Monotonicity and Efficient Computation of Bounds with Time Parallel Simulation

Jean-Michel Fourneau and Franck Quessette

PRiSM, Université de Versailles-Saint-Quentin, CNRS UMR 8144, France

Abstract. We adapt Nicol's approach for the time parallel simulation with fix-up computations. We use the concept of monotonicity of a model related to the initial state of the simulation to derive bounds of the sample-paths. We prove several algorithms with fix-up computations which minimises the number of runs before we get a consistent sample-path. We obtain proved upper or lower bounds of the sample-path of the simulation and bounds of some estimates as well.

1 Introduction

A parallel discrete event simulation is the construction of the slices of the sample-path on a set of processors. These slices can be obtained by a state decomposition or a decomposition of the time interval. The most common approach is the space decomposition, i.e. a grouping of state variables into subsets which are affected to parallel processors. These processors exchange messages about the scheduling of the future events to avoid temporal faults or to correct them. Unfortunately the spatial decomposition approach has a limited parallelism and has in general an important overhead due to this synchronisation of future events.

Time Parallel Simulation (TPS in the following) follows a different approach considering a decomposition of the time axis and performing the simulations on time intervals in parallel (see [11] chap. 6 and references therein). Afterwards the simulation results are combined to build the overall sample-path. TPS has a potential to massive parallelism [18] as the number of logical processes is only limited by the number of times intervals which is a direct consequence of the time granularity and the simulation length. But the final and initial states of adjacent time intervals do not necessarily coincide at interval boundaries, possibly resulting in incorrect state changes. The efficiency of TPS depends on our ability to guess the state of the system at the beginning of the simulation intervals or to efficiently correct the guessed states to finally obtain a consistent sample-path after a small number of trials. Several properties had already been studied: regeneration [17], efficient forecasting of some points of the sample-path [12], parallel prefix computation [13], a guessed initial state followed up by some fix-up computations when the first state has a weak influence on the sample-path [18]. Another approach consists in relaxing these assumptions to obtain an approximation of the results [3,16,21].

We have previously introduced two properties of the model both related to monotonicity (inseq-monotonicity in [7] and hv-monotonicity in [6,8]), which can

N. Thomas (Ed.): EPEW 2011, LNCS 6977, pp. 57–71, 2011.

be used to improve the efficiency of TPS. We have developed in these publications the basic theory needed to compare the sample-paths of simulations and to improve the speed of the TPS. In our approach, the simulation model is seen as a deterministic black box with an initial state and one input sequence which computes one output sequence. All the randomness of the model is in the input sequence. We define some orderings on the sequences and the states. A model is monotone when it preserves the ordering. If this property holds, we have proposed new approaches to improve the efficiency of TPS. We obtain an upper or a lower bound of the output sequence. Then we compute a reward on the output sequence. The ordering on the sequences are defined in such a way that typical rewards such as the moments or the tail of the distributions are consistently ordered when the sequences are ordered. In performance evaluation or reliability studies, we must prove in general that the systems we analyze satisfy some quality of service requirements. Thus it is sufficient to prove that the upper bound (or lower bound, depending on the measure of interest) of the output sequence satisfies the constraint. We have already proved in [10] that many queueing network models are hv-monotone.

Here we develop the ideas presented in [6,8] giving much more details, some new algorithms for speculative computations and some numerical results. We show how we can build a sample-path for a bound of the exact simulation using some simulated parts and ordering relations without some of the fix-up phases proposed in [18]. The inseq-monotonicity and hv-monotonicity concepts are both related to the stochastic monotonicity used to compare random variables and stochastic processes [9,20] and the event monotonicity which is the main assumption for the monotone Coupling From The Past algorithm for perfect simulation [19]. Hv-monotonicity is related to the non crossing property used in [1]. For these various notions of monotonicity of stochastic processes, one can also refer to [15] for a comparison.

The rest of the paper is as follows. In section 2, we give a detailed presentation of Nicol's approach of TPS with fix-up computation phases. Then in section 3, we define the comparison of sequences, the inseq-monotonicity and the hv-monotonicity and their relations with stochastic ordering. The approach is illustrated with examples to clarify the concepts. In Section 4 we first present the initial algorithm proposed in [8] and we extend it in several directions proving the convergence of some algorithms in any number of runs smaller than the number of processors. Section 5 is devoted to a simple model of a Web server. We prove that the system is hv-monotone and we provide some numerical results for the speed up and the efficiency of the approach.

2 Time Parallel Simulation with Fix-Up Phases

Let us now detail the classical fixed up approach [18] before we introduce the first modification presented in [8] and the new extensions which are the main results of this paper. For the sake of completeness we begin by a short introduction to the fixed up approach proposed by Nicol. In a first step, the interval $[0, T)$ is divided

into K equal intervals $[t_i, t_{i+1})$. K is the number of processors. Let $X(t)$ be the state at time t during the exact simulation obtained in a centralised manner. The aim is to build $X(t)$ for t in $[0, T)$ through an iterative distributed algorithm. For the sake of simplicity, we assume that for all i between 1 and K, logical process LP_i simulates the i-th time interval. We denote as $Y_j^i(t)$ the state of the system at time t obtained during the j-th iteration of logical process LP_i. The initial state of the simulation is known and is used to initialise LP_1. During the first run, the other initial states (say $Y_1^i(t_i)$) for the initial state of simulation at LP_i are chosen at random or with some heuristics. Simulations are ran in parallel. The ending states of each simulation (i.e. $Y_1^i(t_{i+1})$) are computed at the end of the simulation of the time intervals. They are compared to the initial state we have previously used (i.e. $Y_1^{i+1}(t_{i+1})$). The equality of both states is a necessary condition for consistency of the path. Otherwise one must run a new set of parallel simulations for the inconsistent parts using $Y_1^i(t_{i+1})$ as starting point (i.e. $Y_2^{i+1}(t_{i+1}) \leftarrow Y_1^i(t_{i+1})$) of the next run on logical process LP_{i+1}. These new runs are performed with the same sequence of random inputs. The simulations are ran until all the parts are consistent. The number of rounds before the whole simulation is consistent is smaller than the number of LP. It is clear that at the end of the first run, the simulation performed at LP_1 is consistent. Similarly by induction on i, at the end of round i, LP_i is consistent. It is the worst case we may obtain, and in that case the time to perform the simulation in parallel is equivalent to the time in sequential.

Note that performing the simulation with the same input sequence may speed up the simulation due to coupling. Suppose that we have stored the previous sample-paths computed by LP_i. Suppose now that for some t, we find that the new point $Y_k^i(t)$ is equal to a formerly computed point $Y_m^i(t)$. As the input sequence is the same for both runs, both sample-paths have now merged:

$$Y_m^i(u) = Y_k^i(u) \quad \forall u \geq t.$$

Thus it is not necessary to build the new sample-path. Such a phenomenon is defined as the coupling of sample-paths. Note that it is not proved that the sample-paths couple and this is not necessary for the proof of the TPS that it happens. Clearly, coupling allows to speed up the computations performed by some LP and also reduces the number of rounds before the whole simulation becomes coherent. Indeed a coupling means that the state reached at the end of a time interval is not that dependent of the initial state of the simulation.

Consider an example of TPS with one fix-up step in Figure 1. During the first step, five parts of the simulation are ran in parallel. Part 1 and Part 2 are consistent as the initial point of LP2 is the final point of LP1 due to a correct guess. Other parts are not consistent and the simulations are ran again with a correct initialisation (A instead of B for the second run of part 4 for instance) and the same random inputs to obtain new sample-paths. The former sample-paths are compared to the new ones and as soon that the simulations couple, they are stopped as simulations after coupling will follow the same paths. If the simulations do not couple, we only keep the new sample-path. After each

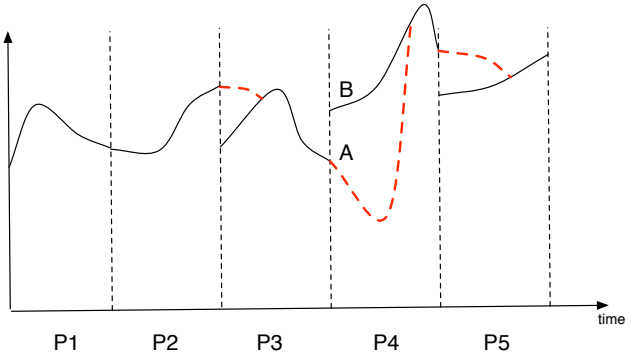

Fig. 1. TPS, coupling and fixing the sample-path

step, the number of consistent sample-paths will increase. The efficiency of the approach is related to the number of consistent parts at each step. In Fig. 1 the new runs are represented in dotted red lines and we see that all the simulations couple during the first fix-up step.

We assume that the logical processes LP_i $(i = 1..K)$ perform simulations and exchange information with a Master process which must check the consistency of the partial sample-paths. Thus the simulation effort is distributed among $K+1$ processes. The whole approach is summarised in the following algorithm for LP_i $(K \geq i > 1)$ and the Master process. The first logical process in charge of the simulation of the first time interval slightly differs for this general pseudo-code: $Y_1^1(t_1)$ is equal to $X(t_1)$ the initial state of the whole simulation.

Algorithm LP_i

1. $k \leftarrow 0$. Read in shared memory input sequence $I(t)$ for all t in $[t_i, t_{i+1}[$.
2. $Y_1^i(t_i) \leftarrow Random$.
3. Loop
   (a) k++.
   (b) Perform run k of Simulation with Coupling on the time interval $[t_i, t_{i+1}[$ to build $Y_k^i(t)$ using input sequence $I(t)$.
   (c) Send to the Master: the state $Y_k^i(t_{i+1})$.
   (d) Receive from the Master: Consistent(i) and a new initial state U.
   (e) If not Consistent(i) $Y_{k+1}^i(t_i) \leftarrow U$.
4. Until Consistent(i).
5. $\forall t \in [t_i, t_{i+1}[$, $X(t) \leftarrow Y_k^i(t)$ and write $X(t)$ in shared memory.

Algorithm Master

1. For all i Consistent(i) $\leftarrow$False.
2. Consistent(0) $\leftarrow$ True; LastConsistent $\leftarrow 0$; $k \leftarrow 0$.
3. Loop
   (a) k++.

(b) For all i, if not Consistent(i) Receive from LP_i the state $Y_k^i(t_{i+1})$.
(c) $Y_1^0(t_1) \leftarrow Y_1^1(t_1)$.
(d) $i \leftarrow$ LastConsistent.
(e) Loop
  i. i++.
  ii. If $(Y_k^i(t_i) = Y_k^{i-1}(t_i))$ and Consistent(i-1) then Consistent(i) $\leftarrow$ True.
(f) Until (not Consistent(i)) or (i>K).
(g) LastConsistent $\leftarrow i - 1$.
(h) For all i send to LP_i Consistent(i) and the state $Y_k^{i-1}(t_i)$.
4. Until LastConsistent = K.

---

3 Monotone Systems, Comparison of States and Comparison of Sequences

A simulation model is defined as an operator on a sequence of parameters (typically the initial state) and an input sequence (typically inter arrival times and service times) and produces an output sequence (typically the state of the system or a reward). Let $\mathcal{M}$ be a simulation model. We denote by $\mathcal{M}(a, I)$ the output of model $\mathcal{M}$ when the parameter sequence is a and the input sequence I. As usual an operator is monotone iff its application on two comparable sequences provides two output sequences which are also comparable. We have used the following point ordering (denotes as $\preceq_p$ in [7], but various orders are possible. This order is interesting because it implies a comparison on the rewards computed on the output sequence.

Definition 1. *Let I_1 and I_2 be two sequences with length n. $I(m)$ is the m-th element of sequence I. $I_1 \preceq_p I_2$ if and only if $I_1(t) \leq I_2(t)$ for all index $t \leq n$.*

Property 1. *Assume that the rewards are computed with function r applied on state $O(t)$ at time t. If the rewards are non negative, then:*

$$O_1 \preceq_p O_2 \implies R(O_1) = \sum_t r(t, O_1(t)) \leq R(0_2) = \sum_t r(t, O_2(t)).$$

Many rewards such as moments and tails of distribution are non negative.

Based on this simple idea, we can define two properties which allows to compare the outputs of a simulation model when the change the input sequence (inseq-montone) or the initial state of the simulation (hv-monotone). In the context of queueing networks for instance, this will describe how evolve the sample-path of the population when we change the arrivals or the initial population in the queue. We consider two arbitrary orderings $\preceq_\alpha$ and $\preceq_\beta$.

Definition 2 (inseq-monotone Model). *Let $\mathcal{M}$ be a simulation model, $\mathcal{M}$ is input sequence monotone (or inseq-monotone in the following) with respect to orderings $\preceq_\alpha$ and $\preceq_\beta$ if and only if for all parameter sequence a and input sequences I and J such that $I \preceq_\alpha J$, then $\mathcal{M}(a, I) \preceq_\beta \mathcal{M}(a, J)$.*

Definition 3 (hv-monotone Model). *Let $\mathcal{M}$ be a simulation model, $\mathcal{M}$ is monotone according to the input state or hidden variable (hv-monotone in the following) with respect to orderings $\preceq_\alpha$ and $\preceq_\beta$ if and only if for all parameter sets a and b such that $a \preceq_\alpha b$, then $\mathcal{M}(a, I) \preceq_\beta \mathcal{M}(b, I)$ for all input sequence I.*

The inseq-monotonicity and the hv-monotonicity are related to the stochastic monotonicity used to compare Markov chains [20].

Definition 4 (Stochastic comparison). *Let X and Y be random variables taking values on a totally ordered finite space $\{1, 2, \ldots, n\}$ with p and q as probability distribution vectors, then X is said to be less than Y in the strong stochastic sense (denoted as $X \preceq_{st} Y$), if and only if $\sum_{j=k}^{n} p_j \leq \sum_{j=k}^{n} q_j$ for $k = 1, 2, \ldots, n$. The stochastic ordering is also used for distribution and we note that $p \preceq_{st} q$.*

Example 1. *One can easily check that:* $(0.1, 0.3, 0.2, 0.4) \preceq_{st} (0., 0.4, 0., 0.6)$.

The relation between inseq-monotonicity and the strong stochastic ordering is described in [7]. We now prove that stochastic monotonicity of Discrete Time Markov Chains (DTMC in the following) implies hv-monotonicty of the simulation model when the state space is fully ordered. Let us first define stochastic monotone Markov chains using the matrix formulation presented in [9]. It is known for a long time [20] that monotonicity and comparability of the one step transition probability matrices of time-homogeneous MCs yield sufficient conditions for their stochastic comparison.

Definition 5 (Monotone DTMC). *Let M_Z be the stochastic matrix associated to Markov chain Z_t on a totally ordered state space $\mathcal{S}$, M_Z is stochastically monotone if and only if for all probability distributions u and v such that $u \preceq_{st} v$, then $uM_Z \preceq_{st} vM_Z$.*

Property 2. *Let M_Z be a stochastic matrix, let $M(i, *)$ be the i-th row of M and may be seen as a probability distribution, M_Z is stochastically monotone iff $M_Z(i, *) \preceq_{st} M_Z(i + 1, *)$.*

Example 2. $M1 = \begin{bmatrix} 0.1 & 0.2 & 0.6 & 0.1 \\ 0.1 & 0.1 & 0.2 & 0.6 \\ 0.0 & 0.1 & 0.3 & 0.6 \\ 0.0 & 0.0 & 0.1 & 0.9 \end{bmatrix}$ *is monotone while* $M2 = \begin{bmatrix} 0.1 & 0.2 & 0.6 & 0.1 \\ 0.2 & 0.1 & 0.1 & 0.6 \\ 0.0 & 0.1 & 0.3 & 0.6 \\ 0.2 & 0.0 & 0.0 & 0.8 \end{bmatrix}$ *is not.*

We may use an event representation of the simulation model. Events are associated with transitions. Let ev be an event in this model and let x be a state, we denote by $ev(x)$ the state obtained by application of event ev on state x. It is more convenient that some events do not have any effect (for instance the end of service event on an empty queue will be a loop). Monotonicity property has already be defined for events and their links with stochastic monotonicity are now understood (see [15]).

Definition 6 (event -monotone). *An event ev is monotone if its probability does not depend on the state and for all states x and y, $x \preceq y$ implies that $ev(x) \preceq ev(y)$.*

It is now possible to show the links between strong stochastic ordering of DTMCs and hv-monotonicity of their simulations.

Property 3. *Let M_Z be a monotone stochastic matrix on a totally ordered state space (denoted by $\leq$ to emphasis the total ordering) and let $\mathcal{M}()$ be the simulation of this chain. a is the initial state and I is a sequence of uniform random values in $[0,1]$. Assume that the output $O = \mathcal{M}(a, I)$ is the state of the chain at each time. Thus, $\mathcal{M}()$ is hv-monotone w.r.t. the total ordering on the states (i.e. $\leq$) and the point ordering $\preceq_p$ on the output sequence.*

Proof: It is proved in [15] that a stochastically monotone DTMC on a totally ordered state space admits a set of event $(e_1, ..., e_m)$ which is monotone. Suppose that $a \geq b$ and consider $O1 = \mathcal{M}(a, I)$ and $O2 = \mathcal{M}(b, I)$. As we use the same $I(t)$, both simulations perform the same event at time t. Let e_k be this event. By construction $O1(t+1) = e_k(O1(t))$ and $O2(t+1) = e_k(O2(t))$ The events are all monotone. Thus is if $O1(t) \leq O2(t)$, we get $O1(t+1) \leq O2(t+1)$. By induction the model is monotone for the point ordering on the output sequence. □

Note that hv-monotonicity or inseq-monotonicity do not imply strong stochastic monotonicity in general. Let us now consider one basic example to illustrate the approach.

Example 3 (DTMC). *Consider again matrix $M1$ defined in Example 2. Assume that the input sequence consists in uniform random values in $[0,1]$ used in an inverse transform method to find the transitions according to matrix $M1$. Assume that this sequence I is equal to $(0.5, 0.05, 0.15, 0.06, 0.25, 0.45)$. Assume that the output of the model is the state of the Markov chain at each step and that the hidden variable is the initial state of the chain. As $M1$ is stochastically monotone, the simulation $\mathcal{M}_1()$ of matrix $M1$ is hv-monotone w.r.t. natural ordering on the states and point ordering on the output sequence. Therefore:*

$$\mathcal{M}_1(1, I) \preceq_p \mathcal{M}_1(2, 1).$$

Indeed, one can easily check that the output of $\mathcal{M}_1(1, I)$ is the following sequence of states $(1, 3, 2, 2, 1, 2, 4)$ while the output of $\mathcal{M}_1(2, I)$ is $(2, 4, 3, 3, 2, 3, 4)$; the verification of the point ordering is readily made. Note also that both simulations have coupled at time 6 on state 4.

Remark 1. *Note that increasing the initial state does not in general results in an upper bounding sample-path as shown in the following example.*

Example 4 (DTMC 2). *Consider now matrix $M2$ defined in Example 2. We make the same assumptions on the input and output sequences as in the previous example. Let $I = (0.15, 0.5, 0.15)$. The sample-path beginning with state 1 is $(1, 2, 4, 1)$ while beginning with state 2 it is $(2, 1, 3, 3)$. Clearly the two vectors cannot be compared with the $\preceq_p$ ordering.*

4 Fast Parallel Computation of Bounds with TPS

We assume in this section that we consider an arbitrary hv-monotone model and we develop the approach introduced in [6,8]. We show how to modify the TPS to converge faster on a bound of the sample-path. We have already proved in [10] that many queueing networks model are hv-monotone. To speed up the computation we change the rules about the consistency of the simulations performed by the logical processes. We now say that two parts are consistent if they are a proved bound of the sample-path that we do not have computed. To illustrate the approach let us suppose that we want to compute faster a lower bound of the sample-path. We modify the TPS as follows: the simulation processes LP_i ($i = 1..N$) are not modified but the Master process performs a slightly different computation. The first assumption is, as before, that all parts except the first one are marked as not consistent while the first one is consistent. Suppose that at the end of round k, the points $Y_k^i(t_{i+1})$ have been computed using $Y_k^i(t_i)$ as starting points. If $Y_k^i(t_{i+1}) = Y_k^{i+1}(t_{i+1})$ and part i is consistent, mark part $i+1$ as consistent. Furthermore if $Y_k^i(t_{i+1}) > Y_k^{i+1}(t_{i+1})$ and part i is consistent, the concatenation of the parts provide a sample-path of a lower bounding sequence. Thus mark part $i+1$ as consistent. Two consecutive parts are consistent if the first one is consistent and if the second one is a proved lower bound. Let us more precisely describe the new version of the Master Process, we only report the inner loop.

- Lower Bounding sample-path -

3.e Loop
  i) i++;
  ii) if ($Y_k^i(t_i) = Y_k^{i-1}(t_i)$) and Consistent(i-1) then Consistent(i) ← True;
  iii) elsif ($Y_k^i(t_i) < Y_k^{i-1}(t_i)$) and Consistent(i-1) then Consistent(i) ← True;
3.f Until (not Consistent(i)) or (i>K);

-

In the original method [18], instruction 3.e.iii) of our approach is not included. This version of the method is proved to provide a lower bound of the sample-path (see [8] for the proof of this first algorithm).

Theorem 1. *Assume that the simulation model is hv-monotone, the new version of the Master for the TPS algorithm makes the logical simulation processes build a point-wise lower bound of the sample-path faster than the original approach. Thus the number of runs is smaller than K.*

For instance, Fig. 1 shows that after step 2, we have coupled and we have obtain an exact solution. But at the end of step 1 we have obtained a proved lower bound if the system is hv-monotone. Thus we can immediately stop the computation and consider the sample-path after the first run. This sample-path is not an exact one but it is a point-wise lower bound of the exact sample-path. Due to

Property 1 the expectations of non negative rewards on this sample-path are lower bounds of the exact expected rewards. Note that the bound is proved while the approximations provided by other methods [3,21] do not provide such an information (see also Remark 1 to show that changing the initial state does not always result in a bound of the sample-path).

Designing an algorithm for computing an upper bound is straightforward. But instead we show how to improve the method and obtain a trade-off between the number of runs and the accuracy. We can again improve the speed of convergence with a clever choice of the input points sent by the Master to the simulation processes LP_i.

In the original approach when process LP_{i-1} is not consistent, then LP_i is not consistent either, even if the condition $(Y_k^i(t_i) = Y_k^{i-1}(t_i))$ holds. However it is useless to compute again a new part of the simple path using $(Y_k^i(t_i))$ as a starting point. Indeed the former run has already computed this particular part and beginning with the same state and using the same sequence of inputs, we will again obtain exactly the same results. Thus the processor is idle and it can be used instead for some speculative computations: it may typically compute a new part for the same sequence of inputs using a speculative initial point. Thus one must assume that the Master process stores several parts of the sample-path using the same sequence of inputs but initialised with several starting points. It checks the consistency of the parts and it asks to create new speculative parts when needed.

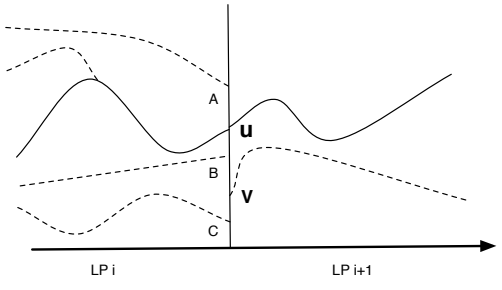

Fig. 2. How to chose a new initial point for a speculative computation

It remains to describe how to chose a new starting point for the next run. The configuration is depicted in Fig. 2. Assume that we want to compute a lower bound and that Min is the smallest state. Consider the results of run k. Assume that $(Y_k^{i+1}(t_{i+1}) = Y_k^i(t_{i+1})) = \mathbf{u}$. The next initial value is chosen at random between Min and $\mathbf{u}$ ($\mathbf{u}$ excluded). Let $\mathbf{v}$ be this random value. We will run a new simulation with the same input sequence $I(t)$ starting form state $\mathbf{v}$. At the end of the run we have at least two speculative parts (one beginning with $\mathbf{u}$ and one beginning with $\mathbf{v}$, and maybe some other ones previously computed). Assume that the state space is totally ordered. Now consider the four cases for

the simulation performed during that run on LP_i. Assume that at the end of that run, part i is found consistent. This simulation may end in:

- $A > \mathbf{u}$. In that case, part $i+1$ beginning with $\mathbf{u}$ is consistent.
- $\mathbf{u}$ because of coupling. Again part $i+1$ beginning with $\mathbf{u}$ is consistent.
- B such that $\mathbf{u} > B \geq \mathbf{v}$. In that case part $i+1$ beginning with $\mathbf{v}$ is consistent.
- $C < \mathbf{v}$. None of the two speculative simulations of part $i+1$ is consistent. One needs another run.

Clearly choosing a new speculative part beginning with a state smaller than $\mathbf{u}$ will help to find a lower bound of the sample-path while a simulation initialised with a state larger than $\mathbf{u}$ will not.

When the state space is only partially ordered, we have one more case where the states are not comparable and one more run is needed. Thus we use a similar set of rules to decrease the number of runs before the whole system has converged to a bound of the sample-path. We present in the next algorithm how such a speculative computation can be organised by the Master process. Again we only present the inner loop computation. The remaining part of the Master algorithm is kept unchanged.

——————————- Speculative Runs when idle ——————————

3.e Loop

i) i++; Let k be the last run index

ii) if Consistent(i-1) then
- Let l be the run index for which part $i-1$ is consistent.
- If $\exists$ m such that $Y_m^i(t_i) = Y_l^{i-1}(t_i)$ then Consistent(i) $\leftarrow$ True and the m-th run is used for consistency of part i.
- Elsif $\exists$ m such that $Y_m^i(t_i) < Y_l^{i-1}(t_i)$ then Consistent(i) $\leftarrow$ True and the m-th run is used for consistency of part i.
- Else $Y_{k+1}^i(t_i) \leftarrow Y_l^{i-1}(t_i)$.

iii) Else
- If $\exists$ m such that $Y_m^i(t_i) = Y_k^{i-1}(t_i)$ then $Y_{k+1}^i(t_i) \leftarrow$ Random state between Min and $Y_k^{i-1}(t_i)$.
- Else $Y_{k+1}^i(t_i) \leftarrow Y_k^{i-1}(t_i)$.

3.f Until (i>K);

3.g LastConsistent $\leftarrow i-1$.

3.h For all i send to LP_i Consistent(i) and the state $Y_{k+1}^i(t_i)$.

Finally we also prove an algorithm to compute a lower bound path with a convergence in any fixed number of round (say R) under the same assumptions on state Min. During the first $R-1$ runs, the Master Process acts as before but if the simulation has not converged to a consistent sample-path (i.e. a bound of the exact one) at the end of run $R-1$, the Master sends to the simulation processes LP_i state Min as a starting point for the next run.

Convergence in R runs

3.e Loop
  i) i++; Let k be the last run index
  ii) if Consistent(i-1) then
    – Let l be the run index for which part $i-1$ is consistent.
    – If $\exists\ m$ such that $Y_m^i(t_i) = Y_l^{i-1}(t_i)$ then Consistent(i) $\leftarrow$ True and the m-th run is used for consistency of part i.
    – Elsif $\exists\ m$ such that $Y_m^i(t_i) < Y_l^{i-1}(t_i)$ then Consistent(i) $\leftarrow$ True and the m-th run is used for consistency of part i.
    – Else $Y_{k+1}^i(t_i) \leftarrow Y_l^{i-1}(t_i)$.
  iii) Else
    – if $k = R-1$ then $Y_{k+1}^i(t_i) \leftarrow Min$.
    – Elsif $\exists\ m$ such that $Y_m^i(t_i) = Y_k^{i-1}(t_i)$ then $Y_{k+1}^i(t_i) \leftarrow$ Random state between Min and $Y_k^{i-1}(t_i)$.
    – Else $Y_{k+1}^i(t_i) \leftarrow Y_k^{i-1}(t_i)$.
3.f Until (i>K);
3.g LastConsistent $\leftarrow i-1$.
3.h For all i send to LP_i Consistent(i) and the state $Y_{k+1}^i(t_i)$.

Property 4. *The Master algorithm with inner loop "Convergence in R runs" needs less than R runs to build a consistent lower bound of the sample-path.*

Proof: due to computation at the end of run $R-1$, the logical processes are initialised with the minimal state. Thus the consistency condition holds after one more run and all the parts are consistent at the end of run R. □

We now describe briefly more improvements that we cannot detail for the sake of conciseness.

1. When a part is consistent, the process is idle and it can be used after reading in shared memory a new input sequence to build other speculative paths for a remaining part of the simulation which is not consistent at that time.
2. Remember that the initial approach provides an exact solution. Thus we can obtain a time versus accuracy trade-off by using the traditional approach during the first $R1$ runs and our approach with bounds during the next $R2$ runs and completing the simulation in one last run where the initial states for non consistent parts are initialised with minimal or maximal states. We obtain a proved convergence in $R1 + R2 + 1$ steps and the parts computed and found consistent at the end of run $R1$ are exact.

5 Modelling a Web Server

Some numerical results have already been published for monotone queueing networks such as Jackson networks [10]. To illustrate the approach with a new example, we analyse now a simple model of a web server. The system consists in a scheduler and a set of stations (see Figure 3). All these components are modelled by FIFO queues with infinite capacity. Web servers have receive

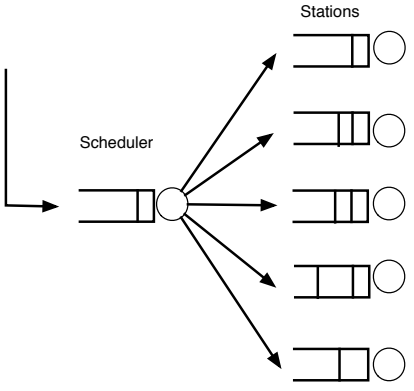

Fig. 3. Web Server as a queueing network

considerable attention (see for instance [14]) to optimise the impact of the scheduler on the performance of the system taking into account some partial information on the load of the queues and the work requested by the customers.

We first describe the model, we prove that the system is hv-monotone and we report some numerical results for the speed-up. Here we consider a discrete time model where the slot time is equal to the duration of the service time at the scheduler. The page arrivals follow a Poisson distribution with rate λ. Pages contain an HTTP GET for a file. Thus the important features of a page is the sum of the sizes of the objects included in the page. The sizes are estimated by the scheduler. The service times in the scheduler are constant and all equal to 1. Let T_0, T_1,..,T_N be $N+1$ increasing number where T_N (resp. T_0) is the biggest (resp. smallest) size estimation. We assume that $T_0 > 0$. The pages are sent to the stations according to the estimation of their size. Station $S1$ serves pages whose size estimation is in the interval $[T_0, T_1)$. Similarly, station Si receives the pages with size in the interval $[T_{i-1}, T_i)$. The sizes of the page are independent and identically distributed. They follow a simple distribution with a large variance: $\lceil \frac{1}{u} \rceil$ with $0 < u \leq 1$. The services are supposed to be geometric with rate μ_i at queue Si. Due to these assumptions, the state of the system is the number of customers at each queue (i.e. a $N+1$ tuple of integers). Let u be a state; $u(0)$ will denote the size of the scheduler queue while $u(i)$, $i = 1..N$ will be the size of the queue associated with server Si. The output of the model is the state of the system at time t. We use the following ordering on the initial states.

Definition 7. *We define the $\preceq_\beta$ ordering on states as the product of the natural orderings on each component of the tuple: $u \preceq_\beta v$ iff $u(i) \leq v(i)$ for all i.*

Property 5. *This model of a web server is hv-monotone with $\preceq_\beta$ ordering on the parameter sequence and $\preceq_p$ on the output sequence.*

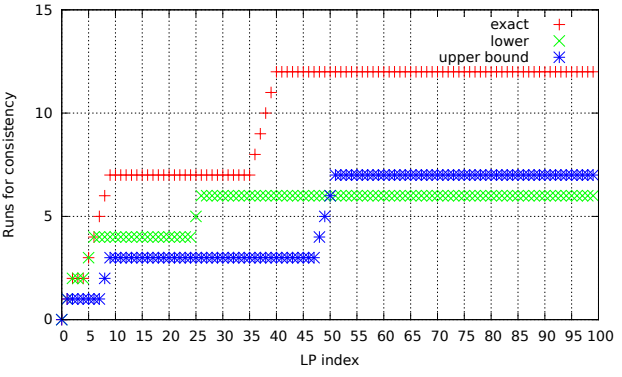

Fig. 4. Number of runs before global consistency, $\mu = 0.56$, $\lambda = 0.8$, uniform initial state

Proof: Let u_0 and v_0 be two states such that $u_0 \preceq_\beta v_0$ and let us denote by u_t and v_t the states at time t. Let us first consider $u_0(0)$ and $v_0(0)$. As $u_0(0) \leq v_0(0)$ and as the service of the scheduler is constant and the arrivals are the same because we use the same input sequence for both simulations, we have $u_t(0) \leq v_t(0)$ for all t. And in the simulation beginning with u_0, all the customers leave the scheduler at the same time they leave in the simulation beginning with v_0. Furthermore the routing is the same in both simulations. Note however that we must use the same amount of random values in the input sequence in both simulations. Thus we assume that the input sequence is slotted and all values produced at time t are only available for transitions at time t. If they are not used, they will be deleted and a new set of random values will be provided for time slot $t + 1$.

Now consider $u_0(i)$ and $v_0(i)$ for an arbitrary i. Due to the previous remarks, the arrivals of customers in Si in simulation beginning with u_0 also happen at the same time and at the same queue in simulation beginning with v_0. The service rates are the same in both simulations. Therefore $u_0(i) \leq v_0(i)$ implies that $u_t(i) \leq v_t(i)$. Finally we get: $u_t \preceq_\beta v_t$ for all t. □

We present in the next figures some numerical results obtained for some parameters for λ and μ_i. The number of simulation processes K is equal to 100. We give for some typical simulations, the time necessary for simulation processes LP_i to be consistent for all i. We report the results for the usual algorithm and for the bounding algorithms (upper and lower bounds) without the improvements on the computation of speculative parts by idle processors. We also used two distributions for the Random states used in the initialisation part. Clearly both bounding algorithms are more efficient than the usual approach. They provide a bound of the paths and the rewards while exact methods are slower and simple approximations [3] do not give a guarantee on the performance. We only need 7 runs for obtaining 100 consistent parts of the sample path.

6 Conclusion

Many queueing systems are known to be monotone and such a property has not been used for simulation except within some Coupling From The Past algorithms. We think that monotonicity of the model has many applications in simulation which remain to be studied, especially for TPS or the space decomposition approaches. We will apply this technique for stochastic model checking by simulation and for the analysis of queueing elements with sophisticated service or access discipline. Indeed in stochastic model checking, we often have to compute bounds of probability for long paths [22,5] and TPS looks like a very efficient solution. Similarly, queues with complex discipline designed for service differentiation are difficult to analyse exactly. Improved TPS may be an alternative to stochastic bound and numerical analysis or fluid methods, see for instance [4] for a diffusion model of the Pushout mechanism and [2] for the analysis of the delays in WFQ queues.

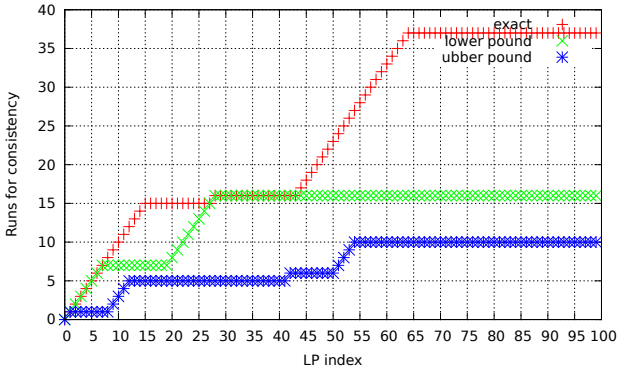

Fig. 5. Number of runs before each process is consistent, $\mu = 0.55$, $\lambda = 0.8$, second distribution for initial state

References

1. Andradottir, S., Hosseini-Nasab, M.: Parallel simulation of transfer lines by time segmentation. European Journal of Operational Research 159(2), 449–469 (2004)
2. Ben Mamoun, M., Pekergin, N.: Stochastic delay bounds of fair queuing policies by analyzing weighted round robin-related policies. Performance Evaluation 47(2), 181–196 (2002)
3. Bölöni, L., Turgut, D., Wang, G., Marinescu, D.C.: Challenges and benefits of time-parallel simulation of wireless ad hoc networks. In: Valuetools 2006: 1st International Conference on Performance Evaluation Methodologies and Tools, p. 31. ACM, New York (2006)
4. Czachórski, T., Fourneau, J.-M., Pekergin, F.: Diffusion model of the push-out buffer management policy. In: INFOCOM, pp. 252–261 (1992)
5. El Rabih, D., Pekergin, N.: Statistical model checking using perfect simulation. In: Liu, Z., Ravn, A.P. (eds.) ATVA 2009. LNCS, vol. 5799, pp. 120–134. Springer, Heidelberg (2009)

6. Fourneau, J.-M., Kadi, I.: Time parallel simulation of monotone systems. In: Poster Session, IFIP Performance Conference, Namur, Belgium (2010)
7. Fourneau, J.-M., Kadi, I., Pekergin, N.: Improving time parallel simulation for monotone systems. In: Turner, S.J., Roberts, D., Cai, W., El-Saddik, A. (eds.) DS-RT, pp. 231–234. IEEE Computer Society, Los Alamitos (2009)
8. Fourneau, J.-M., Kadi, I., Quessette, F.: Time parallel simulation and hv-monotonicity. In: Proceedings of the 26th Internation Symposium on Computer and Information Sciences. LNEE. Springer, Heidelberg (2011)
9. Fourneau, J.-M., Pekergin, N.: An algorithmic approach to stochastic bounds. In: Calzarossa, M.C., Tucci, S. (eds.) Performance 2002. LNCS, vol. 2459, pp. 64–88. Springer, Heidelberg (2002)
10. Fourneau, J.-M., Quessette, F.: Monotone queuing networks and time parallel simulation. In: Al-Begain, K., Balsamo, S., Fiems, D., Marin, A. (eds.) ASMTA 2011. LNCS, vol. 6751, pp. 204–218. Springer, Heidelberg (2011)
11. Fujimoto, R.M.: Parallel and Distributed Simulation Systems. Wiley Series on Parallel and Distributed Computing (2000)
12. Fujimoto, R.M., Cooper, C.A., Nikolaidis, I.: Parallel simulation of statistical multiplexers. J. of Discrete Event Dynamic Systems 5, 115–140 (1994)
13. Greenberg, A.G., Lubachevsky, B.D., Mitrani, I.: Algorithms for unboundedly parallel simulations. ACM Trans. Comput. Syst. 9(3), 201–221 (1991)
14. Harchol-Balter, M., Schroeder, B., Bansal, N., Agrawal, M.: Size-based scheduling to improve web performance. ACM Trans. Comput. Syst. 21, 207–233 (2003)
15. Kadi, I., Pekergin, N., Vincent, J.-M.: Different monotonicity definitions in stochastic modelling. In: Al-Begain, K., Fiems, D., Horváth, G. (eds.) ASMTA 2009. LNCS, vol. 5513, pp. 144–158. Springer, Heidelberg (2009)
16. Kiesling, T.: Using approximation with time-parallel simulation. Simulation 81, 255–266 (2005)
17. Lin, Y., Lazowska, E.: A time-division algorithm for parallel simulation. ACM Transactions on Modeling and Computer Simulation 1(1), 73–83 (1991)
18. Nicol, D., Greenberg, A., Lubachevsky, B.: Massively parallel algorithms for trace-driven cache simulations. IEEE Trans. Parallel Distrib. Syst. 5(8), 849–859 (1994)
19. Propp, J., Wilson, D.: Exact sampling with coupled Markov chains and applications to statistical mechanics. Random Structures and Algorithms 9(1&2), 223–252 (1996)
20. Stoyan, D.: Comparison Methods for Queues and Other Stochastic Models. John Wiley and Sons, Berlin (1983)
21. Turgut, D., Wang, G., Boloni, L., Marinescu, D.C.: Speedup-precision tradeoffs in time-parallel simulation of wireless ad hoc networks. In: DS-RT 2006: Proceedings of the 10th IEEE International Symposium on Distributed Simulation and Real-Time Applications, pp. 265–268. IEEE Computer Society Press, Los Alamitos (2006)
22. Younes, H.L., Simmons, R.G.: Statistical probabilistic model checking with a focus on time-bounded properties. Information and Computation 204(9), 1368–1409 (2006)