MASSIVELY TIME-PARALLEL, APPROXIMATE SIMULATION OF LOSS QUEUEING SYSTEMS

Jain J. WANG and Marc ABRAMS
Department of Computer Science, Virginia Polytechnic Institute and State University, Blacksburg, VA 24061-0106

Abstract

A time-parallel simulation obtains parallelism by partitioning the time domain of the simulation. An approximate time-parallel simulation algorithm named GG1K is developed for acyclic networks of loss FCFS G/G/1/K queues. The GG1K algorithm requires two phases. In the first phase, a similar system (i.e., a $G/G/1/\infty$ queue) is simulated using the GLM algorithm. Then the resultant trajectory is transformed into an approximate G/G/1/K trajectory in the second phase. The closeness of the approximation is investigated theoretically and experimentally. Our results show that the approximation is highly accurate except when K is very small (e.g., 5) in certain models. The algorithm exploits unbounded parallelism and can achieve near-linear speedup when the number of arrivals simulated is sufficiently large.

1. Introduction

Computer simulation is a process of computing a *sample path*, or *trajectory*, of a target simulation model, representing the time evolution of the state of the model over a period of simulation time. In this paper, the term *simulation* refers to discrete event simulation. The state space and time domain of a simulation model form a *space-time* region [4]. The state space is defined as the set of state variables representing the simulation model. Parallel simulation refers to the parallel execution of a simulation on multiple processors for the purpose of reducing the execution time.

A number of parallel simulation algorithms have been proposed in the past. These algorithms, with few exceptions, can be classified into three broad categories, or *frameworks*, with respect to how parallelism is obtained. They are: *parallel replication*, *space-parallel*, and *time-parallel*. These frameworks will be reviewed briefly in section 2.

In this paper, we investigate a time-parallel algorithm, called GG1K, for simulating the arrival of N jobs to a loss FCFS G/G/1/K queue, in which arriving jobs that find the queue full are lost rather than blocked. Let α_j denote the interarrival time between job j and job $j+1$ to a queue, and δ_j denote the service time of job j, where $1 \leq j \leq N$. Also, let A_j, D_j, and $L(t)$ denote the arrival time of job j, the departure time of job j, and the queue length at simulation time t, respectively, for all t in the simulation time interval $[0, \tau]$ (where $\tau > 0$). Job interarrival and service times are continuous random variables whose values can be pre-sampled. Given $\alpha_j > 0$ and $\delta_j > 0$, we wish to find A_j and D_j, for all $1 \leq j \leq N$, and $L(t)$, for all t in $[0, \tau]$, so that measures such as average queue length and job loss rate can be obtained. We assume that an arrival and departure can not simultaneously occur.

Loss G/G/1/K and G/D/1/K queues are considered because they are important in modeling packet switched communication networks. Simulating these models, however, can be a time consuming process. For example, in an ATM (asynchronous transfer mode) network [13], data loss rate can be on the order of 10^{-9} or lower [19]. Without an appropriate variance reduction technique (e.g., importance sampling [5]), simulating such an ATM switch on a single processor workstation may take days of CPU time to obtain a reliable loss rate estimate. Our goal is to devise algorithms that can dramatically reduce the long simulation execution time using the power of parallel processing.

The rest of this paper is organized as follows. Section 2 reviews parallel simulation frameworks and discusses why a time-parallel approach is considered. Section 3 reviews related time-parallel simulation algorithms. Section 4 presents the GG1K algorithm. The algorithm is non-iterative and is based on an idea of simulating a similar system ($G/G/1/\infty$) and then transforming the resultant trajectory into an approximate G/G/1/K trajectory. Some experimental results of the GG1K algorithm are also given in this section. Conclusions are given in section 6.

2. Parallel Simulation Frameworks

2.1. PARALLEL REPLICATION SIMULATION

The most intuitive way of executing a simulation using multiple processors is to run a replication of the simulation model independently and concurrently on each processor. The estimates obtained from all independent runs are averaged together when the runs complete. If the real (i.e., wall-clock) execution times of these independent runs have a small variance, good processor efficiency (hence speedup) will be achieved. However, care must be taken in starting and terminating a simulation, collecting samples, and calculating estimates from the samples. Otherwise, serious statistical problems may result. Parallel replication simulation and their statistical issues have been studied extensively by Glynn and Heidelberger [7, 9, 10] and some other researchers (e.g., [3]) in recent years.

Parallel replication is attractive because it is simple and has low overhead. Each processor simply performs a sequential simulation. However, the applicability of parallel replication simulation is limited by the following constraints:

C1: The simulation model must fit into the memory of a single processor.

C2: Each processor must run the simulation for a sufficiently long time to dissipate the initial transient [9].

C3: For a simulation that uses data traces as input (i.e., trace-driven simulation), multiple independent traces that share the same set of statistical parameters are required.

2.2. SPACE-PARALLEL SIMULATION

A number of space-parallel algorithms have been proposed in the literature (see [6] for a survey). Space-parallel algorithms decompose a simulation model into components based on the state space of the model. Each component is mapped into a *logical process* which is responsible for computing the values of the state variables of the corresponding component over a simulation time interval. Logical processes communicate with each other by passing time-stamped *messages*. The speedup of a space-parallel simulation is bounded by the number of logical processes. For models with few components, such as a single server queue, space-parallel simulation is not viable simply because the limited number of components limits parallelism.

2.3. TIME-PARALLEL SIMULATION

Chandy and Sherman [4] propose a space-time conceptual framework, which views simulation as a process of *filling in* a space-time rectangle, and suggest that parallelism can be obtained through decomposing the space domain as well as the time domain of a simulation model. In this paper, a time-parallel simulation refers to one that obtains parallelism purely from the time domain.

Time-parallel simulation can be generally described as follows. Suppose that a simulation model has M state variables: $v_1, v_2, \ldots, v_M$. Let $v_k(t)$ denote the value of state variable v_k *immediately after* t. The *system state* at time t is represented by an M-tuple: $S(t) = (v_1(t), v_2(t), ...v_M(t))$. Let $t_1, t_2, \ldots, t_n$, where $0 < t_1 < t_2 \ldots < t_n < \tau$, be the time points when the system changes its state. The *trajectory* of a simulation model can then be represented by the sequence $S(0), S(t_1), \ldots, S(t_n)$, where $S(0)$ is the initial state of the system. In a time-parallel simulation, the simulation time interval is partitioned into P intervals: $[b_0, b_1], (b_1, b_2], \ldots, (b_{P-1}, b_P]$, where $0 = b_0 < b_1 \ldots < b_P = \tau$, and $\{b_0, \ldots, b_P\} \cap \{t_1, \ldots, t_n\} = \emptyset$. The subtrajectory in interval $[b_{k-1}, b_k]$ for $k = 1$, and $(b_{k-1}, b_k]$ for $1 < k \leq P$, is referred to as *batch* k. Then $S(b_{k-1})$ and $S(b_k)$ are the *initial* state and *final* state, respectively, of batch k. All batches can be simulated in parallel.

This paper examines time-parallel simulation in preference to parallel replication simulation due to constraints C2 and C3. Furthermore, time-parallel simulation is examined in preference to space-parallel simulation because the later exploits practically no parallelism in the models of interest (i.e., single server queues). A key problem in time-parallel simulation is determine the batch initial states. Solutions to this problem are usually model dependent (e.g., [2, 11, 15, 17, 20]).

3. Related Work in Time-Parallel Simulation

This section reviews some related time-parallel simulation algorithms: (1) Greenberg, Lubachevsky, and Mitrani's (GLM) prefix algorithm [8], (2) Lin's network multiplexer algorithm [15], (3) Nikolaidis and Fujimoto's bursty arrival algorithm [17], and (4) Andradóttir and Ott's time segmentation algorithm [2]. The GLM algorithm is discussed with more detail because it forms the basis of our approximation algorithms.

3.1. THE GLM ALGORITHM

The GLM algorithm [8] efficiently simulates queueing network models which can be expressed as recurrence relations and transformed into a parallel prefix problem. Let $\Re$ be a domain and $\circ$ be any associative operator on $\Re$. Let N be any positive integer and let $x_1, x_2, \ldots, x_N$ be a sequence, where $x_j \in \Re$ for $1 \le j \le N$. The prefix problem [12] is to compute $s_j = x_0 \circ x_1 \circ \ldots \circ x_j$, for $1 \le j \le N$. We first review how the GLM algorithm is applied to a FCFS G/G/1/∞ queue.

Let D_j^∞ denotes the departure time of job j from a G/G/1/∞ queue. If A_1 (i.e., the arrival time of the first job) is given, the arrival and departure time sequences $A_1, A_2, \ldots, A_N$ and $D_1^\infty, D_2^\infty, \ldots, D_N^\infty$ are the solution of the following recurrence relations [16]:

$$A_j = A_{j-1} + \alpha_{j-1} \quad 1 < j \le N, \tag{1}$$

$$D_j^\infty = \begin{cases} A_j + \delta_j & j = 1, \\ \max(D_{j-1}^\infty, A_j) + \delta_j & 1 < j \le N. \end{cases} \tag{2}$$

From (1), $A_j = A_1 + \alpha_1 + \ldots + \alpha_{j-1}$. Thus, solving $A_1, \ldots, A_N$ is a prefix problem. For the departure time sequence, equation (2) can be rewritten by a matrix recurrence relation [8] to which a prefix algorithm can be applied directly. In the rest of this paper, a sequence $x_a, x_{a+1}, \ldots, x_{b-1}, x_b$, where $a < b$, will be denoted by $\langle x_a, x_b \rangle$.

In a G/G/1/∞ model, an *event time* sequence $\langle E_1^\infty, E_{2N}^\infty \rangle$ is the result of merging two sequences $\langle A_1, A_N \rangle$ and $\langle D_1^\infty, D_N^\infty \rangle$ in ascending time order. *Event* i (for $1 \le i \le 2N$) is a job arrival or a departure corresponding to E_i^∞. When there are N arrivals, a total of $2N$ events will be simulated. Let t^- and t^+ represent the instances immediately *before* and *after* t, respectively, such that $t^+ - t = t - t^- \approx 0$ and no event occurs in the intervals $[t^-, t)$ and $(t, t^+]$. For example, $E_i^{\infty+}$ represents the instance immediately after the occurrence of event i and A_j^- represents the instance immediately before job j arrives. Then the G/G/1/∞ queue length at $E_i^{\infty+}$, denoted L_i^∞, is obtained by solving the following recurrence relation:

$$L_i^\infty = \begin{cases} L_{i-1}^\infty + 1 & \text{if event } i \text{ is an arrival}, \\ L_{i-1}^\infty - 1 & \text{if event } i \text{ is a departure}, \end{cases} \tag{3}$$

where L_0^∞ is defined to be the initial queue length of the G/G/1/∞ queue. Therefore, the computation of the queue length sequence is again a prefix problem.

Suppose that the number of processors, denoted P, divides N evenly, and there exists an identity element $\iota \in \Re$, such that $\iota \circ x_j = x_j$, for $1 \le j \le N$. To compute $s_j = x_0 \circ x_1 \circ \ldots \circ x_j$ for $1 \le j \le N$ in parallel, the GLM algorithm performs the following steps:

Algorithm GLM

input: $\langle x_1, x_N \rangle$.

output: $\langle s_1, s_N \rangle$.

1. Partition the sequence $\langle x_1, x_N \rangle$ into P batches evenly such that batch k, $1 \le k \le P$ contains $\langle x_{((k-1)N/P)+1},\ x_{kN/P} \rangle$.

2. Compute $f_k = x_{((k-1)N/P)+1} \circ \ldots \circ x_{kN/P}$ for all $1 \le k \le P$.

3. Compute

$$\beta_k = \begin{cases} \iota & k = 1, \\ f_1 \circ \ldots \circ f_{k-1} & 1 < k \le P. \end{cases}$$

4. Let $q(j)$ and $r(j)$ be the quotient and the remainder of $\frac{j}{N/P}$, respectively. For $1 \le j \le N$, compute

$$s_j = \begin{cases} \beta_{q(j)+1} & r(j) = 0, q(j) \le P, \\ \beta_{q(j)+1} \circ x_{q(j)(N/P)+1} \circ \ldots \circ x_j & r(j) \ne 0, q(j) < P. \end{cases}$$

The GLM algorithm can be directly applied to compute $\langle A_2, A_N \rangle$, and $\langle D_1, D_N \rangle$ (by solving (1) and (2)) in time $O(N/P + \log P)$. Computing $\langle E_1^\infty, E_N^\infty \rangle$ requires merging two N-element sorted lists (i.e., $\langle A_1, A_N \rangle$ and $\langle D_1, D_N \rangle$). This can be done in time $O(N/P + \log N)$ using a parallel merging algorithm described in [1]. While computing $\langle E_1^\infty, E_N^\infty \rangle$, we associate each event time with its corresponding event type (i.e., arrival or departure). Then the GLM algorithm can again be applied to compute $\langle L_1^\infty, L_{2N}^\infty \rangle$ (by solving (3)) in time $O(N/P + \log P)$. In total, the GLM algorithm requires time $\Theta(N/P + \log P + \log N)$ to compute the job arrival time sequence, the job departure time sequence, the event time sequence, and the queue length sequence for a FCFS G/G/1/∞ queue [8].

3.2. THE NETWORK MULTIPLEXER ALGORITHM

Lin [15] has developed an approximate trace-driven algorithm for simulating network multiplexers which are modeled by loss G/D/1/K queues. The algorithm first partitions the input trace consisting of N arrivals into P subtraces as evenly as possible. Then the algorithm computes a lower bound and an upper bound on the number of lost jobs by assuming, respectively, at each of the partition time points the queue length is zero (for the lower bound), and the remaining service time of the first job in the queue has a full service time quantum (for the upper bound). Overall, Lin's algorithm requires time $\Theta(N/P + P)$.

When $N \gg P$, Lin's algorithm produces accurate results and can achieve near-linear speedup. However, with a fixed trace length, the accuracy decreases as P increases because when fewer jobs are simulated by each processor, the initial state approximation will increasingly affect the accuracy of the simulation.

3.3. THE BURSTY ARRIVAL ALGORITHM

Nikolaidis and Fujimoto [17] model a network multiplexer as a finite FCFS queue which receives packets from several sources. The packets generated by each source arrive at the queue in *bursts*. In each burst, the packet lengths and interarrival times are assumed to be constant so that the input trace can be *compressed* by recording the arrivals on a burst basis rather than tracking each individual packet. This characterization of the workload allows us to obtain a set of simulation times (through a pilot simulation) when the queue is guaranteed to be empty or full regardless of the initial queue length. The input trace is then partitioned into subtraces at these time points, hence permitting time-parallel simulation. Therefore, the amount of parallelism obtained by this algorithm relies on the workload of the system. In a circumstance where the multiplexer becomes idle or saturated very often, the amount of parallelism permitted by the bursty arrival could be significant, and hence good speedup can be achieved. However, the bursty arrival simulation assumes constant interarrival times and constant job service times within each traffic burst which, in turn, limits the applicability of this approach.

3.4. THE TIME SEGMENTATION ALGORITHM

Recently, Andradóttir and Ott [2] have proposed a time-parallel algorithm for systems that can be modeled by a positive recurrent Markov Chain. Andradóttir and Ott show that for certain queueing systems, there exists a finite time beyond which the trajectory of the system is guaranteed to converge regardless of the initial state of the system. Therefore, it is possible to correctly 'predict' a state at time $t > 0$ without simulating the system for the entire time interval $[0, t)$, hence permitting temporal parallelism.

4. The GG1K Algorithm

This section presents the GG1K algorithm for loss FCFS G/G/1/K queues in which the arriving jobs that find the queue full are lost. Let $L(t)$ denote the loss FCFS G/G/1/K queue length at time t. For convenience, the departure time of a lost job is defined to be its arrival time because a lost job leaves the

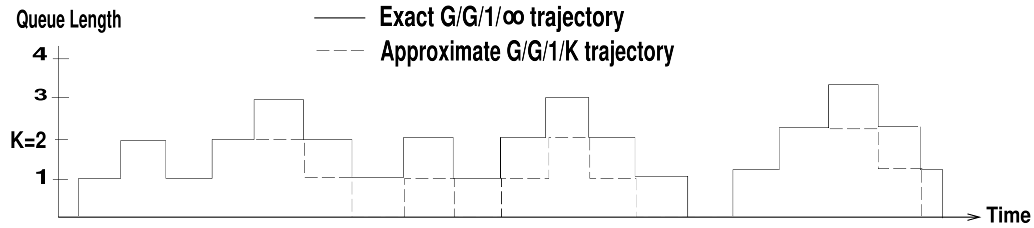

Fig. 1: An example of computing an approximate trajectory based on the transformation technique.

queue immediately upon arrival. The departure time of any job j is:

$$D_j = \begin{cases} \max(D_1, \ldots, D_{j-1}, A_i) + \delta_j & L(A_j^-) < K, \\ A_j & L(A_j^-) = K. \end{cases} \tag{4}$$

An open question is whether there exists an efficient exact parallel algorithm to solve (4). The GG1K algorithm presented next provides an approximate solution to (4).

The GG1K algorithm (stated in section 4.2) consists of two phases. The first phase relaxes the finite buffer constraint and simulates a FCFS G/G/1/∞ queue using the GLM algorithm. If the queue length computed in the first phase never exceeds K, phase two is skipped because the trajectory generated by phase one is also an exact trajectory for the corresponding FCFS G/G/1/K queue. Otherwise, the second phase is performed, which takes the finite buffer storage into account and *transforms* the phase one trajectory into an approximate trajectory for the corresponding G/G/1/K queue.

The transformation technique is illustrated in fig. 1 and 2. This transformation is based on the following assumptions:

A1: Whenever an arrival occurs in the G/G/1/∞ queue, an arrival will also occur in the corresponding G/G/1/K queue.

A2: Whenever a departure occurs in the G/G/1/∞ queue, a departure will also occur in the corresponding G/G/1/K queue except when the G/G/1/K queue is empty. No departure will occur in the G/G/1/K queue at any other times.

For the models of interest (i.e., acyclic networks of G/G/1 queues), A1 is always true. However, when job losses occur in the G/G/1/K queue, A2 may not hold (fig. 2). Hence the GG1K algorithm is approximate. The next section shows that A2 enables us to approximately simulate a G/G/1/K queue efficiently with unbounded parallelism. Sections 4.3 and 4.5 show that the error arising from A2 is small in general.

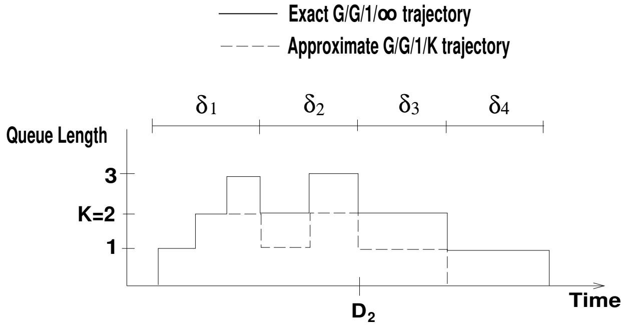

Fig. 2: An example of computing an approximate trajectory based on the transformation technique. In the exact G/G/1/K queue, because job 3 is lost, job 4 should depart at $D_2 + \delta_4$. Thus, A2 holds if and only if $\delta_3 = \delta_4$ (e.g., when service times are constant).

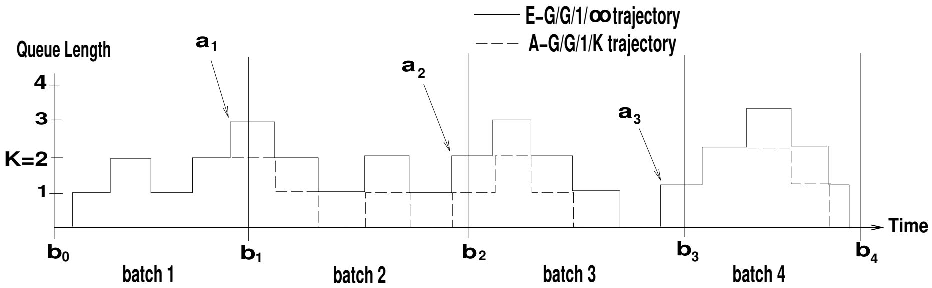

Fig. 3: Each of the P=4 batches contains 5 events. The last event of each batch k is labeled by a_k, for $1 \leq k < 4$.

4.1. DETERMINING THE INITIAL QUEUE LENGTHS FOR THE GG1K TRANSFORMATION

Before presenting the algorithm, we consider a key problem: how to determine the initial queue length for each batch required by time-parallel simulation. This section presents an efficient *approximate* solution to this problem. The rest of this paper uses the FCFS queueing discipline. In addition, we use prefixes "A-" and "E-" to distinguish between an approximate simulation, which uses the GG1K algorithm (section 4.2), and its corresponding exact simulation. For instance, "A-G/G/1/K trajectory" and "A-G/G/1/K queue length" represent the approximate trajectory and queue length, respectively, resulting from the GG1K algorithm, while their exact G/G/1/∞ counterparts will be referred to as "E-G/G/1/∞ trajectory" and "E-G/G/1/∞ queue length," respectively. Finally, *all approximate quantities are denoted by bold (and non-italics) font.*

Let $\mathbf{L}_i^K$ denote the A-G/G/1/K queue length at $E_i^{\infty+}$. Let $\Delta\mathbf{L}_i^K$ denote the difference between the E-G/G/1/∞ queue length (L_i^∞) and the A-G/G/1/K queue length ($\mathbf{L}_i^K$) at $E_i^{\infty+}$. Formally:

$$\Delta\mathbf{L}_i^K = L_i^\infty - \mathbf{L}_i^K. \tag{5}$$

Let $a_0 = 0$ and let event a_k be the last event of batch k, for $1 \leq k < P$, as illustrated in fig. 3. Formally, a_k satisfies $a_k \in \{1, 2, \ldots, 2N\}$; $a_1 < a_2, \ldots, < a_{P-1}$; and $E_{a_k}^\infty < b_k < E_{a_k+1}^\infty$. Our goal is to efficiently determine $\Delta\mathbf{L}^K_{a_k}$, for $1 \leq k < P$, as a function only of quantities known at the end of an exact G/G/1/∞ simulation using the GLM algorithm: $\langle L^\infty_0, L^\infty_{2N} \rangle$ and K. From $L^\infty_{a_k}$ and $\Delta\mathbf{L}^K_{a_k}$, equation (5) yields the desired quantity: $\mathbf{L}^K_{a_k}$ (the initial A-G/G/1/K queue length of batch $k+1$). Our method computes $\Delta\mathbf{L}^K_{a_k}$ by deriving lower and upper bounds on $\Delta\mathbf{L}^K_{a_k}$. The bounds are computed using only K and the E-G/G/1/∞ queue lengths in batch k (i.e., $\langle L^\infty_{a_k-1}, L^\infty_{a_k} \rangle$).

Consider for a moment the problem of finding the initial queue length of batch $k+1$ (i.e., $\mathbf{L}^K_{a_k}$) from batch k. Consider any event i in batch k (i.e., $a_{k-1} < i \leq a_k$). Let $\underline{\Delta\mathbf{L}}^K_{a_{k-1},a_k}$ (respectively, $\overline{\Delta\mathbf{L}}^K_{a_{k-1},a_k}$) denote a lower (upper) bound on $\Delta\mathbf{L}^K_i$ computed using only K and the E-G/G/1/∞ queue length in $\langle L^\infty_{a_{k-1}}, L^\infty_i \rangle$. Formally,

$$\underline{\Delta\mathbf{L}}^K_{a_{k-1},i} \leq \Delta\mathbf{L}^K_i \leq \overline{\Delta\mathbf{L}}^K_{a_{k-1},i} \quad \text{for all} \quad a_{k-1} \leq i \leq a_k. \tag{6}$$

Then $\underline{\Delta\mathbf{L}}^K_{a_{k-1},i}$ and $\overline{\Delta\mathbf{L}}^K_{a_{k-1},i}$, for $i = a_k$, yields the desired quantity, $\Delta\mathbf{L}^K_{a_{k-1}}$. The subsequent four equations will express $\underline{\Delta\mathbf{L}}^K_{a_{k-1},i}$ and $\overline{\Delta\mathbf{L}}^K_{a_{k-1},i}$ as recurrences.

The first recurrence for the lower bound arises because immediately after event a_{k-1}, for any $1 \leq k < P$, if the E-G/G/1/∞ queue has more than K jobs in the buffer, then the E-G/G/1/∞ queue length and the A-G/G/1/K queue length must differ by at least the amount by which the E-G/G/1/∞ queue length exceeds K. Otherwise, they differ by at least 0. Formally:

$$\underline{\Delta\mathbf{L}}^K_{a_{k-1},a_{k-1}} = \max\{0, L^\infty_{a_{k-1}} - K\}. \tag{7}$$

The first recurrence for the upper bound arises because immediately after event a_{k-1}, the queue length difference must be no more than the E-G/G/1/∞ queue length. Formally:

$$\overline{\Delta\mathbf{L}}^K_{a_{k-1},a_{k-1}} = L^\infty_{a_{k-1}}. \tag{8}$$

Now, consider any event i in batch k (i.e., $a_k < i \leq a_{k+1}$). To compute a lower and an upper bound on $\Delta\mathbf{L}^K_i$, we consider two cases: (1) event i is an arrival, (2) event i is a departure. In case (1), after event i occurs, the queue length difference either remains the same or increases, and must be no less than the amount by which the E-G/G/1/∞ queue length exceeds K. In case (2), the queue length difference either remains the same or decreases, and must be no more than the E-G/G/1/∞ queue length. These observations are formally stated in (9) and (10). For $a_{k-1} < i \leq a_k$,

$$\underline{\Delta\mathbf{L}}^K_{a_{k-1},i} = \begin{cases} \max\{L^\infty_i - K, \underline{\Delta\mathbf{L}}^K_{a_{k-1},i-1}\} & L^\infty_i > L^\infty_{i-1} \\ \min\{L^\infty_i, \underline{\Delta\mathbf{L}}^K_{a_{k-1},i-1}\} & L^\infty_i < L^\infty_{i-1} \end{cases} \tag{9}$$

and

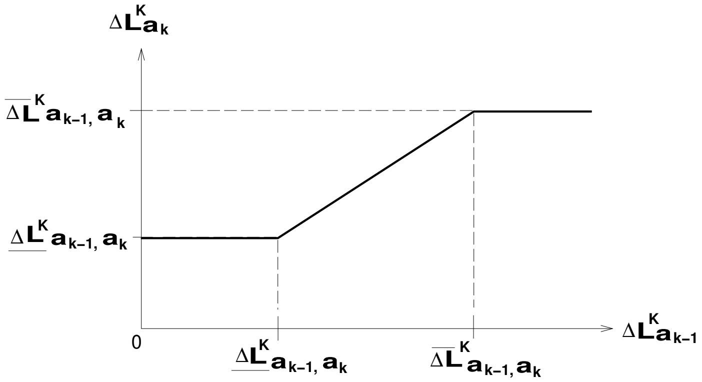

Fig. 4: Relationship of $\Delta\mathbf{L}^K_{k-1}$ and $\Delta\mathbf{L}^K_{a_k}$, which are the initial queue length differences of the E-G/G/1/∞ and the A-G/G/1/K queues for batches k and $k+1$, respectively.

$$\overline{\Delta\mathbf{L}}^K_{a_{k-1},i} = \begin{cases} \max\{L^\infty_i - K, \overline{\Delta\mathbf{L}}^K_{a_{k-1},i-1}\} & L^\infty_i > L^\infty_{i-1} \\ \min\{L^\infty_i, \overline{\Delta\mathbf{L}}^K_{a_{k-1},i-1}\} & L^\infty_i < L^\infty_{i-1}. \end{cases} \tag{10}$$

The fact that $\underline{\Delta\mathbf{L}}^K_{a_{k-1},i}$ and $\overline{\Delta\mathbf{L}}^K_{a_{k-1},i}$ are indeed lower and upper bounds on $\Delta\mathbf{L}^K_{a_{k-1}}$ (i.e., (6) holds) can be proven from (7) to (10) by induction on i.

Lemma 1 states that the difference between the E-G/G/1/∞ and the A-G/G/1/K initial queue lengths for batch k (i.e., $\Delta\mathbf{L}^K_{a_k}$) can be one of the following three values: the lower bound $\underline{\Delta\mathbf{L}}^K_{a_{k-1},a_k}$, the difference between the E-G/G/1/∞ and the A-G/G/1/K initial queue lengths for batch $k-1$ (i.e., $\Delta\mathbf{L}^K_{a_{k-1}}$), or the upper bound $\overline{\Delta\mathbf{L}}^K_{a_{k-1},a_k}$. These three cases occur when, respectively, $\Delta\mathbf{L}^K_{a_k}$ is in the interval $(0,\underline{\Delta\mathbf{L}}^K_{a_{k-1},a_k})$, $[\underline{\Delta\mathbf{L}}^K_{a_{k-1},a_k},\overline{\Delta\mathbf{L}}^K_{a_{k-1},a_k}]$, or $(\overline{\Delta\mathbf{L}}^K_{a_{k-1},a_k}, +\infty)$ (fig. 4).

Lemma 1 *For all* $1 \le k \le P-1$,

$$\Delta\mathbf{L}^K_{a_k} = \max\{\min\{\Delta\mathbf{L}^K_{a_{k-1}}, \overline{\Delta\mathbf{L}}^K_{a_{k-1},a_k}\}, \underline{\Delta\mathbf{L}}^K_{a_{k-1},a_k}\}. \tag{11}$$

Proof: See Appendix A.

Equation (11) is a linear recurrence in the semiring on non-negative integers, where *max* is the addition operator with identity $-\infty$, and *min* is the multiplication operator with identity $+\infty$. That is, (11) can be expressed as:

$$\begin{bmatrix} \Delta\mathbf{L}^K_{a_k} \\ \infty \end{bmatrix} = \begin{bmatrix} \overline{\Delta\mathbf{L}}^K_{a_{k-1},a_k} & \underline{\Delta\mathbf{L}}^K_{a_{k-1},a_k} \\ 0 & \infty \end{bmatrix} \cdot \begin{bmatrix} \Delta\mathbf{L}^K_{a_{k-1}} \\ \infty \end{bmatrix}. \tag{12}$$

Solving equation (12) is a parallel prefix problem. Using Stone's combining scheme [18, pp. 200-206] to compute $P-1$ values of $\Delta\mathbf{L}^K_{a_k}$ for all k requires $\Theta(\log P)$ of execution time. Note that we only compute $P-1$ initial queue lengths because the initial state of the first batch is the initial state of the system. We next describe how to use (5) to (12) to compute the initial A-G/G/1/K queue lengths.

Table 1
Computing $\underline{\Delta \mathbf{L}}^K_{a,i}$ and $\overline{\Delta \mathbf{L}}^K_{a,i}$ for the trajectory in fig. 3.

(a,i)	(0,0)	(0,1)	(0,2)	(0,3)	(0,4)	(0,5)
$\overline{\Delta \mathbf{L}}^K_{a,i}$	0	0	0	0	0	1
$\underline{\Delta \mathbf{L}}^K_{a,i}$	0	0	0	0	0	1
(a,i)	(5,5)	(5,6)	(5,7)	(5,8)	(5,9)	(5,10)
$\overline{\Delta \mathbf{L}}^K_{a,i}$	3	2	1	1	1	1
$\underline{\Delta \mathbf{L}}^K_{a,i}$	1	1	1	1	1	1
(a,i)	(10,10)	(10,11)	(10,12)	(10,13)	(10,14)	(10,15)
$\overline{\Delta \mathbf{L}}^K_{a,i}$	2	2	2	1	0	0
$\underline{\Delta \mathbf{L}}^K_{a,i}$	0	1	1	1	0	0

Algorithm IQL (to compute initial A-G/G/1/K queue lengths):

input: $\langle L_0^\infty, L_{2N}^\infty \rangle$, K.

output: $\langle \mathbf{L}^K_{a_1}, \mathbf{L}^K_{a_{P-1}} \rangle$.

1. Compute $\underline{\Delta \mathbf{L}}^K_{a_{k-1},a_k}$ and $\overline{\Delta \mathbf{L}}^K_{a_{k-1},a_k}$ using (7) to (10), for all $1 \leq k < P-1$.

2. Compute $\Delta \mathbf{L}^K_{a_k}$, for all $1 \leq k < P$, using (12).

3. Compute $\mathbf{L}^K_{a_k}$ for all $1 \leq k < P$, (i.e., the initial A-G/G/1/K queue lengths of batches 2 to P) using (5).

Assume that each batch contains approximately the same number of events. Then with P processors, step 1 requires time $\Theta(N/P)$. Step 2 requires $\Theta(\log P)$ as discussed above. Step 3 requires constant time. In total, the algorithm takes $\Theta(N/P + \log P)$.

We use the example in fig. 1 to illustrate the computation of the initial A-G/G/1/K queue lengths using the algorithm described above. Suppose $P = 4$. Thus, the trajectory is partitioned into four batches (fig. 3). Each batch contains 5 events. The last event of the first three batches are numbers 5, 10, and 15, respectively. Therefore, we are solving for $\Delta \mathbf{L}^K_5$, $\Delta \mathbf{L}^K_{10}$, and $\Delta \mathbf{L}^K_{15}$. Each of these is computed by a processor independently. We first (step 1 of algorithm IQL) compute in parallel three pairs of upper and lower bounds: ($\underline{\Delta \mathbf{L}}^K_{0,5}$,$\overline{\Delta \mathbf{L}}^K_{0,5}$), ($\underline{\Delta \mathbf{L}}^K_{5,10}$,$\overline{\Delta \mathbf{L}}^K_{5,10}$), and ($\underline{\Delta \mathbf{L}}^K_{10,15}$,$\overline{\Delta \mathbf{L}}^K_{10,15}$) by applying (7) to (10) as shown in table 1. Then (step 2 of IQL) we apply (12) to obtain $\Delta \mathbf{L}^K_5$=1, $\Delta \mathbf{L}^K_{10}$=1, and then $\Delta \mathbf{L}^K_{15}$=0, respectively. Note from fig. 3 that $\mathbf{L}^K_5$ and $\mathbf{L}^K_{10}$ differ by one from the exact values while $\mathbf{L}^K_{15}$ matches the exact value. Finally, (step 3 of IQL) using (5), we obtain $\mathbf{L}^K_5$=2, $\mathbf{L}^K_{10}$=1, and $\mathbf{L}^K_{15}$=1.

4.2. THE GG1K ALGORITHM

The complete GG1K algorithm is given below:

Algorithm GG1K

input: $\langle \alpha_1, \alpha_N \rangle$, $\langle \delta_1, \delta_N \rangle$, K, A_1, L_0^∞.

output: $\langle E_1^\infty, E_{2N}^\infty \rangle$, $\langle \mathbf{L}_1^K, \mathbf{L}_{2N}^K \rangle$.

Phase I: G/G/1/∞ Simulation

1. Use algorithm GLM to compute $\langle E_1^\infty, E_{2N}^\infty \rangle$, and $\langle L_0^\infty, L_{2N}^\infty \rangle$.

2. If there exists no i that satisfies $L_i^\infty > K$, for $1 \le i \le 2N$, skip phase II.

Phase II: Transformation

1. Partition $\langle L_0^\infty, L_{2N}^\infty \rangle$ evenly into P batches such that each batch contains about the same number of events.

2. Use algorithm IQL (section 4.1) to compute $\mathbf{L}_{a_1}^K, \mathbf{L}_{a_2}^K, \ldots, \mathbf{L}_{a_{P-1}}^K$ (i.e., the A-G/G/1/K initial queue lengths for P-1 batches).

3. Each processor k, for $1 \le k \le P$, performs the following computation in parallel:

$$\mathbf{L}_i^K = \begin{cases} \mathbf{L}_{i-1}^K + 1 & \text{if } \mathbf{L}_{i-1}^K < K \text{ and event } i \text{ is an arrival,} \\ K & \text{if } \mathbf{L}_{i-1}^K = K \text{ and event } i \text{ is an arrival,} \\ \mathbf{L}_{i-1}^K - 1 & \text{if } \mathbf{L}_{i-1}^K > 0 \text{ and event } i \text{ is a departure,} \\ 0 & \text{if } \mathbf{L}_{i-1}^K = 0 \text{ and event } i \text{ is a departure.} \end{cases}$$

In the GG1K algorithm, phase I requires time $\Theta(N/P + \log N + \log P)$. In phase II, step 1 requires constant time and step 3 requires time $\Theta(N/P)$. Section 4.1 shows that step 2 requires time $\Theta(N/P + \log P)$. Therefore, in total, the GG1K algorithm requires time $\Theta(N/P + \log N + \log P)$, which is the same as the GLM algorithm. Thus, near-linear speedup can be achieved when $N \gg P$. In addition, because the GG1K algorithm allows arbitrary batch partitioning (section 4.1), "unbounded" parallelism (as defined in [8]) is achieved.

4.3. ANALYSIS OF THE APPROXIMATION TECHNIQUE

This section analyzes the approximation technique of the GG1K algorithm. Consider some simulation time t at which both the E-G/G/1/∞ simulation and the A-G/G/1/K simulation experience a departure.

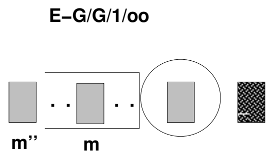

Fig. 5: Case 1: When job m'' arrives, the E-G/G/1/∞ queue length and the A-G/G/1/K queue length are both greater than zero.

Fig. 6: Case 2: When job m'' arrives, the E-G/G/1/∞ queue length and the A-G/G/1/K queue length are both zero.

Let jobs m and m' be the departing jobs, respectively. Recall from A2 that for each departure in a A-G/G/1/K trajectory, there exists a departure in the corresponding E-G/G/1/∞ trajectory at the same simulation time. Let job m'', where $m'' \geq m' + 1$, be the first non-lost job that arrives after job m' arrives in the A-G/G/1/K simulation. Computing the departure time of job m'' can be described by the following four cases:

Case 1: When job m'' arrives, the E-G/G/1/∞ queue length and the A-G/G/1/K queue length are both greater than zero (fig. 5).

In this case, the departure time of job m'' should occur at time $t + \delta_{m''}$ (i.e., $\delta_{m''}$ time units after the previous departure). However, the GG1K algorithm will assign $t + \delta_{m+1}$ as the departure time for job m''. If $m'' \neq m + 1$, the algorithm effectively *reassigns* the service time of job $m + 1$ to job m''. Assume that job service times are independent and identically distributed random variables and are uncorrelated with job arrival times. Then such re-association of service times is unbiased since the asymptotic job service time distribution will not be changed. Asymptotically, no approximation error is introduced in this case.

Case 2: When job m'' arrives, the E-G/G/1/∞ queue length and the A-G/G/1/K queue length are both zero (fig. 6).

In this case, the departure time of job m'' should occur at time $A_{m''} + \delta_{m''}$ (i.e., $\delta_{m''}$ time units after job m'' arrives). Since both systems are idle at $A_{m''}$, the GG1K algorithm will assign $A_{m''} + \delta_{m''}$ as the departure time for job m''. Therefore, no approximation error will be introduced.

Fig. 7: Case 3: When job m'' arrives, the E-G/G/1/∞ queue length is greater than zero while the A-G/G/1/K queue length is zero.

Case 3: When job m'' arrives, the E-G/G/1/∞ queue length is greater than zero while the A-G/G/1/K queue length is zero (fig. 7).

In this case, the departure time of job m'' should occur at time $A_{m''} + \delta_{m''}$ (i.e., $\delta_{m''}$ time units after job m'' arrives). Let t' denote the simulation time of the first departure that occurs in the E-G/G/1/∞ queue after job m'' arrives. In this case, the GG1K algorithm will assign t' as the departure time for job m''. In other words, the GG1K algorithm effectively replaces the service time of job m'' with the remaining service time of the job that is in service in the E-G/G/1/∞ server when job m'' arrives. If job service times are constant, the approximation is biased since a service time will be under calculated each time this case occurs.

Case 4: When job m'' arrives, the A-G/G/1/K queue length is greater than zero while the E-G/G/1/∞ queue length is zero.

This case is impossible with the GG1K algorithm.

Therefore, only case 3 causes an approximation error. As a result, the occurrence frequency of case 3 governs the accuracy of the GG1K simulation. In the rest of this paper, case 3 is referred to as the *case-3 condition*. Also, $A_{m''}$ is referred to as the *occurrence time* of the case-3 condition, and job m'' is said to *cause* the case-3 condition.

4.4. ERROR ANALYSIS

This section considers the approximation error of the GG1K algorithm in terms of the number of job losses for G/D/1/K queues. Let A_c and $A_{c'}$ be two simulation times at which the following conditions are satisfied:

1. $A_c < A_{c'}$.

2. Case-3 condition occurs (hence a job arrives at A_c and $A_{c'}$, respectively).

3. The E-G/G/1/K queue is empty immediately before the time (i.e., $L(A_c^-) = 0$ and $L(A_{c'}^-) = 0$).

4. There exists no other simulation time that satisfies conditions two and three.

Let θ and θ^K denote the exact and the approximate *loss count* (i.e., the number of lost jobs) computed by the GG1K algorithm, respectively, in the time interval $(A_c, A_{c'})$. Then for any pair of A_c and $A_{c'}$, the following lemma holds:

Lemma 2 *If* $\delta_0 = \delta_1 =, \ldots, \delta_N$, $\theta^K \leq \theta \leq \theta^K + 1$.

Proof: See Appendix B.

Let $\Delta\theta^K_{t,t'}$ denote the amount by which the E-G/G/1/K loss count exceeds the A-G/G/1/K loss count in the interval $[t, t']$. Also, let $C_{t,t'}$ denote the number of case-3 condition occurrences in $[t, t']$, and I_0 be the number of times that the E-G/G/1/K queue becomes idle in $[0, \tau]$. The following lemma gives an upper bound on the approximation error in terms of the total loss count for the entire simulation.

Lemma 3 $0 \leq \Delta\theta^K_{0,\tau} \leq \min\{I_0, C_{0,\tau}\}$.

Proof: Follows from Lemma 2.

Lemma 3 suggests that the GG1K algorithm produces less error when the value of λ/μ is either very large or small, where λ and μ are the average job arrival rate and service rate, respectively. This is because when λ/μ is large (e.g., 2) both the A-G/G/1/K queue and the E-G/G/1/K queue will rarely become idle. Thus, both I_0 and $C_{0,\tau}$ will be small. When λ/μ is small (e.g., 0.5), the G/G/1/∞ system *regenerates* (i.e., becomes idle) more frequently so that the probability of G/G/1/∞ queue length being greater than the A-G/G/1/K queue length (required by the case 3 condition) is low. Thus, $C_{0,\tau}$ will be small.

4.5. APPLICATION

4.5.1. Single server queues

The GG1K simulation is compared with a sequential simulation for a set of G/G/1/K queues. Normalized approximation errors of the average queue lengths for three models are shown in table 2, in which $\Gamma(2)$ represents a gamma distribution with a shape parameter equal to 2.

The results show that the approximation errors are negligible except when the buffer size is smaller than 5 for $\Gamma(2)/\Gamma(2)$/1/K and M/D/1/K models. The errors decrease as the value of K increases. Also, for all values of K, the approximation errors peak when the ratio of λ/μ is 1. This is because the case-3 condition occurs more frequently as λ/μ approaches 1. Consequently the error increases. Also, as indicated in section 4.3, when the service times are constant, the approximation is biased. Thus, the results for the M/D/1/K model in table 2 are in general the worst. Because G/D/1/K is an important class of queueing

Table 2

Normalized approximation errors on the average queue length using the GG1K algorithm. Each table entry is the average value of 100*(exact value - approximate value)/(exact value) obtained from 10 simulation runs of 10^6 jobs each.

	λ/μ	K=2	K=5	K=20	K=100
	.1	0.03	0.0	0.0	0.0
	.5	-0.01	0.01	0.0	0.0
M/M/1/K	.9	0.17	0.83	-0.35	0.0
	1.0	1.2	0.39	1.35	0.49
	1.3	0.17	0.38	0.08	0.16
	.1	0	0.0	0.0	0.0
	.5	0.93	0.007	0.0	0.0
$\Gamma(2)/\Gamma(2)/1/K$	.9	5.99	2.24	0.22	0.0
	1.0	7.68	3.84	0.99	0.002
	1.3	4.64	1.04	-0.005	0.005
	.1	0.028	0.0	0.0	0.0
	.5	2.90	0.80	0.0	0.0
M/D/1/K	.9	12.5	4.66	0.17	0.0
	1.0	15.32	8.0	2.22	0.36
	1.3	9.0	1.73	0.02	0.003

models (e.g., useful in modeling ATM switches), we evaluate the GG1K algorithm by also comparing the errors on job loss rates and average job waiting times with sequential simulations for some G/D/1/K queues. Table 3 shows results for some M/D/1/K queues. The GG1K algorithm produces very accurate results except when the buffer size is very small (e.g., $K = 5$). The results for the $\Gamma(2)$/D/1/K model are similar to those shown in table 3.

4.5.2. Feed-forward queueing networks

The GG1K algorithm can simulate acyclic feed-forward, finite buffer queueing networks. In an acyclic feed-forward queueing network, jobs enter the system from *source* queues and depart the system from *destination* queues. Because the network contains no cycles, each job visits each queue at most once. There are three basic components for feed-forward networks: fork, merge, and tandem. All feed-forward queueing networks are compositions of these three basic structures.

We present results of simulating three networks (a fork, merge, and tandem network) using the GG1K algorithm. Each of these three models contains seven homogeneous M/M/1/K queues (fig. 8). We assume that a job departing from a fork queue has equal probability of joining any of the queues connected to the server output. Also, the service times of a job at each server are *independent.* That is, each job is assigned with a new service time when arriving at a server. For any of these three models, because there are no cycles, the GG1K algorithm can be applied queue by queue in a breadth-first order starting from any of the the source queues.

In our experiments, we let $\lambda = 100$ for all source queues and $\mu = 100$, $K = 10$ for all queues. Each

Table 3

A comparison of normalized approximation errors on job loss rate and average waiting time for M/D/1/K queues for $N = 10^5$. The numbers in column 'SEQ' are the estimates (with 90% confidence intervals) obtained from sequential simulations, and each number in columns 'GG1K' is the average value of 100*(approximate value - sequential estimate) /(sequential estimate) from 10 simulation runs.

		Loss Rate		Avg. Waiting Time	
	λ/μ	SEQ(90%c.i.)†	GG1K	SEQ(90%c.i.)†	GG1K
K=5	.3	$9.8(\pm 2)\times 10^{-3}$	.0	1.215	-.08
	.5	.22(±.02)	-.04	1.484	-.13
	.9	6.5(±.06)	-5.46	2.527	-4.94
	.95	8.3(±.07)	-7.27	2.685	-6.92
	1.0	10.4(±.07)	-9.35	2.841	-8.87
	2.0	50.0(±.06)	-.03	4.375	-.04
K=10	.3	0	0	1.215	.0
	.5	$5.0(\pm 5)\times 10^{-4}$	.0	1.502	.0
	.9	1.7(±.03)	-1.35	3.930	-10^{-3}
	.95	3.1(±.05)	-2.74	4.618	-2.70
	1.0	5.1(±.07)	-4.50	5.345	-4.62
	2.0	50.0(±.06)	.0	9.371	-.04
K=20	.3	0	0	1.215	.0
	.5	0	0	1.502	.0
	.9	.20(±.01)	-.14	5.189	-.13
	.95	.86(±.04)	-.72	7.415(±.1)	-.76
	1.0	2.6(±.07)	-2.18	10.379(±.1)	-2.22
	2.0	50.0(±.06)	.0	19.362	.04
K=100	.3	0	0	1.215	.0
	.5	0	0	1.502	.0
	.9	0	0	5.602(±.1)	.0
	.95	0	0	10.89(±.89)	.0
	1.0	.58(±.10)	-.27	51.40(±3.9)	-.34
	2.0	49.9(±.06)	.0	99.211	.0

† The 90% half confidence intervals of those not shown are negligible.

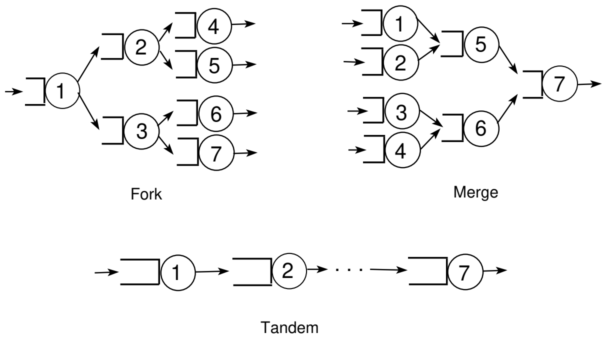

Fig. 8: Feed-forward queueing models. Each model contains 7 queues.

Table 4

A comparison of the queue lengths for the feed-forward queueing network models in which K=10 for all queues. Each number is the average of 10 runs.

	Fork		Merge		Tandem	
Queue	GG1K	Exact	GG1K	Exact	GG1K	Exact
1	5.00	5.01	5.02	5.01	4.97	4.99
2	0.82	0.82	5.01	5.02	4.05	4.09
3	0.82	0.81	4.95	5.04	3.59	3.58
4	0.29	0.29	5.05	5.00	3.30	3.30
5	0.29	0.29	8.79	8.79	3.08	3.09
6	0.29	0.29	8.79	8.78	2.93	2.92
7	0.29	0.29	8.96	8.97	2.79	2.77

model is simulated 10 times. Each run simulates 10^5 external arrivals. In the merge network model, queues 1 to 4 are independent source queues, each admitting 25,000 arrivals for each simulation run. Table 4 compares the mean number of jobs in the system from the GG1K simulation to an exact simulation. The table shows that the GG1K simulations produce very accurate results for all three models. The approximation errors are mostly well under 1%, and never exceeds 1.8%.

5. The GD1K Algorithm

We have shown that the GG1K algorithm generates more significant errors for G/D/1/K queues with a small K (e.g., not exceeding 5). This section extends the GG1K algorithm and presents a GD1K algorithm for G/D/1/K queues that can produce highly accurate results for any value of K. The GD1K algorithm improves the accuracy of the GG1K algorithm by accurately estimating the initial queue length for each batch.

Before deriving the GD1K algorithm, we first examine the differences between an A-G/D/1/K trajectory and its corresponding E-G/D/1/K trajectory. It can be observed that when no case-3 condition occurs during the simulation, these two trajectories are the same. Therefore, we consider only the situations where the case-3 condition occurs at least once. Suppose job k causes the first occurrence of case-3 condition. Then the GG1K algorithm will effectively *underestimate* the service time of job k by some amount $\delta'_k < \delta$, where δ is the job service time. Thus, in the A-G/D/1/K trajectory the departure time of job k should be *delayed* for $d_k = \delta'_k$ time units to compensate the error. This delay should also be *propagated* along the A-G/D/1/K trajectory for the subsequent departures by following the rules below:

R1: The propagation of a departure delay should stop at the point when the the corresponding E-G/G/1/∞ queue becomes empty.

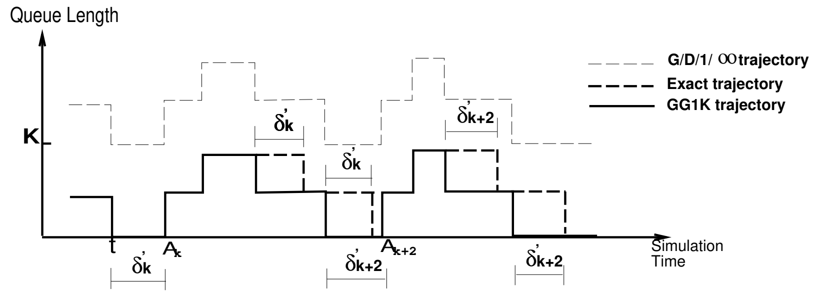

Figure 9: At A_k a case-3 condition occurs. The departure delay δ'_k propagates along the trajectory until another case-3 condition occurs at A_{k+2} causing a new departure delay δ'_{k+2}.

R2: When a job k' ($k' > k$) which causes another case-3 condition arrives, and of the shortfall of the service time $\delta'_{k'}$ is greater than the current delay d_k, then $\delta'_{k'}$ should be used as the delay for the subsequent departures until either R1 or R2 condition is met again.

For R1, because when the E-G/G/1/∞ queue becomes empty the A-G/D/1/K queue will become empty too. Therefore, the effect (which causes the delay propagation) of the case-3 condition disappears. To illustrate R2, we consider the example shown in Figure 9, in which a case-3 condition occurs at A_{k+2} and the shortfall of the service time (of job $k+2$) δ'_{k+2} is greater than the current propagation delay δ'_k. Therefore, δ'_{k+2} should be used as the new departure time delay for the subsequent jobs (including job $k+2$) until either R1 or R2 condition is met again. It is easy to show that the modified trajectory resulted from R1 and R2 is exact.

The system state of the G/D/1/K queue can be specified by a 2-tuple: {L(t), FRST(t)}, where L(t) and FRST(t) denote the length and the first job remaining service time of the queue at simulation time t, respectively. The GD1K algorithm is based on a key property of the GG1K algorithm: In any A-G/D/1/∞ trajectory, the queue length immediately after any departure event is either the same or exactly one less than the exact queue length. This is formally stated in the following lemma:

Lemma 4 *For any A-G/D/1/K trajectory, if $E_j^{\infty+}(1 \le j \le 2N)$ corresponds to a departure event in the A-G/D/1/K trajectory, then $L(E_j^{\infty+}) - 1 \le \mathbf{L}_j^K \le L(E_j^{\infty+})$.*

Proof: See Appendix C.

Lemma 4 enables us to predict the queue length at any GG1K-computed departure time with a maximum error of one. Given lemma 4, the GD1K algorithm can be described as follows:

Algorithm GD1K

1. Compute an A-G/D/1/K trajectory using the GG1K algorithm. If no case-3 condition occurs in the

resultant GD1K trajectory, the trajectory is exact; hence exit.

2. Partition the resultant A-G/D/1/K trajectory (approximately) evenly into P batches such that each pair of batches are partitioned at a time point immediately after an occurrence of a (GG1K-computed) departure.

3. For each batch i, $1 \leq i \leq P$, apply lemma 4 to compute the batch initial queue lengths $\{\ell_i - 1, \ell_i\}$, where the exact initial queue length is in $\{\ell_i - 1, \ell_i\}$.

4. For each processor $i, 1 \leq i \leq P$, sequentially compute batch i using the initial states $\{\ell_i, \delta\}$, and $\{\ell_i - 1, \delta\}$, respectively.

5. Construct a complete trajectory in the following manner: Let B_i and B_i' be the subtrajectories computed by processor i obtained from the previous step. For all $1 \leq i < P$, select a subtrajectory $B_i'' \in \{B_i, B_i'\}$ such that the final queue length of B_i'' is equal to the initial queue length of B_{i+1}''.

In step 4, we approximate the first job remaining service time with δ. Thus, the GD1K algorithm produces approximate results. Nevertheless, because each batch is re-computed using a sequential simulation, case-3 approximation error is eliminated. The total execution time required by the GD1K algorithm is $O(N/P + P + logP + logN)$, in which step 1, steps 2, 3, and 4 combined, and step 5 take time $O(N/P + logP + logN)$, $O(N/P)$, and $O(P)$, respectively. When $N \gg P$, near-linear speedup can be obtained.

Table 5 shows some experimental results for the M/D/1/K model. It shows that the GD1K algorithm obtains near-exact results despite the fact that only an average of 100 and 1000 arrivals, respectively, are simulated by each processor for P=1000 and P=100. When K is small, the accuracy significantly improves over the GG1K algorithm. However, the approximation error grows as P increases due to the increasing effect of the approximate initial FRST.

6. Concluding Remarks

Two approximate time-parallel simulation algorithms (GG1K and GD1K) for FCFS G/G/1/K and G/D/1/K queues, respectively, are proposed. The algorithms exploit massive parallelism (unbounded for the GG1K algorithm) and hence are suitable for massively parallel computers. Experimental results show that the GG1K algorithm can achieve highly accurate results except when both K is small (e.g., 5) and the job service times are constant. The GD1K algorithm is developed for such case. When $N \gg P$, both algorithms can achieve near-linear speedup.

Table 5: Normalized approximation errors of the GD1K algorithms in job loss rate and in average waiting time for M/D/1/K queues for $N = 10^5$. The numbers in column 'SEQ' are the estimates (with 90% confidence intervals) obtained from a sequential simulation, and each number in column 'GD1K' is the average value of 100*(approximate value - sequential estimate) /(sequential estimate) from 10 simulation runs.

			Loss Rate		Avg. Waiting Time	
		λ/μ	SEQ(90%c.i.)†	GD1K	SEQ(90%c.i.)†	GD1K
P= 100	K=5	.3	$9.8(\pm 2)\times 10^{-3}$	.0	1.215	.0
		.5	$.22(\pm .02)$	-.04	1.484	.0
		.9	$6.5(\pm .06)$	.05	2.527	.0
		.95	$8.3(\pm .07)$	.07	2.685	.0
		1.0	$10.4(\pm .07)$	.13	2.841	.07
		2.0	$50.0(\pm .06)$	.02	4.375	.04
	K=10	.3	0	0	1.215	0
		.5	$5.0(\pm 5)\times 10^{-4}$	.0	1.502	.0
		.9	$1.7(\pm .03)$	.04	3.930	.0
		.95	$3.1(\pm .05)$	.09	4.618	.06
		1.0	$5.1(\pm .07)$	.34	5.345	.11
		2.0	$50.0(\pm .06)$	-10^{-4}	9.371	.0
	K=20	.3	0	0	1.215	0
		.5	0	0	1.502	0
		.9	$.20(\pm .01)$	.0	5.189	.0
		.95	$.86(\pm .04)$	0.10	$7.415(\pm .1)$	.02
		1.0	$2.6(\pm .07)$	.72	$10.379(\pm .1)$	.24
		2.0	$50.0(\pm .06)$	-10^{-4}	19.362	.04
	K=100	.3	0	0	1.215	0
		.5	0	0	1.502	0
		.9	0	0	$5.602(\pm .1)$	0
		.95	0	0	$10.89(\pm .89)$	0
		1.0	$.58(\pm .10)$	1.5	$51.40(\pm 3.9)$	.39
		2.0	$49.9(\pm .06)$	-10^{-4}	99.211	.0
P=1000	K=5	.3	$9.8(\pm 2)\times 10^{-3}$	-1.02	1.215	-.16
		.5	$.22(\pm .02)$	-.22	1.484	-.13
		.9	$6.5(\pm .06)$	.39	2.527	.11
		.95	$8.3(\pm .07)$	.80	2.685	.29
		1.0	$10.4(\pm .07)$	1.36	2.841	.63
		2.0	$50.0(\pm .06)$	.22	4.375	.22
	K=10	.3	0	0	1.215	0
		.5	$5.0(\pm 5)\times 10^{-4}$	.0	1.502	.0
		.9	$1.7(\pm .03)$	.26	3.930	.07
		.95	$3.1(\pm .05)$	1.08	4.618	.38
		1.0	$5.1(\pm .07)$	3.04	5.345	1.10
		2.0	$50.0(\pm .06)$	-10^{-3}	9.371	.0
	K=20	.3	0	0	1.215	0
		.5	0	0	1.502	0
		.9	$.20(\pm .01)$	-.14	5.189	-.03
		.95	$.86(\pm .04)$	1.02	$7.415(\pm .1)$	.24
		1.0	$2.6(\pm .07)$	5.77	$10.379(\pm .1)$	1.81
		2.0	$50.0(\pm .06)$	-10^{-3}	19.362	.04
	K=100	.3	0	0	1.215	0
		.5	0	0	1.502	0
		.9	0	0	$5.602(\pm .1)$	0
		.95	0	0	$10.89(\pm .89)$	0
		1.0	$.58(\pm .10)$	4.19	$51.40(\pm 3.9)$	.55
		2.0	$49.9(\pm .06)$	-10^{-3}	99.211	.0

† The 90% half confidence intervals of those not shown are negligible.

References

[1] S.G. Akl, *The Design and Analysis of Parallel Algorithms* (Prentice Hall, 1989).

[2] S. Andradóttir and T.J. Ott, Parallel Simulation of Communication Networks through Time Segmentation, working paper.

[3] V.C. Bhavsar and J.R. Isaac, Design and Analysis of Parallel Monte Carlo Algorithms, SIAM J. Sci. Stat. Comput. 8(1987)s73-s95.

[4] K.M. Chandy and R. Sherman, Space-Time and Simulation, *Proc. of the 1989 SCS Multiconference on Distributed Simulation* (1989), pp. 53-57.

[5] C. Chang and P. Heidelberger, Fast Simulation of Packet Loss Rates in a Shared Buffer Communications Switch, ICASE Report 93-79, ICASE, Hampton, VA (1993).

[6] R.M. Fujimoto, Parallel Discrete Event Simulation, Commun. ACM 33(1990)31-53.

[7] P. Glynn and P. Heidelberger, Analysis of Parallel Replicated Simulations Under a Completion Time Constraint, ACM TOMACS 1(1991)3-23.

[8] A.G. Greenberg, B.D. Lubachevsky and I. Mitrani, Unboundedly Parallel Simulations via Recurrence Relations, *Proc. of the Conf. on Measurement and Modeling of Computer Systems* (1990), pp. 1-12.

[9] P. Heidelberger, Statistical Analysis of Parallel Simulations, *Proc. of the 1986 Winter Simulation Conference* (1986), pp. 290-295.

[10] P. Heidelberger, Discrete Event Simulations and Parallel Processing: Statistical Properties, SIAM J. Stat. Comput. 9(1988)1114-1132.

[11] P. Heidelberger and H.S. Stone, Parallel Trace-Driven Cache Simulation by Time Partitioning, *Proc. of the 1990 Winter Simulation Conference* (1990), pp. 734-737.

[12] R.E. Lander and M.J. Fischer, Parallel Prefix Computation, J. of the ACM 27(1980)831-838.

[13] J. Lane, ATM Knits Voice, Data on Any Net, IEEE Spectrum 2(1994)42-45.

[14] Y.B. Lin and E.A. Lazowska, Time-Division Algorithm for Parallel Simulation, ACM TOMACS 1(1991)73-83.

[15] Y.B. Lin, Parallel Trace-Driven Simulation for Packet Loss in Finite-Buffered Voice Multiplexers, Parallel Computing 19(1993)219-228.

[16] D. Mitra and I. Mitrani, Control and Coordination Policies for System with Buffers, ACM SIGMETRICS Performance Evaluation Review 17(1989)156-164.

[17] I. Nikolaidis and R. Fujimoto, Parallel Simulation of High-Speed Network Multiplexers, *IEEE Conference on Decision and Control* 3(1993), pp. 2224-2229.

[18] H.S. Stone, *High Performance Computer Architecture* (Addison Wesley, 1987).

[19] W. Verbiest, L. Pinnoo and B. Voeten, The Impact of the ATM concept on Video Coders, IEEE Journal on Selected Areas in Communications 6(1988)1623-1631.

[20] J.J. Wang and M. Abrams, Determining Initial States for Time-Parallel Simulations, *Proc. of the 1993 Workshop on Parallel and Distributed Simulation* (1993), pp. 19-26.

A Proof of Lemma 1

The correctness of (11) can be proven by showing that for any pair of integers a and a', for $1 \le a \le a' \le 2N$, given $\mathbf{L}_a^K$ the following equation always holds:

$$\Delta\mathbf{L}_{a'}^K = \max\{\min\{\Delta\mathbf{L}_a^K, \overline{\Delta\mathbf{L}}_{a,a'}^K\}, \underline{\Delta\mathbf{L}}_{a,a'}^K\}. \tag{13}$$

The proof uses induction on a'. It is clear that equation (13) holds (induction base) for $a = a'$ for any $1 \le a \le 2N$ because $\underline{\Delta\mathbf{L}}_{a,a}^K \le \Delta\mathbf{L}_a^K \le \overline{\Delta\mathbf{L}}_{a,a}^K$. Assume that equation (13) holds for some integer y satisfying $a < [a' = y]$ (induction hypothesis). Then:

$$\Delta\mathbf{L}_y^K = \max\{\min\{\Delta\mathbf{L}_a^K, \overline{\Delta\mathbf{L}}_{a,y}^K\}, \underline{\Delta\mathbf{L}}_{a,y}^K\}. \tag{14}$$

The proof completes if we show that (13) holds for $a' = y+1$. Consider event $y+1$. There are four possible cases:

Case 1: Event $y+1$ is an arrival and $\mathbf{L}_y^K < K$

In this case, $L_{y+1}^\infty = L_y^\infty + 1$. From A1, $\mathbf{L}_{y+1}^K{=}\mathbf{L}_y^K{+}1$ and hence $\Delta\mathbf{L}_{y+1}^K{=}\Delta\mathbf{L}_y^K$. Because $\mathbf{L}_y^K < K$, from (6) and (5), we have $\overline{\Delta\mathbf{L}}_{a,y}^K \ge \Delta\mathbf{L}_y^K \ge L_y^\infty - K + 1 = L_{y+1}^\infty - K$. Thus, $\overline{\Delta\mathbf{L}}_{a,y}^K \ge \Delta\mathbf{L}_y^K \ge L_{y+1}^\infty - K$. Applying (10) yields:

$$\overline{\Delta\mathbf{L}}_{a,y+1}^K = \overline{\Delta\mathbf{L}}_{a,y}^K. \tag{15}$$

Because event $y+1$ is an arrival, from (9), we have:

$$\underline{\Delta\mathbf{L}}_{a,y+1}^K \ge \underline{\Delta\mathbf{L}}_{a,y}^K. \tag{16}$$

Now consider ΔL_a. There are three possible situations:

1. $\Delta L_a < \Delta\mathbf{L}_y^K$**:** From (6) and (14), $\underline{\Delta\mathbf{L}}_{a,y}^K{=}\Delta\mathbf{L}_y^K$. Because $\underline{\Delta\mathbf{L}}_{a,y+1}^K \le \Delta\mathbf{L}_{y+1}^K$, from (16), $\underline{\Delta\mathbf{L}}_{a,y+1}^K = \underline{\Delta\mathbf{L}}_{a,y}^K$. Thus, (13) yields $\Delta\mathbf{L}_{y+1}^K = \underline{\Delta\mathbf{L}}_{a,y+1}^K = \Delta\mathbf{L}_y^K$.

2. $\Delta L_a > \Delta\mathbf{L}_y^K$**:** From (6) and (14), $\overline{\Delta\mathbf{L}}_{a,y}^K{=}\Delta\mathbf{L}_y^K$. From (15), $\overline{\Delta\mathbf{L}}_{a,y+1}^K = \overline{\Delta\mathbf{L}}_{a,y}^K$. Thus, (13) yields $\Delta\mathbf{L}_{y+1}^K = \overline{\Delta\mathbf{L}}_{a,y+1}^K = \Delta\mathbf{L}_y^K$.

3. $\Delta L_a = \Delta\mathbf{L}_y^K$**:** From (15) and (14), $\underline{\Delta\mathbf{L}}_{a,y}^K{\le}\Delta\mathbf{L}_y^K{\le}\overline{\Delta\mathbf{L}}_{a,y}^K$. But because $\overline{\Delta\mathbf{L}}_{a,y}^K{=}\overline{\Delta\mathbf{L}}_{a,y+1}^K$ (15) and $\underline{\Delta\mathbf{L}}_{a,y+1}^K{\le}\Delta\mathbf{L}_y^K$, (13) yields $\Delta\mathbf{L}_{y+1}^K{=}\Delta\mathbf{L}_y^K$.

Case 2: Event $y+1$ is an arrival and $\mathbf{L}_y^K{=}K$

In this case, $L_{y+1}^\infty = L_y^\infty + 1$. From A1, $\mathbf{L}_{y+1}^K = \mathbf{L}_y^K = K$. Thus, $\Delta\mathbf{L}_{y+1}^K = \Delta\mathbf{L}_y^K + 1 = L_{y+1}^\infty - K$. Now consider ΔL_a. There are three possible situations:

1. $\Delta L_a < \Delta\mathbf{L}^K_y$**:** From (6) and (14), $\underline{\Delta\mathbf{L}}^K_{a,y} = \Delta\mathbf{L}^K_y = L^\infty_y - K$. From (9), $\underline{\Delta\mathbf{L}}^K_{a,y+1} = L^\infty_y + 1 - K = \underline{\Delta\mathbf{L}}^K_{a,y}+1$. Thus, (13) yields $\Delta\mathbf{L}^K_{y+1} = \underline{\Delta\mathbf{L}}^K_{a,y+1} = \underline{\Delta\mathbf{L}}^K_{a,y}+1 = \Delta\mathbf{L}^K_y+1$.

2. $\Delta L_a > \Delta\mathbf{L}^K_y$**:** From (6) and (14), $\overline{\Delta\mathbf{L}}^K_{a,y}=\Delta\mathbf{L}^K_y=L^\infty_y - K$. From (10), $\overline{\Delta\mathbf{L}}^K_{a,y+1}= L^\infty_y + 1 - K$ $=\overline{\Delta\mathbf{L}}^K_{a,y}+1$. Thus, (13) yields $\Delta\mathbf{L}^K_{y+1} =\overline{\Delta\mathbf{L}}^K_{a,y+1}= \overline{\Delta\mathbf{L}}^K_{a,y}+1= \Delta\mathbf{L}^K_y+1$.

3. $\Delta L_a = \Delta\mathbf{L}^K_y$**:** From (6) and (14), $\underline{\Delta\mathbf{L}}^K_{a,y}\leq\Delta\mathbf{L}^K_y= L^\infty_y - K$. From (9), $\underline{\Delta\mathbf{L}}^K_{a,y+1}=L^\infty_y + 1 - K$. Thus, (13) yields $\Delta\mathbf{L}^K_{y+1} = \underline{\Delta\mathbf{L}}^K_{a,y+1}= L^\infty_y + 1 - K =\Delta\mathbf{L}^K_y+1$.

An analogous argument holds for case 3 (event $y + 1$ is a departure and $\Delta L^K_y > 0$) and case 4 (event $y + 1$ is a departure and $\mathbf{L}^K_y = 0$). □

B Proof of Lemma 2

Let δ denote the constant job service time. Let $\hat{\delta}(t)$ denote the remaining service time of the first job in the E-G/G/1/K queue at simulation time t. That is, $\hat{\delta}(t)$ is the amount of time between t and the next departure time. Define $R(t)$ to be the *total remaining service time* of all jobs in the E-G/G/1/K queue at simulation time t. Formally,

$$R(t) = \begin{cases} \hat{\delta}(t) + (L(t) - 1)\delta & L(t) \geq 1, \\ 0 & L(t) = 0. \end{cases}$$

Analogously, define $\mathbf{R}^K(t)$ to be the total remaining service time of all jobs in the A-G/G/1/K queue at simulation time t. Let $\mathbf{L}^K(t)$ denote the A-G/G/1/K queue length at simulation time t. Then the following corollary holds:

Corollary 1 *$R(t) \geq \mathbf{R}^K(t) \Rightarrow L(t) \geq \mathbf{L}^K(t)$; $R(t) \leq \mathbf{R}^K(t) \Rightarrow L(t) \leq \mathbf{L}^K(t)$.*

A proof of Corollary 1 follows directly from the fact that the service times are constant.

Suppose a case-3 occurs at A_j (hence $\mathbf{L}^K(A_j^-)$=0). Assume $L(A_j^-) = 0$. Based on the GG1K transformation technique (section 4.3), the GG1K algorithm will *underestimate* the service time of job j by some amount δ', where $0 < \delta' < \delta$ (as illustrated in fig. 10). Then $R(A_j^+) = \mathbf{R}^K(A_j^+) + \delta'_j$.

Corollary 2 *For all t in the time interval $(A_c, A_{c'})$, exactly one of the following situations must be true:*

S1: *$0 \leq R(t) - \mathbf{R}^K(t) < \delta$ and $\Delta\theta^K_{A_c,t}$=0.*

S2: *$0 < \mathbf{R}^K(t) - R(t) < \delta$ and $\Delta\theta^K_{A_c,t}$=1.*

Corollary 2 can be proved by induction on t. Because $R(A_c^+) > \mathbf{R}^K(A_c^+)$ and $\Delta\theta^K_{A_c^+,A_c^+}$=0, S1 holds at A_c^+ (induction base). Assume that an event occurs at some time t, $A_c^+ < t < A_{c'}^-$, and S1 or S2 is true at

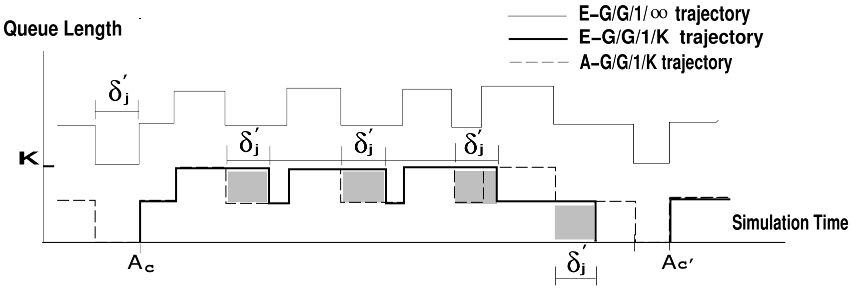

Fig. 10: Sample trajectories for E-G/G/1/∞, E-G/G/1/K, and A-G/G/1/K queues.

t^+ (induction hypothesis). Now, consider time t', $t < t' < A_{c'}$, when the next event occurs. Then this event can be an arrival (to both the E-G/G/1/K and the A-G/G/1/K queues) or a departure (to either or both the E-G/G/1/K and the A-G/G/1/K queues). For the event that occurs at t', there are four cases:

Case 1: S1 holds at t^+ and an arrival occurs (to both queues) at t':

S1 holds in $[t^+, t']$. From Corollary 1, if the job that arrives at t' is lost in the E-G/G/1/K queue and is not lost in the A-G/G/1/K queue, then S2 holds at t'^+. Otherwise, S1 holds at t'^+.

Case 2: S1 holds at t^+ and a departure occurs (to either queue) at t':

From Corollary 1, S1 holds in $[t^+, t'^+]$.

Case 3: S2 holds at t^+ and an arrival occurs (to both queues) at t':

S2 holds in $[t^+, t']$. From Corollary 1, if the job that arrives at t' is lost in the A-G/G/1/K queue and is not lost in the E-G/G/1/K queue, then S1 holds at t'^+. Otherwise, S2 holds at t'^+.

Case 4: S2 holds at t^+ and a departure occurs (to either queue) at t':

From Corollary 1, S2 holds in $[t^+, t'^+]$.

Therefore, Corollary 2 holds. Note that because both queues are empty at A_c^- and $A_{c'}^-$, the jobs arrive at A_c and $A_{c'}$ are not lost. Therefore, Lemma 2 follows directly from Corollary 2. $\square$.

C Proof of Lemma 4

Proof: A proof can be carried out by induction on departure events. Suppose event j_0 is the first departure event in the corresponding E-G/D/1/∞ trajectory. It is clear that the $\mathbf{L}_{j_0}^K = L(E_{j_0}^{\infty+})$ (induction base). Let event j and j', $j < j'$, be any two consecutive departure events in the E-G/D/1/∞ trajectory. Assume that $L(E_j^{\infty+}) - 1 \leq \mathbf{L}_j^K \leq L(E_j^{\infty+})$ holds for some j, $1 \leq j < N$ (induction hypothesis). The proof completes if we show that $L(E_{j'}^{\infty+}) - 1 \leq \mathbf{L}_{j'}^K \leq L(E_{j'}^{\infty+})$ holds. Let n_j denote the number of jobs that arrive in the simulation time interval $(E_j, E_{j'}]$. If $n_j = 0$, clearly, $L(E_{j'}^{\infty+}) - 1 \leq \mathbf{L}_{j'}^K \leq L(E_{j'}^{\infty+})$. If $n_j > 0$, there are two cases:

Case 1: $\mathbf{L}_j^K = L(E_j^{\infty+})$

If $n_j + L(E_j^{\infty+}) \leq K$, no job will be lost in $(E_j, E_{j'}]$ and $\mathbf{L}_{j'}^K = L(E_{j'}^{\infty+})$. If $n_j + L(E_j^{\infty+}) > K$, at least one job will be lost. Let d_j and a_j denote the departure delay for event j and the last job arrival time in $(E_j, E_{j'}]$, respectively. Then if $d_j = 0$, obviously, $\mathbf{L}_{j'}^K = L(E_{j'}^{\infty+})$. Otherwise, if $E_j^{\infty} + d_j > a_j$, $\mathbf{L}_{j'}^K = L(E_{j'}^{\infty+})$. On the other hand, if $E_j^{\infty} + d_j < a_j$, $\mathbf{L}_{j'}^K = L(E_{j'}^{\infty+}) - 1$.

Case 2: $\mathbf{L}_j^K = L(E_j^{\infty+}) - 1$

If $n_j + L(E_j^{\infty+}) \leq K$, no job will be lost in $[E_j, E_{j'})$ and $\mathbf{L}_{j'}^K = L(E_{j'}^{\infty+}) - 1$. Otherwise, at least one job will be lost. If $E_j^{\infty} + d_j > a_j$, $\mathbf{L}_{j'}^K = L(E_{j'}^{\infty+})$. On the other hand, if $E_j^{\infty} + d_j < a_j$, $\mathbf{L}_{j'}^K = L(E_{j'}^{\infty+}) - 1$.

□