An Analytic Method for Predicting Simulation Parallelism

Hong WANG, Yong Meng TEO and Seng Chuan TAY
Department of Computer Science
National University of Singapore
3 Science Drive 2
Singapore 117543
teoym@comp.nus.edu.sg

Abstract

The ability to predict the performance of a simulation application before its implementation is an important factor to the adoption of parallel simulation technology in industry. Ideally, a simulationist estimates the inherent parallelism *of a simulation problem to determine whether it is worthwhile to invest resources to carry out a parallel simulation. In this paper, we proposed an analytic method for predicting the simulation parallelism of a simulation problem that is independent of implementation details. We assume that the system to be simulated is modelled as a network of logical processes, and each logical process models a queuing server center. Unlike many analytic models reported in the literature, we consider the causal relations among events in a simulation. Causality effects reduce event parallelism. Our proposed analytic method gives a tighter upper bound on performance speedup. Validation experiments show that our analytic prediction of simulation parallelism differs from that of critical path analysis by 2.9% and 18.8% in open and closed systems respectively.*

1 Introduction

In the last two decades, parallel and distributed simulation has been intensively studied. The two major approaches of asynchronous parallel simulation are conservative [4] and optimistic [11]. Synchronization protocols for parallel simulation have been proposed [4, 11, 16, 18, 19], and many experimental parallel simulation suites are available [2, 6, 14, 20].

Performance of parallel simulation has also been extensively studied in the past [7]. Much work has been devoted to the evaluation of synchronization protocols [1, 3, 8, 5, 10, 12, 15, 23]. Ferscha et al. [5] proposed a test-bed (called N-MAP) for performance prediction of parallel simulation protocols. The method considers the characteristics of simulated problems, synchronization protocols and execution platforms, and performance prediction is dependent on the implementation of the test-bed and its execution platform. Berry et al. [3] introduced critical path analysis of parallel simulation. Critical path analysis is performed based on an implementation of a sequential simulation. Gupta et al. [8] proposed a Markovian model to analyze the performance of Time Warp. The performance and dynamics of the processes are analyzed under the following assumptions: exponential task times and time-stamp increments of events, fixed number of events, negligible rollback, state saving, communication delay, unbounded message buffers and homogeneous processors. Based on pre-sampled activities and non-preemptive queue, Nicol [15] analyzed the performance of a conservative parallel simulation protocol using time window to synchronize the parallel simulation, and a lower bound on the average number of events processed per window was determined.

Performance study without considering the details of synchronization protocols reveals the inherent parallelism of a simulated problem. The parallelism index can be used to assess the suitability of applying parallelism simulation, and to estimate the overhead of the parallel implementation after it is implemented. In addition, performance evaluation at the modeling stage can be relaxed from the constraints imposed by synchronization protocols and execution platforms, and thus is easier to be carried out, especially using analytic methods.

Lin [13] proposed parallelism analyzers for parallel simulation. The parallelism analyzers for three different event-scheduling (process scheduling) policies in parallel simulation are based on critical path analysis (CPA). These CPA algorithms can be integrated with sequential simulation programs and the optimal parallel simulation time (i.e. critical time of the corresponding parallel simulation) can be com-

puted. However, the parallelism analysis of a simulation relies completely on the implementation of a sequential version of the simulation.

As alternatives, analytic methods are often used to evaluate the performance of parallel simulation. Wagner et al. [22] proposed an analytic method to study the limitation and potentials of parallel simulation of queuing networks, and introduced logical process (LP) utilization as the metric of parallelism of a parallel simulation. The *normalized utilization* of an LP, which is obtained by setting the largest LP utilization to 1 and changing other LP utilizations proportionally, is used to determine the asymptotic upper bound of parallelism in a simulation model. Nevertheless, the causality effect of event executions has not been considered.

In this paper, we propose an analytic method for predicting simulation parallelism. In section 2, we define simulation parallelism. Section 3 describes the analytic method for determining simulation parallelism. Based on an *aggressive schedule ahead strategy* for events, we discuss the effect of causality for two main LP interconnection structures: merge and fork-and-merge. Using these two structures, we propose algorithms for computing the degree of simulation parallelism for both open and closed systems. In section 4, we validate the analytic model against critical path analysis. Section 5 gives the conclusions.

2 Parallelism of Simulation

We assume that

- for a given problem, the simulation model is composed of a network of LPs that communicate with others by sending time-stamped messages;
- any operational characteristics of the simulation model are theoretically simplified, i.e. implementation issues such as synchronization protocols and communication costs are not considered.

Each LP executes events received from other LPs or generated by itself, and schedules new events. The parallelism of a simulation can be defined as the number of simultaneous events that can be processed by all LPs at a given time of the simulation. As the degree of parallelism varies with time, it may be difficult to determine the parallelism at a given time except by taking measurements using a real simulator. At the modeling stage, the expected parallelism is studied, and is used to characterize the model performance. Based on the conventional definition for program parallelism, simulation parallelism (π) can be expressed as

$$\pi = \frac{T_1}{T_n} \tag{1}$$

where T_1 is the total time taken to execute a sequential simulation for the problem, and T_n is the total time required to execute the simulation model consisting of n LPs using n processing elements (PEs).

We define the parallelism in terms of the utilization of its PEs. For simplicity, these utilizations are called LP's utilizations. Thus the parallelism of the simulation is defined as the total utilizations of all LP's in the simulation, that is,

$$\pi = \sum_{\substack{\text{LP}_i \in \\ \text{the LP network}}} U_i \tag{2}$$

where U_i is the utilization of LP_i during the parallel simulation. The proof of equivalence between definitions (1) and (2) is shown in [21]. In next section, we discuss our analytic method for determining simulation parallelism.

3 The Analytic Method

An LP is more than an ordinary queueing service center. In parallel simulation, an LP can be blocked due to the causality of events. However, an LP keeps events in its future event list, and executes the event with the smallest timestamp. To model an LP as a queueing service center for executing events, we separate the blocking phenomenon. The blocking phenomenon is modeled as event occurrence delays outside the queueing service center. With this abstraction, event flow at a queueing service center and LP is similar, i.e. same performance.

Assume that an LP models a server in the real world. Based on the simulation parallelism defined by equation (2), an analytic model for simulation parallelism can be outlined as follows,

- LPs in the simulation model are modeled by queueing service centers for executing events, and the LP network is mapped onto a queueing service network;
- each queueing service center has equal service time for executing an event;
- workload for the queueing network is event flows in the LP network.

For simplicity, an LP refers to the corresponding queueing service center in the analytic model. The analysis for the above model can be carried out as follows,

- determine the causality effects on the event flows of an LP network;
- determine workload in terms of event flows for the analytic model with respect to the workload for the simulation model of the given problem;
- apply analytic methods for queueing systems to the analytic model with the workload, and then determine the performance measurements (i.e. utilizations for LPs).

The following terminologies are used in this paper. A *customer* denotes a temporary entity in the real world problem (i.e. simulated system). A *server* is a resource in the real world system, and it corresponds to an LP in the simulation. *Simulation time* models the time in the real-world problem. *Wall clock time* of simulation is the time for the simulation to run on an execution platform. A simulation is *steady* if the event flows of each LP in the simulation are balanced. *Event* and *message* are used interchangeably in this paper.

3.1 Event Scheduling Strategies

In the simulation of a queueing service system, the problem is modeled mainly by two types of events, i.e. a customer entering a server (arrival) or leaving a server (departure). The typical event scheduling policy used in sequential simulation is as follows: an arrival event at a idle server results in the scheduling of a departure event, otherwise the departure event that corresponds to the arrival event will be scheduled by another departure event preceding the customer. The executions of events is performed in non-decreasing order of time-stamps. However, this event scheduling strategy may not be suitable for parallel simulation because it imposes strict causal relation on events. For instance, the departure of a customer usually does not have causal relations with the arrivals of its successors, i.e. independent. Using the conventional event scheduling policy, the execution of the departure event may have to depend on the execution of an arrival of its successor because the time-stamp of the arrival event is smaller than that of the departure event.

Other event scheduling strategies have been proposed based on the causality relaxation for some physical processes in the simulation. We propose an event scheduling strategy called **aggressive schedule ahead strategy** that exploit lookahead information. In a First Come First Serve (FCFS) queueing network, when a customer schedules its departure event, an arrival event at its destination server can be generated. This allows the arrival and departure events belonging to the same customer to be run in parallel. This strategy makes use of the lookahead information to increase the degree of exploitable parallelism.

The departure time of the last customer is kept at the server (Server A in figure 1). Suppose customer N leaves from the server A (see figure 1, snapshot 2). The execution of a new arrival event (customer $N+1$) at server A can be used to determine the departure time of the customer and in turn the arrival of the customer to the next destination (server B). Thus two events can be scheduled when the arrival event is executed, and simulation progresses for both servers are driven by arrival events only. In addition, the flow of arrival events in the simulation resembles more closely the movements of the customer in the real-world system.

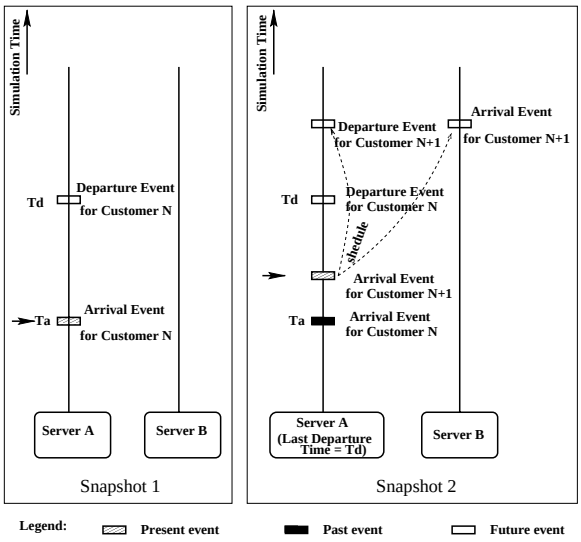

Figure 1. Event Scheduling with Lookahead

3.2 Causality Effects

If an event arrives in the "past" clock time of the receiver LP, the causality constraint is violated thus the simulation will produce wrong results. Therefore, appropriate synchronization protocols are used to avoid or to undo causality violations. The causality violation caused by normal events is called *direct causality violation*, and that caused by other causality violations is called *indirect causality violation*. As discussed in [21], direct causality violations may only occur at merge points of several message streams in LP networks.

We assume the aggressive schedule ahead strategy discussed. Thus causality violation occurs at the explicit message stream merge points (see figure 2). Indirect causality violations are not discussed in this paper. We further assume that service times in the real-world problems are exponentially distributed, and event execution times for LPs are exponentially distributed with the same mean time (homogeneous LPs). In figure 2, each LP simulates a FCFS queueing server. The customer arrival rates for server i is denoted by λ_i, and event arrival rates for LP$_i$ is X_i. We assume that inter-arrival times are exponentially distributed. We assume that the first k LPs are at steady states, and the real-world problem is also a steady problem. Thus the throughputs for server i ($i = 1 \ldots k$) are λ_i ($i = 1 \ldots k$), and throughputs for LP$_i$($i = 1 \ldots k$) are X_i($i = 1 \ldots k$). The server $k+1$ has a customer arrival rate of $\lambda_{k+1} = \sum_{i=1}^{k} \lambda_i$. Shorey and Kumar [17] presented a method for computing the throughput for LP$_{k+1}$, and showed that configuration shown in figure 2 is not steady without careful control on the events arrival rates for LP$_i$($i = 1 \ldots k$).

Theorem 1 (Shorey & Kumar [17]). *The throughput*

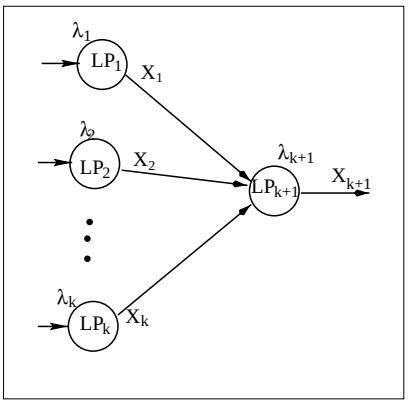

Figure 2. An LP Network with a Merge Point

for the LP_{k+1} in the simulation model depicted in figure 2 without causality violation is

$$X_{k+1} = \min_{j=1}^{k}\{\frac{X_j}{\lambda_j}\} \times \lambda_{k+1} \tag{3}$$

The proof for Theorem 1 is given in [17]. We also have the following relation,

$$X_{k+1} \leq \sum_{i=1}^{k} X_i \tag{4}$$

However, when the simulation is at steady state, message flows in the simulation are balanced, that is, the equality in (4) is true. Therefore, the relationship among X_i is redefined.

Corollary 2 *For a simulation at steady state and considering the effect of causality such as the LP network as shown in figure 2, the relations among X's and λ's are*

$$X_{k+1} = \sum_{i=1}^{k} X_i \tag{5}$$

and

$$\frac{X_1}{\lambda_1} = \frac{X_2}{\lambda_2} = \ldots = \frac{X_k}{\lambda_k} = \frac{X_{k+1}}{\lambda_{k+1}} \tag{6}$$

Proof: The first formula holds directly by the assumption of steady simulation. Now we show that the second one also holds.

Let $q_i = \frac{X_i}{\lambda_i}$ for $i = 1 \ldots k$, and $q^* = min\{q_i\}$. From Theorem 1, we have

$$X_{i+1} = q^* \times \lambda_{k+1} \tag{7}$$

and through (6) and (7), we have

$$q^*\lambda_{k+1} = \sum_{i=1}^{k} q_i\lambda_i$$

Since $\lambda_{k+1} = \sum_{i=1}^{k} \lambda_i$, we have,

$$\sum_{i=1}^{k} \lambda_i(q^* - q_i) = 0$$

Since $\lambda_i > 0 (i = 1 \ldots k)$, and $q^* \leq q_i$ $(i = 1 \ldots k)$, we have $q^* = q_i (i = 1 \ldots k)$. Thus $X_i/\lambda_i = q^*$ $(i = 1 \ldots k + 1)$ and this justifies the second formula in the corollary.

Corollary 2 helps to determine the causality effects for simulation with several external event generators. By adjusting the event generating rates for these generators according to the causality effects, the characteristics of workloads for the simulation could be partially determined.

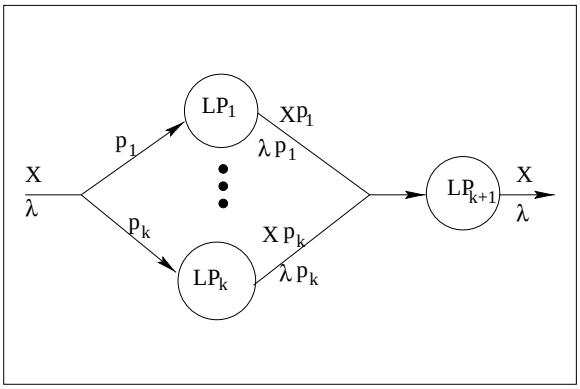

Figure 3. Fork and Merge LP Network

A simple "fork-and-merge structure" shown in figure 3 is a special case of the merge structure. Let p_i denote the routing probability for the route i where LP_i resides. From Theorem 1, this structure is a steady structure by its nature. Nevertheless, such a structure still could impacts the performance of parallel simulation. The causality effects of the fork-and-merge structure will be characterized by *the number of lost events* in the structure.

We first explain the lost events in the fork-and-merge structure. The customer arrival rates in the real world problem are denoted by λ's, and the event arrival rates in the simulation are denoted by X's. Let r_i denote the response time for server i. Let $ts_f(c)$ denote the time-stamp of the event corresponding to the arrival of a customer c at the fork point, and $ts_m(c)$ denote the time-stamp of the same customer's arrival at the merge point. If the customer takes the route i between the fork and merge points, we have $\Delta ts(c) = ts_m(c) - ts_f(c) = r_i$.

Suppose that there are two customers (c_1 and c_2) traveling through the fork-and-merge structure simultaneously during a simulation run, and they take different routes (assume that there are only two routes in the structure), where $\Delta ts(c_1) = r_1$ and $\Delta ts(c_2) = r_2$ and $r_1 > r_2$. We assume that these two customers arrive at the merge LP at the same

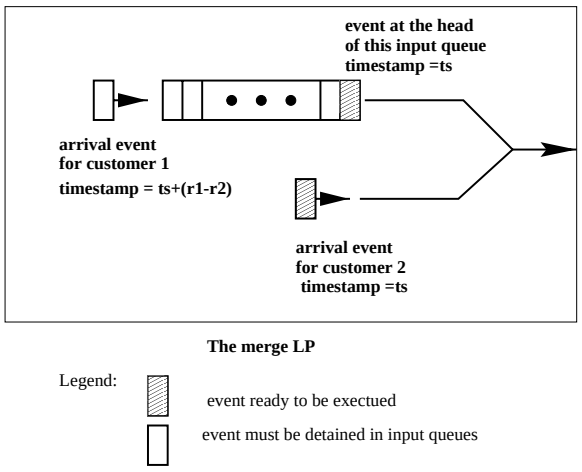

Figure 4. Simultaneous Arrival Events

time. Thus, at the merge LP, the event corresponding to the customer with larger Δts will have a larger time-stamp. It is not sure whether the event can be executed, even if it is the first event in the event list of the merge LP. Thus this event is withheld at the LP until it is safe to be executed. We refer to such an event as a *lost event*, since it seems that the event is lost temporarily.

However, if there are more than one lost event from the same route at the merge LP, it is possible that the first of these events can be executed, i.e. when its time-stamp is the smallest among all events on the LP (since its time-stamp is not greater than that of the arrival event for customer c_2 that is coming from the other route, see figure 4).

Suppose the merge LP has one dedicated queue for each route (or input channel), and the queue is referred to as **input queue**. When a new event arrives at one of these queues, the first event in the queue could be executed because incoming events will not have smaller time-stamps. However, the number of events withheld in the queue remains fixed during that period of simulation. Therefore, the total number of lost events will remain constant at steady state. Furthermore, the difference of wall clock times for a customer to travel in the simulation from the fork point to the merge point by taking different routes will affect the number of the lost events.

To estimate the number of the lost events, it is necessary to compare the advancement of simulation times on different routes before the merge point. The overall response time for servers along the route i in the fork-and-merge structure is denoted by γ_i, and the smallest among all routes is denoted by γ_{base}. The route with γ_{base} is referred to as **base route**. The total *wall clock time* for a customer to travel from the fork point to merge point using route i is denoted by R_i, and R_{base} is that of the *base route*. Let X_{base} and λ_{base} denote the event throughput in the simulation and customer throughput in the simulated problem along the *base route* respectively. The difference in simulation time (or event time-stamp advancement) between the route i and the base route is denoted by J_i. Let

$$H_i = \gamma_i - \gamma_{base} \quad (8)$$

$$I_i = (R_{base} - R_i) X_{base} \quad (9)$$

$$J_i = H_i + \frac{I_i}{\lambda_{base}} \quad (10)$$

$$J' = min\{J_i\}, \text{ (for all routes)} \quad (11)$$

$$L_i = J_i - J' \quad (12)$$

where H_i is the difference of server response times between route i and the *base route*, I_i is the relative speed for a customer to travel between the fork point and merge point. It can be scaled by the simulation time increment rate of the base route (i.e. $1/\lambda_{base}$) to reflect the difference of simulation time. Therefore the overall difference of simulation time at the merge point between route i and the base route can be determined by formula (10). L_i is used to avoid the occurrence of negative numbers. The input queue length for route i at the merge LP is denoted by G_i, and

$$G_i = L_i \lambda_i \quad (13)$$

Let the total number of lost events at the merge point be denoted by G. We have

$$G = \sum G_i, \text{ (for all routes in the fork-merge pair)} \quad (14)$$

In the simulation of open systems, the population of simultaneous events is determined mainly by the event generators at steady state. Therefore the number of events lost in the merge points will not affect the workload characteristics. It means that causality effects in the fork-and-merge structures can be neglected in the parallelism analysis for the simulation of open systems. This will be validated by the experiments in the next section.

For closed systems, the population of simultaneous events for the simulation is fixed. The number of lost events caused by fork-and-merge structures affects the parallelism of the simulation. In the following section, we propose two algorithms to estimate the simulation parallelism with causality effects.

3.3 The Method for Determining Parallelism

Algorithm for Open Systems

- **Step 1.** Analyze the problem.
  Assume balanced job flows in the problem. Apply operational laws and queueing theory [9] to the problem to obtain the throughputs for all servers in the system. Throughput for server i is λ_i.

- **Step 2.** Find the maximum among λ's.
  Let
  $$\lambda^* = max\lambda_i \text{ (for all servers)}$$

- **Step 3.** Compute the utilization for LPs.
  Let U_i be the utilization of LP$_i$, and
  $$U_i = \frac{\lambda_i}{\lambda^*} \quad (15)$$

- **Step 4.** Determine the degree of simulation parallelism (π).
  $$\pi = \sum_{LP_i \in \{LPs\}} U_i$$

Remarks:
Step 3 is based on the following reasoning. From Corollary 2, we have
$$X_i = \lambda_i q \quad (16)$$
for all LP$_i$ in the LP network, where X_i is the event throughput for LP$_i$, and q is a constant. Let X^* denotes the maximum of all these event throughputs for LPs, and λ^* is the corresponding customer throughput in the problem. Clearly, λ^* is the maximum among all λ's. Suppose each LP takes the same mean time (denoted by $\bar{t}$) to execute an event. The utilization of LP$_i$ during the simulation is defined as,
$$U_i = X_i \bar{t} \quad (17)$$
The utilization for the LP that has the maximal event throughput X^* is denoted by U^*, and it is the maximum among all U's for LPs. Since the system to be simulated is open, we can assume that the customer generators will produce sufficient customers to the simulator. Thus the U^* may reach 1. By setting $U^* = 1$, we have
$$\bar{t} = \frac{1}{X^*} \quad (18)$$
Substituting (16) and (18) to (17), we have the equation converge to formula (15).

Algorithm for Closed Systems

- **Step 1**. Analyze the problem.
  Use Mean Value Analysis(MVA) [9] to analyze the queueing characteristics of the problem, i.e response time, utilization, queue length, and throughput for each server.

- **Step 2.** Find fork-and-merge structures in the LP network and determine the number of lost events in the simulation. Set the tolerance ϵ, $\pi_0 = 0$, $G_{all} = 0$.

- **Step 3.** Determine the effective simultaneous event population for the simulation.
  With aggressive schedule ahead strategy, the overall population of simultaneous events in the simulation is equal to the customer population in the real world problem, which is denoted by N. The effective simultaneous event population, denoted by N^{eff}, is derived as follows,
  $$N^{eff} = N - G_{all}$$

- **Step 4.** Analyze the LP network with the effective event population as its workload.
  Apply N^{eff} to Approximate Mean Value Analysis [9] on the LP network by assigning each LP with the same event execution time $\bar{t}$, and compute the utilization, event throughput and event response time for each LP.

- **Step 5.** Obtain the simulation parallelism.
  Sum up utilizations for LPs in the last step to get the estimation for the simulation parallelism π.

- **Step 6.** Condition for termination.
  If $||\pi - \pi_0|| \leq \epsilon$ stop,
  else
  $\pi_0 = \pi$, and
  for {every fork-merge pair} do
  { Compute G by applying (8) – (14)}
  Sum up all G's for all merge points to obtain G_{all}.
  Goto Step 3.

Remarks:
Iterations in the above algorithm are necessary because I_i in equation (9) cannot be determined from Step 1. Thus, in the algorithm, initial values for I_i's are set by analyzing the LP network without taking the causality effects into account. In the following iterations, the I_i's are adjusted until the condition for termination is met. We use the Approximate MVA in step 4 because of that N^{eff} may not be an integer.

4 Experiments and Validation

To validate the analytic method, we implemented an analyzer for determining simulation parallelism using critical path analysis(CPA). The analyzer implements CPA algorithm [13] and our aggressive schedule ahead strategy.

Two sets of experiments were designed. An open queueing system as depicted in figure 5, and a closed queueing system as depicted in figure 6.

The configuration for the open system is:
inter-arrival time to server 1,2: {60, 60}
service time for server 1-5 : {2, 10, 20, 20, 2}
routing probabilities at server 1: {0.7, 0.3}
routing probabilities at server 2: {0.7, 0.3}
routing probabilities at server 3: {0.6, 0.4}
Simulation parallelism measured using CPA and computed using the analytic model (ALGO) is 3.675 and 3.685 respec-

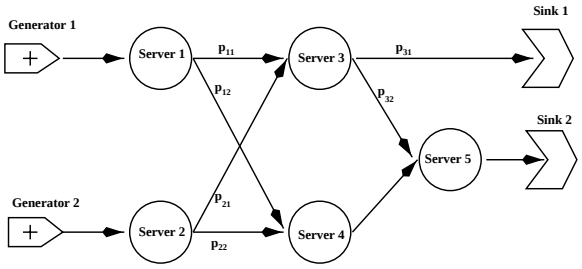

Figure 5. An Open Queueing System

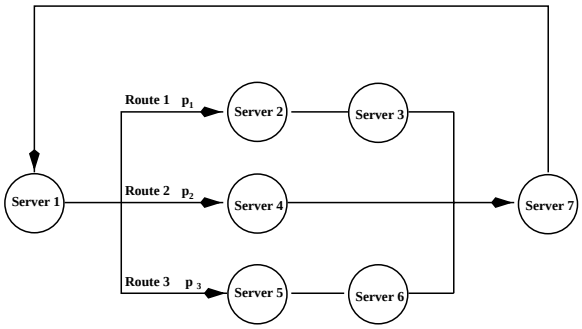

Figure 6. A Closed Queueing System

tively. The relative error δ[1] is 0.2%. The largest relative errors for varying the routing probability and the inter-arrival time at server 1 (see figure 7) are 2.1% and 2.9% respectively.

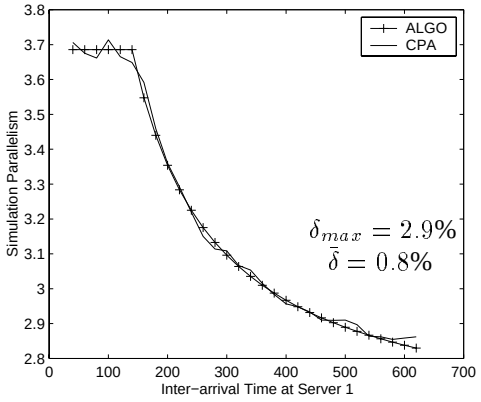

Figure 7. Open System - Varying Customer Inter-arrival Times to Server 1

The closed queueing system has the following configuration:

service time for server 1-7 :{5, 15, 20, 60, 5, 10, 25}
routing probabilities for routes 1,2,3:{0.3333, 0.3333, 0.3334}
population size of customer: 30

Simulation parallelism is 3.486 and 3.501 using CPA and analytic model respectively, and with relative error of 0.4%. We conducted three sensitively experiments by varying the service time at server 7 (see figure 8), the number of customers, and the routing probability for route 2. The largest relative errors are 18.8%, 9.8% and 18.2% respectively. The average relative errors in these experiments are 4.8%, 2.9% and 3.7% respectively. In summary, the analytic model deviates from CPA by less than 5% on average.

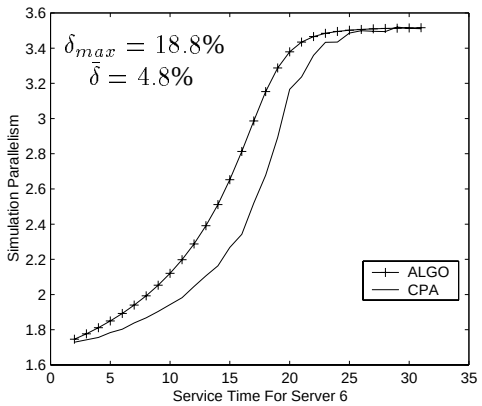

Figure 8. Closed System - Varying Service Time of Server 7

In the open systems, the analytic method considers the causality effects of merge structures, i.e. formula (6), and ignores those of fork-and-merge structures. In the closed systems, the analytic method considers the causality effects of fork-and-merge structures. The causality effects are characterized by the number of *lost events*. In estimating the number of lost events, we assumed the independence of input event stream and output event stream. This assumption may be violated in the case of closed systems and affects the accuracy of estimating the size of the lost events. This is revealed in the above experiments where there exist several configuration settings whose relative errors are nearly 18.8%. Nevertheless, when the causality effects of the fork-and-merge structures are ignored, the experiments with those settings show that the relative errors exceed 70%. Thus our model for the causality effects of fork-and-merge structure is effective.

5 Conclusions

In this paper, we proposed an analytic method for estimating the simulation parallelism of FCFS queueing systems. The method incorporates our proposed *aggressive schedule ahead strategy* for scheduling events, and considers the effects of causality in the simulation. We discussed how to deal with causality effects for LPs with

[1]The relative error is defined as $\delta = ||\frac{\pi_{ALGO} - \pi_{CPA}}{\pi_{CPA}}||$

a merge and a fork-and-merge structures, and present the algorithms to analyze open and closed systems. We validated the proposed analytic method by comparing the parallelism obtained with that from critical path analysis. Experimental results show that the analytic method has a higher accuracy in open systems (the largest relative error is 2.9%), while the accuracy of the method for closed systems varies (relative error as high as 18.8%). The higher relative error for closed systems is due to our assumption of the independence of input event and output event streams. Nevertheless, in the case of analytic method without considering the causality effects (such as in the normalized utilization approach proposed by [22]), the relative error is as high as 70%. Thus, our proposed analytic method is comparatively more accurate in predicting the simulation parallelism of closed systems. Work is in progress to improve the accuracy of the model for generalized closed systems.

Acknowledgement

This research is supported by a grant form the Ministry of Education and PSA Corporation under grant RP960715.

References

[1] I.F. Akyildiz, L. Chen, S. Das, R. M. Fujimoto, and R. Serfozo. The Effect of Memory Capacity on Time Warp Performance. *Journal of Parallel and Distributed Computing*, 18(4):411–422, August 1993.

[2] R. Bagrodia, V. Jha, and J. Waldorf. The Maisie Environment for Parallel Simulation. In *Proceedings of the 27th Annual Simulation Symposium*, 1994.

[3] O. Berry and D. Jefferson. Critical Path Analysis of Distributed Simulation. In *Proceedings of the 1985 SCS Conference on Distributed Simulation*, pages 57–60, 1985.

[4] K.M. Chandy and J. Misra. Asynchronous Distributed Simulation via a Sequence of Parallel Computations. *Communication of ACM*, 24(11):198–206, 1981.

[5] A. Ferscha, J. Johnson, and S.J. Turner. Early Performance Prediction of Parallel Simulation Protocols. In *Proceedings of World Conference on Systems Simulation*, pages 282–287, September 1997.

[6] R. M. Fujimoto. Time Warp on a Shared Memory Multiprocessor. *Transactions of the Society for Computer Simulation*, 6(3):211–239, Jul. 1989.

[7] R.M. Fujimoto. Parallel Discrete Event Simulation. *Commu. of ACM*, 33(10):30–53, 1990.

[8] A. Gupta, I.F. Akyildiz, and R.M. Fujimoto. Performance Analysis of Time Warp With Multiple Homegeneous Processors. *IEEE Transactions on Software Engineering*, 17(10):1013–1027, October 1991.

[9] R. Jain. *The Art of Computer Systems Performance Analysis – Techniques for Experimental Design, Measurement, Simulation, and Modeling*. John Wiley and Sons, Inc., 1991.

[10] D. Jefferson and P. Reiher. Supercritical Speedup. In *24th Annal Simulation Symposium*, pages 159–168, 1991.

[11] D.R. Jefferson. Virtual Time. *ACM Transaction on Programming Languages and Systems*, 7(3):404–425, 1985.

[12] C-C Lim and W Cai. A Parallelism Analyzer for a Conservative Super-Step Simulation Protocol. In *Proceedings of the 32nd Hawii International Conference on System Sciences*, pages 1–10, 1999.

[13] Y-B. Lin. Parallelism Analyzers for Parallel Discrete Event Simulation. *ACM Transactions on Modeling and Computer Simulation*, 2(3):239–264, 1992.

[14] D. E. Martin, T. J. McBrayer, and P. A. Wilsey. warped: A Time Warp Simulation Kernel for Analysis and Application Development. In *29th Hawaii International Conference on System Sciences (HICSS-29)*, pages 383–386, Jan. 1996.

[15] D.M. Nicol. The Cost of Conservative Synchronous in Parallel Discrete Event Simulation. *Journal of ACM*, 40(2):304–333, 1993.

[16] P.L. Reiher, S. Fujimoto, R.M. abd Bellenot, and D.R. Jefferson. cancellation Strategies in Optimistic Execution Systems. In *Proceedings of the SCS Multiconference on Distributed Simulation 22*, pages 112–121, 1988.

[17] R Shorey, A Kumar, and K.M. Rege. Instability and Performance Limits of Distributed Simulators of Feedforward Queueing Networks. *ACM TOMACS*, 7(2):210–238, 1997.

[18] L.M. Sokol, D.P. Briscoe, and A.P. Wieland. MTW: A Strategy For Scheduling Discrete Event Simulation Events For Concurrent Execution. In *Proceedings of the SCS Multiconference on Distributed Simulation 19*, pages 34–42, 1988.

[19] W.K Su and C.L. Seitz. Variants of the Chandy-Misra-Bryant Distributed Discrete-Event Simulation Algorithm. In *Proceedings of the SCS Multiconference on Distributed Simulation 21*, pages 38–43, 1989.

[20] Y.M. Teo, S.C. Tay, and S.T. Kong. Structured Parallel Simulation and Programming. In *Proceedings of the 31st Annual Simulation Symposium*, pages 135–142. IEEE Computer Society Press, April 1998.

[21] Y.M. Teo, H. Wang, and S.C. Tay. A Framework for analyzing Parallel Simulation Perforamance. In *Proceedings of the 32nd Annual Simulation Symposium*, San Diego, U.S., April 1999. IEEE Computer Society.

[22] D.B. Wagner and E.D. Lazowska. Parallel Simulation of Queuing Networks : Limitations and Potentials. *ACM Performance Evaluation Review*, 17(1):146–155, 1989.

[23] Y-C Wong, S-Y Hwang, and Y-B Lin. A Paralleism Analyzer for Conservative Parallel Simulation. *IEEE Transaction On Parallel And Distributed Systems*, 6(6):628–638, June 1995.