# Efficient Distributed Queuing System Simulation

Tobias Kiesling
Institut für Technische Informatik

Thomas Krieger
Institut für Technik Intelligenter Systeme e.V.

Bericht 2006-01
Februar 2006

Fakultät für INFORMATIK

der Bundeswehr Universität München

Werner-Heisenberg-Weg 39 • D-85577 Neubiberg

# Efficient Distributed Queuing System Simulation

Tobias Kiesling[1] and Thomas Krieger[2]

[1] Fakultät für Informatik
Universität der Bundeswehr München
85577 Neubiberg, Germany
kiesling@informatik.unibw-muenchen.de
[2] Institut für Technik Intelligenter Systeme
Universität der Bundeswehr München
85577 Neubiberg, Germany
krieger@informatik.unibw-muenchen.de

**Abstract.** Queuing systems are an important building block for performance evaluation in various application areas, due to their powerful, yet simple nature. Although it is often possible to perform an analytical evaluation of a queuing model, simulation of queuing systems remains an important technique in the context of performance evaluation. In order to speed up queuing simulation executions, parallel and distributed simulation techniques have been devised. Unfortunately, existing methods are complex in nature, leading to increased development costs. Moreover, most of these approaches have been developed for tightly coupled parallel processing machines. Consequently, they are not suited for distributed execution. This paper investigates an alternative approach based on the technique of time-parallel simulation with fix-up computations. The salient features of this novel approach are its simplicity and its suitability for efficient distributed simulation execution.

## 1 Introduction

Queuing systems are an important tool in the area performance evaluation of computer and communication systems, especially as the building blocks of queuing networks. Although many queuing systems can be solved analytically, simulation remains an important tool for the computation of various statistics of queuing systems.

Parallel and distributed simulation techniques [1] are applied to decrease the run times of sequential simulation algorithms. This is desirable in cases, where the efficiency of sequential algorithms is not sufficient due to economical reasons or real-time constraints. Generally, there are two different approaches to exploit parallel processing power for a simulation problem: *Parallel replicated simulation* performs multiple replications of the same simulation model in parallel, *model parallelization* decomposes a simulation task in multiple subtasks for parallel or distributed simulation. Parallel replicated simulation is easy to implement, but its applicability is restricted, depending on the strength of the initial transient, the variability of the simulation, and the available simulation run length [2].

Furthermore, the execution of a parallel replicated simulation or of a parallelized model is classified by the utilized processing hardware. *Parallel simulation* is concerned with the concurrent execution of a simulation model on a tightly coupled parallel computer (e.g. employing shared memory). In *distributed simulation*, a model is executed concurrently on a number of independent computing resources communicating by the use of standard interconnection networks (e.g. clusters of workstations or grid computing [3]). This distinction is reasonable due to significant differences in the characteristics of the corresponding hardware. Distributed systems are characterized by high communication latencies and a general lack of shared memory. This has a significant impact on the applicability of various parallel algorithms.

In the field of parallel discrete-event simulation [4], the classical method for the parallelization of a model is a spatial model decomposition. The state space of the model is divided into a number of substates to be simulated concurrently by parallel processes. This requires decomposability of the model state space. Unfortunately, in many cases a model exhibits only a limited amount of decomposability. A single queuing system has a very simple state space, often consisting only of the number of jobs currently present. Furthermore, even in queuing networks with a more complex state space, the amount of parallelism achievable with spatial model decomposition is limited [5].

An alternative to spatial decomposition is the temporal decomposition of the simulation task [6]. The simulated time is split into a number of slices and the responsibility for the calculation of state changes inside each slice is assigned to a separate parallel process. This technique of *time-parallel simulation* is promising for the simulation of queuing systems, as the achievable degree of parallelism is not limited by the decomposability of the model state space. Therefore, all of the approaches for efficient queuing system simulation that have been developed, use a variant of time-parallel simulation. Greenberg et al. [7, 8] use parallel prefix computations [9, 10] to calculate G/G/1 queuing system dynamics via recurrence relations. Furthermore, they investigate techniques to apply this approach to several different types of queuing networks, including acyclic fork-join networks, series of queues with bounded buffers, closed cyclic networks, and several types of acyclic networks of G/G/1 queues. They also introduce the method of iterative folding, which can be used to simulate arbitrary G/G/1 queuing networks, albeit with varying efficiency. Andradottir et al. [11, 12] introduce a complex algorithm for the simulation of queuing networks with either loss or communication blocking. The structure of supported queuing networks is not restricted, but only markovian queues with bounded buffers are supported. Wang and Abrams [13, 14] modify the approach of Greenberg et al. for the simulation of bounded G/G/1/K queuing systems. Their approximate algorithm estimates the correct results in two phases: First, the G/G/1 algorithm of Greenberg et al. is applied. Then, the resulting trajectory of simulation states is transformed into an approximate G/G/1/K trajectory. Chen [15] presents an approach for the parallel simulation of G/G/1 queues and certain networks of these queues, including systems with loss or communication blocking. Instead of

using recursive equations, longest-path distances in directed graphs are utilized to compute queuing system statistics.

All of the mentioned approaches for parallel or distributed queuing system simulation are rather complex and thus difficult to be implemented efficiently on all types of parallel processing architectures. Furthermore, they have been designed for parallel simulation (i.e. for an execution on tightly coupled multiprocessor machines), thus not being well suited for distributed simulation (e.g. using grid computing resources). With the rising significance of distributed processing architectures, this happens to be a serious impediment for parallel or distributed queuing simulation. Therefore, an alternative approach for G/G/1 queuing system simulation based on time-parallel simulation with fix-up computations [16] is introduced in this paper. The salient features of this method are its simplicity and its suitability for distributed simulation.

The rest of the paper is structured as follows. Section 2 introduces the method of time-parallel simulation with fix-up computations. The central part of this paper is the definition of an efficient state match criterion and the proof of its correctness in Section 3. Section 4 gives indications on the performance of the introduced approach by the example of M/M/1 queues. Finally, Section 5 concludes the work.

## 2 Time-Parallel Simulation

In classical parallel simulation [4], the set of state variables of a simulation model is decomposed into subsets. Each of these is assigned to a logical process that manages the corresponding part of the global state. These logical processes are then executed concurrently on parallel processing nodes. The drawbacks of this approach are the introduction of an overhead for the synchronization between logical processes and the limited amount of achievable parallelism, which is restricted by the number of state variables and the decomposability of states in the model. Time-parallel simulation [6] is a different approach that decomposes the time axis and performs simulations of resulting time intervals in parallel. Afterwards, the results of all intervals are combined to create the overall simulation result. This has the potential for massive parallelism, as the maximum number of logical processes is determined by the number of possible time intervals, which is only restricted by the granularity of the time representation in the simulation implementation.

However, without further mechanisms, the final and initial states of adjacent time intervals do not necessarily coincide at interval boundaries, possibly resulting in incorrect state changes. Several different solutions of this problem have been proposed. Lin and Lazowska [17] introduce the notion of *regeneration points*, which are states that keep reoccurring throughout a simulation execution. If such a state can be identified a priori, a number of simulations can be executed concurrently, starting from the regeneration point and continuing until the regeneration point is reached again. Afterwards, the traces of the parallel simulations are concatenated to a correct trace of the simulation over the whole time

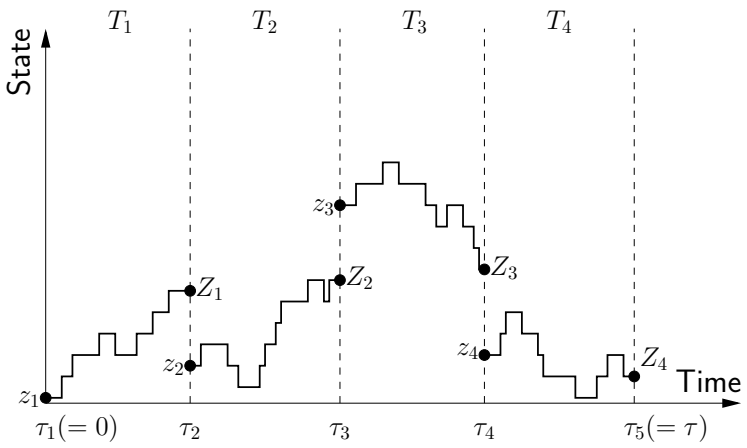


**Fig. 1.** Time-parallel simulation with guessed states

period. The drawback of this approach is the difficulty to identify regeneration points, especially for models with complex states. Among other applications, this approach was used successfully for the simulation of cascaded statistical multiplexers [18].

Heidelberger and Stone [16] introduce another solution using *fix-up computations*, which has been used for the simulation of caching in computer systems [16] and the simulation of CSMA/CD [19], among others. Compared to the approach utilizing regeneration points, fix-up computations are a more general construct, as they can be applied to almost any simulation model. Therefore, the rest of the paper is only concerned with time-parallel simulation using fix-up computations.

Let $T = [0, \tau]$ be the interval of the whole simulation time. $T$ is decomposed into the *time intervals* $T_1, \ldots, T_m$, where $T_k = [\tau_k, \tau_{k+1}]$, with $\tau_1 = 0 < \tau_2 < \ldots < \tau_m < \tau_{m+1} = \tau$, such that every point in simulated time is contained in some interval and the intervals overlap only at boundaries $\tau_2, \ldots, \tau_m$.

Now, simulations of time intervals are executed concurrently by corresponding processes $p_1, \ldots, p_m$ in the *initial simulation phase*. Unfortunately, the correct states at time interval boundaries $\tau_2, \ldots, \tau_m$ are unknown prior to the simulation. Therefore, these states are guessed and used as initial states $z_2, \ldots, z_k$ of the simulation processes. However, if the simulations are executed with these (possibly incorrect) states, the situation may occur, that the final state $Z_k$ of a simulation execution for time interval $T_k$ does not match the initial state $z_{k+1}$ that has been used for the simulation of the following time interval $T_{k+1}$. Figure 1 illustrates these issues.

If it is the case that $z_{k+1} \neq Z_k$ for at least one time interval $T_k$, overall simulation results might be incorrect. Therefore, a *fix-up phase* is utilized after the initial simulation phase to amend these illegal state changes. Fix-up computations for an interval $T_{k+1}$ (see Figure 2 for an illustration) are just a continuation

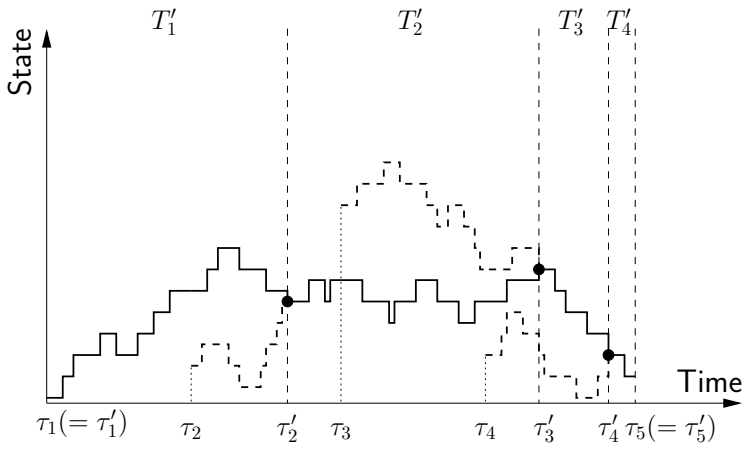


**Fig. 2.** Time-parallel simulation with fix-up computations

of the simulation of the preceding interval $T_k$ by process $p_k$ until a time $\tau'_k$, where the state of the initial simulation performed by process $p_{k+1}$ matches the corresponding state calculated during the fix-up computations by process $p_k$. As can be noted in Figure 2, the correct sequence of states results from a concatenation of the sequences of states of intervals $T'_k = (\tau'_k, \tau'_{k+1}]$ calculated by process $p_k$ for all $k \in \{1, \ldots, m\}$. An interval $[\tau_k, \tau'_k]$ can be interpreted in two ways: as the *fix-up phase* of process $p_{k-1}$ and as a *warm-up phase* of process $p_k$.

Figure 2 also shows the case, where state matching does not occur during the fix-up computations of process $p_2$ for interval $T_3$. In that case, process $p_2$ continues fix-up computations beyond $\tau_4$. State matching is now attempted against the sequence of states calculated by process $p_3$ for interval $T_4$. The fix-up phase $[\tau_3, \tau'_3]$ of process $p_2$ now overlaps with the warm-up phases of $p_3$ as well as $p_4$, and three different sequences of states have been calculated for the interval $[\tau_4, \tau'_3]$.

The amount of fix-up computations depends on the state calculations done by the corresponding processes. Hence, the length of interval $T'_k$ handled by a process $p_k$, is not known a priori and does not necessarily have the same length as the intervals of the other processes. The computational overhead $O$ of the parallel simulation can be defined as the lengths of fix-up phases of processes. It can be measured as the sum of the lengths of fix-up phases and expressed relative to the length $\tau$ of the whole simulation:

$$O := \frac{1}{\tau} \sum_{k=2}^{m} (\tau'_k - \tau_k) \,. \tag{1}$$

For equidistant process start times $\tau_k$, i.e. $\tau_k = (k-1)\frac{\tau}{m}$, (1) can be simplified to

$$O = \frac{1}{\tau}\sum_{k=2}^{m}\left(\tau'_k - (k-1)\frac{\tau}{m}\right) = \frac{1}{\tau}\sum_{k=2}^{m}\tau'_k - \frac{m-1}{2}\,. \tag{2}$$

The worst case for the overhead is that all processes perform fix-up computations until the end of the simulation time, i.e. for all $k \in \{2,\ldots,m\} : \tau'_k = \tau$. The overhead $\widetilde{O}$ in this case can be calculated from (2):

$$\widetilde{O} := \frac{1}{\tau}\sum_{k=2}^{m}\tau - \frac{m-1}{2} = \frac{m-1}{2}\,. \tag{3}$$

As an example, consider a simulation with $\tau = 100$ and four parallel processes ($m = 4$). In the worst case, $\widetilde{O} = 1.5$, i.e. the computational overhead for the parallel calculation is 1.5 times the length of the overall simulation interval. The total work done is 2.5 times the work in the sequential case. To determine the overall efficiency of the parallel algorithm, additional knowledge about the communication and synchronization overheads is necessary.

During the fix-up phase, fix-up computations are performed until the simulation state matches the corresponding state that had been calculated in the initial simulation phase. Therefore, a central aspect of fix-up computations is the detection of state matching. State matching might occur in both deterministic and stochastic simulation models, but in the latter case it is harder to anticipate and sometimes leads to obscure results. Fortunately, it is easy to transform a stochastic simulation model into an equivalent deterministic one by pre-sampling of random numbers. This leads to a trace-driven simulation, where the now *repeatable* simulation executions enable a direct comparison of the simulation states of initial simulation and corresponding fix-up computation. The details how this presampling can be performed depend on the specificities of the simulation model.

## 3 Parallel Queuing System Simulation

Using the method of time-parallel simulation with fix-up computations for the simulation of G/G/1 queuing systems is straightforward: Job arrival and service times are presampled and stored in an input trace. The trace is split into a number of subtraces to be simulated concurrently. For each parallel process, it is supposed that the system is empty just before the first job arrival of the corresponding subtrace occurs. Simulation is executed as usual until the end of the subtrace is reached. Now, each process performs fix-up computations by continuing the simulation with the input subtrace of the following parallel process until state matching occurs. Afterwards, the simulation results are collected from the processes and the overall simulation result is calculated. Two open issues still remain: the detection of state matching and the efficient collection of results from the parallel processes.

The rest of this section is concerned with the detection of state matching. In fact, state matching occurs exactly at the time when a process performing fix-up computations encounters an empty queue. It is easy to see that after this time, states will be identical. However, it can be shown that this is also the earliest time, when the state of the process performing fix-up computations matches the state calculated by the following process in the initial simulation phase. This result is stated by Theorem 1, which is derived hereafter.

### 3.1 Foundations

In the following, the problem of queuing simulation is reduced to the problem of determining the departure times of a number of jobs with associated arrival instants and service times. Based on this sequence of departure instants, the trajectory of the number of jobs in the system over time can be reconstructed easily [7]. The calculation of the job departure instants of a G/G/1 queue can be represented by a recursive function that relates a job with its departure time [20]. The basic observation underlying this formulation is, that the departure instant $d(j)$ of job $j$ is determined by its own service time $s(j)$ and either its arrival instant $a(j)$ or the departure instant of the previous job $d(j-1)$ (depending on which occurs later in time):

$$d(j) = max(a(j), d(j-1)) + s(j)\,. \tag{4}$$

This recursive formula for the calculation of departure times is easily understandable and can be implemented efficiently as a sequential computer program. In the following, (4) is modified to support the parallel calculation of departure times, defined later.

**Definition 1 (departure function).** *Let $N := \{1, \ldots, n\}$ be the set of jobs to simulate. Let $s : N \to \mathbb{R}^+$ be a positive function of job service times. Let $a : N \to \mathbb{R}_0^+$ be a stricly monotonic increasing function of job arrival instants. The departure function $d_u^i : \{i-1, \ldots, n\} \to \mathbb{R}_0^+$ for $u \in \mathbb{R}_0^+$ and $i \in N$, is defined as follows:*

$$d_u^i(j) := \begin{cases} u & , \quad \textit{if } j = i-1 \\ max(a(j), d_u^i(j-1)) + s(j), & \textit{otherwise} \end{cases}\,.$$

The parameter $i$ is used to restrict the domain of the function to the set $\{i-1, \ldots, n\}$, starting at job $i$, which is a part of the overall simulation domain. The function is defined for job $i-1$, as well. However, this value is only to be used as the basic case of the recursion. The parameter $i$ is relevant in the context of time-parallel simulation of the queuing system, where the simulation of a parallel process is started with an initial job representing the time interval boundary. The case $i = 1$ is used for a sequential calculation of departure times as well as for the first process in a time-parallel simulation execution. The parameter $u$ of the departure function is used to indicate an *initial delay* for the queuing system. In a sequential queuing system which is typically empty at the

beginning of the observed time, $u = 0$. However, if there is an initial load of the system, the queue starts with at least one existing job. This property can be captured by introducing an initial delay with $u > 0$ that must pass until the first job can be served. This is used in Section 3.2 to represent the performance of fix-up computations in a time-parallel simulation, where a process starts its fix-up computation phase with a number of jobs in its queue, which has been determined during its initial simulation phase. As can easily be seen, $d_u^i$ is a strictly monotonic increasing function for all parameters $i$ and $u$.

In time-parallel queuing simulation, it is necessary to compare the calculations of two adjacent processes to decide on the termination of fix-up computations. The following lemmata provide the foundation for the following formalization of time-parallel queuing simulation.

**Lemma 1.** *Let $i \in N$ and $u, v \in \mathbb{R}_0^+$ with $u \geq v$. Then for all $j \in \{i-1, \ldots, n\}$*

$$d_u^i(j) \geq d_v^i(j)\,.$$

*Proof.* Proof by induction over $j$ with the basic case $j = 0$ trivially met.
Let $d_u^i(j-1) \geq d_v^i(j-1)$ hold.
Case 1 ($d_u^i(j-1) \leq a(j) (\Rightarrow d_v^i(j-1) \leq a(j))$):
$d_u^i(j) = a(j) + s(j) = d_v^i(j) \Rightarrow d_u^i(j) \geq d_v^i(j)$
Case 2 ($d_u^i(j-1) > a(j)$ and $d_v^i(j-1) \leq a(j)$):
$d_u^i(j) = d_u^i(j-1) + s(j) > a(j) + s(j) = d_v^i(j)$
Case 3 ($d_u^i(j-1) > a(j)$ and $d_v^i(j-1) > a(j)$):
$d_u^i(j) = d_u^i(j-1) + s(j) \geq d_v^i(j-1) + s(j) = d_v^i(j)$
Case 4 ($d_u^i(j-1) \leq a(j)$ and $d_v^i(j-1) > a(j)$):
Cannot occur due to $d_u^i(j-1) \geq d_v^i(j-1)$.

Lemma 1 indicates that for two simulation executions starting with different initial delays, the departure times of all jobs of the simulation with the lower initial delay are always dominated by the departure times of jobs in the other execution. However, it is still unknown how departure functions with different parameters $i$ can be compared. The following lemma shows how this can be done.

**Lemma 2.** *Let $i, i' \in \mathbb{N}_0$ with $i \leq i'$ and $u \in \mathbb{R}_0^+$. Let $u' := d_u^i(i'-1)$. Then the following equality holds for all $j \in \{i'-1, \ldots, n\}$:*

$$d_u^i(j) = d_{u'}^{i'}(j)$$

*Proof.* Proof by induction over $j$ with the base case $j = i' - 1$ met due to Definition 1. Suppose, that for any $j \in N_{i'}$, $d_u^i(j) = d_{u'}^{i'}(j)$ holds.
Then $d_u^i(j+1) = max(a(j+1), d_u^i(j)) + s(j+1) = max(a(j+1), d_{u'}^{i'}(j)) + s(j+1) = d_{u'}^{i'}(j+1)$.

Lemma 2 shows, that a departure function $d_u^i$ is equivalent to a function $d_{u'}^{i'}$ with another parameter $i' > i$ for all $j \geq i' - 1$. This can be achieved by a simple adjustment of the initial delay $u$. Hence, if two departure functions with

differing parameters $i$ and $i'$ are to be compared, the function with the smaller parameter can be adjusted properly.

Now, a strong relationship between simulation executions with the same parameter $i$ has been established and it has been shown how departure functions with different parameters $i$ and $i'$ can be compared. The purpose of the next section is to show how state match detection in time-parallel queuing simulation can be performed. State matching occurs if the states of adjacent simulations are identical. In fact, it is shown in Section 3.2, that matching between the states calculated by two processors $p_k$ and $p_{k+1}$ occurs exactly when the queuing system gets empty the first time during the fix-up computations of $p_k$. In the case of calculation of departure times, as presented in Definition 1, the property of an empty queue is represented by the expression $d_u^i(j-1) \leq a(j)$, in which case the service of job $j$ is not influenced at all by the history of the queuing system, as job $j-1$ departs from the system before job $j$ arrives. The following lemma is an important step to the property of match detection discussed above. It gives a characterization of the state match time for two simulations with differing initial delays, but the same domain.

**Lemma 3.** *Let $i \in \mathbb{N}_0$ and $u, v \in \mathbb{R}_0^+$ with $u \geq v$. Then for all $j \in \{i-1, \ldots, n\}$, the following two statements are equivalent:*

$$\begin{aligned} &\text{(i)} \quad d_u^i(j) = d_v^i(j)\,, \\ &\text{(ii)} \quad d_u^i(j-1) = d_v^i(j-1) \vee d_u^i(j-1) \leq a(j)\,. \end{aligned}$$

*Proof.* Let $i \in \mathbb{N}$ be an arbitrary natural number. Furthermore, choose an arbitrary $j \in \{i-1, \ldots, n\}$.
Case 1 ($d_u^i(j-1) \leq a(j)$ and $d_v^i(j-1) \leq a(j)$):
$d_u^i(j) = max(a(j), d_u^i(j-1)) + s(j) = a(j) + s(j)$
$= max(a(j), d_v^i(j-1)) + s(j) = d_v^i(j)$
$\Rightarrow$ (i) always holds.
(ii) trivially holds due to $d_u^i(j-1) \leq a(j)$.
Case 2 ($d_u^i(j-1) > a(j)$ and $d_v^i(j-1) \leq a(j)$):
$d_u^i(j) = max(a(j), d_u^i(j-1)) + s(j) = d_u^i(j-1) + s(j)$
$> a(j) + s(j) = max(a(j), d_v^i(j-1)) + s(j) = d_v^i(j)$
$\Rightarrow$ (i) never holds.
(ii) never holds due to $d_u^i(j-1) > a(j) \geq d_v^i(j-1)$.
Case 3 ($d_u^i(j-1) > a(j)$ and $d_v^i(j-1) > a(j)$):
(i) is reduced to $d_u^i(j-1) = d_v^i(j-1)$.
$d_u^i(j) = max(a(j), d_u^i(j-1)) + s(j) = d_u^i(j-1) + s(j)$
$d_v^i(j) = max(a(j), d_v^i(j-1)) + s(j) = d_v^i(j-1) + s(j)$
Thus, it holds that $d_u^i(j) = d_v^i(j) \Leftrightarrow d_u^i(j-1) = d_v^i(j-1)$,
which settles Case 3.
Case 4 ($d_u^i(j-1) \leq a(j)$ and $d_v^i(j-1) > a(j)$):
Cannot occur due to Lemma 1.

### 3.2 Time-Parallel Queuing Simulation

With the foundations defined in the previous section, it is now possible to define time-parallel queuing system simulation. Let $N := \{1, \ldots, n\}$ be the set of jobs to simulate with corresponding arrival instants $a : N \to \mathbb{R}_0^+$ and service times $s : N \to \mathbb{R}^+$. The responsibility for the calculation of departure times of jobs is assigned to $m$ processes $p_1, \ldots, p_m$, assigning start job $j_k \in \{1, \ldots, n\}$ to every process $p_k$ ($j_1 = 1 < j_2 < \ldots < j_m \leq n$). Furthermore, each logical process $p_k$ is assigned a simulation interval $N_k := \{j_k - 1, \ldots, n\}$ and an initial delay $u_k := a(j_k)$. This value of the initial delay represents an empty queue at the beginning of the simulation of each simulation interval, as the departure time of the first job $j_k$ is in any case $a(j_k) + s(j_k)$ and the job is not influenced by any of the preceding jobs in the system.

The following lemma is given to simplify the proofs of Lemma 5 and Theorem 1.

**Lemma 4.** *Let $k, l \in \{1, \ldots, m\}$ with $k < l$ and $z := d_{u_k}^{j_k}(j_l - 1)$. Then the following implication holds for all $j \in \{j_l, \ldots, n\}$:*

$$z \leq u_l \Rightarrow d_{u_k}^{j_k}(j) = d_{u_l}^{j_l}(j)\,.$$

*Proof.* First of all, note that $k < l$ directly leads to $j_k < j_l$ due to definition, which is silently exploited in all of the following proofs. Due to definition, it holds that $u_l = a(j_l)$ and $d_z^{j_l}(j_l - 1) = z$. Hence, $d_z^{j_l}(j_l - 1) \leq a(j_l)$. Therefore, $d_z^{j_l}(j_l) = a(j_l) + s(j_l) = d_{u_l}^{j_l}(j_l)$. Repeated application of Lemma 3 leads to $d_z^{j_l}(j) = d_{u_l}^{j_l}(j)$ for all $j \in \{j_l, \ldots, n\}$. An application of Lemma 2 settles the proof.

The value $a(j_l)$ of the initial delay $u_l$ of a process $p_l$ is reasonable, as it leads to the domination of $d_{u_l}^{j_l}$ by $d_{u_k}^{j_k}$ for every $k < l$. This property is shown in the following lemma and can be exploited later for the determination of the match detection criterion.

**Lemma 5.** *Let $k, l \in \{1, \ldots, m\}$ with $k < l$. Then the following inequality holds for all $j \in N_l$:*

$$d_{u_k}^{j_k}(j) \geq d_{u_l}^{j_l}(j)\,.$$

*Proof.* Let $z := d_{u_k}^{j_k}(j_l - 1)$. There are two cases for the value of $z$:
Case 1 ($z > u_l$):
Then, due to Lemma 1, $d_z^{j_l}(j) \geq d_{u_l}^{j_l}(j)$ for all $j \in N_l$. With an application of Lemma 2, this translates directly to $d_{u_k}^{j_k}(j) \geq d_{u_l}^{j_l}(j)$ for all $j \in N_l$.
Case 2 ($z \leq u_l$):
Follows directly from Lemma 4.

Every one of the functions $d_{u_k}^{j_k}$ has a domain that extends up to the last job $n$. It is not possible to reduce the domain further, as for any process $p_k$,

fix-up computations might be necessary up to $n$. However, it is not necessary for every process $p_k$ to perform fix-up computations up to job $n$, but only until a job arrives after the previous job departed, i.e. until the process has an empty queue. This is formalized in the following theorem.

**Theorem 1.** *Let $k, l \in \{1, \ldots, m\}$ with $k < l$. Then for all $j \in \{j_l, \ldots, n\}$, the following two statements are equivalent:*

$$\text{(i)} \quad d_{u_k}^{j_k}(j) = d_{u_l}^{j_l}(j)\,,$$
$$\text{(ii)} \quad d_{u_k}^{j_k}(j-1) = d_{u_l}^{j_l}(j-1) \vee d_{u_k}^{j_k}(j-1) \leq a(j)\,.$$

*Proof.* Choose an arbitrary $j \in \{j_l, \ldots, n\}$. Let $z := d_{u_k}^{j_k}(j_l - 1)$. There are two cases for $z$:
Case 1 ($z > u_l$):
Due to Lemma 3, $d_z^{j_l}(j) = d_{u_l}^{j_l}(j)$ is equivalent to $(d_z^{j_l}(j-1) = d_{u_l}^{j_l}(j-1)) \vee (d_z^{j_l}(j-1) \leq a(j))$. An application of Lemma 2 settles the proof of this case.
Case 2 ($z \leq u_l$):
(i) holds due to Lemma 4. If $j > j_l$, (ii) holds due to Lemma 4. Hence, let $j = j_l$. Due to $z \leq u_l$, $d_z^{j_l}(j_l - 1) \leq u_l = a(j_l)$. Thus, (ii) holds because of Lemma 2.

Theorem 1 has three implications:

- Once the departure times of a job match between initial simulation and corresponding fix-up computation, the departure times of all following jobs match as well.
- If in the fix-up computations, a job $i$ arrives after or at the time when the previous job departs, the departure times of the job match between initial simulation and corresponding fix-up computation.
- State matching cannot occur before in the fix-up computations, a job arrives at or after the time when the previous job departs.

## 4 Computational Overhead

The previous section introduced an alternative parallel processing scheme for queuing system models. An important property of that approach is the simple state match criterion, which allows for an efficient state match detection. However, as discussed in Section 2, a large part of the overall overhead of a time-parallel simulation execution is determined by the amount of fix-up computations that have to be performed. These in turn depend on the times of state match occurences in the time intervals. The aim of this section is to exemplarily investigate the computational overhead $O$ of the novel queuing system simulation approach. This is done by an examination of the expectation of (the random variable) $O$ in case of an $M/M/1$ queue with parameters $\lambda$ and $\mu$ representing arrival and service rates and under the restriction $\lambda < \mu$.

### 4.1 Facts on $M/M/1$ Queues

In this section we review some formulas concerning $M/M/1$ queues, which are needed in the considerations of Section 4.2.

Let $N(t)$ be the random number of customers in an $M/M/1$ queuing system at time $t$. We consider for all $i = 1, 2, \ldots$ the busy period initiated by $i$ customers at time $t = 0$ (cf. [21])

$$T(i) := \inf\{t \geq 0 : N(t) = 0 \quad \text{and} \quad N(0) = i\}$$

and define $T(0) := 0$. We obtain for the expectation of $T(i)$ in case of $\lambda < \mu$ (see [22] p. 128, problem 16)

$$\mathbf{E}\, T(i) = \frac{i}{\mu - \lambda}\,, \quad i = 0, 1, \ldots\,. \tag{5}$$

To calculate the computational overhead of the parallel simulation, we need a further definition. For $i = 0, 1, 2, \ldots$ and given $\tau_2, \ldots, \tau_m$, let

$$T_k(i) := \inf\{t \geq \tau_k : N(t) = 0 \quad \text{and} \quad N(\tau_k) = i\}\,, \quad k = 2, \ldots, m$$

be the busy time initiated by $i$ customers at time $\tau_k$. Since $\{N(t) : t \in \mathbb{R}_0^+\}$ is a continuous time Markov chain (see [21]), it follows that $T(i)$ and $T_k(i)$ $(k = 2, \ldots, m)$ have the same distribution. Therefore we obtain with (5) in case of $\lambda < \mu$ for the expectation of $T_k(i)$

$$\mathbf{E}\, T_k(i) = \tau_k + \mathbf{E}\, T(i) = \tau_k + \frac{i}{\mu - \lambda}\,, \quad k = 2, \ldots, m\,. \tag{6}$$

In case of $\lambda < \mu$ we know, that for all $i, j \in \{0, 1, \ldots\}$

$$P_{ij}(t) := \mathbf{P}(N(t) = j | N(0) = i) \to \pi_j := \left(1 - \frac{\lambda}{\mu}\right)\left(\frac{\lambda}{\mu}\right)^j$$

for $t \to \infty$ (cf. [21]). We now suppose that $t_0$ is sufficiently large to allow the assumption, that the system is encountered in steady state. Referring to this time point $t_0$, we obtain with (6) for the expectation of the busy time $T_k^{(s)}$ in steady state

$$\mathbf{E}\, T_k^{(s)} = \sum_{j=0}^{\infty} \pi_j\, \mathbf{E}\, T_k(j) = \tau_k + \frac{\lambda}{(\mu - \lambda)^2}\,, \quad k = 2, \ldots, m\,. \tag{7}$$

### 4.2 Expectation of the Computational Overhead

In Section 2 we introduced the computational overhead $O$ for the parallel simulation of general systems. We now want to specialize our considerations by considering the computational overhead $O_{M/M/1}^{(s)}$ of an $M/M/1$ queue in steady state.

Let $\tau_k$ $(k = 2, \ldots, m)$ be given. Then $[N(\tau_k)|N(0) = i]$ is the random number of customers in the system at time $\tau_k$ if the system had started with $i$ customers at time $\tau_1(= 0)$. Therefore, $T([N(\tau_k)|N(0) = i])$ is the random variable representing the first occurrence of state 0 after time $\tau_k$ under the condition $N(0) = i$. Using the total law of expectation (see [23]) we obtain with (6) for $k = 2, \ldots, m$

$$\begin{aligned}\mathbf{E}\,T([N(\tau_k)|N(0) = i]) &= \sum_{j=0}^{\infty} \mathbf{E}\,T_k(j)\,P_{ij}(\tau_k) \\ &= \sum_{j=0}^{\infty} (\tau_k + \mathbf{E}\,T(j))\,P_{ij}(\tau_k) \\ &= \tau_k + \sum_{j=0}^{\infty} \mathbf{E}\,T(j)\,P_{ij}(\tau_k)\,. \end{aligned} \tag{8}$$

Together with (1), the definition of $T([N(\tau_k)|N(0) = i])$ leads to

$$O_{M/M/1}(i) = \frac{1}{\tau}\sum_{k=2}^{m}(T([N(\tau_k)|N(0) = i]) - \tau_k)$$

for all $i = 0, 1, \ldots$ and finally with (8) to

$$\begin{aligned}\mathbf{E}\,O_{M/M/1}(i) &= \frac{1}{\tau}\sum_{k=2}^{m}(\mathbf{E}\,T([N(\tau_k)|N(0) = i]) - \tau_k) \\ &= \frac{1}{\tau}\sum_{k=2}^{m}\sum_{j=0}^{\infty} \mathbf{E}\,T(j)\,P_{ij}(\tau_k)\,.\end{aligned}$$

Note that $\mathbf{E}\,O_{M/M/1}(i)$ does not depend on the time interval boundaries $\tau_k$ with the exception of the transition probabilities $P_{ij}(\tau_k)$.

If we assume that $\tau_2$ is sufficiently large to allow for $P_{ij}(\tau_2) \approx \pi_j$ then this relation holds also for every $\tau_k$ $(k = 3, \ldots, m)$. With (7) we get for the expectation of the computational overhead in steady state $O^{(s)}_{M/M/1}$

$$\begin{aligned}\mathbf{E}\,O^{(s)}_{M/M/1} &= \frac{1}{\tau}\sum_{k=2}^{m}\left(\mathbf{E}\,T^{(s)}_k - \tau_k\right) \\ &= \frac{1}{\tau}\sum_{k=2}^{m}\left(\tau_k + \frac{\lambda}{(\mu-\lambda)^2} - \tau_k\right) \\ &= \frac{m-1}{\tau}\frac{\lambda}{(\mu-\lambda)^2}\,.\end{aligned}$$

The most important observation regarding the expected overhead is a strong dependency on the distance between the arrival rate $\lambda$ and the service rate $\mu$. The expected overhead tends to grow quadratically with a decreasing distance. Furthermore, increasing $\lambda$ and $\mu$ while keeping a fixed distance between both

of the parameters also increases the expected overhead. Finally, as could have been guessed, the overhead tends to grow linearly in the number $m$ of time intervals. These observations can be used to evaluate the intended application of the parallel simulation approach to a given queuing system model.

Note that in order to provide a confidence interval for the overhead, the calculation of the variance of $O$ is necessary. Unfortunately, this calculation is a complex task which cannot be easily solved.

## 5 Conclusions

Queuing networks and their building blocks, the atomic queuing systems are an important tool in the area of performance evaluation of computer and communication systems. Although analytic solutions exist for many types of queuing networks, simulation remains an important tool in this area. Parallel and distributed simulation methods can be applied to queuing simulation models if there are time constraints on the execution of the simulations. It is possible to utilize parallel replicated simulations to execute multiple independent simulation experiments in parallel. However, depending on the conditions of the simulation experiments, the parallelization of the sequential model might be more convenient. Efficient parallelization approaches exist for various types of queuing networks. Unfortunately, these are complex and difficult to be implemented on parallel and distributed computing systems. Moreover, they rely on a tight coupling of processing nodes, thus being unsuited for distributed simulation execution. Therefore, this paper introduces a new approach for parallel G/G/1 queuing system simulation based on time-parallel simulation with fix-up computations.

In time-parallel queuing system simulation, job arrival instants and service times are presampled and stored in an input trace. The trace is decomposed into a number of subtraces to be assigned to parallel processes. Processes start simulation of the corresponding subtraces, supposing that the system is empty at simulation start. This produces incorrect simulation results, being corrected by the use of fix-up computations. The central aspect of the approach is the detection of state matching, i.e. the time when fix-up computations of a process can be stopped. It turns out that state matching is easy to detect in G/G/1 queuing systems, as it occurs exactly at the time a process performing fix-up computations finds the system empty. After all of the processes have finished fix-up computations, simulation results have to be collected from the parallel processes.

An analytical evaluation of the computational overhead of the parallel processing of an $M/M/1$ queuing system model reveals the relationship between the simulation efficiency and the arrival and service rates of the model. The expected computational overhead due to fix-up computations tends to grow quadratically with a decreasing difference between the arrival rate $\lambda$ and the service rate $\mu$. Hence, the utilization of the novel approach is reasonable with a sufficiently large distance between arrival and service rates.

In contrast to the other approaches for parallel queuing simulation, time-parallel queuing system simulation is a simple method, which can be easily implemented and is also suited for distributed simulation execution. Another advantage is the possibility of an application of approximate simulation techniques, which have been developed for time-parallel simulation with fix-up computations [24, 25].

## References


1. Fujimoto, R.M.: Parallel and Distributed Simulation Systems. John Wiley & Sons, New York (2000)
2. Heidelberger, P.: Statistical analysis of parallel simulations. In: Proceedings of the 1986 Winter Simulation Conference. (1986) 290–295
3. Foster, I., Kesselman, C., Tuecke, S.: The anatomy of the grid: Enabling scalable virtual organizations. International Journal of High Performance Computing Applications **15** (2001) 200–222
4. Fujimoto, R.M.: Parallel discrete event simulation. Communications of the ACM **33** (1990) 30–53
5. Wagner, D.B., Lazowska, E.D.: Parallel simulation of queueing networks: Limitations and potentials. In: SIGMETRICS. (1989) 146–155
6. Chandy, K., Sherman, R.: Space-time and simulation. In: Proceedings of the SCS Multiconference on Distributed Simulation. (1989) 53–57
7. Greenberg, A.G., Lubachevsky, B.D., Mitrani, I.: Algorithms for unboundedly parallel simulations. ACM Transactions on Computer Systems **9** (1991) 201–221
8. Greenberg, A.G., Lubachevsky, B.D., Mitrani, I.: Superfast parallel discrete event simulations. ACM Transactions on Modeling and Computer Simulation **6** (1996) 107–136
9. Kruskal, C.P., Rudolph, L., Snir, M.: The power of parallel prefix. IEEE Transactions on Computers **34** (1985) 965–968
10. Ladner, R.E., Fischer, M.J.: Parallel prefix computation. Journal of the ACM **27** (1980) 831–838
11. Andradóttir, S., Hosseini-Nasab, M.: Efficiency of time segmentation parallel simulation of finite markovian queueing networks. Operations Research **51** (2003) 272–280
12. Andradóttir, S., Ott, T.J.: Time-segmentation parallel simulation of networks of queues with loss or communication blocking. ACM Transactions on Modeling and Computer Simulation **5** (1995) 269–305
13. Wang, J.J., Abrams, M.: Approximate time-parallel simulation of queueing systems with losses. In: Proceedings of the 1992 Winter Simulation Conference. (1992) 700–708
14. Wang, J.J., Abrams, M.: Massively time-parallel, approximate simulation of loss queueing systems. Annals of Operations Research **53** (1994) 553–575
15. Chen, L.: Parallel simulation by multi-instruction, longest-path algorithms. Queueing Systems **27** (1997) 37–54
16. Heidelberger, P., Stone, H.S.: Parallel trace-driven cache simulation by time partitioning. In: Proceedings of the 1990 Winter Simulation Conference. (1990) 734–737
17. Lin, Y., Lazowska, E.: A time-division algorithm for parallel simulation. ACM Transactions on Modeling and Computer Simulation **1** (1991) 73–83

18. Nikolaidis, I., Fujimoto, R.M., Cooper, C.A.: Time-parallel simulation of cascaded statistical multiplexers. In: Proceedings of the 1994 SIGMETRICS Conference on Measurement and Modeling of Computer Systems. (1994) 213–240
19. Wu, H., Fujimoto, R.M., Ammar, M.: Time-parallel trace-driven simulation of CSMA/CD. In: Proceedings of the 17th Workshop on Parallel and Distributed Simulation. (2003) 105–114
20. Krivulin, N.K.: Recursive equations based models of queueing systems. In: Proceedings of the 1994 European Simulation Symposium. (1994) 252–256
21. Prabhu, N.U.: Queues and Inventories. J. Wiley & Sons (1965)
22. Saaty, T.L.: Elements of Queueing Theory with Applications. McGraw-Hill (1961)
23. Rohatgi, V.K.: An Introduction to Probability Theory and Mathematical Statistics. John Wiley & Sons (1976)
24. Kiesling, T.: Using approximation with time-parallel simulation. SIMULATION **81** (2005) 255–266
25. Kiesling, T., Pohl, S.: Time-parallel simulation with approximative state matching. In: Proceedings of the 18th Workshop on Parallel and Distributed Simulation. (2004) 195–202