Distributed Stochastic Discrete-Event Simulation in Parallel Time Streams

K.Pawlikowski, V.Yau and D.McNickle

University of Canterbury
Christchurch, New Zealand

Abstract . Quantitative stochastic simulation as a tool used by engineers of various disciplines for studying and evaluating performance of various systems suffers from the fact that sound simulation studies requires very long runlength to obtain the results with the accuracy. In this paper we look at traditional approaches to distributed quantitative stochastic simulation and propose a new scenario, named Multiple Replications in Paralel Time Streams (MRIP), that solve the problem in an efficient way. An implementation of MRIP in a simulation package AKAROA has been also described. AKAROA accepts ordinary (non-parallel) simulation models and creates fully automatically the environment required for running MRIP on workstations of a local area network. Presented results show that MRIP offers linear speedup of simulation. Limitations of this scenario for running distributed quantitative stochastic simulation are also discussed.

1. Introduction

Over the last decade discrete-event simulation has become probably the most commonly used tool by engineers of various disciplines for studying and evaluating performance of various systems. This is the result of significant achievements in electronic and computer engineering that have led to the broad proliferation of powerful and cheap computers, and significant achievements in software technology, that have resulted in very simple and efficient interactive human-computer interfaces. Today, it is natural for telecommunication engineers to study processes occuring in data communication networks by watching their animated, dynamic graphical representations on monitors, using data generated by computers during simulation runs. There is easy access to various user-friendly simulation packages in which traditional discrete-event simulation modeling is supported by various concepts of artificial intelligence. Some of these packages totally release users from burdens of programming, allowing then to construct simulation models from typical components appearing on screens as icons. This situation has created a climate for spreading a popular impression that simulation is mainly an exercise in computer programming. This is certainly a totally misleading and dangerous opinion if it is applied to quantitative simulation studies, conducted for assessing the performance quality of simulated systems. Certainly, "*succeeding in simulation requires more than the ability to build useful models*" [KIVI91], and some claim that modelling of a simulated system represents only 30-40% of the total effort in most successful simulation projects [LAWM91]. One of the inherent problems of quantitative simulation is credibility of the final results.

This is specially true in the case of stochastic discrete-event simulations, i.e. when simulated events are functions of (pseudo)random numbers. Such simulation should be regarded as a statistical experiment, and, as in any statistical experiment, the final results should be given together with their accuracy, i.e. with the widths of their confidence intervals at an assumed confidence level. As warned by J. Kleijnen [KLEI79]: "*... computer runs yield a mass of data but this mass may turn into a mess <if the random nature of such output data is ignored, and then >... instead of an expensive simulation model, a toss of the coin had better be used*". Unfortunately, ignoring this has become very widespread in many areas. B.Gaither, the Editor-in-Chief of the ACM Performance Evaluation Review, complaining about the way stochastic simulation is used in computer science and engineering, stated in his editorial that he did not know of "*... any other field of engineering or science where similar*

liberties are taken with empirical data ..." [GAIT90]. This situation can be partially explained by the fact that some stochastic simulation studies, in particular if aimed at evaluating systems in their steady-state, might require knowledge of sophisticated statistical techniques.

Another inherent problem of stochastic simulation is that simulation of even moderately complex models can be computationally intensive and require very long simulation runs. Excessive runtimes hinder development and validation of simulation models, and can even totally inhibit some performance evaluation studies. The obvious solution is to speed up simulation by executing it on a multiprocessor or distributed computer system. Traditionally, distributed or parallel[1] stochastic simulation has meant **Single Replication in Parallel** (SRIP), based on many processors cooperating in executing a single replication of a simulated system. An alternative scenario is to run **Multiple Replications in Parallel** (MRIP), with processors engaged into running their own replications of the simulated system but cooperatingwith central analyzers (one central analyzer for each performance measure analyzed) that are responsible for observing the stopping criteria of the simulation.

Research in distributed and parallel simulation has been almost entirely focused on SRIP. In this scenario, a simulation process and/or simulation model is partitioned between a number of processors. When a simulation process is distributed then this distribution is done at a functional level, and the logical topology of interprocessor connections may reflect different functional elements of the simulation (event set processing, input/output processing, etc.); see eg. [BILE85]. It is obvious that this method cannot offer a substantial speedup in itself, since the degree of such distributiveness is limited.

The second option for SRIP is to partition a given simulation model is into a set of submodels to be simulated at different processors, of tightly or loosely coupled multiprocessors systems. The processors responsible for running processes related with different simulation submodels occassionally have to synchronise the advance of simulated processes. Many different methods have been proposed to achieve such a synchronisation; see e.g. [CHAN81, CHAND83, MISR86, REED87, LUBA89, CHAM90, FUJI90]. Generally speaking, it is achieved by exchanging timestamped messages between participating processors. Reasonable speedup is possible, provided that a given simulation model is highly decomposable. Unfortunately, this feature is not frequently observed in practise, thus the efficiency of this approach is strongly application-dependent [WAGN89]. The research in distributed/parallel processing, having successfully solved many related problems, has not led yet to a portable and efficient tool for distributed stochastic simulation.

MRIP has been considered only in a few publication (see [HEID86, HEID88, GLYN91, REGO91, SUND91, PAWL92, REGO92, YAUP93]), despite that, as our experience has shown, it is an attractive alternative scenario for quantitative stochastic simulation that potentially offers good speedup, linear with the number of procesors involved. Also, considering statistical properties of results when applying SRIP and MRIP in steady-state simulation, it is possible to show that the latter scenario is more efficient than the former, in the sense of the mean squared error of final estimates, if the problem of the initialization bias is effectively solved [HEID86].

In this paper we report results of our research project on developing fully automated version of MRIP and its implementation in AKAROA, started in the middle of 1991. AKAROA is a user-friendly package for running distributed quantitative stochastic simulation based on MRIP. It solves both main problems of such a simulation, by applying fully automated control of accuracy of the final results (a parallel version of sequential estimation based on spectral analysis [PAWL90], as described in Sec.2), and fully automated distribution of simulation for concurrent execution on many procesors, see Sec.3. The potentials and limitations of our version of MRIP, together with results of experimental studies of AKAROA are reported in Sec.4. While AKAROA has many features in common with EcliPse [REGO91, SUND91, REGO92], it also creates very different environment for running simulation. A user, having prepared a sequential simulation program has only to

---

[1] The terms "concurrent", "parallel" and "distributed" are differently understood in different areas, appplications and interest groups. For the purpose of this paper they are synonymous, meaning a simulation executed by a number of cooperating processors during the same time interval.

declare the required level of precision of the final results, e.g. 5%, and the maximum possible length of the simulation run (say, no more than 10 000 000 observations to be collected). All other decisions and functions are transparent for users.

Without a loss of generality we limit our discussion to simulation models based on queuing networks. While such models are natural in modelling computer systems, telecommunication networks and manufacturing systems, they also find alpplications in many other areas of science and engineering, e.g. in computational physics [LUBA88]. We will discuss here only the most difficult case of quantitative stochastic simulation: its application in studying performance of systems in steady-state, i.e. over very long period of time, as the time tends to infinity.

2. Preliminaries: Statistical Aspects of Non-distributed Steady-State Simulation

As mentioned, any performance evaluation studies of systems based on quantitative stochastic simulation should include proper statistical analysis of output data. In practise, this means that we should control the precision of steady-state estimators, i.e. the final estimate of an analysed parameter Θ should be determined together with its confidence interval $(\theta-\Delta_1, \theta+\Delta_2)$, at a given confidence level $1-\alpha$, i.e. by

$$P(\theta - \Delta_1 \le \Theta \le \theta + \Delta_2) = 1-\alpha \qquad (1)$$

where θ is the final estimate of Θ, and $\Delta_2+\Delta_1$ is the width of the confidence interval. The precision of estimates can be then measured by

$$\varepsilon = \textbf{Error!} \qquad (2)$$

ε is known as the relative precision. In the context of quantitative steady state simulation, the precision of the final results can be controlled if it is sequentially checked at consecutive checkpoints, and compared with the (worst) acceptable level of precision, ε_{max}. The simulation is stopped at a given checkpoint if the stopping criterion (the current value of relative precision ε being not greater than ε_{max}, for given ε_{max}, $0<\varepsilon_{max}<1$) is satisfied for the first time. In such a case we can claim that $\theta \pm 0.5\,(\Delta_2+\Delta_1)$ contains the correct value of Θ with probability $1-\alpha$.

This approach can be easily applied when analyzing performance measures using cumulative estimators. The simplest performance measure of this type is the sample mean μ_X, that, for given sequence of observations $x_1, x_2, ..., x_n$, is estimated by the arithmetic average, thus

$$\theta = X,^{-}(n) = \textbf{Error!}. \qquad (3)$$

In this case , by the Central Limit Theorem,

$$\Delta_1 = \Delta_2 = t_{n-1,1-\alpha/2}\, \sigma,^{\wedge} [X,^{-}(n)] , \qquad (4)$$

where $\sigma,^{\wedge} [X,^{-}(n)]$ is the estimator of standard deviation of $X,^{-}(n)$ and $t_{n-1,1-\alpha/2}$ is the $(1-0.5\alpha)$ quantile of Student t-distribution. The main analytical problem is to get a reliable estimate of $\sigma, [X,^{-}(n)]$ or, equivalently, of $var[X,^{-}(n)] = \sigma^2[X,^{-}(n)]$, since the classical formulae require that collected observations $x_1, x_2, ..., x_n$ are realizations of independent and normally (or at least identically) distributed random variables $X_1, X_2, ..., X_n$. Unfortunately, observations collected during typical stochastic simulations are neither independent or identically distributed. In the matter of fact, they are usually highly correlated. Also, a well known formula for estimating $var[X,^{-}(n)]$ from an autocorrelated and stationary (thus identically distributed) sequence of observations $x_1, x_2, ..., x_n$ can not be applied because of the dificulties of estimating autocovariances of higher order within finite samples of data.

A number of techniques for accurate estimation of $\sigma^2[X,^{-}(n)]$ has been proposed to be applied in non-distributed sequential simulation; see a survey in [PAWL90]. Our extensive studies of these techniques ([PAWL88, PAWL91]) led us to select SA/HW, the method based on spectral analysis, in its version proposed by Heidelberger and Welch [HEID81], as the method of analysis of $\sigma^2[X,^{-}(n)]$ in fully automated quantitative steady-state simulations.

SA/HW produces reliable estimates in the sense of their coverage (correspondence between the theoretical confidence level and its experimental correspondence, analysed over a long series of simulations) provided that simulated systems are not too heavy loaded, i.e. are utilized in no more than 90%. This is the common weakness of all techniques developed for automated analysis of $\sigma^2[X,^{-}(n)\ ]$, and its removal is one of topics of our current research activities [PAWL93].

SA/HW requires that collected observations belong to stationary time series, thus observations collected during initial transient periods of analysed processes should be discarded. For detecting the length of initial transient period one can use a sequential stationarity test presented in [PAWL90] or [PAWL93]. Only then the proper sequential analysis of steady-state statistics of simulated systems can begin. Under SA/HW only one (long) simulation run is executed at each setting of input variables. Thus, one can distinguish two stages in such automatic sequential steady-state simulations, as illustrated in Fig.1. A useful practical feature of this technique is that it can work with reduced data sets. During the whole course of a single simulation, we can work with a fixed number of (aggregated) output data points, being batch means calculated over batches that have their size increased as new observations are collected.

3. Distributed Quantitative Stochastic Simulation

As mentioned, practical simulations typically require excessively long runs, if executed on single processors. This has motivated applications of distributed and parallel processing for performing discrete-event simulation, and stochastic simulation in particular. Distributed stochastic simulation of queueing networks has also become an attractive domain for experimenting with concurrent processing because of the potentially high degree of parallelism of operations and distributiveness of network structures. Traditionally, applications of distributed, parallel processing in the area of stochastic simulation have focused on speeding up execution of single replications of simulation models. Another approach is to consider any instance of quantitative stochastic simulation as a statistical experiment during which statistical data are generated by a processor(s). Such experiments can be speeded up by generating statistical data in parallel, i.e. running replications of the simulated system in

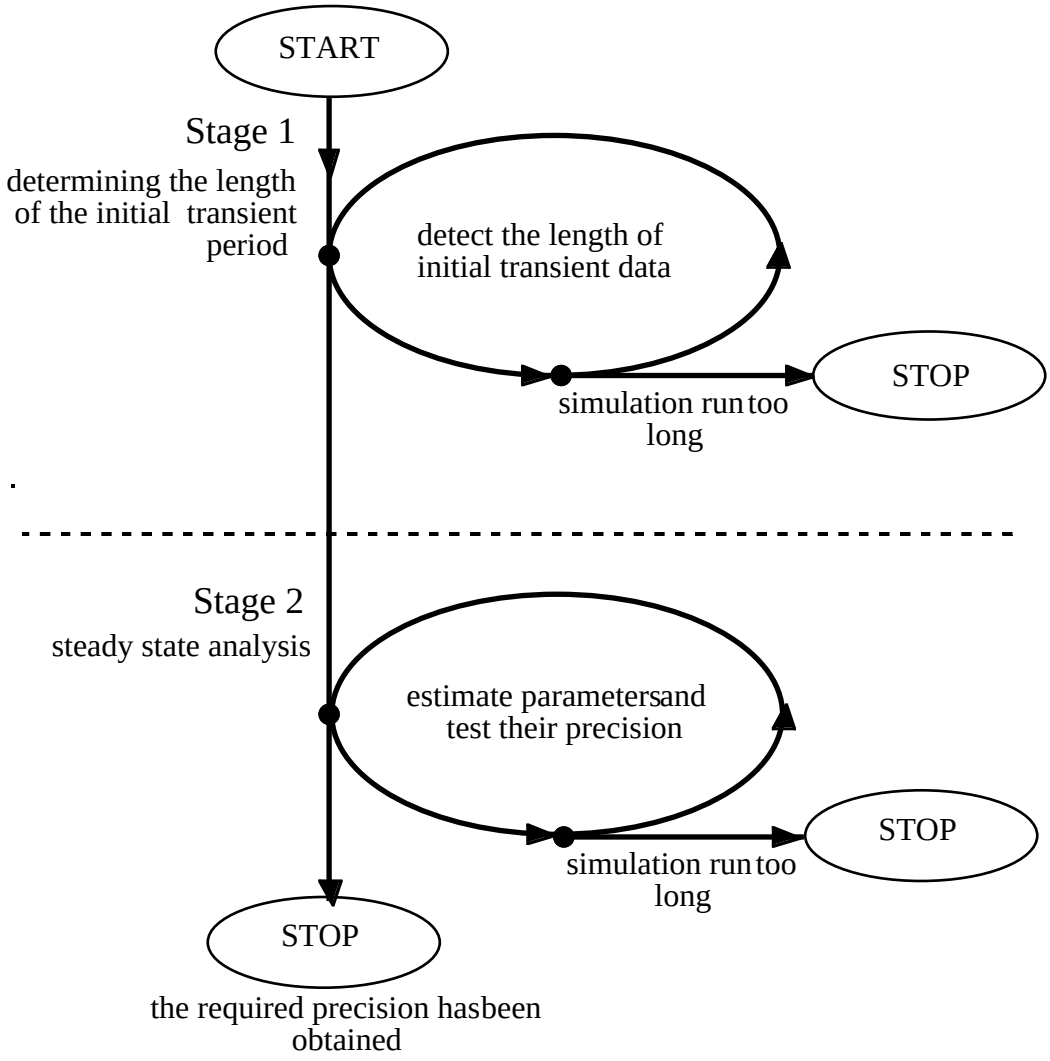


Fig.1. Flowchart of sequential analysis of of one time-series of observations collected during steady-state simulation

parallel, on many processors, under control of a global analyzer(s) responsible for analysing submitted data and detecting when the stopping condition of simulation is satisfied. Here, these two scenarios of distributed quantitative stochastic simulation are called Single Replication in Parallel (SRIP) and Multiple Replications in Parallel (MRIP), respectively **.**

3.1 Single Replication in Parallel. Following this traditional way of applying concurrent processing in quantitative stochastic simulation, it can be achieved at functional level of simulation, or it can be done by distributing simulation model. These two solutions we call here briefly as*SRIP.F* and*SRIP.M*, respectively. In the former, the logical topology of interprocessor connections reflects different functional elements of the simulation and such activities as random variate generation, event list manipulation, and statistical analysis of output data are performed at separate processors; see eg. [WYAT83, BILE85, KRIS85, ZEIG87]. But the extent of functional distributiveness of any simulation is generally not significant [COMF81, BRIN88]. Let us also note that the fine granularity of decomposition of support functions needed for their parallel execution necessitates frequent communication among subprocesses. This is connected with a more general problem of concurrent processing, discussed e.g. in [MCCR89]. Apart from consuming processor power, interprocess communication puts limitations on multiprocessor architectures that can be used in such applications. Communication between processes on different processors may substantially increase total traffic on the time-shared bus and/or multiple-bus of multiprocessor

systems to such extent that it becomes the bottleneck, increasing the amount of time the processors are blocked waiting for access to a common memory module. Even if the number of shared buses were increased, contention for memory (processors queue for accessing common memory module(s)) can create another bottleneck limiting effective processing power. Thus, one should not expect that this strategy allows to achieve a significant speedup [BURK90]. Because of that, SRIP.F is not satisfactory efficient to be used on its own.

In SRIP.M, speedup is achieved by executing (interdependent) parts of the simulation model in parallel, at different processors. It is done by abandoning the concept of shared objects, such as the global simulation clock and event list, and using a synchronisation algorithm instead, to ensure that causality of events is maintained. The synchronisation of parallel (sub)streams of events simulated at different processors is achieved by exchanging time-stamped messages, in an attempt to protect against causality errors, i.e. situations in which wrong synchronization between events causes that future actions affect the past. For this purpose, either an *optimistic* or *conservative* synchronisation algorithm can be used. In the former class of algorithms, event-carrying messages are allowed to be processed as soon as they are available, so there is no strict avoidance of causality errors. If such an error occurs, it is corrected by rolling the local time back. This is the basic idea implemented in the so-called Time Warp algorithm and its numerous variations [FUJI90]. Conservative synchronization algorithms ensure that, before an event is processed, any its prior event has already been executed. The first conservative algorithms were proposed independently by Chandy and Misra, and Bryant in 1977-79. Since then their numerous improvements have been investigated; see [FUJI90].

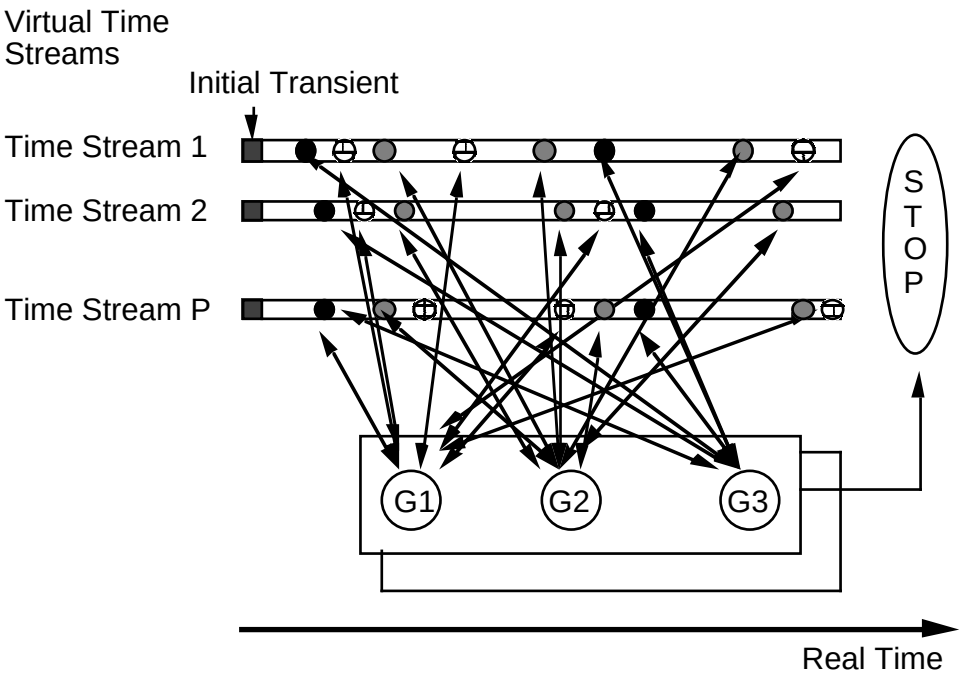
There are natural weaknesses and limitations on efficiency of SRIP.M. Firstly, it has been shown that a high structural parallelism of simulation models does not imply similar high parallelism in the simulation of that model. For example, the maximum speedup achievable when simulating a simple computer system (a CPU, two I/O devices and N terminals forming a closed queueing network) cannot be greater then 3.7 [WAGN89]. On the other hand, there are reports of many successful applications of distributed stochastic simulation executed in SRIP.M scenario, using either optimistic and conservative synchronisation algorithms. For example, a speedup as high as 1900 was demonstrated on a 16384 processor Connection Machine [LUBA89]. Generally, an experienced simulator should be able to obtain a good speedup, provided that a given simulation model is well decomposable. Unfortunately, this feature is rarely observed in simulation practise. Thus, the success is strongly application-related. And, if deadlocks and causality errors occur too often, or mechanisms for their protection against them are too complicated, the resulting simulation can be even slower then non-distributed one. This scenario of distributed stochastic simulation is also generally not well suited for running on distributed computer systems, such as networks of workstations, since there are substantial costs connected with interprocess communication. SRIP.M is also not fault-tolerant. If one processor or workstation running a sub-task fails, then the simulation fails too, due to causality between subtasks. Finally, this is the scenario of distributed stochastic simulation that does not seem to be well suited for automation, despite the fact that it uses single time-streams of data (one per each perormance measure), and, because of that, it could rely on sequential techniques of simulation output data developed for non-distributed simulation, such as SA/HW mentioned in Sec.2. But, the time order in which some observations are collected can differ from the logical order of their occurance in the simulated time, if they are collected from different logical (sub)proceses, or, some observations may need to be discarded if a causality error occurs. A simple solution could be to use time-stamped observations. However, this could generate an overhead consuming additional space and time resourses. Such aspects of sequential output data analysis in SRIP.M scenario, and associated problems, have not been studied yet.

3.2. Multiple Replications in Parallel (MRIP). In the light of these restrictions and limitations of SRIP, we have recognized that the duration of quantitative stochastic simulation directly depends on the time needed for collecting the required number of observations. If a few performance measures are studied during one simulation experiment, then the simulation is finished when all sequences of observations (one sequence per each performance measure) contain sufficiently many data items. Thus, a simple way for increasing the rate at which

observations are generated is to produce them in parallel time streams, i.e. to run statistically different replications on many processors, using the same simulation model. One can view these simulation replications run at different processors as *simulation engines* working in a team and producing samples of output data (one sample per each performance measure). Observations generated by different simulation engines, but representing values of the same performance measure, are submitted to the global analyser responsible for analysing this performance measure. Accepting arguments about random nature of results obtained from stochastic simulation, it is important to produce the final results with the assumed precision, Eq.(2). It requires the current precision of results be checked at consecutive checkpoints. The analysis of each performance measure is then continued until its stopping condition is not satisfied. All simulation engines run as long as the analyses of all performance measures is not finished. At that instant of time all simulation engines are stopped and global analysers produce the final results.

Distributed simulation in MRIP can be carried on with any simulation model, either on multiprocessor computers or multicomputer networks. Very little is know about estimators, even about estimators of sample mean, that could be applied in MRIP. Parallel versions of the method of Independent Replications, in the context of non-steady state simulation was analysed in [HEID88, GLYN91]. It was shown that an extreme care has to be taken when selecting estimators since some obvious choices were shown guarantee convergence to the wrong value when the number of processors increases. On the other hand, studying properties of MRIP as the scenario for running steady-state simulation, it was shown that MRIP can be more efficient that SRIP.M if the problem of initial transient is effectively solved [HEID86].

Relying on our previous experience in non-distributed stochastic simulation, we have decided to adopt the method of SA/HW (discussed in Sec.2) also in distributed simulation, in MRIP scenario. Our proposal is a parallel generalization of SA/HW, named the **Spectral Analysis in Parallel Time Streams** (SA-PTS). Virtual-real time interactions between processes at simulation engines and global analysers in such version of MRIP are depicted in Fig.2. According to SA-PTS, P logically equivalent instances of a simulation model are launched at P processors at the beginning of the simulation. Each instance is run in a time-stream parallel to others, using different sequence of (pseudo)-random numbers. At the beginning, stationarity tests are applied locally within each replication, to determine the outset of steady state conditions in each time-stream separately, and the sequential SA/HW method is used to estimate the variance of local estimates at consecutive checkpoints. At each checkpoint the current local estimate and its variance are sent to the global analyser which computes the current value of the global estimate and its precision. Thus, when the simulation engine i reaches its check point j, it sends a message containing the pair $\{\bar{X}_i(n_j), V_{ij} = \hat{\sigma}^2[\bar{X}_i(n_j)\}$, the mean and itsvariance, to the global analyser responsible for analysis of $\bar{X}$. Thus, the global precision of each estimator is analysed following partially ordered sequence of checkpoints (checkpoints associated with the same simulation engine are ordered in time, but we get a randomly ordered sequence of checkpoints from different processors as they



Spectral Analysis in Parallel Time Streams
(⊖ = local checkpoint parameter 1,
● = local checkpoint parameter 2,
● = local checkpoint parameter 3,
Gi = Global Estimation of parameter i, i=1,2,3.)

Fig.2. Virtual time -real time interactions between processes at simulation engines and global analysers in SA-PTS

followed by a given global analyser). If P processors are used to in a given simulation, each time when the global analyser is active it can use up to P partial estimates of variance, V_{1j_1} , V_{2j_2} , ..., V_{Pj_P}, submitted from independent replications of the simulated process that reached the checkpoints j_1, j_2 , ..., and j_P, respectively. When p of P processors have reached at least the first checkpoint of the process they simulate, the pooled mean is estimated over $k= n_{1j_1}+n_{2j_2}+...+n_{pj_p}$ observations, i.e. using n_{1j_1} data collected by processor 1 at its checkpoint j_1, n_{2j_2} data collected by processor 2 at its checkpoint j_2, etc. In the simplest case, the estimate of polled vaiance can be obtained as

$$\sigma,^{\wedge 2}{}_{,SAPTS}[\,X,^{=}(k)\,] = (\textbf{Error!})^2\, V_{1j_1} + (\textbf{Error!})^2\, V_{2j_2} + ...(\textbf{Error!})^2\, V_{Pj_P} \quad (5)$$

where $X,^{-}{}_{i}(n_{ij_i})$ is the most recent point estimate of the mean obtained from the simulation engine i, at its most recently observed checkpoint j_i. Let us note that for determining the confidence interval for the sample mean μ_X, one has to know the probability distribution of **Error!**. We have assumed that it is approximately governed by Student t-distribution with p times d degrees of freedom (where d equals the number of degrees of freedom of t-statistic coming from one replication[2]), if data from p simulation engines are used for determining precision of a given pooled estimate. Let us note the obvious fault tolerance of MRIP regarding simulation engines. Sudden lost of one or more processors running simulaiton engines is not catastrophic as long as at least one simulation engine remains is able to continue submitting data to global analysers.

[2] In SA/HW the degrees of freedom of this t-statistic do not depend on the number of observations collected but on the way in which these observations are grouped for analysis [HEID81]

At this stage no theoretical studies of SA-PTS are available. But our experimental results, see the next section, show that it produces good estimates in the sense of experimental converage of the final confidence intervals. Further work on improving quality our estimator of the pooled variance is continued. SA-PTS has been implemented in AKAROA, our simulation package for rapid modeling, automatic generation of multiple processes and process control for concurrent stochastic simulation in MRIP scenario.

3.3. An Implementation of MRIP in AKAROA.

AKAROA (a simulation p**a**c**k**age for **a**utomatic generation and cont**r**ol of processes for parallel st**o**chastic simul**a**tion)[3] accepts ordinary (non-parallel) simulation programs, and creates fully automatically the environment required for running MRIP on workstations of a local area network. Our main considerations when selecting a development language and designing programming interface were simplicity, space and code efficiency, as well as compatibility with existing sequential simulation programs. Recognizing naturality of object-oriented approach in constructing simulation models by means of hierarchically encapsulated classes of objects, AKAROA is written in C++. A user of AKAROA is required to add only one extra line of code to his/her sequential simulation program before AKAROA transparently parallelizes it. Thus, users do not even need to be aware of the existence of multiple (parallel) simulation engines and control processes during simulation, since their creation, location (machine and port addresses), cooperation, and inter-machine interprocess communication, are hidden from users. AKAROA consists of three modules: *Control*, responsible for controlling simulation run-time and analysis of output data collected during MRIP-type steady-state simulation; *Parallel Simulation Manager* (PSM), responsible for automatic initialization of parallel simulation processes, process management and interprocess communication; and *Build*, a module which can be used for speeding up construction of typical simulation models.

Sequential precision control services are arranged by declaring an object for output data analysis. Its member function responsible for precision control is later called whenever a new observation is recorded. The function accepts the value of a new observation as parameter and returns one of two values that either orders the simulation to be continued (desired precision of estimates has not been achieved) or to be terminated (all estimates reached the required level of precision). Such implementation of MRIP simulation is semantically identical to a normal non-distributed simulation; only the type of object that needs to be declared is different. The syntax for object declaration and calls of object's member functions are also identical to those in the non-parallel case. Internal binding of simulation processes to various control processes is performed dynamically, yielding a flexible and fault-tolerant system, featuring totally transparent parallelization from users' point of view, both in semantic and syntactic sense.

PSM automatically creates and maintains an environment in which MRIP can be executed (a collection of support processes distributed among the computers), parallelizes and runs the simulation. Launching one simulation replication by activating a simulation program equipped with necessary objects of Control and PSM creates a simulation engine. PSM provides dynamic binding between simulation engines and global control processes. Development of an efficient, portable and flexible Interprocess Communication (IPC) subsystem of PSM was regarded as the critical factor for achieving high efficiency of AKAROA. It is known that a careless implementation of IPC can result even in a negative speedup of parallel processing, if high IPC overhead is generated. UNIX facilities for implementing IPC mechanisms include streams, pipes, socket-pairs, and various types of sockets [QUAR85]. AKAROA's IPC subsystem must support communicating processes located on different machines, and possibly belonging to different file systems. Having considered basic IPC facilities, the IPC selected for AKAROA uses an extension of a (synchronous) Remote Procedure Call (RPC) mechanism ([BIRR84], [BRIA90]) and appears as an inter-machine interprocess communication based on UNIX Internet domain datagram-type sockets that allow for fully file-less exchange operations. Selection of RPC was

[3] Also: a nice spot on Banks Peninsula in the South Island of New Zealand.

motivated by the fact that it has simpler semantics than the Rendezvous model [GEHA88], and that, as a higher level of construct, it better encapsulates (simulation engine, control process) interactions than alternative solutions based on point-to-point message-passing [HOAR78]. Further possible improvements could be achieved by introducing an asynchronous RPC [TAYK92], which is one of our future plans.

When N different parameters are estimated while P simulation engine processes are maintained, then these P simulation engines have to communicate with N global control processes. When initiating communication between a pair (simulator, control process), nine pieces of PSM program are involved: simulator_stub, simulator_RPC_Runtime, Director/Launcher RPC_Runtime, Director_Launcher_stub, Director/Launcher server, plus global_control_runtime, global_control_stub, and global_control_server.

Whenever a simulation engine produces sufficiently many observations for reaching a checkpoint of sequential estimtion, a new call to the global control process responsible for gathering local estimates from the simulations engines can be initiated. The caller_engine, i.e. the simulator_stub of the simulation engine making the call, generates a *locate_request* datagram to a known director/launcher process, specifying its (the caller_engine's) machine address and port number, the type of call and the instance of global_estimation control process it wishes to be connected with. The director/launcher, once receiving the caller_engine's datagram, searches its Active_Control_Process (ACP) Database for the requested instance of the process. If found, it transmits a datagram to the caller_engine with the location of that process. Otherwise, the director/launcher launches new global SA-PTS process, updates its ACP Database, and transmits a datagram to the caller_engine with the location of newly created control process. Then, the caller_engine transmits a *global_estimation_request* datagram with appropriate data (parameter's identification, its current value and precision, the length of initial transient, and the length of simulation) to the located instance of global control process. Upon receiving the datagram from the caller_engine, the control process updates its local estimation database and computes a combined estimate of the estimate and its relative precision. If the desired level of precision is achieved, a *stop* datagram is returned to the caller_engine; otherwise a *continue* datagram is transmitted. Subsequent calls to *stub.global_estimation* are directed to the *global_control* process, without going through the Director/Launcher.

To use the objects provided by Control and PSM for automatic statistical analysis of simulation output data, dynamic data precision control, and parallelization of a simulator for parallel execution, a user's program should have

#include <AKAROA.h>

at its beginning. To transform an existing sequential (non-parallel) simulator into an MRIP simulator, a parallel simulation interface object

sapts sa(int num_parameters, double max_precision, double confidence) ;

should be declared, before the desired service can be requested by calling the member function(s) of the object. *max_precison* means the maximum acceptable value of the final precision of results, while *confidence* is the assumed level of confidence. The only other statement required is

result = sa1.processnewobs(new_observation, parameter_number) ;

that should precede the generation of an observation. After that, all tasks relating to precision control and parallel execution will be performed without further user involvement.

For better memory utilization, control objects presented above were implemented dynamically, i.e. were composed of two (sub)objects for each parameter being estimated in the simulation run, each of which being created only when its services are required. The first constituent object is for testing for the end of transient phase, and the other is for steady state precision control. The transient object is created only when the first observation has been obtained, and does not occupy resources before then. This transient object is disposed after steady state is detected, relinquishing the resources it occupied (its instruction and data space) to the system. In a similar manner, the steady-state output analysis and precision control object is created only after steady state is reached, and is disposed of when the desired precision is obtained. As soon as an estimate achieves the required precision, objects devoted to its precision analysis are freed. All these operations are transparent to the user.

Making full use of the adopted object-oriented programming paradigm, AKAROA has been also equipped with an object-oriented toolkit, called Build, which allows users to work at a high level of abstraction when constructing new simulation models, either by using already existing building components (classes) or defining new components in terms of existing ones. A component may model any entity of a given simulated system, for example a priority queue, encapsulating its attributes, and procedures for their manipulation. One immediate benefit of modular construction is that any component of the model can be readily and independently tested. Its data can be then manipulated through specific access functions, such as functions for enqueueing and dequeueing, helping to protect its consistency. Lastly, components can be easily reused, either directly in another simulation or in definitions of new components. Of course, there is no need for using Build with simulation models constructed without its help.

4. Performance of MRIP

Dynamic properties of AKAROA, and the quality of SA-PTS estimators, were tested in a series of 1600 benchmark simulation experiments using P=1, 2, 4, and 6 processors. Initial studies of AKAROA's performance were done on a local computer network (a multiprocessor SUN Server with two SPARC CPUs, various SUN 4 and SUN SPARC workstations) based on Ethernet. Apart from the obvious differences in processing power between the workstations available for our investigations, none of the machines was dedicated solely to AKAROA's use. In this situation, simulation experiments that we conducted for evaluating AKAROA on single processors (P=1) s were done using the fastest machine available, during its low load period's, while all multimachine experiments involved a mix of the fast and lower rated workstations during normal working hours. Further, the priority of simulation processes engaged in parallel simulations was lower, to accommodate other users of the network, while the non-parallel simulations (P=1) were run at the highest priority level. Thus, the results reported here are very conservative.

Performance of the MRIP implemented in AKAROA was assessed by measuring the *real time speedup*, defined as the ratio of mean real times needed for stopping the same steady-state simulations on one and P processors, for P= 1, 2, 4, 6. Note that it gives the reduction in real time of simulation, as observed by the user, and accounts for the overhead incurred in concurrent processing, including time required for creating parallel processes, management, and inter-machine interprocess communication, as well as the delays caused by (non-AKAROA) processes of generated by other users, sharing the machines used in the experiments.

All presented here results were obtained from steady-state simulations of $M/M/1/\infty$ queueing systems with traffic load ρ=90%; each result is an average over 200 experiments, and the mean time spent in the system was estimated. The required level of precision of final estimates was ±5%, at the 0.95 level of confidence. We were assuming also two different strategies for determining the distance between consecutive checkpoints during the simulaiton: one in which that distance was geometrically increasing and another one, in which it was kept constant. While former strategy is typically used in sequential steady-state simulations run on single processors, our results clerily show that the strategy to keep checkpoints uniformly distributed is much better in the MRIP scenario.

The results showing the real time speedup of simulations achieved with AKAROA, as a function of the number of workstations used, are depicted in Figs. 3 and 4. Both figures clearly show the nearly linear (with the number of processors involved) speedup offered by AKAROA in the case of geometrically distributed checkpoints, and even better speedup in the case of uniformly distributed checkpoints.

The CPU-times, normalized to the average time required for generating an observation, for different levels of parallelization are be compared in Figs.5. and 6. There are three results for each value of P: the average minimum run length of a simulation engine within 200 repeated MRIP simulations; the average number of observations per simulation engine produced during an experiment (averaged over 200 replications); and the average maximum run length of a simulation engine. Comparing the maximum replication lengths as a function of P, one can see that the reduction in CPU time with P workstations is greater than 1/P,

suggesting super-linear speedup ! This may be due to the fact that AKAROA uses CPU time more efficiently, and n observations generated by P workstations in parallel case (P>1) have higher entropy than if they were collected from a single replication. The results also show that using uniformly distributed checkpoints we can achieve substantially shorten simulation runs. In the quantitative stochastic simulation there is a danger that shorter simulation runs produce less reliable final resuls. But, as the results of coverage analysis presented in the Table 1 show, the quality of estimators (better coverage) improves with the degree of parallelism, what is probably caused by the fact that entropy of SA-PTS global estimates grows with P. Further research in this area is needed.

Finally, Figs.7 and 8 show the average numbers of messages (datagrams) exchanged in AKAROA as a function of P. One can see, the communication overhead grows slower than linearly with the number of communicating processors in the case of geometrically distributed checkpoints, and becomes practically constant for uniformly distributed checkpoints, showing a clear advantage of MRIP over traditional SRIP scenario, and a good efficiency of our IPC subsystem implemented in AKAROA. These figures show too, that higher speedups characterizing uniformly distributed checkpoints result from the fact that simulations with uniformly distributed checkpoints go through more checkpoints before they are stopped than in the case of geometrically distributed checkpoints (in the former more datagrams are transmitted.

Limitations of MRIP. Let us note that it is possible that when MRIP is applied in a heterogenous network, with one processor much faster than others, slower processors may not be able to contribute in parallel production of data since none of them would reach its first checkpoint when the fastest processor stops the whole simulation by generating the required number of observations.

On the other hand, the best speedup should be observed in homogeneous netwroks, with all processors (smulation engines) operating at the same speed. It is possible that in such situation all P processors would equally contribute in MRIP simulation, submitting, on average, the same number of observations to global analysers. Since the number of observations needed to obtain the required precision of results is fixed, at some stage each processor will be able to reach only the first checkpoint, and the simulation will be stopped. Let P_{max} be the minimum number of processors when it happens. In such a situation, adding more processors would not increase the speedup that has already reached its limit value equal P_{max}. The only effect of having more data (generated by $P>P_{max}$ processors) would be better final precision of results.

If n_1 means the mean number of observations, per simulation engine, needed to reach the first checkpoint, and N_{max} is the total number of observations needed for stopping simulation with the required precision of results, then

$$P_{max} = \lceil N_{max}/n_1 \rceil \qquad (6)$$

Basing on our experimental, it means that simulating such dynamic queueuing system as M/M/1, loaded in 90 %, for typical values n_1=1 000, and N_{max}= 1 000 000, we get P_{max} = 1000. When simulating data communication networks such as DQDB (a standard for fiber optic metropolitan area networks, at 150 Mbps) loaded in 50 %, for typical values n_1=500 and N_{max}= 50 000, the maximum speedup P_{max} = 100. These observations should yet to be confirmed by experimental studies.

Let us note that if a distributed simulation originally based on SRIP is speeded up by factor S_{SRIP} , then applying additionally MRIP, i.e. parallelizing P times a simulation model already distributed in lines of SRIP.F or SRI.M, will additionally increase the speedup P times, i.e. the final speedup would be PS_{SRIP} , as long as $P \le P_{max}$.

5. Conclusions.

We have discussed main features of a new scenario for distributed quantitative stochastic simulation, named Multiple Replications in Parallel, and compare it with traditional Single

Replication in Parallel. Most important features of MRIP are its universality, as it can be applied without exemption to any simulation model, and the high level of speedup it offers.

Whe presenting an application of MRIP in AKAROA, a new technique of outpt data analysis (SA-PTS) has been introduced and assessed. The selection of this technique for analysing output data in AKAROA was motivated by our intention of full automation of distributed quantitative steady-state simulation. This eliminated for example the method of regenerative cycles and independent replications, both adopted in EcliPse [REGO91, REGO92, SUND92], that require some decisions to be undertaken by simulators before a simulation, and, to make these decisions properly, one should know well the dynamics of the simulated system. Nevertheless, we are investigating different solutions for full automation of the independent replications [YAUP91], [PAWL93].

Further design issues of AKAROA that are under our current considerations include equipping this package with a graphical interface, for visualization of various stages of simulation, sequential data analysis and the final results.

Acknowledgement. This research was partially supported by the Research Laboratories of Australian and Overseas Telecommunications Co. in Melbourne, Australia. The permission of the Managing Director of AOTC to publish this paper is hereby acknowledged.

References

BILE85 Biles, W.E, Daniels, C.M., and T.J. O'Donnell. " Statistical Considerations in Simulation on a Network of Multicomputers". Proc. 1985 Winter Simulation Conf., IEEE Press, 1985, 388-393

BIRR84 Birrell A.D., and B.J.Nelson, "Implementing Remote Procedure Calls". *ACM Trans on Comput. Systems*, vol.2, Feb.1984, pp.39-59

BRIA90 Brian N.B., T.E.Anderson, and E.D.Lazowska, "Lightweight Remote Procedure Call". *ACM Trans. on Comput. Sys.*, vol.8, Feb.1990, 37-55

BRIN88 Briner, J. "A Framework for Analysing Parallel Discrete Event Simulation". Proc. Int. Conf. Manag. and Perf. Evaluation of Computer Systems, CMG'88, Dallas, 1988, 180-185.

BURK90 Burk, W.H. "Limitations to parallel Processing". Proc. 9th Int. Phoenix Conf. Comput. and Commun., 1990, 86-93

CHAM90 Chamberlain, R.D., and M.A.Franklin. "Huerarchical Discrete-Event Simulation on Hypercube Architecture". *IEEE Micro*, August 1990, 10-20

CHAN81 Chandy, K.M., and J Misra. "Asynchronous Distributed Simulation via a Sequence of Parallel Computations". *Commun. of the ACM*, 1981, vol.24, No.11, 198-205

CHAN83 Chandy, K.M., J. Misra, and L.M.Haas. "Distributed Deadlock Detection". *ACM Trans. on Comp. Systems*, vol.1, No.2, May 1983, 144-156

COMF81 Comfort, J., and A.Miller. "Considerations in the Design of a Multiprocessor-Based Simulation Computer". In *Modelling and Simulation on Microcomputers*, ed. L. Leventhal, So. of Computer Simulation, laJolla, 1981.

FUJI90 Fujimoto, R. "Parallel Discrete Event Simulation". *Comm. of the ACM*, vol.33, Oct.1990, 30-60

GAIT90 Gaither, B. "Empty Empiricism". *ACM Performance Evaluation Review*, vol.18, No.2, August 1990, 2-3

GLYN91 Glynn, P.W., and P.Heidelberger. "Analysis of Parallel Replicated Simulations under a Completion Time Constraint". *ACM Trans. on Modeling and Computer Simulation*, vol.1, no.1, Jan.1991, 3-23.

HEID81 Heidelberger, P., and P.D. Welch. "A Spectral Method for Confidence Interval Generation and Run Length Control in Simulations". *Comm. of the ACM*, 1981, 233-245.

HEID86 Heidelbereger, P. "Statistical Analysis of Parallel Simulations". Proc. 1986 Winter Simulation Conf., IEEE Press, 1986, 290-295

HEID88 Heidelbereger, P. "Discrete Event Simulations and Parallel Processing: Statistical Properties". SIAM J. Stat. Comput.. vol.9, no.6, Nov. 1988, pp. 1114-1132

HOAR78 Hoare, C.A.R. “Communicating sequential Processes”. *Comm. ACM*, Feb. 1978, 75-83

KIVI91 Kiviat, P.J. "Simulation, Technology and the Decision Process". *ACM Trans. on Modeling and Computer Simulation*, vol.1, No.2, April 1991, pp.89-98.

KLEI82 Kleijnen, J.P.C., R. Van der Ven and B. Saunder. "Testing Independence of Simulation Subruns: A Note on the Power of the von Neumann Test". *Eur. J. Operational Res.*, 1982, 92-93.

KRIS85 Krishnamurthi, M., U.Chandrasekaran, and S.Sheppard. "Two Approaches to the Implementation of a Distributed Simulation System". *Proc. 1985 Winter Simulation Conf.*, IEEE, Nov. 1984, 463-464

LAWM91 Law, A.M., and M.G. McComas. "'Secrets of Successful Simulation Studies". Proc. 1991 Witer Simulation Conf., IEEE Press, 1991, 21-27

LUBA88 Lubaczevsky, B.D. "Efficient Parallel Simulation of Dynamic Ising Spin Systems". *J.Comp. Physics*, vol.75, no.1, March 1988, 103-122

LUBA89 Lubachevsky, B.D. "Efficient Distributed Event-driven Simulations of Multiple-Loop Networks". *Commun. of the ACM*, Jan. 1989, 111-123

MCCR89 McCreary, J.D., and H.Gill. "Automatic Determination of Grain Size for Efficient Parallel Processing". *Commun. of the ACM*, vol.32, No.9, 1989, 1073-1078

MISR86a Misra, M. "Distributed Discrete-Event Simulation".*ACM Computing Surveys*, March 1986, 39-65.

PAWL88 Pawlikowski, K., and M. Asgarkhani. "Sequential Procedures in Simulation Studies of Satellite Protocols". *Proc. ITC'12*, Torino, Italy, 1988, vol.6, 4.3B.3.1- 7.

PAWL90 Pawlikowski, K. "Steady-State Simulation of Queueing Processes: a Survey of Problems and Solutions". *ACM Computing Surveys*, vol.22, No.2, June 1990, 123-170.

PAWL91 Pawlikowski , K., and V.Yau."Independent Replications versus Spectral Analysis of Output Data in Steady-State Simulation of High Speed Data Networks". Proc. 6th Australian Teletraffic Research Sem., Wollongong, Australia, Nov. 1991, 322-330.

PAWL92 Pawlikowski, K., and V.Yau."An Automatic Partitioning, Runtime Control and Output Analysis Methodology for Massively Parallel Simulations". Proc. European Simulation Symp., ESS'92, Dresden, Germany, Nov.1992.

PAWL93 Pawlikowski , K., and C.Stacey. "Detection and Significance of Initial Transient Period in Quantitative Steady-State Simulation". Will be published in Proc. 7th Australian Teletraffic Research Sem., Melbourne, Australia, Dec. 1993

QUAR85 Quarterman, J.S., A.Silberschatz, and J.L.Peterson, "4.2BSD and 4.3BSD as Examples of the UNIX System", ACM Computing Surveys, vol.17, Dec.1985, 379-418

REED87 Reed, D.A., and A.D.Malony. "Parallel Discrete Event Simulation: a Shared Memory Approach". *IEEE Tran.s. Software Eng.*, vol.14, no.4, 1988, 541-553

REGO91 Rego, V.J., and V.S.Sunderam. "Concurrent Stochastic Simulation: Experiments with Eclipse". Proc. Int. Conf. Perf. of Distributed Systems and Integrated Commun. Networks, 1991, 253-271.

REGO92 Rego, V.J., and V.S.Sunderam. "Experiments in Concurrent Stochastic Simulation: the Eclipse Paradigm". *J. of Parallel and Distributed Comp.*, vol.14, 1992, 66-84

SUND91 Sunderam, V.S., and V.J.Rego. "EcliPse: a System for High Performance Concurrent Simulation". *Software-Practise and Experience*, vol.21, Nov. 1991, 1189-1219

TAYK92 Tay, B.H., E.K.Koh, and A.L.Ananda. "An Asynchronous Remote Procedure Call Mechanism for Distributed Computing". Tech.Rep. TRB4/92, Dept. of Information Systems and Computer Science, National Univ. of Singapore, 1992

WYAT8 Wyatt, D.L., and S.Sheppard. "A Language Directed Distributed Discrete Simulation System". *Proc. 1984 Winter Simulation Conf.*, IEEE, Nov. 1984, 463-464

WAGN89 Wagner, D.B., and E.Lazowska. "Parallel Simulation of Queueing Networks: Limitations and Potentials". *Performance Evaluation Review*, vol.17, May 1989, 146-155.

YAUP93 Yau, V., and K.Pawlikowski. "AKAROA: a Package for Automatic Generation and Process Control of Parallel Sotochastic Simulation". Proc. 16th Australian Computer Science Conf., Brisbane, Australia, Feb. 1993, 71-82

ZEIG87 Zeigler, B.P. "Hierarchical, Modular Discrete Event Modelling in an Object-Oriented Environment". Simulation, vol.49, no.5, 1987, 219-230

Distributed Stochastic Discrete-Event Simulation in Parallel Time Streams

K.Pawlikowski, V.Yau and D.McNickle

University of Canterbury
Christchurch, New Zealand

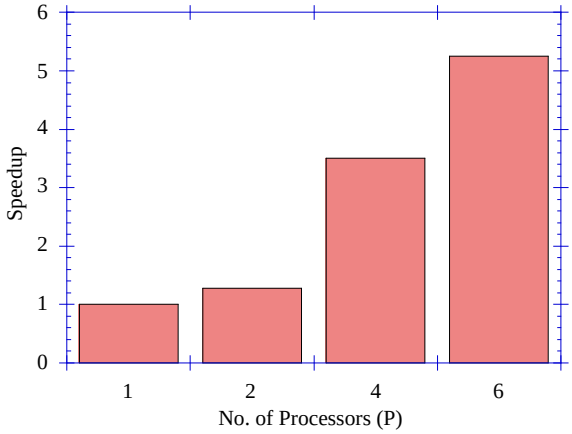


Fig.3. Real time speedup vs the number of processors employed.
Geometrically distributed checkpoints.

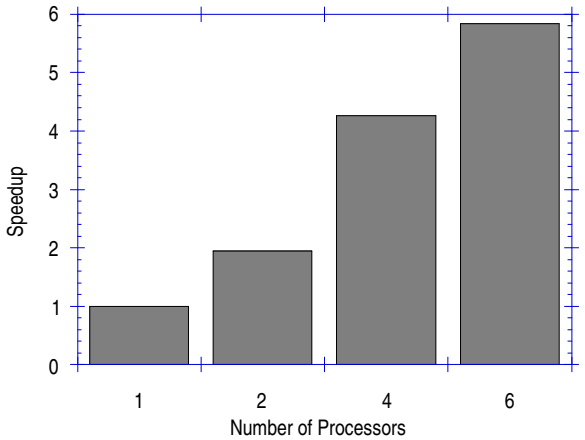


Fig. 4. Real time speedup vs the number of processors employed.
Uniformly distributed checkpoints.

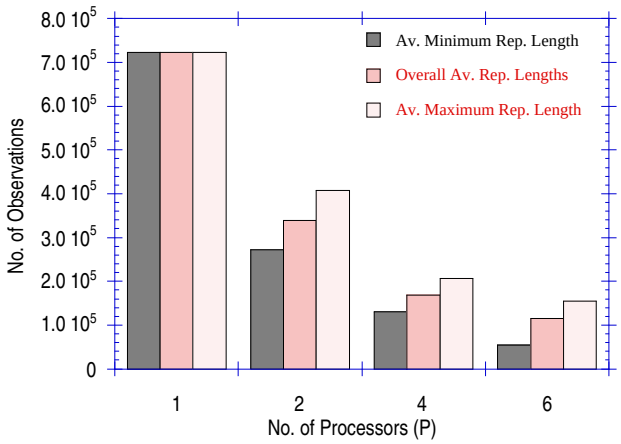


Fig.5. Speedup measured by reduction in CPU timevs the number of processors employed. Geometrically distributed checkpoints.

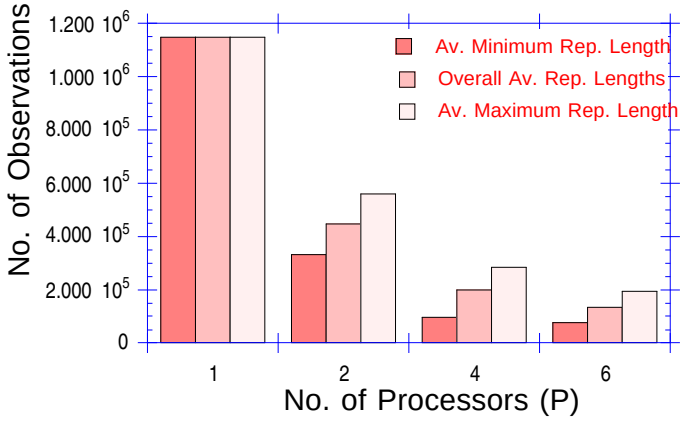


Fig.6. Speedup measured by reduction in CPU time vs the number of processors employed. Uniformly distributed checkpoints.

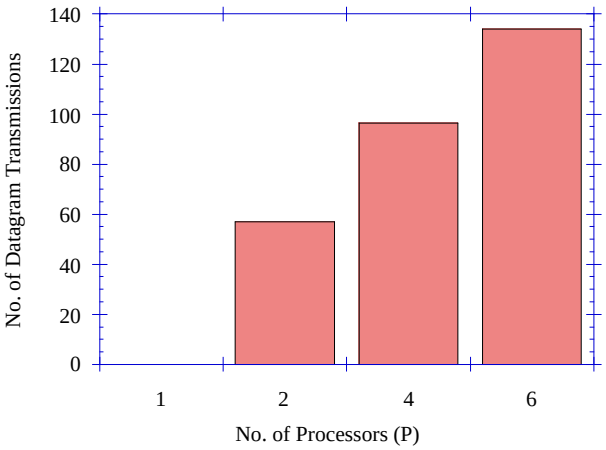


Fig.7. Average number of datagrams exchanged vs the number of processors employed. Geometrically distributed checkpoints.

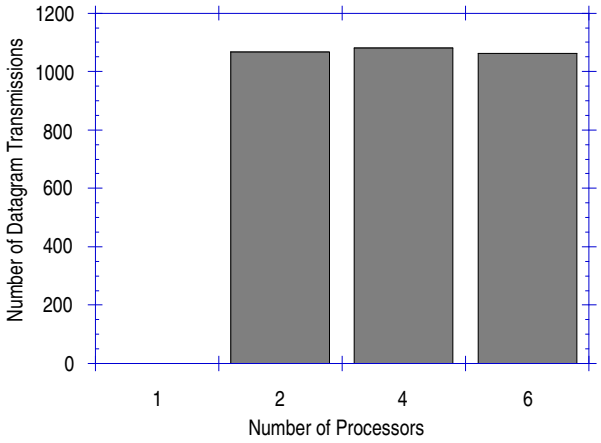


Fig.8. Average number of datagrams exchanged vs the number of processors employed. Uniformly distributed checkpoints.

TABLE 1. Coverage and its confidence intervals for SA-PTS at the confidence level =0.95. Uniformly distributed checkpoints.

P	Mean	95% Confidence Interval
1	0.91270	(0.865000, 0.817630)
2	0.94000	(0.907079, 0.972921)
4	0.93000	(0.894631, 0.965369)
6	0.92000	(0.882393, 0.957607)