the estimator processes. From each theoretical law of each estimator process depending on some parameter λ_i (for criterion i), nice computations for optimal size of batches and optimal number of batches could be derived but these numbers would be different for each criterion i. Our method is intended to be simpler.

This algorithm has been implemented in MIMESIS (Machine-Implemented Modeling and Exploratory Simulation from Initial Specification), which is a CASE (Computer Aided Software Environment) designed for computer architectures performance evaluation. It was initially developed at EDF [7] and is now supported by Delta Partners. Potential users are not supposed to master simulation techniques. Therefore, it is important to hide from them a simulation-specific problem such as simulation length choice.

However, a stop condition for the method has to be specified in some way. In this algorithm, the user chooses some performance criteria (C_i) and indicates, for each one, a confidence level 1-η_i and a relative confidence interval radius ρ_i to be reached. The algorithm computes confidence intervals $\rho_{i,T}$ at certain instants T for each C_i and stops the simulation when all confidence levels are smaller than those specified (*i.e.* when $\forall i,\ \rho_{i,T} < \rho_i$). In MIMESIS, default value for 1-η_i is 95% and default value for ρ_i is 10%. The performance criteria than can be chosen are those of the components of the MIMESIS-specific formalism [8].

Our method is relevant only for second order asymptotically stationary processes. Only the simulation length is validated, not the accuracy of the model. Let us consider the example of a simulation that yields a response time estimation of 100 ms, with a 8.3% confidence interval at a 95% confidence level. This means that there is a 95% probability that the result of an infinite-length simulation of *this* model is between 91.7 ms and 108.3 ms (but an infinite-length simulation is impossible, hence the interest of the algorithm). Let us note that, even with this method, the necessary simulation length can be too long in the case of too rare events or too bursty processes.

The algorithm is iterative. A first simulation time T_1 is determined. When T_1 is reached, confidence intervals are computed. If they are all small enough, the simulation is stopped and results are displayed. If at least one of them is too large, a new simulation time T_2 is determined, as a function of T_1 and $(\rho_{i,T_1})_i$. When T_2 is reached, $(\rho_{i,T_2})_i$ are computed and compared to the $(\rho_i)_i$, and so on. The method usually converges in only 2 or 3 iterations. In the following, the choice of $T_1, T_2, \ldots, T_j$ and the method for estimating $(\rho_{i,T_j})_i$ are emphasized.

The paper is organized as follows. In Section 2, we present the theoretical justification of the method (how and when to compute confidence intervals, how to choose the next iteration length). The algorithm itself is described in Section 3. For validation purpose, a FORTRAN program was written. The algorithm was also implemented in MODSIM, in a modular and re-usable way, so that it could be integrated in MIMESIS. Section 4 reports about these implementations. A few results tables are also presented.

2. ALGORITHM JUSTIFICATION

2.1. Introduction

For complex systems performance evaluation, stochastic models are often well suited, since they allow to study access concurrency. This paper is concerned with models studied by simulations rather than by analytical solutions. We focus on validating a simulation in relation to the steady-state problem, which consists in determining confidence intervals and deciding whether or not the simulation should be stopped.

There are two kinds of processes:

- $X(t)$, function of the time;
- X_k, function of k, the number of events.

All the processes are assumed to be strictly ergodic.

The time average of process $X(t)$ is defined as $Z_T = \frac{1}{T}\int_0^T X(t)dt$. Z_T depends on the model and on the simulation execution. Strict ergodicity implies that for any function Y of $X(t)$:

$$E\left[\frac{1}{T}\int_0^T Y(X(t))dt\right] \xrightarrow[T\to\infty]{} Z$$

where $Z = \lim_{t\to\infty} E[Y(X(t))]$.

However, a variable in a computer program is a step-function: its value is changed when it is left-accessed (*i.e.* modified) at instants t_i; it is a constant otherwise (variables in a simulation could be defined in a more complicated way but only simple variables are considered here).

Therefore, in a discrete-event simulation,

$$T = t_p \qquad \text{and} \qquad Z_T = \frac{1}{t_p}\sum_{i=1}^{p} X(t_i)\cdot(t_{i+1} - t_i),$$

where p is the number of left-accesses.

A customer number in a queue or the occupied size of a memory are two examples of such variables.

We are also interested in variables X_k that are not directly function of the time: X_k is only defined when certain events occur. For this kind of performance criteria, the average is not computed with respect to time, but with respect to events. The event average is simply the arithmetic mean: $Z_N = \frac{1}{N}\sum_{k=1}^{N} X_k$, where N is the number of different values for X_k. In this case, strict ergodicity means that for any function Y of X_k :

$$E\left[\frac{1}{N}\sum_{k=1}^{N} Y(X_k)\right] \xrightarrow[T\to\infty]{} Z$$

where $Z = \lim_{t\to\infty} E[Y(X_k)]$.

A message transfer delay is an example of such a variable.

The problem is the following: if a variable is observed during a simulation, what confidence do we have in the time average? To put it in another way, how *close* are Z_T or Z_N (which are known from the simulation) from Z (which is the figure we want to know)?

The distance between Z_T (or Z_N) and Z can be precisely known with confidence intervals. How they are defined is described in Section 2.2. How they are used to choose a simulation length is described in Section 2.3. In order to make this paper clearer, only the time average case (Z_T) is

presented. As for the event average case (Z_N), the method is the same for the following reason: to determine a large enough events number N (for confidence intervals to be small enough) is equivalent to determine a large enough simulation length T since N and T tend to be proportional (the ratio $\frac{N}{T}$ tends to the density of events), because of the assumptions that were made.

2.2. Confidence intervals

Let us remind the well-known batch mean method for estimating confidence intervals [9, 10].

Z_T is an estimator of $Z = E[X(t)]$. The absolute confidence interval radius on Z_T, with a confidence level of 1-η, is a number $\varepsilon > 0$ such that:

$$\Pr(|Z_T - Z| < \varepsilon) \geq 1\text{-}\eta \tag{1}$$

The relative confidence interval radius is $\frac{\varepsilon}{Z_T}$. The value of 1-η can be chosen and Z_T is obtained from the simulation. We intend to estimate Z. How can ε be found?

Let us apply Tchebychev's theorem:

$$\Pr[|Z_T - Z| \geq \varepsilon] \leq \frac{E[|Z_T - Z|^\alpha]}{\varepsilon^\alpha} \tag{2}$$

(on condition that $E[|Z_T - Z|^\alpha] < +\infty$).

If $\alpha = 2$ and if it is assumed that $Z \approx E[Z_T]$ (when T is large enough), then:

$$E[|Z_T - Z|^2] \simeq E[|Z_T - E[Z_T]|^2] = \sigma^2(Z_T)$$

(variance of the estimator Z_T, which is not the variance of the random variable $X(t)$).

Therefore, inequality (2) is equivalent to:

$$\Pr(|Z_T - Z| < \varepsilon) \geq 1 - \frac{\sigma^2(Z_T)}{\varepsilon^2}$$

which is equivalent to inequality (1) for $\varepsilon = \frac{\sigma(Z_T)}{\sqrt{\eta}}$.

If the confidence level is 95%, then:

$$\eta = 0.05 \qquad \text{and} \qquad \frac{1}{\sqrt{\eta}} < 4.5$$

As a consequence, $\varepsilon = 4.5\sigma(Z_T)$ can be chosen as a confidence interval for a confidence level of 95%.

Let us note that, with a high probability, the estimator tends to a normal law and this confidence interval could be reduced to $2\sigma(Z_T)$. Since we

wanted our confidence to be very safe, we chose to keep the coefficient 4.5. The user will nevertheless be notified that the confidence interval could probably be divided by 2.25.

Now the question is: how can $\sigma(Z_T)$ be estimated?

If $X(t)$ is a second order asymptotically stationary process, it can be proved that $\sigma^2(Z_T) \approx \frac{A}{T}$, where A is some positive constant. Neither $\sigma^2(Z_T)$ nor A can be directly computed in general.

In order to compute $\sigma^2(Z_T)$ in spite of this, the simulation is cut into n batches of length $T_{\text{batch}} = \frac{T}{n}$ so as to estimate $\sigma^2(Z_{T_{\text{batch}}})$, variance of $Z_{T_{\text{batch}}}$.

Since $\sigma^2(Z_{T_{\text{batch}}}) \approx \frac{A}{T_{\text{batch}}}$, then:

$$\sigma^2(Z_T) = \frac{T_{\text{batch}}}{T}\,\sigma^2(Z_{T_{\text{batch}}}) = \frac{1}{n}\,\sigma^2(Z_{T_{\text{batch}}}).$$

The variance of $Z_{T_{\text{batch}}}$ is estimated with the well-known formula: $\frac{1}{n-1}\sum_{i=1}^{n} Z_i^2 - nZ_{T^2}$, where Z_i is the time average of $X(t)$ on the $i^{\underline{\text{th}}}$ batch:

$$\frac{1}{T_{\text{batch}}}\int_{(i-1)\cdot T_{\text{batch}}}^{i\cdot T_{\text{batch}}} X(t)dt$$

and Z_T is the arithmetic average $\frac{1}{n}\sum_{i=1}^{n} Z_i$.

Thus, only n and T_{batch} have to be chosen to get the confidence interval this way:

- simulate the system until the conditions to define T_{batch} are true;
- simulate the system for $n-1$ batches of length T_{batch} (total length: $n \cdot T_{\text{batch}}$);
- at the end of each batch, compute Z_i;
- at $T = n \cdot T_{\text{batch}}$, compute:

$$Z_T = \frac{1}{n}\sum_{i=1}^{n} Z_i \quad \text{(estimation of } Z\text{)}$$

$$\sigma^2(Z_{T_{\text{batch}}}) = \frac{1}{n-1}\sum_{i=1}^{n} Z_i^2 - nZ_T^2$$

$$\varepsilon = \frac{1}{\sqrt{\eta}}\,\frac{\sigma(Z_{T_{\text{batch}}})}{\sqrt{n}}$$

$$\left(\text{or } \varepsilon = 4.5\,\frac{\sigma(Z_{T_{\text{batch}}})}{\sqrt{n}} \text{ if } \eta = 0.05\right).$$

The conclusion is: "With a confidence level of $(1-\eta)$, $Z \in [Z_T - \varepsilon; Z_T + \varepsilon]$."

Let us note that the estimation of $\sigma^2(Z_{T_{\text{batch}}})$ is approximate but the error can be neglected if T_{batch} is large enough. The equation $\sigma^2(Z_T) \approx \frac{A}{T}$ assumes strict ergodicity and asymptotic stationarity.

2.3. Use in the algorithm

How can this material be used for an automatic simulation length choice? For each performance criteria C_i, at instant T, the estimator $Z_{i,T}$ can be computed with a confidence interval $\varepsilon_{i,T}$ at a confidence level 1-η_i. Relative confidence interval radius $\rho_{i,T}$ is $\frac{\varepsilon_{i,T}}{Z_{i,T}}$ and we can compare $\rho_{i,T}$ to the desired ρ_i.

The first step is to dimension T_{batch} for the first iteration. For this purpose, a counter with a limit value is attached to each observed variable C_i. Counters can also be attached to other variables in the model. When all counters have reached their limit values, the current simulation time is chosen as the first T_{batch} value. What is the best limit value? On the one hand, the variables must vary often enough so that estimations on T_{batch} are significant. On the other hand, they must not vary too often so that $n \cdot T_{\text{batch}}$ is not too large. Our experiments showed that good counter values are of the order of 500 to 1000. Smaller values can be chosen for quite rare events, for which only a rough idea is required. Counter values also depend on $(\varepsilon_i)_i$.

Let us explain now the main loop in the algorithm. If, at $T = n \cdot T_{\text{batch}}$, confidence intervals are not small enough, the simulation must proceed until an instant T' greater than T. Since the variance of each estimator $Z_{i,T}$ is $\sigma^2_{i,T} \approx \frac{A_i}{T}$, the confidence interval is $\varepsilon_{i,T} \approx \frac{B_i}{\sqrt{T}}$, where B_i is a constant. If $\varepsilon_{i,T} > \varepsilon_i$, performance criterion C_i is not well enough estimated at T. Let δ_i be the necessary time to obtain the confidence interval ε_i for C_i. δ_i and T are related: $\delta_{i,\varepsilon_i^2} = T\varepsilon^2_{i,T}$ or:

$$\delta_i = \left(\frac{\varepsilon_{i,T}}{\varepsilon_i}\right)^2 T.$$

Any length $T' > \delta_i$ is also suitable. In order to reach good confidence intervals for each C_i, we therefore choose $T' = \text{Max}_i(\delta_i)$, or:

$$T' = \left[\text{Max}_i\left(\frac{\varepsilon_{i,T}}{\varepsilon_i}\right)\right]^2 T.$$

Since it is an approximation, it may be necessary to iterate once more. The coefficient $\theta = \frac{T'}{T}$ also changes batch size: $T'_{\text{batch}} = \theta \cdot T_{\text{batch}}$. We will see in the next section why it is more interesting to choose $\lceil \theta \rceil$ (smaller integer greater or equal to θ) as an iteration coefficient.

There are further refinements (in order to spare CPU time), such as batch re-use, which consists in "recycling" old batches to construct new batches, and batches number increase, which consists in increasing n (number of batches) when $\theta < 2$.

3. ALGORITHM

The algorithm has to choose the simulation size, *i.e.* the batch size, since the batches number is a parameter. However, the batches number can be increased during the simulation under certain circumstances.

3.1. Sparing CPU time

3.1.1. *Batch re-use*

If the system has been simulated until T and if $T' = \theta T$ has been computed as described before, n batches of size $T'_{\text{batch}} = \theta T_{\text{batch}}$ could be started again between T and $T + T'$. We rather try not to lose the simulation work already done, especially if θ is small. Simulation length could be T' instead of $T + T'$. Since all computations make use of averages on batches, it would be easy to gather them to form a new average on a new batch, provided θ were an integer, which is not the case. For this reason, we choose $\beta = \lceil \theta \rceil$ as the next iteration coefficient: $T' = \beta T$ and $T'_{\text{batch}} = \beta T_{\text{batch}}$. The averages on β last iteration batches are grouped and the average on next iteration batches are: $A'_1 = \frac{1}{\beta}(A_1 + A_2 + \ldots + A_\beta)$, $A'_2 = \frac{1}{\beta}(A_{\beta+1} + A_{\beta+2} + \ldots + A_{2\beta})$, and so on. Grouping is made according to the following cases:

- if n is a multiple of β, then $\frac{n}{\beta}$ new batches can be retrieved and only $(n - \frac{n}{\beta})$ new batches have to be simulated until T' is reached;
- if n is not a multiple of β, a few more "old-sized" batches are simulated so as to complete a new group of β batches. Thus, $\lfloor \frac{n}{\beta} \rfloor + 1$ batches can be retrieved;
- finally, if $\beta > n$, T was definitely too small (smaller than a new batch) and no batch can be re-used. In this case, the simulation between 0 and T is dropped and n new batches will be simulated between T and $T + T'$. This case should not occur if the counter values are chosen large enough.

Also, if the first T_{batch} is chosen too short, the estimation of β can be very wrong and be extremely large. Some authors then suggest to limit β to a maximum value in order to avoid a uselessly long simulation [11]. This problem does not occur with the algorithm thanks to the counter feature (but a maximum β value is implemented anyway).

Finally, let us note that rounding θ up can avoid a useless iteration. If we choose exactly $T' = \theta T$, then $\varepsilon_{i,T'}$ can be slightly greater than ε_i, therefore leading to a new iteration whereas confidence interval is nearly correct. If we rather choose $T' = \lceil\theta\rceil T$, $\varepsilon_{i,T'}$ is more likely to be greater than ε_i, for a low extra simulation cost (especially if θ is not too small).

The batch re-use is not possible with replicated simulations. Thus, a single long simulation is less expensive not only because of a single steady-state establishment, but also because it allows batch re-use as described above.

3.1.2. *Batches number increase*

As $\beta = \lceil\theta\rceil$ is an integer, 2 is the smallest value for β for which the simulation proceeds (if $\beta = 1$, stop condition is reached). If the simulation has already lasted long and if θ is only slightly greater than 1, to round θ up will probably result in CPU time waste. This is why the batch size is not changed when $\beta = 2$. Instead, the number of batches is increased. If n is the default number (usually 16), a $(n+1)^{\underline{\text{th}}}$ batch is simulated and the $(\varepsilon_{i,(n+1)T_{\text{batch}}})_i$ are compared to the $(\varepsilon_i)_i$. If necessary, a $(n+2)^{\underline{\text{th}}}$ batch is simulated, and so on. If $nT_{\text{batch}} = T$ and if $\theta \in [1+\frac{k}{n}; 1+\frac{k+1}{n}](k < n)$, we may need to simulate only $(k+1)$ batches (instead of n if we had chosen $T'_{\text{batch}} = 2T_{\text{batch}}$). Since this is only a guess, the increase had better be limited: a maximum of n other batches is allowed (simulation time would then be of $2T$). If confidence intervals are not reached at this limit, batches are grouped for the next iteration.

3.1.3. *Algorithm convergence*

Non-convergence can occur if required $(\varepsilon_i)_i$ are too small or if confidence levels $(1-\eta_i)_i$ are too high. In this case, even if the method assumptions are verified (processes are strictly ergodic), theoretical results apply but are out of reach with the computer's resolution: numeric precision is too low [12]. Averages and confidence intervals may however be presented, even if the latter are not those specified.

Non-convergence can also occur if the method assumptions are not verified. For example, the modelled system can be unstable (and a variable of a bottlenecked region has been chosen as a performance criterion) or the

program may include a bug (causing for example a variable chosen as a performance criterion to keep increasing). In this case, theoretical results do not apply and no result should be presented.

A simple mechanism prevents infinite loops in both cases: the iteration number is limited to a maximum. The chosen maximum value is four completed iterations.

3.2. Algorithm description

- simulate the system until each counter has reached its limit value
- T_{batch} := current simulation time
- compute and save estimators $Z_{i,1}$ on this first batch
- TheEnd := FALSE
- FirstBatch := 2; LastBatch := n;
- main loop:

while NOT TheEnd
  for NumBatch=FirstBatch **to** LastBatch
    simulate the system during T_{batch}
    compute and save estimators $Z_{i,\text{NumBatch}}$
  end for
  compute estimators $Z_{i,T}$
  compute confidence intervals $\varepsilon_{i,T}$ and β
  case of β:
    $\beta = 1$: TheEnd := TRUE
    $\beta = 2$: batch number increase (batch size: T_{batch})
    $\beta > n$: **if** $\beta >$ Maxβ **then** $\beta :=$ Maxβ **end if**
      $T_{\text{batch}} := \beta T_{\text{batch}}$
      FirstBatch :=1; LastBatch := n
    $2 < \beta \leq n$:
      if $\frac{n}{\beta}$ is integer **then**
        $T_{\text{batch}} := \beta T_{\text{batch}}$
        $K := \frac{n}{\beta}$
        retrieve K batches (size: T_{batch})
        FirstBatch : =$K + 1$; LastBatch : =n
      else
        $K := \lfloor \frac{n}{\beta} \rfloor$
        retrieve K batches (size: βT_{batch})
        FirstBatch := $n + 1$; LastBatch := $(K + 1)\beta$
        for NumBatch=FirstBatch **to** LastBatch
          simulate the system during T_{batch}
          compute and save estimators $Z_{i,\text{NumBatch}}$
        end for
        retrieve another batch (size: βT_{batch})
        $T_{\text{batch}} := \beta T_{\text{batch}}$
        FirstBatch := $K + 2$; LastBatch := n
      end if
  end case
  stop simulation if IterationNumber > MaxIterations
end while

- display results (or write in a file): $Z_{i,T}$ and $\varepsilon_{i,T}$

4. IMPLEMENTATION

4.1. FORTRAN

The first implementation of this algorithm was developed in FORTRAN. It did not include the batches number increase and maximum β value features. The purpose of this implementation was to validate the method.

The chosen example was a $M/M/1$ queue simulation. In this model, customers arrive in a queue and are served according to a First-In-First-Out policy. The server is characterized by its service rate.

Customers arrival process is of Poisson type, with parameter λ. The service time is exponential, with parameter μ.

Queue capacity is infinite. The chosen performance criteria are:

- mean customer number in the queue (time average);
- mean response time (event average);
- mean waiting time (event average);
- occupation rate of the server (time average);
- throughput (time average).

The throughput is a special case: the estimator on a batch is the served customers total number divided by batch length. The batch mean is computed as for the other criteria. No program variable can represent the throughput and be time-averaged. However, it is not an event average since the served customers total number is divided by a time.

The theoretical values can be analytically computed [13]. The performance processes are strictly ergodic on condition that $\lambda < \mu$. In the results tables, the total simulation time is presented, along with the values of θ_i and β_i at the end of each iteration i. The chosen values for the example were $\lambda = 0.5$ and $\mu = 0.8$ (see Tables 1 and 2).

Table 1 shows the counter limit value influence. This value determines the first iteration batch size. As expected, large values result in very good confidence intervals but also in too long simulation times. Small values result in more iterations (3 instead of 2 in this example), because T is not well estimated at the first iteration. This is not a CPU-time waste since second iteration batches are most often re-used or the third iteration (whereas first iteration batches are most often disposed of).

This table allows us to somewhat quantify the "large" and "small" adjectives: good values are in the order of 500 to 1000.

TABLE 1
Results as a function of counter limit value: $\lambda = 0.5$, $\mu = 0.8$, BatchNumber = 16, $\varepsilon = 10\%$, $1 - \eta = 95\%$ (FORTRAN implementation).

Counter	*Iterations*	T	θ_1	β_1	θ_2	β_2	θ_3
100000	1	3192039	0.059				
10000	1	319706	0.527				
3600	1	114353	0.815				
2000	2	187012	2.883	3	0.536		
1000	3	127558	1.893	2	1.216	2	0.867
500	3	231499	4.971	5	2.106	3	0.573
100	3	204959	26.482	27	1.138	2	0.532
10	3	298354	138.411	139	2.686	3	0.343

Table 2 shows the desired relative confidence interval value influence. Of course, the smaller ε, the larger T: this is the estimator property used in the method for the next iteration size estimation. Let us note that $\varepsilon = 5\%$ is the only case that required a fourth iteration.

TABLE 2
Results as a function of ε : $\lambda = 0.5$, $\mu = 0.8$, Counter = 1000, BatchNumber = 16, $1 - \eta = 95\%$ (FORTRAN implementation).

ε	*Iterations*	T	θ_1	β_1	θ_2	β_2	θ_3	β_3	θ_4
0.25	1	31899	0.303						
0.10	3	127558	1.893	2	1.216	2	0.807		
0.05	4	1020462	7.571	8	1.427	2	1.632	2	0.666
0.03	3	2104703	21.030	22	2.464	3	0.467		
0.02	3	4592080	47.318	48	2.201	3	0.680		
0.01	3	18176976	189.272	190	2.369	3	0.483		

The algorithm was also tested for different service rates in the model. Table 3 shows that the method was able to find a satisfactory simulation length in each case.

If the system is heavily loaded (load ρ near to 1), the steady-stage establishment is quite long: a large simulation length is necessary. The algorithm found large enough simulation lengths in three iterations.

TABLE 3
Results as a function of the model (load ρ): $\lambda = 0.5$, Counter = 1000, Batches Number = 16, $\varepsilon = 10\%$, $1 - \eta = 95\%$ (FORTRAN implementation).

μ	(ρ)	*Iterations*	T	θ_1	β_1	θ_2	β_2	θ_3
0.52	(0.96)	3	20651762	78.340	79	7.516	8	0.481
0.55	(0.91)	3	2314551	35.535	36	4.036	5	0.739
0.60	(0.83)	3	1361343	13.748	14	2.676	3	0.695
0.70	(0.71)	2	384161	11.981	12	0.577		
0.80	(0.63)	3	127558	1.893	2	1.216	2	0.867
1.00	(0.50)	3	193115	2.602	3	1.465	2	0.828
2.00	(0.25)	2	62677	1.931	2	0.599		
5.00	(0.10)	2	157955	4.256	5	0.587		
50.0	(0.01)	2	913143	28.294	29	0.426		

If the load is low (ρ near to 0), the required simulation length is larger than with an average load (ρ is near to 0.5) for the following reason: very few customers need to wait in the queue, so the simulation length is increased in order to reach the confidence interval on the mean waiting time estimation. Nonetheless, only 2 iterations were necessary to find the correct simulation lengths.

4.2. MODSIM

A MODSIM [14] implementation was developed for the integration in MIMESIS. This implementation needs to be independent from the model, since the performance criteria choice is made *after* the model development.

The basic idea is to create two *monitor* types (a counter monitor and a mean-computer monitor) and to attach them to the relevant program variables. A monitor is a special object able, among other things, to execute functions at each left or right access to the variable it is attached to. In MODISM, the left-accesses count, the time average and the arithmetic mean are available from pre-defined monitor objects. For instance, the time average $Z_T = \frac{1}{t_p} \sum_{i=1}^{p} X(t_i) \cdot (t_{i+1} - t_i)$, mentioned in Section 2.1, is the MODISM definition of the TimedStatObj monitor's Mean field. The two types defined in our implementation inherit from the MODSIM objects. These monitors are further specialized so as to make a difference between integers and reals, and also between event averages and time averages.

All model variables that can be chosen as performance criteria are declared as *monitored* by a counter monitor and a mean-computer monitor. When the main program is generated, the model description is only linked without any update due to user's choices. In the main program, the performance criteria monitors are given a reference to an object named AutoLength which is in charge of controlling the simulation (*i.e.* implementing the algorithm) and synchronizing the monitors. AutoLength is given the number of performance criteria. This referencing system does not demand to re-compile the model description, thus allowing the independence.

Counter monitors signal to AutoLength that their limit value is reached. When AutoLength has received all expected signals, current simulation time is chosen as the first batch size. Mean-computer monitors are then signalled to write the means (with respect to time or to events accordingly) in an AutoLength table. Mean-computer monitors are synchronized with a trigger mechanism, activated by AutoLength when a batch is finished. After having written the batch mean in the table, mean-computer monitors reset their statistical gatherings and wait for the next batch end signal.

Monitors of variables that are not chosen as performance criteria are deactivated so as to spare CPU time (they would uselessly execute left and right accesses functions otherwise).

The batches number increase was tested with this implementation. Results are in Tables 4 to 6. The simulated model consisted in two independent $M/M/1$ queues and only four time-average performance criteria were considered, namely the mean customer number and the occupation rate of each queue.

The variable parameter in Table 4 is the counter limit value. This table shows that only two iterations were necessary for counter limit values greater than 100. A very small counter limit value of 10 made the algorithm go for a third iteration because θ_1 could not be correctly estimated. However, θ_3 is much closer to 1 than in any other case and simulation length T is the smallest. In this example, the estimation of T at T_2 (simulation time at the end of the second iteration) with a small counter limit value proved to be better than the estimation of T at T_1 (simulation time at the end of the first iteration) with a large counter limit value.

A batches number increase was observed for a counter limit value of 50. At the end of iteration #2, seven supplementary batches were simulated, thus sparing a substantial (16 – 7)/16 = 56% of the third iteration work (this

TABLE 4

Results as a function of counter limit value: $\lambda = 0.5$, $\mu = 0.8$, Default Batches Number = 16, $\varepsilon = 10\%$, $1 - \eta = 95\%$, Max$\beta = 100$ (MODSIM implementation).

Counter	*Iterations*	T	θ_1	β_1	θ_2	β_2	θ_3
1000	2	108149	3.88	4	0.59		
750	2	122770	5.97	6	0.47		
500	2	111472	7.97	8	0.42		
200	2	100497	16.44	17	0.84		
100	2	93045	30.05	31	0.83		
50	2 (+ 7B)	92353	47.38	48	1.74	2	
10	3	68954	60.47	61	2.95	3	0.959

fraction is approximate because of the start-up that had been dropped in the first iteration).

The θ decrease is presented in Table 5. Value of T_2 was 64,647. Without the batches number increase mechanism, final simulation length $T = T_3$ would have been 129,295. Thanks to the mechanism, T is only 92,353 (an effective sparing of 57.1% in the third iteration, or 28.6% for the whole simulation).

TABLE 5

θ evolution in a batches number increase: $\lambda = 0.5$, $\mu = 0.8$, Counter = 50, Default Batches Number = 16, $\varepsilon = 10\%$, $1 - \eta = 95\%$ (MODSIM implementation).

Number of batches	θ
17	1.538
18	1.369
19	1.265
20	1.138
21	1.086
22	1.011
23	0.926

Table 6 shows the results for a smaller required confidence interval, for different counter limit values. The batches number increase occurred more

often, with only one supplementary batch in 3 cases (1000, 500 and 10). The mechanism is the most profitable when the batches number is increased only by one. This table also illustrates the maximum β value control mechanism, for counter limit values of 50 and 100.

TABLE 6

Results as a function of counter limit value: $\lambda = 0.5$, $\mu = 0.8$, Default Batches Number = 16, $\varepsilon = 5\%$, $1 - \eta = 95\%$, Maxβ = 100 (MODSIM implementation).

Counter	*Iterations*	T	θ_1	β_1	θ_2	β_2	θ_3	β_3
1000	2 (+ 1B)	459633	15.52	16	1.08	2		
750	2	511543	23.87	24	0.41			
500	2 (+ 1B)	487690	31.86	32	1.10	2		
200	2 (+ 4B)	466196	65.75	66	1.20	2		
100	2 (+ 2B)	330019	120.21	100	1.15	2		
50	3 (+ 6B)	545545	189.54	100	2.16	3	1.69	2
10	2 (+ 1B)	438367	10.32	11	1.10	2		

4.3. Integration in MIMESIS

This method has been integrated in MIMESIS so as to prevent users from finding themselves a fitting simulation length. In this implementation, default values are:

- 95% confidence level;
- 10% relative confidence interval threshold;
- 16 batches;
- 200 variations to determine the first iteration batch length.

In the following, the activation and the operation of the algorithm within the tool are emphasized. In the first example, a very simple software is modelled (Fig. 1).

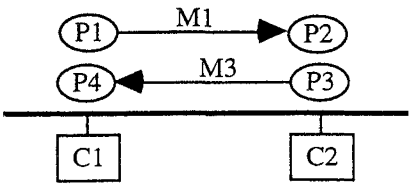

Figure 1. – Software and hardware model (MIMESIS).

It consists of two processes P1 and P2. P1 sends messages M1 to P2 in the following loop:

```
loop
   WAIT exp(X1) seconds
   EXECUTE exp(X1) seconds
   SEND M1 TO P2
end loop
```

where (X1=1) or (X1=2) and "exp" designates an exponential distribution. "WAIT", "EXECUTE" and "SEND" are MIMESIS instructions. The WAIT instruction causes a process to be idle during the specified duration. The EXECUTE instruction blocks the calling process until the specified time has been spent on a processing unit, with possible competition.

Upon receiving a message M1, P2 performs some work but does not send any message back to P1.

The hardware model, also very simple, consists of two computers C1 and C2 connected to a network. When the user selects the network so as to probe it, two simulation results are selected as performance criteria: the network activity rate and the mean messages number in the network.

P1 is placed on C1 and P2 on C2. Therefore, the message sent by P1 to P2 causes some traffic on the network.

If (X1=1), a message is sent approximately every two seconds. If (X1=2), a message is sent approximately every four seconds. It takes 1.3 s for a message to be processed by the network, but only one message at a time can be served.

With this information, a user familiar with simulation theory might estimate a correct simulation length in the two cases. In MIMESIS, the user only has to specify a maximum duration. Simulation is stopped either by the algorithm or if the maximum simulation length is reached. Other monitors compute the confidence intervals in case the maximum is reached, so that they can be displayed as they are at this point.

Let us suppose that the user, having no idea of what simulation length would be suitable, enters 1,000,000,000 as a maximum duration and launches the simulation. MIMESIS then automatically activates the relevant monitors.

When (X1=1), and with this very large TMAX value, the simulation was stopped by the algorithm at $T = 3911.17$ and the relative confidence intervals are (R1 = 3.67%) and (R2 = 5.98%). The algorithm stopped at the end of its first iteration. We have restarted the simulation of the same model with other maximum lengths so as to check this result (see Table 7).

TABLE 7
Relative confidence intervals for X1=1 (16 batches).

TMAX	R1	R2
100	44.7%	51.9%
500	14.8%	21.2%
1000	9.7%	13.0%
2000	6.8%	10.8%
2500	4.6%	7.6%

With these TMAX values smaller than 3911, the simulation was stopped by TMAX and the confidence intervals are computed with 16 batches. We can observe that a simulation length between 2000 and 2500 would be optimal (R1 and R2 would be both smaller than 10%). The length 3911 is larger than the optimum because of the first batch size estimation: it was decided at (T = 244.4 = 3911/16).

Nonetheless, the results were obtained very quickly from the user point of view.

When (X1 = 2) and TMAX is still very large, the simulation is stopped by the algorithm at T = 6796.19 and the relative confidence intervals are (R1 = 5.59%) and (R2 = 6.01%). Here again, the algorithm stopped at the end of its first iteration. We did the same kind of checking:

TABLE 8
Relative confidence intervals for X1=2 (16 batches).

TMAX	R1	R2
500	32.8%	34.9%
1000	21.5%	21.8%
2000	11.8%	12.2%
3000	9.8%	10.0%
4000	7.3%	7.3%
6000	6.0%	6.4%

The results in Table 8 show that the optimal simulation length would have been 3000.

Let us check that the algorithm is executing as expected. When (X1 = 2), a message is emitted every four seconds in average, *i.e.* 0.25 messages/s are processed by the network. The program variables used for estimating the activity rate and the messages number are modified twice for each message (arrival and departure). As a consequence, these two variables are each modified approximately 0.5 times a second. 200 left-accesses are thus made in approximately 400 seconds, which determines the first batch size. Finally, 16 batches of 400 seconds result in a 6400-second simulation (the actual simulation length turned out to be 6796 seconds, *i.e.* a first batch size of 424.8 s).

In the two previous examples, the simulation was stopped at the end of the first simulation. In order to observe more iterations in the algorithm, we added another pair of processes (P3 and P4) in the software model. P3 sends messages M3 to P4 in the following loop:

```
loop
   WAIT exp(X3) seconds
   EXECUTE exp(X3) seconds
   SEND M3 TO P4
end loop
```

The size of message M3 is ten times that of message M1. The values chosen for X1 and X3 are respectively 2 and 25. In average, P1 sends a (short) message to P2 every 4 seconds whereas P3 sends a (long) message to P4 every 50 seconds. The network model is still assumed to process only one message at a time. P1 (sender) and P4 (receiver) are placed on computer C1. P2 (receiver) and P3 (sender) are placed on computer C2.

The following observations were made during the algorithm execution:

- the simulation was stopped at T = 53,104 by the algorithm;
- two iterations were completed;
- the first iteration batch size was 465.8;
- the second iteration batch size was 2,794.9;
- at the end of the first iteration, two more batches have been simulated (for batch re-use purpose, see 3.1.1);
- at the end of the second iteration, three more batches have been simulated (batches number increase, see 3.1.2).

Tables 9 and 10 illustrate a few results for this execution. Table 9 shows the algorithm iterations. Table 10 shows the evolution of θ when the number of batches increases.

TABLE 9
Algorithm execution results: X1=2, X3=25 (MIMESIS implementation).

Iterations	T	θ_1	β_1	θ_2	β_2	θ_3
2 (+ 3B)	53104	5.65	6	1.07	2	

TABLE 10
θ evolution in the batches number increase (MIMESIS implementation).

Number of batches	θ
16	1.068
17	1.096
18	1.005
19	0.929

Let us compare the results of the algorithm to the effective confidence intervals found for predefined simulation lengths.

Table 11 confirms that the simulation was optimally stopped by the algorithm. The 10% confidence interval for the first result (activity rate) is reached before 10,000, but the system has to be simulated a bit more than 50,000 so that the 10% threshold is reached for the second criterion

TABLE 11
Relative confidence intervals for X1=2 and X3=25 (16 batches).

TMAX	R1	R2
5000	13.1%	24.4%
10000	9.6%	21.4%
20000	7.5%	17.5%
30000	6.6%	13.2%
40000	5.4%	11.0%
50000	5.5%	10.6%

(mean messages number). The dynamic batches number increase proved to be efficient, since the second iteration ended at T = 44,719. With three more batches, the simulation ended at T = 53,104. Without this, there would have been a third iteration until T = 89,438.

5. CONCLUSION

An automatic run-time choice algorithm for simulation length has been designed and implemented. The processes are supposed to be second order asymptotically stationary. The method consists in running a single simulation until some predefined relative confidence intervals on the performance criteria are reached. At run-time, the simulation is cut into equi-sized batches. The batch mean method is used for estimating confidence intervals which are in turn used for iteratively determining a sufficient simulation length.

In all our tests, the algorithm was able to stop the simulation when necessary, reaching the required confidence intervals without wasting too much CPU time. However, it might diverge should the user ask really too small confidence intervals and/or too high confidence levels.

Some practical features have been added to the theoretical idea so as to make the implemented method more efficient. The simulation length multiplying coefficient θ is rounded up in order to re-use batches and also to avoid an iteration in some cases. This coefficient is limited to a maximum value so that the first iteration is not too long. Also, the iteration number itself is limited to avoid divergence. In the same vein, the batches number is dynamically modified when the coefficient is small.

The method rests on the definition of the states of a system and the automatic observation of the changes in the states, which are called *events*. Therefore, the method can be applied to any system that can be described in terms of states, which is the case of a large range of complex systems. It is a useful feature in a simulation tool such as MIMESIS. The user no longer has to worry about finding a satisfactory simulation length: performance criteria are nearly always obtained with small enough confidence intervals.

Further work should be done in two directions. Firstly, for some complex systems, especially those involving rare events, simulation is not a performance evaluation method to resort to. Our algorithm does not solve this problem: other techniques must be used to deal with rare events. Secondly, the method is valid only for steady state simulations. A study of the automatic detection of non steady state period is in progress. Later on, it will be possible to analyze simulations during a non steady state period.

REFERÈNCES

1. A. Law and W. D. Kelton, Simulation Modeling & Analysis, Mc Graw Hill, 1991.
2. M. Badel, Quelques problèmes liés à la simulation de modèles de systèmes informatiques, *Thèse de Docteur-Ingénieur*, Université de Paris VI, France, 1975.
3. A. Law and W. D. Kelton, Confidence Intervals for Steady-State Simulations: I. A. Survey of Fixed Sample Size Procedure, *Operations Research*, 1984, *32*, n° 6, pp. 1221-1239.
4. H. Damerdji, Strong Consistency of the Variance Estimator in Steady-State Simulation Output Analysis, *Mathematics of Operations Research*, 1994, *19*, n° 2, pp. 494-512.
5. C. Chien, Batch Size Selection for the Batch Means Method, *Proceedings of the 1994 Winter Simulation Conference*, 1994, pp. 345-352.
6. M. K. Nakayama, Two-Stage Stopping Procedures Based on Standardized Time Series, *Management Science*, 1994, *40*, n° 9, pp. 1189-1206.
7. W. Y. Thang and G. Damm, An Environment for Performance Evaluation: MIMESIS (Machine-Implemented Modelling and Exploratory Simulation from Initial Specification), *Proceedings of the Summer Computer Simulation Conference 1996*, Portland, Oregon, USA, July 1996.
8. G. Damm, W. Y. Thang, A. L. Beylot and M. Becker, Generic Components for a Library of Parallel Architectures Models in MIMESIS, *Proceedings of the Summer Computer Simulation Conference 1995*, Ottawa, Ontario, Canada, July 24-26, 1995, pp. 773-778.
9. M. Becker, Validité des Simulations de Files d'Attente, *Thèse de Doctorat d'État*, Université Paris VI, France, 1976.
10. P. Le Gall, L'ergodisme et la convergence des méthodes aléatoires, *Annales des Télécommunications*, 1967, *22*, n° 7-8, pp. 221-228.
11. M. Becker and P. Douillet, Hierarchical Simulation for Rare Events, *IASTED Modelling and Simulation*, May 10-12, 1993.
12. M. Becker and P. Becker, Optimal Simulation Lengths for Various Algorithms Computing the Mean, *IMACS*, 1978, *20*, n° 1, pp. 44-52.
13. L. Kleinrock, Queueing Systems, Volume 1: Theory, *John Wiley & Sons*, 1975.
14. CACI Products Company, MODSIM II, The Language for Object-Oriented Programming: Reference Manual, La Jolla, CA, USA, 1993.