Approximate Time-Parallel Simulation of Queueing Systems with Losses

Jain J. Wang and Marc Abrams

TR 92-08

Department of Computer Science
Virginia Polytechnic Institute and State University
Blacksburg, Virginia 24061

March 19, 1992

Approximate Time-Parallel Simulation of Queueing Systems with Losses

Jain J. Wang and Marc Abrams

Department of Computer Science
Virginia Tech, Blacksburg, VA 24061-0106

TR 92-08

March 19, 1992

Abstract

This paper presents a guideline of a partial state matching approach for time-parallel simulation. Two algorithms using this approach to simulate FCFS G/G/1/K and G/D/1/K queues in which arriving customers that find the queue full are lost are proposed. Experiments with M/M/1/K and M/D/1/K models show that the performance of the algorithms in terms of convergence speed and accuracy is good in general cases. The worst performance of the algorithms occur when traffic intensity approaches one. An argument is made to explain this phenomenon.

1 Introduction

Computer simulation is a process of computing a *sample path*, or *trajectory*, of a target simulation model, consisting of the time evolution of the states of the model over a period of simulation time. The state space and time domain of a simulation model form a *space-time* region [1]. The state space is typically defined as the set of components, processes (in the process-oriented world view), or state variables comprising a simulation model. For parallel distributed simulations, parallelism can be obtained through decomposing the state space (space-parallel) or the time domain (time-parallel). Space decomposition will be briefly reviewed in the following sections, while this paper will concentrate on time-parallel simulations.

Space-Parallel Simulation

Many space-parallel algorithms have been proposed [2]. These algorithms decompose the target simulation model into a number of components based on its state space.

Each component is modeled by a *logical process*. Logical processes communicate with each other by sending messages [5] or through sharing some state variables [6]. In this approach, speed-up is bounded by the number of logical processes. For example, in a queueing network model, each queue is usually modeled by a logical process. When the network consists of fewer queues than available processors, speed-up is limited to the number of queues. In addition, speed-up is further limited by the overhead involved in coordinating multiple processors and by the structural dependencies that exist within some models [12].

Time-Parallel Simulation

Time-parallel approaches exploit parallelism by partitioning the trajectory into a number of *sub-trajectories*, or *batches*, along the time domain. Each batch is assigned to one or more processors that simulate the batch independently (and possibly asynchronously) of other batches. Thus, multiple processors simulate the system at different points in simulation time concurrently. The simulation proceeds in an iterative manner. Initially, each process is assigned a *guessed* initial state. After simulation of one or more batches complete, the initial states of the completed batches are updated based on the previous results and then the batches with updated initial states are again simulated. The process is repeated until no changes occur in any initial states, at which point the simulation *converges* or reaches a *fix* point. In this paper, the terms *estimated trajectory* and *true trajectory* are used to refer to an intermediate trajectory before convergence and the final trajectory at convergence, respectively.

The efficiency of time-parallel simulations rests on the ability to select the initial states of each batch to minimize the number of iterations required for convergence. A poor selection can make a time-parallel simulation take longer to execute than a sequential simulation. For many simulation models, however, predicting future system states is difficult.

In this paper we study a *partial state matching* approach for time-parallel simulation. Simulations using this approach are *approximate* in the sense that they may not generate the same results as sequential simulation runs using the same sequence of random numbers. The approach sacrifices simulation accuracy in exchange for higher parallelism. In this paper, two time-parallel algorithms using the partial state matching approach are introduced. One is for FCFS G/G/1/K queues and the other is for FCFS G/D/1/K queues with losses. In both cases, arriving customers that find the queue full are lost rather than block.

Our motivation for studying finite storage queues with losses is that they model switching nodes in packet-switching data communication networks which lose packets due to buffer overflow. If the packets are served in order of their arriving times, a node of such a packet switching network can be modeled by an FCFS G/G/1/K queue. In fact, combining the algorithms presented here with the time-parallel simulation method for other queueing systems devised by Greenberg, Lubachevsky, and Mitrani

[3] could allow simulation of datagram service in data communication networks with far more nodes than is possible using sequential simulation or space-parallel simulation.

The rest of this paper is organized as follows. In section 2, we review two related time-parallel simulation algorithms. Section 3 introduces the proposed partial state matching simulation. Two time-parallel algorithms using the partial state matching approach for FCFS G/G/1/K and G/D/1/K queues with losses are presented in section 4 an 5. Conclusions are given in section 6.

2 Related Work in Time-Parallel Simulation

For some time-parallel algorithms, correct initial states can be *predicted* in a deterministic number of iterations. Two examples are Greenberg, Lubachevsky, and Mitrani's (GLM) parallel prefix algorithm [3] and Lin and Lazowska's regeneration state matching algorithm [10]. For others, convergence time may be non-deterministic [9]. In this section, we review the GLM and Lin and Lazowska algorithms. The GLM algorithm is discussed with more detail because it forms the basis of one of our partial state matching simulations.

2.1 The GLM Algorithm

The GLM algorithm provides an efficient way to simulate a class of queueing network models which can be expressed as recurrence relations and transformed into a parallel prefix problem. Let D be a domain and $\circ$ be any associative operator on that domain. Let N be any positive integer. A prefix problem is to compute each of the products $a_0 \circ a_1 \circ \ldots \circ a_k, 1 \leq k \leq N$ [4,8]. We first review how the GLM algorithm is applied to an FCFS G/G/1/∞ model.

For an FCFS G/G/1/∞ model, let A_i and D_i denote the arrival time and the departure time, respectively, of job i for i=1,2,...,N. Let α_i denote the interarrival time between job i and job $i+1$, and δ_i denote the service time of job i. If A_1, the arrival time of the first job, is given, the arrival and departure time sequences $(A_1, A_2, \ldots, A_N)$ and $(D_1, D_2, \ldots, D_N)$ are the solution of the following recurrence relations [11]:

$$A_i = \; A_{i-1} + \alpha_{i-1} \quad 1 < i \leq N, \tag{1}$$

$$D_i = \begin{cases} A_i + \alpha_i & i = 1, \\ max(D_{i-1}, A_i) + \delta_i & 1 < i \leq N. \end{cases} \tag{2}$$

It is assumed that job interarrival and service times are random variables whose values can be pre-sampled; therefore sequences $(\alpha_1, \alpha_2, \ldots, \alpha_N)$ and $(\delta_1, \delta_2, \ldots, \delta_N)$ are known in advance.

Define an *event* to be a job arrival or a departure. The *event sequence* is the result of merging sequences A_i and D_i in time order and is denoted $(E_1, E_2, \ldots, E_{2N})$. The *queue length trajectory* is the sequence $(L_1, L_2, \ldots, L_{2N})$, where L_j , for $1 \leq j \leq 2N$, is the queue length immediately after event E_j. Sequence (L_j) is the solution of the following recurrence relation:

$$L_j = \begin{cases} 0 & j = 0, \\ L_{j-1} + 1 & 1 \leq j \leq 2N \text{ and } \mathrm{E_j} \text{ is an arrival,} \\ L_{j-1} - 1 & 1 \leq j \leq 2N \text{ and } \mathrm{E_j} \text{ is a departure.} \end{cases} \qquad (3)$$

Therefore, the computation of sequence (L_j) is again a prefix problem. To solve the prefix problem in parallel, the GLM algorithm performs the following steps:

1. Partition the sequence of $a_0, a_1, \ldots a_N$ into P *batches* of same length, where P is the number of processors available. Assume that P divides N evenly. Then batch i contains $a_{((i-1)*N/P)+1}, \ldots, a_{i*N/P}$.

2. Assume that there exist an $\iota \in D$, such that $\iota \circ a_i = a_i$, for $1 \leq i \leq N$. Let ι be the initial state of all batches and start the computation. If f_i denotes the final state of batch i, then $f_i = a_{((i-1)*N/P)+1} \circ \ldots \circ a_{i*N/P}$.

3. Compute the correct initial states. Since operator $\circ$ is associative, the correct initial state of batch i, denoted by b_i, is $f_1 \circ \ldots \circ f_{i-1}$, for $1 < i \leq P$ and $b_1 = \iota$.

4. Correct the batches based on the updated initial states. For any batch i, the product of sequence $a_{((i-1)*N/P)+1}, \ldots, a_k$, denoted by s_k, should be corrected to $b_i \circ s_k$ for all $((i-1) * N/P) + 1 \leq k \leq i * N/P$.

The GLM algorithm requires $O(N/P + logN + logP)$ of time to compute a queue length trajectory with N jobs [3].

2.2 Lin and Lazowska's State Matching Algorithm

Lin and Lazowska's algorithm [10] partitions the time domain at a set of *regeneration points*. Regeneration points are time points such that the system behavior between a pair of two successive regeneration points is a statistical replica of the process between any other such pair of regeneration points. For example, a regeneration point occurs in a G/G/1 model when the server becomes idle. A *batch* is a sub-trajectory between two neighboring regeneration points. Without knowing the simulation time of each regeneration point, each batch can be simulated independently with its initial simulation times set to zero. Following simulation of each batch, the simulation time of each batch is corrected by simply adding to the time of each event the final simulation time of the preceding batch. For example, consider two processors, p_1 and p_2, that both simulate a batch with initial simulation time, zero. If p_1 finishes first and the final simulation time of its batch is t, then t is added to the time of each event in the batch computed by p_2, and p_1 can initiate another batch whose correct initial time can be decided when p_2 finishes.

3 Partial State Matching Simulation

The GLM algorithm and Lin and Lazowska's regeneration state matching approach both provide a way to correct initial states quickly. However, to apply the GLM algorithm, recurrence relations solvable as a prefix problem must be identified. For Lin and Lazowska's algorithm, regeneration points must exist in the trajectory. Simulation models may not fit these assumptions. The partial state matching simulation proposed in this paper extends the class of models to which time-parallel simulation can be applied.

3.1 Terminology

Before discussing the partial state matching simulation, some definitions are required. Let $S = \{v_i | i = 1, \ldots, M\}$ be the state space of a simulation model which contains M state variables. Let $v_{i,j}(t)$ denote the value of state variable v_i at simulation time t after j iterations (for $j = 0, 1, \ldots$) where $0 \leq t \leq \tau$ for some $\tau > 0$. Then the *system state* at time t after iteration j and before iteration $j+1$ is represented by an M-tuple $S_j(t) = (v_{1,j}(t), v_{2,j}(t), ... v_{M,j}(t))$. For discrete event simulation, the system state changes only in some discrete points $t_0, t_1, \ldots$. The *trajectory* of the simulation after j iterations is represented by the sequence: $(S_j(t_0), \ldots, S_j(t_N))$, where $t_0 = 0$ and $t_N \leq \tau < t_{N+1}$. Let P be a positive integer. The interval $[0,\tau]$ is partitioned into P intervals: $[b_0, b_1), [b_1, b_2), \ldots, [b_{P-1}, b_P]$, where $b_0 = 0$ and $b_P = \tau$. The *sub-trajectory* in the interval $[b_{i-1}, b_i)$ for $0 < i < P$, and $[b_{i-1}, b_i]$ for $i = P$ is referred to as batch i. The initial state of batch $i+1$ at iteration j is $S_{j-1}(b_i)$ for $j \geq 1$. The state of batch $i+1$ has *converged* after j iterations if $S_{j-1}(b_i) = S_j(b_i)$. The simulation is said to be *exactly completed* after T iterations for $T \geq 0$, if $S_{T-1}(b_i) = S_T(b_i)$, for all $1 \leq i \leq P$. Let T_{min} be the smallest T that the simulation exactly completes after T iterations. Then the sequence $(S_{T_{min}}(t_0), \ldots, S_{T_{min}}(t_N))$ is a *true trajectory* and $(S_j(t_0), \ldots, S_j(t_N))$ for $0 \leq j < T_{min}$ is an *estimated trajectory*. In the worst case, a simulation requires P iterations to complete. If all state variables $v_1, \ldots, v_M$ are recomputed at all iterations, each iteration requires a constant amount of wall clock time to execute. To gain any speed-up, T_{min} must be smaller than P.

For a simulation model, if there exists a set $U_j \in S$, such that $\forall v_i \in U_j$, $v_{i,j}(t) = v_{i,j'}(t)$ for all $j' \geq j$ and $t \geq 0$, the simulation is said to have *partially converged* on subset U_j after j iterations. We call $\{S - U_j\}$ the *unmatched set* of the simulation after j iterations, where $-$ is the set difference operator. The number of state variables in the unmatched set is called the *degree of freedom* of the simulation. Therefore, the degree of freedom defines the number of states that must converge for the simulation to be exactly completed.

The idea of partial state matching simulation is to artificially *fix* some state variables in the unmatched set with some approximated values so that these state variables can be removed from the unmatched set. Therefore, the simulation converges on fewer variables. If these approximate values can be computed efficiently and are relatively close to the true values, the simulation may require less time to converge and still obtain accurate simulation results. In the following sections we discuss two examples of this partial state matching approach.

4 FCFS G/G/1/K Queues with Losses

In this section we discuss an algorithm using the partial state matching approach for an FCFS G/G/1/K model with losses in which the arriving jobs that find the queue full are lost. For a G/G/1/K model, we assume that job interarrival times and service times can be modeled by some independent, identically distributed random variables. Unless mentioned otherwise, in the rest of this paper a G/G/1/K queue refers to one with an FCFS queueing discipline.

Recall that the GLM method described in section 2 provides an efficient algorithm for the G/G/1/∞ model. However, it presents a problem when applying the GLM method to a G/G/1/K model with losses. Let L_i denote the queue length when job i arrives. If a job is lost, its departure time will be defined to be ∞. Also, let D'_i denote the departure time of the last job that is not lost before job i. Then we have the following relation:

$$D_i = \begin{cases} max(D'_i, A_i) + \delta_i & L_i < K, \\ \infty & L_i = K. \end{cases} \tag{4}$$

Job departure times depend on queue lengths. However, to compute queue lengths, knowledge about previous departures is necessary. Departure times and queue lengths form a circular dependency. The GLM algorithm is not directly applicable here.

A trajectory of the G/G/1/K model can be represented by a compound of a job arrival time sequence, a job departure time sequence, and a queue length sequence. Our first partial state matching algorithm consists of two phases. In the first phase, the buffer space constraint is relaxed. That is, a G/G/1/∞ model is simulated. If the queue length never exceeds K, the resulting trajectory is then correct and further computation is not required. Otherwise, a second phase is required in which the estimated trajectory computed in phase one is *transformed* into an approximate true trajectory. The degree of freedom of the second phase computation is two since the job arrival sequence is pre-determined in phase one. The unmatched set contains departure times and queue lengths. The proposed partial state matching simulations artificially fixes the departure times so that the state variable associated with departure times can be removed from the unmatched set. As a result, the simulation needs to match only on queue lengths and the degree of freedom of the simulation is reduced to one. The proposed algorithm is shown in Figure 1.

4.1 Algorithm

```
1. Simulate the G/G/1/∞ queue using the GLM algorithm
                                       {Phase 1 computation}

2. If queue_length at any simulation time > K then
                                       {Start phase 2 computation}
   begin
3.   batch_length = floor(2N/P)
4.   for all batch i do_par
5.      lowi = (i-1)*(batch_length)+1 {lowi defines the low-end boundary of batch i}
6.      highi = i*(batch_length)      {highi defines the high-end boundary of batch i}
7.      if(i=P)
8.             highi= 2N
9.      b(i,0) = min(queue_length[lowi-1],K)
                                       {Set the initial queue length for batch i}
10.  end_for

11.  j = 0                             {j is an iteration counter}
12.  repeat
13.      j = j + 1
14.      for all batch i do_par
15.         for m = lowi to highi do
16.            if(event[m] is an arrival event)
17.               queue_length[m] = min(queue_length[m-1]+1,K)
                                       {The current event is an arrival event}
                              {If queue_length[m]=queue_length[m-1], a loss occurs}
18.            else
19.               queue_length[m] = max(queue_length[m-1]-1,0)
                                       {The current event is a departure event}
                                       {If queue_length[m]=queue_length[m-1],}
                                         {this departure corresponds to a lost job}
20.            endif
21.         end_for
22.         b(i,j) = queue_length[(i-1)*batch_length)
                                       {Update the initial state for batch i}
23.      par_end

24.      Bj = (b(1,j),b(2,j), ... ,b(p,j))

25.  until(Bj = Bj-1)                  {Convergence check}
   end
```

Figure 1: The first partial state matching algorithm for the FCFS G/G/1/K model.

In phase one, the G/G/1/∞ model is simulated using the GLM algorithm of section 2.1 to produce a job arrival time sequence, an estimated job departure time sequence, and an estimated queue length sequence. Phase two computes an approximate true queue length sequence. Let N be the number of total arrivals. Then $2N$ events are simulated in phase one, since each job introduces an arrival and a departure event. Phase two partitions the trajectory into P batches. Each batch contains $\lfloor 2N/P \rfloor$ events, except the last one which receives whatever is left over (line 5 - line 8). The initial queue length of each batch is obtained from the corresponding estimated queue length resulted from phase one computation. If the corresponding estimated queue length is greater than K then K is used (line 9). If the current event is an arrival and the queue is not full then the queue length is increased by one. Otherwise, the job is lost and the queue length is not changed (line 17). If the current event is a departure and the queue is not empty then the queue length is decreased by one. Otherwise (the queue is empty), the departure event, which corresponds to a lost job, is ignored and the queue length is not changed (line 19). In phase two, the estimated departure time sequence computed in phase one is not re-computed. In other words, the estimated departure time sequence is used to compute an approximate true queue length sequence in phased two. The approximate true departure time sequence can be obtained by simply excluding those departures corresponding to some lost jobs from the estimated departure time sequence.

Some properties of the algorithm follow:

1. At any simulation time t, when an arrival event occurs in the G/G/1/∞ model, there is an arrival event in the corresponding G/G/1/K model at t. If a job arrival occurs when the G/G/1/K queue is full, the job is dropped.

2. At any simulation time t, when a departure event occurs in the G/G/1/∞ model, a departure event will also occur in the corresponding G/G/1/K model at t. If a job departure occurs when the G/G/1/K queue is empty, the departure is ignored.

3. At any iteration during the simulation, the queue length of the G/G/1/∞ model at any simulation time t is no less than the queue length of the G/G/1/K model. That is, the queue length of the G/G/1/K model as a function of time is a lower bound on the queue length of the G/G/1/∞ model.

Properties 1 and 2 correspond to line 17 and line 19 of the algorithm, respectively. Property 2 is an assumption made by the algorithm. Property 3 follows properties 1 and 2. Obviously, property 2 is a false assumption because after the first loss, the estimated trajectory and the true trajectory would no longer be synchronous on departure times. Nevertheless, the use of such an approximate departure time sequence is justified if phase two completes quickly and the results are close to the results of an exact simulation. The approximation technique used for the departure time sequence is explained below.

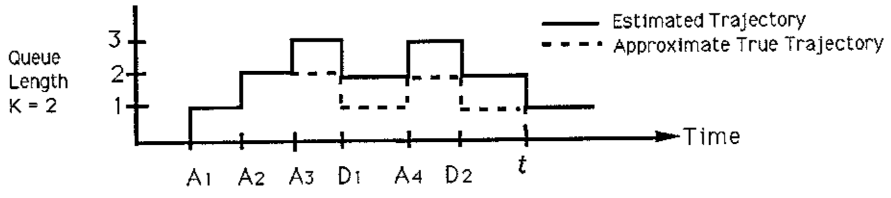

Figure 2: The departure event at t corresponds to job 3 in the G/G/1/∞ model, and job 4 in the G/G/1/K model because job 3 is lost in the G/G/1/K model. Thus, C(4)=3.

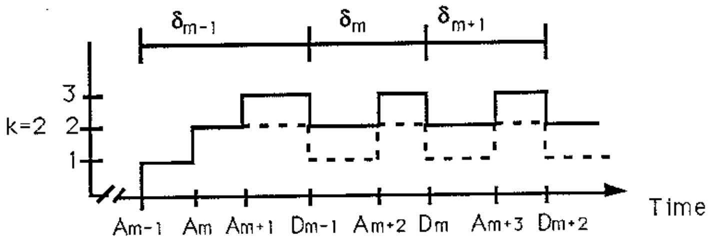

Figure 3: An example of case 1. Here, $f(m) = m + 2$. The service time of job $m + 1$, which is lost, is re-assigned to job $m + 2$. No approximation error is introduced in this case.

4.2 Approximation Technique

Let $f(i)$ denote the next job which enters the queue after job i (i.e. jobs $i+1, \ldots, f(i)-1$ are lost.). Also, let $C(i)$ denote the corresponding job in the G/G/1/∞ model that departs at the same time as job i in the G/G/1/K model. Figure 2 shows an example in which $C(4) = 3$. Now, assume that a prefix of m jobs of a true trajectory has been computed. To compute $D_{f(m)}$, there are four cases:

Case 1: When job $f(m)$ arrives, both the $G/G/1/\infty$ queue and the $G/G/1/K$ queue are not empty.

In this case (Figure 3), when job m departs, job $f(m)$ has already joined the queue. Then, $D_{f(m)}$ should be equal to $D_m + \delta_{f(m)}$. From property 3, we know that at time D_m, when job $C(m)$ departs in the G/G/1/∞ queue, the G/G/1/∞ queue must not be empty. That is, job $C(m) + 1$ must have entered the G/G/1/∞ queue by D_m. Thus, the algorithm will assign $D_m + \delta_{C(m)+1}$ to $D_{f(m)}$, as opposed to $D_m + \delta_{f(m)}$. That is, the service time of job $C(m)+1$ is *re-assigned* to job $f(m)$. Since it is assumed that the job service times are identical, independently distributed random variables, such re-association of service times will not affect the stochastic characteristics of the service time process. By so doing, re-computation of $D_{f(m)}$ is avoided.

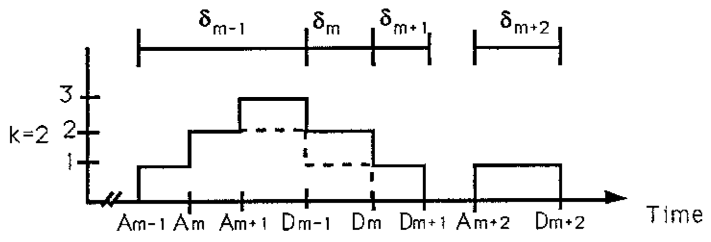

Figure 4: An example of case 2. Here, $f(m) = m+2$ and D_{m+2} is given as $A_{m+2}+\delta_{m+2}$. No approximation error is introduced in this case.

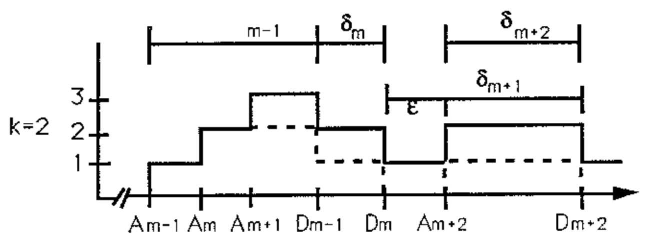

Figure 5: An example of case 3. Here, $f(m) = m + 2$. The approximate service time of job $m + 2$ is given as δ_{m+2}. An expected approximation error $\varepsilon = A_{f(m)} - D_m$ is introduced.

Case 2: When job $f(m)$ arrives, both the $G/G/1/\infty$ queue and the $G/G/1/K$ queue are empty.

In this case (Figure 4), job $f(m)$ job arrives at a point when the server of the G/G/1/∞ model is idle. Thus $C(m) = m$. The algorithm will assign $A_{C(m)} + \delta_{C(m)} = A_m + \delta_m$ to $D_{f(m)}$, which is correct.

Case 3: When job $f(m)$ arrives, the $G/G/1/K$ queue is empty but the $G/G/1/\infty$ queue is not empty.

In this case (Figure 5), the algorithm will assign $D_m + \delta_{m+1}$ to $D_f(m)$. However, the true value of $D_f(m)$ is $A_{f(m)} + \delta_{f(m)}$. The expected error of $D_f(m)$ in this case can be given by:

$$E(\varepsilon_{D_{f(m)}}) = E[(A_{f(m)} + \delta_{f(m)}) - (D_m) + \delta_{m+1})] = E[(A_{f(m)} - D_m)]. \qquad (5)$$

Case 4: When job $f(m)$ arrives, the $G/G/1/K$ queue is not empty but the $G/G/1/\infty$ queue is empty.

From property 3, we know that this case is impossible.

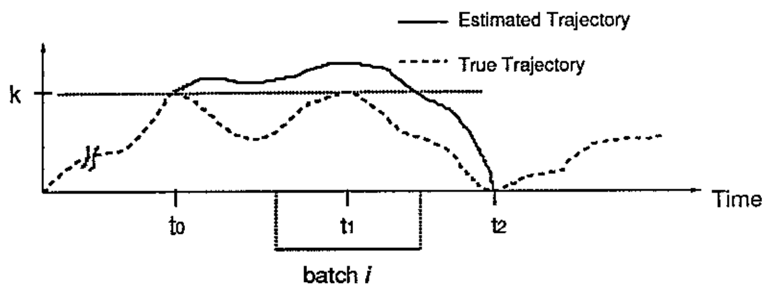

Figure 6: The first loss occurs at t_0. An E_F occurs at t_0 and t_1 and an E_E occurs at t_2. Sub-trajectories from t_0 to t_1 and from t_1 to t_2 are two propagation segments.

In these cases, case 3 causes an error hence makes the proposed algorithm approximate. Using this approximation technique, the simulation needs to converge only on queue lengths rather than on both departure times and queue lengths. The execution time required by phase two is at most $O(N)$ because correct initial batch states propagate at least one batch per iteration. Therefore, if the convergence takes T iterations, the execution time spent by phase two is $O[T * (N/P)]$, $0 \leq T \leq P$. This yields a total simulation time of $O[(N/P + logP + logN) + T * (N/P)]$, where the first term is the time required by phase one using the GLM algorithm.

4.3 Experiment Results and Analysis

An experiment of the algorithm described above is carried out for an M/M/1/K model. Table 1 shows some results of the experiment. In general, it takes only a few iterations to converge. Many iterations are required only when the *traffic intensity* (ratio of λ to μ) is close to 1 and the buffer size is large. We discuss this phenomenon next.

Recall that phase two of the algorithm is a process of transforming an initial estimated trajectory into a true trajectory. It is obvious that when there is no job being lost, an estimated trajectory is the same as a true trajectory. However, when losses do occur, the trajectory after the first loss has to be modified. This is illustrated in Figure 6. At t_0 where the G/G/1/K queue is full and a job is lost. A change in queue length at t_0 has to be made and the change needs to be *propagated* along the queue length trajectory. At time t_1 the queue becomes full again. By property 3, the queue length at time t_1 will be changed to K in the first iteration in phase two regardless of the initial queue length of batch i which contains t_1. Therefore, no matter how many iterations it takes for a change to propagate from t_0 to t_1, it is guaranteed that the propagation *segment* between t_1 to t_2 would start with a correct initial queue length, namely K. Similarly, at t_2, where the G/G/1/∞ queues become idle, the estimated queue length at any iteration will always be zero as well. Thus, the propagation starting from t_1 will not pass beyond t_2. A time point at which the estimated and true trajectory coincide, such as t_1 and t_2, is called a *synchronization*

k	λ/μ	loss rate	P=4	P=16	P=64	P=256	P=1024
1	0.1	0.092	(1.2,0.2)	(1.7,0.3)	(2.0,0.0)	(2.0,0.0)	(2.0,0.0)
	0.3	0.232	(1.9,0.2)	(2.0,0.0)	(2.0,0.0)	(2.0,0.0)	(2.0,0.0)
	0.5	0.334	(2.0,0.0)	(2.0,0.0)	(2.0,0.0)	(2.0,0.0)	(2.0,0.0)
	0.7	0.413	(2.0,0.0)	(2.0,0.0)	(2.0,0.0)	(2.0,0.0)	(2.0,0.0)
	0.9	0.475	(2.0,0.0)	(2.0,0.0)	(2.0,0.0)	(2.0,0.0)	(2.0,0.0)
	0.95	0.487	(2.0,0.0)	(2.0,0.0)	(2.0,0.0)	(2.0,0.0)	(2.0,0.0)
	1.0	0.500	(2.0,0.0)	(2.0,0.0)	(2.0,0.0)	(2.0,0.0)	(2.0,0.0)
	1.05	0.512	(2.0,0.0)	(2.0,0.0)	(2.0,0.0)	(2.0,0.0)	(2.0,0.0)
	1.1	0.524	(2.0,0.0)	(2.0,0.0)	(2.0,0.0)	(2.0,0.0)	(2.0,0.0)
	1.3	0.565	(2.0,0.0)	(2.0,0.0)	(2.0,0.0)	(2.0,0.0)	(2.0,0.0)
	1.5	0.3328	(2.0,0.0)	(2.0,0.0)	(2.0,0.0)	(2.0,0.0)	(2.0,0.0)
10	0.1	0	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)
	0.3	0	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)
	0.5	0.0004	(1.0,0.0)	(1.0,0.0)	(1.2,0.2)	(1.6,0.3)	(2.0,0.0)
	0.7	0.008	(1.1,0.2)	(1.6,0.3)	(2.0,0.0)	(2.0,0.0)	(2.0,0.0)
	0.9	0.05	(1.9,0.2)	(2.0,0.0)	(2.0,0.0)	(2.0,0.0)	(2.2,0.2)
	0.95	0.069	(1.9,0.2)	(2.0,0.0)	(2.0,0.0)	(2.0,0.0)	(2.3,0.3)
	1.0	0.091	(2.0,0.0)	(2.0,0.0)	(2.0,0.0)	(2.0,0.0)	(2.7,0.3)
	1.05	0.115	(2.0,0.0)	(2.0,0.0)	(2.0,0.0)	(2.0,0.0)	(2.6,0.3)
	1.1	0.14	(2.0,0.0)	(2.0,0.0)	(2.0,0.0)	(2.0,0.0)	(2.6,0.3)
	1.3	0.245	(2.0,0.0)	(2.0,0.0)	(2.0,0.0)	(2.0,0.0)	(2.1,0.2)
	1.5	0.3328	(2.0,0.0)	(2.0,0.0)	(2.0,0.0)	(2.0,0.0)	(2.0,0.0)
50	0.1	0	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)
	0.3	0	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)
	0.5	0	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)
	0.7	0	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)
	0.9	0.0006	(1.0,0.0)	(1.4,0.3)	(2.0,0.0)	(2.6,0.3)	(7.7,1.1)
	0.95	0.004	(1.6,0.3)	(2.0,0.0)	(2.1,0.2)	(4.7,0.5)	(15.8,2.1)
	1.0	0.02	(2.0,0.0)	(2.0,0.0)	(2.5,0.3)	(6.3,0.4)	(21.1,1.5)
	1.05	0.052	(2.0,0.0)	(2.0,0.0)	(2.1,0.2)	(5.1,0.5)	(18.2,1.2)
	1.1	0.092	(2.0,0.0)	(2.0,0.0)	(2.0,0.0)	(4.1,0.4)	(13.3,1.6)
	1.3	0.231	(2.0,0.0)	(2.0,0.0)	(2.0,0.0)	(2.0,0.0)	(3.8,0.4)
	1.5	0.3328	(2.0,0.0)	(2.0,0.0)	(2.0,0.0)	(2.0,0.0)	(2.4,0.3)
100	0.1	0	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)
	0.3	0	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)
	0.5	0	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)
	0.7	0	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)	(0.0,0.0)
	0.9	0.00002	(0.1,0.2)	(0.1,0.2)	(0.2,0.4)	(0.3,0.5)	(1.0,1.8)
	0.95	0.0005	(1.1,0.3)	(1.3,0.4)	(2.1,0.5)	(4.8,1.6)	(17.0,6.1)
	1.0	0.0106	(2.0,0.0)	(2.2,0.2)	(4.9,0.6)	(17.3,1.7)	(66.6,6.6)
	1.05	0.0484	(2.0,0.0)	(2.0,0.0)	(3.2,0.5)	(5.1,0.5)	(36.5,4.9)
	1.1	0.091	(2.0,0.0)	(2.0,0.0)	(2.0,0.0)	(9.9,1.1)	(16.5,1.5)
	1.3	0.231	(2.0,0.0)	(2.0,0.0)	(2.0,0.0)	(4.9,0.5)	(3.8,0.4)
	1.5	0.3328	(2.0,0.0)	(2.0,0.0)	(2.0,0.0)	(2.0,0.0)	(2.4,0.3)

Table 1: Numbers of iterations for an M/M/1/K model using the first partial state matching algorithm. Each data point is an average of 10 runs. Each run simulates 10^5 jobs which are divided into 2^{10} batches. For each entry (a, b), a is the average iteration number and $(a - b, a + b)$ is the 90-percent confidence interval for a.

point. Formally, t is a synchronization point of state v_i if $v_{i,j}(t) = v_{i,j'}(t)$, for all $j' > j \geq 0$. An event which leads to a synchronization point is called a *synchronization event*. Therefore, if we let E_F denote a (job arrival) event immediately after whose occurrence the G/G/1/K queue becomes full, and let E_E denote a (job departure) event immediately after whose occurrence the G/G/1/∞ queue becomes empty, E_F and E_E are synchronization events.

It becomes evident that the number of iterations required for convergence, denoted by T, is bounded by the number of batches spanned by the longest propagation segment. Therefore we have:

$$1 \leq T \leq \lceil \frac{max[d(E_F, E_E), d(E_F, E'_F)]}{l_b} \rceil + 1 \tag{6}$$

where (E_F, E_E) and (E_F, E'_F) are any pair of neighboring synchronization events, l_b is the batch length, and $d(E_i, E_j)$ is the number of events between the occurrence of E_i and E_j.

Therefore, when some losses occur, if the traffic intensity is low or is very high, E_E and E_F tend to occur more frequently, respectively, and hence the maximum propagation length is shorter. As a result, the number of iterations required for convergence becomes small. When the traffic intensity is neither high or low both synchronization events occur more sparsely and the longest propagation length increases. Thus, the number of iterations increases. Once the propagation length becomes longer than a batch size, adding more processors becomes unfruitful because the number of iterations will grow linearly with the number of processors used. This situation can be seen when K is 50 and 100, and λ/μ is close to 1 in Table 1 .

In Table 1, the worst M/M/1/K simulation performance always occurs when $\lambda/\mu = 1$, regardless of the number of processors and the buffer capacity. Actually, when $\lambda/\mu = 1$, it is least likely that the queue will become empty or full. This can be derived as follows.

Let $\rho = \lambda/\mu$. If P_0 is the probability that the M/M/1/∞ queue is idle (when an E_E occurs) and P_K is the probability that the M/M/1/K queue is full (when an E_F occurs),then:

$$P_0 = (1 - \rho) \quad 0 \leq \rho < 1, \tag{7}$$

and

$$P_K = \frac{\rho^K - \rho^{K+1}}{1 - \rho^{K+1}} = \frac{\rho^K}{\sum_{i=0}^{K} \rho^i} \quad \rho \geq 0,\ \rho \neq 1 \tag{8}$$

[7]. Because when the M/M/1/∞ queue is empty, the corresponding M/M/1/K queue must also be empty, the joint probability of E_E and E_F occurrence can be given as:

$$P_s(\rho, K) = \begin{cases} P_0 + P_K & 0 \leq \rho < 1, \\ P_K & \rho > 1. \end{cases} \tag{9}$$

Assume that $P_s(1, K) = lim_{\rho \to 1} P_s(\rho, K)$. It can be shown that P_s has a minimum at $\rho = 1$. A proof is given in the Appendix.

K \ λ/μ	0.1	0.3	0.5	0.7	0.9	1.0	1.1	1.3	1.5
1	-0.03	0.29	0.11	-0.24	0.07	-0.29	0.46	0.43	0.38
5	0.0	-0.07	0.01	0.06	0.83	0.39	0.56	0.38	0.62
10	0.0	0.0	-0.06	-0.03	-0.14	0.09	0.38	0.23	0.19
20	0.0	0.0	0.0	0.07	-0.35	1.35	-0.29	0.08	0.31
40	0.0	0.0	0.0	0.0	-0.37	-0.68	-0.05	0.30	0.12
60	0.0	0.0	0.0	0.0	0.10	0.95	0.50	0.26	0.06
80	0.0	0.0	0.0	0.0	0.0	0.32	0.03	0.16	0.07
100	0.0	0.0	0.0	0.0	0.0	0.49	0.14	0.16	0.07

Table 2: Normalized approximation errors of the M/M/1/K model using the first partial state matching algorithm. Each entry is the value of $100*(E(L)-A(L))/E(L)$, where $E(L)$ and $A(L)$ are the average queue lengths of 10 runs obtained from a sequential simulation and the partial state matching simulation, respectively.

K \ λ/μ	0.1	0.3	0.5	0.7	0.9	0.95	1.0	1.05	1.1	1.3
1	0.88	4.11	9.41	15.92	23.15	24.95	26.64	25.94	25.16	22.35
5	0.0	0.0	0.8	0.94	4.66	6.36	8.00	6.51	4.97	1.73
10	0.0	0.0	0.0	0.03	1.36	2.73	4.42	2.76	1.54	0.16
20	0.0	0.0	0.0	0.0	0.17	0.81	2.22	0.76	0.18	0.02
50	0.0	0.0	0.0	0.0	0.0	0.04	0.85	0.02	0.01	0.01
100	0.0	0.0	0.0	0.0	0.0	0.0	0.36	0.002	0.005	0.003

Table 3: Normalized approximation errors of the M/D/1/K model using the first partial state matching algorithm. Each entry is the value of $100*(E(L)-A(L))/E(L)$, where $E(L)$ and $A(L)$ are the average queue lengths of 10 runs obtained from a sequential simulation and the partial state matching simulation, respectively.

The results of the partial state matching simulation in average queue length are compared with a sequential simulation (Table 2). For a fair comparison, both simulations use the same sequence of random numbers. The results suggest that the approximation of the service times has little impact on the simulation outcome. However, the approximation is *biased*, because whenever case 3 occurs, the approximate service time is always decreased by some amount. If the service time process has a small variance, it is likely that the approximate service time is smaller than the expected service time (i.e. $1/\mu$). When the buffer size is small, case 3 occurs more often, and this error will be more significant. An extreme case is an M/D/1/1 queue, where the service time has a zero variance. Results of simulating an M/D/1/K queue using this algorithm are shown in Table 3. It shows that approximation errors are significant when the buffer size is less than 5. For this case, an alternative algorithm is proposed in the following section.

5 FCFS G/D/1/K Queues with Losses

For a G/D/1/K queue, in which each job has a fixed service time, a trajectory can be represented by a sequence of the first job remaining service time (FRST), a queue length sequence, and a job arrival time sequence. Let N be the number of arrivals, P be the number of processors, S_T be the job service time, and β_i be the initial queue length of batch i. A partial state matching algorithm is described by the following steps:

1. Compute the arrival time sequence using the GLM parallel prefix algorithm.
2. Partition the N arrivals into P batches in the same way as described in the first algorithm.
3. Set β_i to 0 for all $1 \leq i \leq P$.
4. Set the FRST of each batch to 0.
5. For each batch, compute departure times and queue lengths independently via a sequential simulation.
6. Assign the new initial queue length of batch i to β_i'.
7. If $\beta_i{=}\beta_i'$ for all $1 \leq i \leq P$, exit; otherwise go to step 4.

The degree of freedom of the simulation is two since we remove FRST from the unmatched set by fixing the initial FRST's of all batches. Therefore the computation of FRST's is local to each batch. Fixing the FRST's is valid when the batch size is large. Because the transient effect of the approximate FRST will becomes less significant as more jobs are simulated.

λ/μ	K=1	K=5	K=10	K=20	K=50	K=100
0.1	(2.0,0.0)	(2.0,0.0)	(2.0,0.0)	(2.0,0.0)	(2.0,0.0)	(2.0,0.0)
0.3	(2.0,0.0)	(2.0,0.0)	(2.0,0.0)	(2.0,0.0)	(2.0,0.0)	(2.0,0.0)
0.5	(2.0,0.0)	(2.0,0.0)	(2.0,0.0)	(2.0,0.0)	(2.0,0.0)	(2.0,0.0)
0.7	(2.0,0.0)	(2.0,0.0)	(2.0,0.0)	(2.0,0.0)	(2.0,0.0)	(2.0,0.0)
0.9	(2.0,0.0)	(2.0,0.0)	(2.0,0.0)	(2.0,0.0)	(2.0,0.0)	(2.0,0.0)
1.0	(2.0,0.0)	(2.0,0.0)	(2.0,0.0)	(2.1,0.2)	(3.5,0.4)	(7.9,1.1)
1.1	(2.0,0.0)	(2.0,0.0)	(2.0,0.0)	(3.0,0.0)	(3.0,0.0)	(3.0,0.0)
1.5	(2.0,0.0)	(2.0,0.0)	(3.0,0.0)	(3.0,0.0)	(3.0,0.0)	(3.0,0.0)
2.0	(2.0,0.0)	(3.0,0.0)	(3.0,0.0)	(3.0,0.0)	(3.0,0.0)	(3.0,0.0)

Table 4: Numbers of iterations of the M/D/1/K model using the second algorithm. Each data point is an average of 10 runs. Each run simulates 10^5 jobs, which are divided into 64 batches. For each entry (a, b), a is the average iteration number and $(a - b, a + b)$ is the 90-percent confidence interval for a.

λ/μ	K=1	K=5	K=10	K=20	K=50	K=100
0.1	(2.0,0.0)	(2.0,0.0)	(2.0,0.0)	(2.0,0.0)	(2.0,0.0)	(2.0,0.0)
0.3	(2.0,0.0)	(2.0,0.0)	(2.0,0.0)	(2.0,0.0)	(2.0,0.0)	(2.0,0.0)
0.5	(2.0,0.0)	(2.0,0.0)	(2.0,0.0)	(2.0,0.0)	(2.0,0.0)	(2.0,0.0)
0.7	(2.0,0.0)	(2.0,0.0)	(2.0,0.0)	(2.0,0.0)	(2.0,0.0)	(2.0,0.0)
0.9	(2.0,0.0)	(2.0,0.0)	(2.0,0.0)	(2.0,0.0)	(2.0,0.0)	(2.0,0.0)
1.0	(2.0,0.0)	(2.0,0.0)	(2.0,0.0)	(2.4,0.3)	(6.4,0.5)	(16.0,3.1)
1.1	(2.0,0.0)	(2.0,0.0)	(2.0,0.0)	(7.0,0.9)	(61.2,5.1)	(64.0,0.0)
1.5	(2.0,0.0)	(2.2,0.2)	(30.6,4.7)	(64.0,0.0)	(64.0,0.0)	(64.0,0.0)
2.0	(2.0,0.0)	(9.2,0.9)	(64.0,0.0)	(64.0,0.0)	(64.0,0.0)	(64.0,0.0)

Table 5: Numbers of iterations of the M/D/1/K model using full state matching. Each data point is an average of 10 runs. Each run simulates 10^5 jobs, which are divided into 64 batches. For each entry (a, b), a is the average iteration number and $(a - b, a + b)$ is the 90-percent confidence interval for a.

$K \backslash \lambda/\mu$	0.1	0.3	0.5	0.7	0.9	1.0	1.3	1.5	2.0
1	0.00	0.01	0.02	0.04	0.07	0.06	0.08	0.10	0.13
5	0.02	0.11	0.24	0.38	0.53	0.65	0.56	0.29	0.43
10	0.02	0.11	0.26	0.57	0.95	1.05	0.82	0.08	0.20
20	0.02	0.11	0.26	0.58	1.85	2.38	0.83	0.06	0.09
50	0.02	0.11	0.26	0.58	2.48	4.88	0.54	0.03	0.03
100	0.02	0.11	0.26	0.58	2.48	10.28	0.27	0.01	0.02

Table 6: The normalized approximation errors for the M/D/1/K model using the second algorithm. Each entry is the value of $100 * (E(L) - A(L))/E(L)$, where $E(L)$ and $A(L)$ are the average queue lengths of 10 runs obtained from the full state matching simulation and the partial state matching simulation, respectively.

Table 4 and 5 compare the convergence speed of the proposed partial state matching simulation with a ***full state matching*** simulation. The full state matching simulation has the same batch partition and initial state assignment as the partial state matching simulation (step 1 to step 4) described in this section, except that the full state matching simulation does not use approximate FRST's and converges on both FRST and the queue length. That is, in step 7, the full state matching simulation compares both the queue lengths and the FRST's, and goes to step 5 when the program loops back. It is expected that a large number of iterations is required for the full state matching simulation when the value of λ/μ is large. Because for the FRST sequence, synchronization points occur only when the queue becomes empty. Thus, when λ/μ increases, the possibility that the queue becomes empty decreases. This can be seen in Table 5. Linear convergence begins to occur when $\lambda/\mu \geq 1.1$ and $K \geq 50$. For smaller K's, linear convergence occurs at a higher traffic intensity value because queues with smaller buffers have higher possibilities of being empty. The partial state matching simulation, on the other hand, requires only a few iterations when $\lambda/\mu > 1$. In fact, the number of iterations for the partial state matching simulation will converge to 2 as λ/μ increases. In this case, the partial state simulation gains an order P fold speed-up. The longest convergence times for both simulations occur when $\lambda/\mu = 1$. This can be argued similarly as the M/M/1/K simulation (section 4.2). In addition to execution speed, another important consideration is approximation errors. Table 6 shows that the partial state matching simulation produces close results. The normalized approximation errors are less than 1% for most cases.

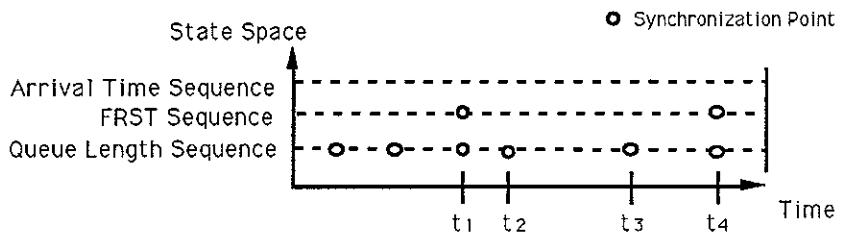

Figure 7: Without approximation, the full state matching simulation converges on both FRST and the queue length. The longest propagation distance is $(t_4 - t_1)$. With approximation on FRST sequence, the partial state matching simulation converges only on the queue length. The longest propagation distance is $(t_3 - t_2)$.

6 Concluding Remarks

Partial state matching simulation artificially fixes some state variables in the unmatched set by using some approximate values so that these state variables can be removed from the unmatched set. As a result, the simulation needs to converge on fewer state variables and thus is likely to converge more quickly.

Two algorithms using this partial state matching approach to simulate FCFS G/G/1/K and G/D/1/K queueing models are proposed in this paper. The first algorithm uses approximate job departure times; the second algorithm uses approximate first job remaining service times. Experiment results of an M/M/1/K and an M/D/1/K model show that the speed-up gained by using these partial state matching simulations against full state matching simulations becomes very significant as λ/μ exceeds 1. Also, both partial state matching simulations produce small approximation errors in general cases. The worst performance for both simulations occurs when $\lambda/\mu = 1$. An argument is made to explain this phenomenon. The first algorithm introduces more significant errors when the model has a small buffer and has a small variance in job service times. The second algorithm is introduced for this situation. Possible future work includes testing a broader class of probability distributions, and investigating networks of queues which contains G/G/1/K queues.

References

[1] Chandy K.M. and Sherman R. Space-Time and Simulation. *Proceedings of the SCS Multiconference on Distributed Simulation.* March 1989, pp 53-57.

[2] Fujimoto R. M. Parallel Discrete Event Simulation. *Commun. ACM.* Vol. 33, No. 10, Oct. 1990, pp 31-53.

[3] Greenberg A. G., Lubachevsky B. D., Mitrani I. Unboundedly Parallel Simulations Via Recurrence Relations. *Proceedings of 1990 Conference on Measurement*

and Modeling of Computer Systems. Boulder, Colorado, May 1990, pp 1-12.

[4] Hills W. D., Steele G. L. Data Parallel Algorithms. *Commun. ACM.* Vol. 29, No. 12, Dec. 1986, pp 1170-1183.

[5] Jefferson D. Virtual Time. *ACM Transactions of Programming Languages and Systems.* Vol. 7, No. 3, July 1985, pp 404-425.

[6] Jones D. W. Concurrent Simulation: An Alternative to Distributed Simulation. *Proceedings of 1986 Winter Simulation Conference,* December 1986. pp 417-423.

[7] Kleinrock L *Queueing Systems.* Vol. 1, Wiley-Interscience, 1975.

[8] Lander R. E., Fischer M. J. Parallel Prefix Computation. *Journal of ACM.* Vol. 27, 1980, pp 831-838.

[9] Bagrodia R., Chandy K. M., Liao W. T. An Experimental Study On the Performance of the Space Time Simulation Algorithm. *Proceedings of the Sixth Workshop of the Parallel and Distributed Simulations.* 1992, pp 159-168.

[10] Lin Y. B. A Time-Division Algorithm for Parallel Simulation. *ACM TOMACS,* Vol. 1, No. 1, Jan. 1991, pp 73-83.

[11] Mitra D., Mitrani I. Control and Coordination Policies for System with Buffers. *ACM SIGMETRICS Performance Evaluation Review,* Vol. 17, No. 1, May 1989, pp 156-164.

[12] Wanger D. B., Lazowska E. D. Parallel Simulation of Queueing Network: Limitation and Potentials. *Proceedings of 1989 ACM SIGMETRICS and PERFORMANCE,* May 1989, pp 146-155.

Appendix

Lemma A.1 For $0 \leq \rho < 1$, $P_s(\rho, K)$ is strictly decreasing.

Proof: We show that for $0 \leq \rho < 1$

$$\begin{aligned}
\frac{\partial P_s}{\partial \rho} &= \frac{K\rho^{K-1}-(K+1)\rho^K}{1-\rho^{K+1}} + \frac{(K+1)\rho^K(\rho^K-\rho^{K+1})}{(1-\rho^{K+1})^2} - 1 \\
&= \frac{[K\rho^{K-1}-(K+1)\rho^K](1-\rho^{K+1})+(K+1)\rho^K(\rho^K-\rho^{K+1})-(1-\rho^{K+1})^2}{(1-\rho^{K+1})^2} < 0.
\end{aligned}$$

Because $(1-\rho^{K+1})^2 \geq 0$, the above inequality holds if the numerator of the left-hand-side expression is less than zero. This is derived as follows:

$$\begin{aligned}
&[K\rho^{K-1} - (K+1)\rho^K](1-\rho^{K+1}) + (K+1)\rho^K(\rho^K - \rho^{K+1}) - (1-\rho^{K+1})^2 \\
&= (K\rho^{K-1} - K\rho^K - \rho^K)(1-\rho^{K+1}) + (\rho^K + K\rho^K)(\rho^K - \rho^{K+1}) - (1 - 2\rho^{K+1} + \rho^{2K+2}) \\
&= -\rho^{2K+2} + \rho^{2K} + 2\rho^{K+1} - K\rho^K - \rho^K + K\rho^{K-1} - 1 \\
&= \rho^{2K}(1-\rho)(1+\rho) + K\rho^{K-1}(1-\rho) - \rho^K(1-\rho) - (1-\rho)(\textstyle\sum_{i=0}^{K}\rho^i) \\
&= (1-\rho)[\rho^{2K}(1+\rho) + K\rho^{K-1} - \rho^K - \textstyle\sum_{i=0}^{K}\rho^i] \\
&= (1-\rho)[(\rho^{2K+1} - \rho^K) + (\rho^{2K} - \rho^K) + (K\rho^{K-1} - \textstyle\sum_{i=0}^{K-1}\rho^i)] < 0. \ \square
\end{aligned}$$

Lemma A.2 For $\rho > 1$, $P_s(\rho, K)$ is strictly increasing.

Proof: When $\rho > 1$, $P_s(\rho, K) = P_K$. Again, we take the first partial derivative of P_K with respect to ρ:

$$\frac{\partial P_K}{\partial \rho} = \frac{(K\rho^{K-1} - K\rho^K) + (\rho^{2K} - \rho^K)}{(1-\rho^{K+1})^2}.$$

Because $(1-\rho^{K+1})^2 > 0$ for $\rho > 1$, to prove the lemma it suffices to show that:

$$(K\rho^{K-1} - K\rho^K) + (\rho^{2K} - \rho^K) > 0 \quad \rho > 1.$$

Dividing both sides of the above inequaity by ρ^{K-1} and letting $g(\rho, K)$ be the resulting left-hand-side expression, we have:

$$g(\rho, K) = (K - K\rho - \rho + \rho^{K+1}).$$

Because $g(1, K) = 0$ and $\partial g/\partial \rho = (K\rho^K - K) + (\rho^K - 1) > 0$, clearly, $g(\rho, K) > 0$ for all $\rho > 1$ and $K \geq 1$. $\square$

Theorem A.1 The minimum of $P_s(\rho, K)$ occurs at $\rho = 1$ for $\rho \geq 0$.

Combining Lemma A.1 and A.2 completes the proof of theorem A.1. $\square$