Proceedings of the 2023 Winter Simulation Conference
C. G. Corlu, S. R. Hunter, H. Lam, B. S. Onggo, J. Shortle, and B. Biller, eds.

FAST APPROXIMATION TO DISCRETE-EVENT SIMULATION OF MARKOVIAN QUEUEING NETWORKS

Tan Wang

School of Data Science
Fudan University
440 Handan Road
Yangpu, Shanghai 200433, CHINA

Yingda Song

Antai College of Economics and Management
Shanghai Jiao Tong University
1954 Huashan Road,
Xuhui, Shanghai 200030, CHINA

Jeff Hong

School of Data Science and School of Management
Fudan University
440 Handan Road
Yangpu, Shanghai 200433, CHINA

ABSTRACT

Simulation of queueing networks is generally carried out by discrete-event simulation (DES), in which the simulation time is driven by the occurrence of the next event. However, for large-scale queueing networks, especially when the network is very busy, keeping track of all events is computationally inefficient. Moreover, as the traditional DES is inherently sequential, it is difficult to harness the capability of parallel computing. In this paper, we propose a parallel fast simulation approximation framework for large-scale Markovian queueing networks, where the simulation horizon is discretized into small time intervals and the system state is updated according to the events happening in each time interval. The computational complexity analysis demonstrates that our method is more efficient for large-scale networks compared with traditional DES. We also show its relative error converges to zero. The experimental results show that our framework can be much faster than the state-of-the-art DES tools.

1 INTRODUCTION

Simulation of queueing networks is a fundamental approach to studying many real-world systems, such as call centers, telecommunication networks, and manufacturing systems. Discrete-event simulation (DES) has emerged as a popular method for queueing networks simulation (Banks et al. 2000). Basically, DES models the dynamics of a system as a sequence of events over time, and hence the execution time of the method is proportional to the number of events that must be processed. As a result, for large-scale systems, particularly those with high activity levels, the number of events can be extremely large, making the tracking of all the events very time consuming.

To solve this problem, a natural idea is to employ parallel-computing environments. However, parallelizing DES is challenging because it requires a global event list to maintain the correct event order so that the effects of the event interactions can be captured (Fujimoto 1990). Nonetheless, many studies have explored parallel DES and proposed two types of algorithms - conservative and optimistic synchronisation strategies. Conservative strategies (Chandy and Misra 1979; Bryant 1977; Nicol and Riffe 1990) ensure the

correct events order by blocking processes, but they cannot fully utilize the available parallelism. Optimistic strategies (Jefferson 1985; Malik et al. 2010; Fujimoto et al. 2017) allow out-of-order event processing, and use rollback to recover from errors, which are more efficient than conservative strategies. Moreover, both strategies depend on the inherent parallelism of the system and most of the literature on parallel DES focus on regular and symmetric network topologies, which are often unrealistic. In contrast, real-world networks often exhibit scale-free property (Pienta and Fujimoto 2013), which limits the parallelism for large-scale networks.

The difficulty of parallelizing traditional DES roots in the adoption of next-event time progression. In DES, once the system state updates, the simulation time jumps directly to the occurrence time of the next event, assuming no change in the state in-between. As the simulation clock is driven by the occurrence time of the series of events, this approach is inherently sequential. Alternatively, we can use fixed-increment time progression to update the simulation time, where the system state updates at equally spaced time-steps according to the events happening in each time interval (Law and Kelton 2000; Yan and Gong 1999). The fixed-increment time progression does not track the order of events in the same time interval, leaving room for enhancing efficiency through parallel simulation.

This paper focuses on simulation of large-scale Markovian queueing networks. We propose a fast approximation framework which adopts the fixed-increment time progression approach and design parallel simulation algorithms, an approach that has been applied successfully to the parallel simulation of large-scale production networks (Wang and Hong 2023). The computational complexity analysis demonstrates that our algorithms are much more efficient for large-scale networks compared with DES. In addition, we prove that the approximation error is bounded by a term that is independent of the simulation horizon, and the relative error converges to zero as the system size increases. We also point out that our framework is not just a trade-off between accuracy and efficiency. With appropriate size of time interval, both the simulation speed and accuracy can be guaranteed simultaneously. The experimental results show that the run-time of the proposed algorithms is superior to DES when the system is large, and the approximation performance is acceptable with appropriately chosen time intervals.

The rest of this paper is organized as follows. In Section 2, we provide the problem formulation and introduce the traditional framework of DES. In Section 3, we introduce the parallel approximation algorithms and compare the complexity of the algorithms with traditional DES algorithms. Preliminary theoretical analysis of approximation error is given in Section 4. Section 5 presents the numerical results, followed by a conclusion and future prospects in Section 6.

2 DES OF QUEUEING NETWORKS

This section considers the traditional framework of DES. In particular, Section 2.1 introduces a generic multi-layer queueing network, and Section 2.2 provides a DES algorithm for simulating the network and analyzes its computational complexity.

2.1 Problem Formulation

We consider a multi-layer queueing network with n nodes (e.g., the network shown in Figure 1), and node i represents a service station consisting of m_i servers. Assume that the buffer is of infinite capacity, so there is no limit on the number of customers in the queue. When a customer finishes service at one node, it can either join another node at the next layer or leave the network according to the routing matrix P. Each element p_{ij} in the routing matrix denotes the probability of joining node j after completing service at node i. In this paper we focus on Markovian queueing networks, i.e., the external arrivals of customers follow Poisson processes, and the service time at each server follows exponential distribution.

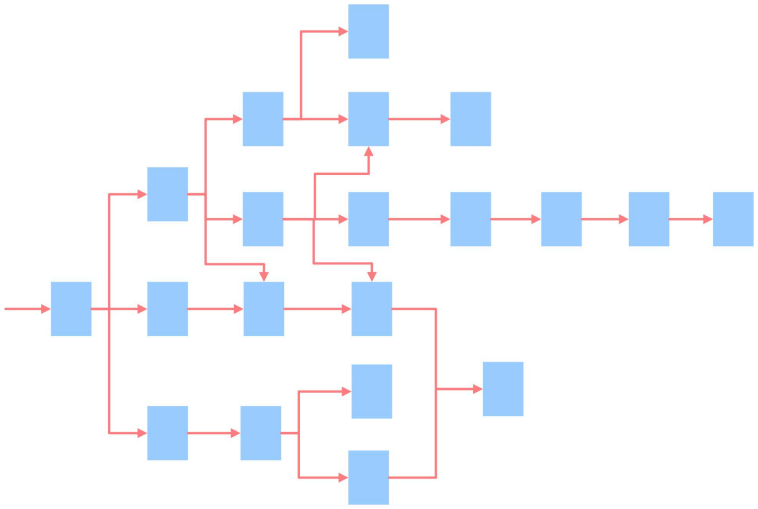

Figure 1: An example of a multi-layer queueing network.

2.2 DES Algorithm and Computational Complexity

As mentioned in Section 1, DES models the dynamics of a system as a sequence of events over time and maintains a global next-event list to preserve the correct event order. In particular, the next-event list tracks the time when the next external arrival occurs at each node and the time when the next departure occurs at each busy server. Figure 2 illustrates the simulation process for the queueing network described in Section 2.1, which is explained as follows.

First, after initialization, we find the next event on the event list and update the simulation time to the occurrence time of that event. If the time does not exceed the maximum time horizon, the simulation continues and the next step depends on the type of the event. If the next event is an arrival, we assign the customer to a free server at the arrived node, change the server state from idle to busy, determine the time when the service will complete, update the event list with a new departure; or add the customer to the end of the waiting queue of the node if all servers are busy. If the next event is a departure, the server who has just completed the service will accept the next customer in the waiting line, or become idle if there is no one waiting in the queue. At the same time, we decide where the departing customer will go according to the routing matrix. If the customer goes to another node, then it triggers an instantaneous internal arrival event, which will be immediately handled as described above. Otherwise, the customer leaves the system. Finally, after updating the event list, the simulation proceeds and jumps to the occurrence of the next event.

Based on the above description, we can analyze the complexity of DES. For simplicity, assume the queueing network is Jacksonian, with m servers for each node; furthermore, the service rate for each server and the utilization of the network are constant as the network scales up. The computational complexity of each update in DES depends on the length of the next-event list, which is composed of arrivals and departures. Since there are n nodes in the network, so the total number of arrival events on the list is at most n. Meanwhile, as the average number of busy servers in a Jacksonian network is proportional to the number of servers, the number of departure events on the list is $O(mn)$. Therefore, the average length of the next-event list is $O(mn)$. To maintain the next-event list, heap is one of the best choices. A heap is a specialized tree-based data structure that satisfies specific property. In a min heap, the value of a parent-node is less than or equal to the value of all the son-nodes. The complexity for every update to the heap-based event list is $O(\log(mn))$ (Leiserson et al. 1994). In addition, as on average there are $O(mn)$ busy servers, each of which finishes service at a fixed rate, therefore the number of departure events is $O(Tmn)$ for a simulation horizon of T. Since the utilization is assumed to be constant, the number of arrival events is also $O(Tmn)$. To sum up, the time complexity of DES is $O(Tmn\log(mn))$.

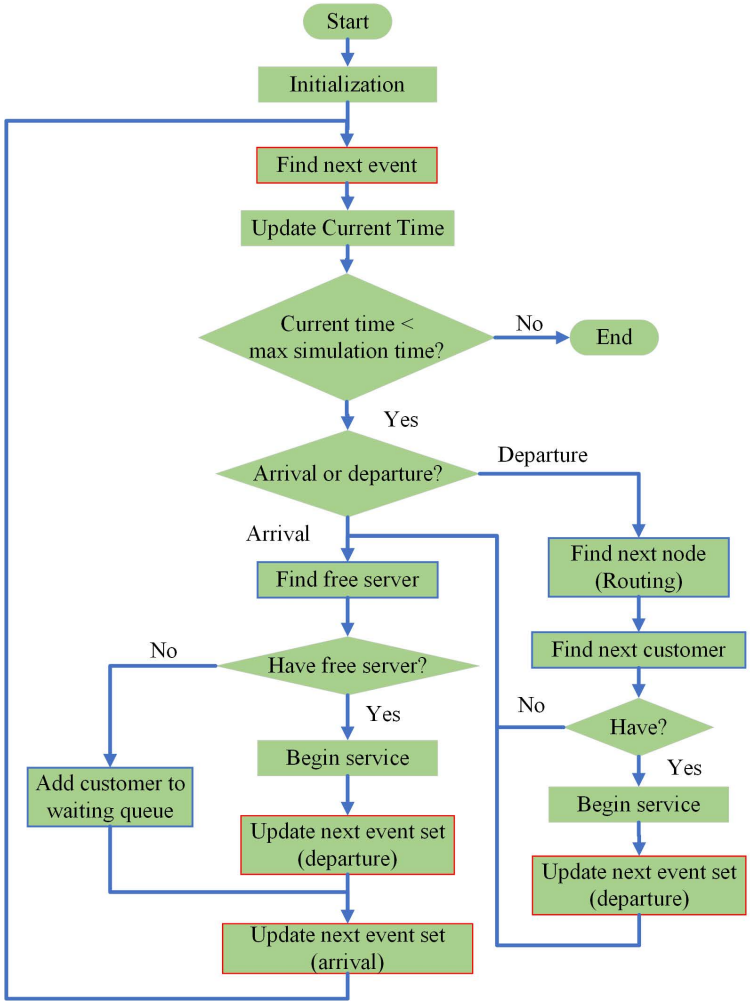

Figure 2: DES flowchart of queueing networks.

3 FAST APPROXIMATION TO DES

This section proposes a fast approximation approach to simulate the queueing networks described in Section 2.1. The basic idea is to adopt fixed-increment time progression, and use parallel processors to speed up the simulation within the time intervals. In this paper, we focus on the simulation of queue lengths, but the algorithm can be extended to the simulation of other quantities like sojourn times.

3.1 Approximation Algorithm

Denote $Q_{\tau,i}$ to be the queue length of the i-th node at the τ-th time step. Given a fixed time increment $h > 0$, we aim to simulate $\{Q_{\tau,i}, i = 1,\ldots,n\}$ at time τh for $\tau = 0,1,\ldots,T/h$. Based on the Markovian property, for each time step, we update the queue lengths based on their values at the end of the last time interval, and the number of arrivals and departures during the current time interval. We ignore the information of the events order and customers routing within each time interval, which has two consequences.

On the one hand, the step-wise update for each node of the queueing network can be carried out independently, so that we can distribute the computations to parallel processors to speed up the simulation (see Figure 3 for a graphical illustration of the parallel simulation scheme). To facilitate implementation, the first step involves initializing a process pool. Subsequently, during each time interval, the main process

allocates tasks to the sub-processes within the pool and collects the results upon completion of their respective calculations. Specifically, each sub-process is responsible for calculating the departures of individual nodes, generating routing results for these departures, and updating the queue length of each corresponding node. Moreover, the main process handles the task allocation and subsequently collects the results, which are utilized to calculate the interval arrivals for each node.

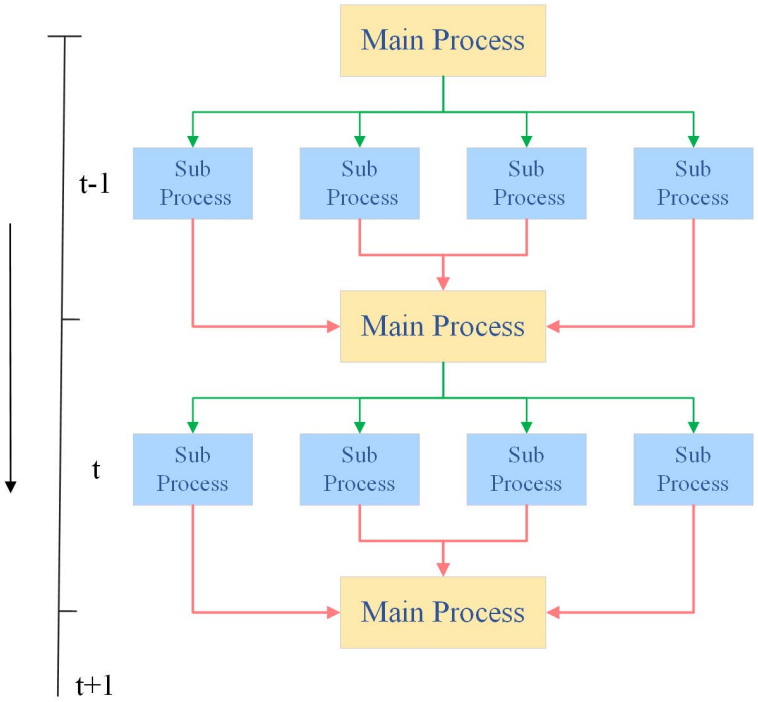

Figure 3: Parallel simulation mode.

On the other hand, the loss of information impedes us from exact simulation of the system dynamics; instead, our algorithm generates upper and lower bounds for the queue lengths, which can be used to construct approximations to the true values.

To explain how to obtain the upper and lower bounds with limited information, consider the update of queue length for node i at the τ-th time step in a multi-layer queueing network. Suppose the queue length at time $(\tau-1)h$ is $Q_{\tau-1,i}$, and during the time period between $(\tau-1)h$ and τh, there are $a^{in}_{\tau,i}$ arrivals from the previous layer in the network and $a^{out}_{\tau,i}$ arrivals from the outside. Denote $s_{\tau,i,j}$ to be the maximum number of customers that can be served by the j-th server ($j=1,\cdots,m_j$). Notice that without the information on the order of events, we cannot determine the exact number of departures during this time interval. Therefore, we push all the new arrivals forward to the end of the time interval and set a maximum capacity for each server to obtain an upper bound for $Q_{\tau,i}$, while pull all the new arrivals backward to the beginning of the time interval and assume the service capacity of each server is fully used to obtain a lower bound. Algorithm 1 exemplifies our approach and generates the upper bound for the queue lengths by setting the maximum capacity for each server to be 2, and hence the service capacity is used only by the customers in the queue at the beginning of the time interval, and the number of departures is determined by sampling from two binomial random variables sequentially.

3.2 Computational Complexity

In our approximation algorithm, the updates for queue lengths of different nodes in each time interval are assigned to parallel processors, so that the computational time can be controlled to $O(1)$. Therefore, the

Algorithm 1: Approximation Algorithm

Initialization: $Q_{0,i} = 0$ (for $i = 1, \cdots, n$);
for $\tau = 1$ **to** T/h **do**
  for *each node i, assign a separate processor and* **do**
    $m_{\tau,i}^{busy} \leftarrow \min\{Q_{\tau-1,i}, m_i\}$;
    $m_{\tau,i}^{1} \leftarrow \text{binomial}\left(m_{\tau,i}^{busy}, Pr(s_{\tau,i,j} \geq 1)\right)$;
    $Q'_{\tau,i} \leftarrow Q_{\tau-1,i} - m_{\tau,i}^{busy}$;
    $m_{\tau,i}^{2} \leftarrow \text{binomial}\left(\min\left\{m_{\tau,i}^{1}, Q'_{\tau,i}\right\}, Pr\left(s_{\tau,i,j} \geq 2 | s_{\tau,i,j} \geq 1\right)\right)$;
    $d_{\tau,i} \leftarrow m_{\tau,i}^{1} + m_{\tau,i}^{2}$;
    $R_{\tau,i} \leftarrow \text{multinomial}(d_{\tau,i}, P_i)$;
  Collect results of each processor;
  for *each node i, assign a separate processor and* **do**
    $a_{\tau,i}^{in} \leftarrow \sum_{k} R_{\tau,k,i}$;
    $Q_{\tau,i} \leftarrow Q_{\tau-1,i} - d_{\tau,i} + a_{\tau,i}^{in} + a_{\tau,i}^{out}$;

computational complexity of our fast approximation mainly depends on the number of time intervals. For a fixed h, the number of time intervals in Algorithm 1 is T/h. Therefore, the overall complexity of our approximation algorithm is $O\left(\frac{T}{h}\right)$. The choice of h controls the tightness of the upper and lower bounds. According to the preliminary results on error analysis in Section 4, we recommend to set $h = o\left(\frac{1}{\sqrt{m}}\right)$ so that the relative error of using the upper bound approximation provided by Algorithm 1 converges to 0. For example, we can choose $h = \frac{1}{\sqrt{m\log(m)}}$, and thus the complexity of our algorithm is $O\left(T\sqrt{m\log(m)}\right)$.

Table 1 compares the complexity of our algorithm with DES algorithm analyzed in Section 2.2. Owing to the adoption of parallel computing, the complexity of our algorithm is independent of n. Even if we consider only the impact of m, the computational complexity of our algorithm is $\sqrt{m\log(m)}$ times lower, with the recommended choice of h. The superiority of our method is more significant when the number of nodes or number of servers in each node is large, which suggests that our method is especially useful for simulating large scale queueing networks.

Table 1: Complexity comparison

Method	Complexity
Parallel Fast Approximation	$O\left(\frac{T}{h}\right) = O\left(T\sqrt{m\log(m)}\right)$ with $h = \frac{1}{\sqrt{m\log(m)}}$
DES-Heap	$O(Tmn\log(mn))$

4 PRELIMINARY ERROR ANALYSIS

In this section, we aim to show that the relative error of our fast approximation algorithm converges to zero as the queueing network scales up when the length of time interval is appropriately chosen. For simplicity, we assume the multi-layer queueing network is Markovian, with external customers arrive according to Poisson processes, and each server finish a job with exponential distributed time. In addition, we assume the utilization is constant and less than 1 as the network scales up.

We start our analysis with the simple case when there is only one node. In this case, the queueing network reduces to a multi-server queue. For a given h and a fixed time point $t = \tau h$, let Q_τ to be the queue

length at time t given that the system is initially idle. Denote Q_τ^{up} and Q_τ^{low} to be the upper bound and lower bound approximations generated by our approximation algorithm. In another word, Q_τ^{up} is obtained by assuming customers arrives at the end of each time interval of length h and each server completes at most 2 services during a time interval, while Q_τ^{low} is obtained by assuming customers arrives at the beginning of each time interval and the service capacities of all servers are fully used. To facilitate our analysis, we use $\tilde{Q}_\tau^{up}$ and $\tilde{Q}_\tau^{low}$ to denote the upper and lower bounds by assuming the customers arrive at the end and beginning of each time interval, respectively, but incorporating the full information about when services are completed. Clearly, we have

$$E[Q_\tau^{low}] \le E[\tilde{Q}_\tau^{low}] \le E[Q_\tau] \le E[\tilde{Q}_\tau^{up}] \le E[Q_\tau^{up}].$$

For a Markovian multi-server queue, we have the following results on the gaps between these upper and lower bounds.

$$E\left[\tilde{Q}_\tau^{up}\right] - E\left[\tilde{Q}_\tau^{low}\right] \le m\mu h,$$
$$E[Q_\tau^{up}] - E[\tilde{Q}_\tau^{up}] = o(mh^2).$$

The first inequality is established by noticing that $\tilde{Q}_\tau^{up}$ and $\tilde{Q}_\tau^{low}$ are the queue lengths of two systems with the identical arrival processes except for a delay period of h. The second result is based on the property that when the service time is exponential distributed, the probability of a server finishes more than two jobs in a period of h is $o(h^2)$. Based on these results, we can prove the convergence of relative error for our fast approximation algorithm.

Theorem 1 Consider a Markovian multi-server queueing system with m servers. Let Q_τ^{up} be the queue length generates by Algorithm 1. Assume the service rate $\mu > 0$ and the utility $\rho < 1$ are fixed, then

(1) Fixing the number of servers m, we have

$$\lim_{h\to 0} E\left[Q_\tau^{up}\right] - E\left[Q_\tau\right] = 0$$

(2) Setting $h = o\left(\frac{1}{\sqrt{m}}\right)$, we have

$$\lim_{m\to\infty} \frac{E\left[Q_\tau^{up}\right] - E\left[Q_\tau\right]}{EQ_\tau} = 0$$

The part (1) of Theorem 1 is a standard result. And the part (2) carries greater significance as it pertains to the performance of our method when applied to large-scale queueing networks. This finding provides evidence that, in the presence of fixed service rate and utility, despite the possibility of the queue length approximation error being unbounded as the number of servers approaches infinity, the relative error converges to 0.

Next, we argue that the above convergence result also holds for the multi-layer Jackson queueing network. It is known that the nodes in a Jackson network can be viewed as interconnected multi-server queues. Denote Q_τ^k to be the queue length of the k-th multi-server queue ($k = 1, \cdots, n$), then the total number of customers in the queueing network is

$$Q_\tau = \sum_{k=1}^{n} Q_\tau^k.$$

Apply Algorithm 1 to generate the upper bound approximation for Q_τ, then the relative error can be bounded by

$$\frac{E\left[Q_\tau^{up}\right]-E\left[Q_\tau\right]}{EQ_\tau}=\frac{\sum_{k=1}^{n}\left(E[Q_\tau^{k,up}]-E\left[Q_\tau^{k}\right]\right)}{\sum_{k=1}^{n}EQ_\tau^{k}} \leq \max_{1\leq k\leq n}\frac{E\left[Q_\tau^{k,up}\right]-E\left[Q_\tau^{k}\right]}{EQ_\tau^{k}}$$

Therefore, when the network scales up in a regulated way, the relative error of the upper bound approximation converges to 0 due to Theorem 1.

Based on the results of approximation error and computational complexity analysis, we can conclude that our proposed framework is not just a trade-off between accuracy and efficiency. With appropriate choice of time interval length, the simulation speed and accuracy can be guaranteed simultaneously.

This paper serves as a preliminary study focused on analyzing queue length. However, our framework exhibits promising potential for simulating a wide range of other performance measures, which we intend to explore in-depth in future work.

5 NUMERICAL RESULTS

In this section, we test the performances of our proposed algorithms from several different perspectives. All the computer programs are coded in Python and the experiments are run on a computer with 64 CPU cores and 512 GB RAM.

Firstly, we compare the run-time of our framework with Ciw, which is one of the-state-of-the-art tools for simulation of queueing networks (Palmer et al. 2019). Ciw is an open-source tool and it adopts ordinary lists to maintain the future event list. We modify the code and employ heap to store the event list to achieve better performance under large-scale networks. Algorithm 1 assigns a dedicated processor to each node in the network. However, due to limited CPU resources, we have to assign multiple nodes to a single CPU core. To assess the performance of our algorithm, we conduct experiments on networks with 100 and 1000 nodes, with 5 and 50 cores respectively, which means that 20 nodes share a CPU core. We use recommended time intervals of 0.2, 0.06, and 0.025 for $m = 20$, 200, and 1000 respectively. The results of our tests with different parameters are presented in Table 2.

Table 2: Run-time comparison with DES

Parameters	T	1000					
	n	100			1000		
	m	20	200	1000	20	200	1000
Algorithm 1		41.1s	136.1s	323.8s	200.3s	650.8s	1538.4s
DES-Heap		34.6s	499.4s	4395.2s	507.1s	7229.2s	99646.6s

We can see that when n remains constant and m is increased tenfold, the run-time for Algorithm 1 becomes approximately $\sqrt{10}$ times the original run-time, which is consistent with the complexity results in Table 1. However, it increases as n increases in Table 2 and the run-time of Algorithm 1 is worse than expected as a whole. The reason is that Algorithm 1 assigns tasks in each interval so that the time spent transmitting data and assigning tasks accounts for a large proportion of the total run-time, which is not included in the complexity analysis. Since more CPU cores are utilized for network with 1000 nodes, data transmission and task assignment take more time, compared with that of 100 nodes. The run time of our approximation algorithm can be improved with more efficient parallel implementation modes. We have

tried some ideas and will discuss them in future work. As for DES, we can see that the run-times under different m and n are also consistent with the complexity analysis. In general, the run-time of the proposed algorithm is superior than DES when the number of nodes or the server number is large.

Table 2 presents a comparison of the run time between the proposed method and a DES tool. As for the parallel DES methods introduced in Section 1, the speedups achievable are dependent on the number of processors and generally lower than the number of processors. For instance, previous literature reports a speedup of 57 with an optimistic approach on 64 processors and a speedup less than 1 with a conservative approach on 5 processors (Fujimoto 1993). Our method's speedup, in contrast, stems not only from parallelization but also from approximation, enabling us to achieve speedups greater than the number of processors. As depicted in Table 2, our proposed method demonstrates a speedup of 13.6 with 5 processors and a speedup of 64.8 with 50 processors. Furthermore, it is worth noting that the existing literature on parallel DES predominantly focuses on regular and symmetric network topologies and the speedup potential is considerably limited when applied to real-world scale-free networks (Pienta and Fujimoto 2013). Our method, however, does not impose any requirements for regular or symmetric network topologies. Consequently, our parallel fast simulation approximation framework exhibits greater promise as a tool for simulating queueing networks.

Secondly, we evaluate the approximation error of our proposed algorithms on 100-nodes networks with different number of servers in each node. The time intervals are set to the recommended values. Relative errors between the average total number of customers in the simulated network and the theoretical value are calculated and the results are presented in Figure 4. It can be seen that as number of servers increases, the relative error gradually converges to zero, which is consistent with our theoretical analysis.

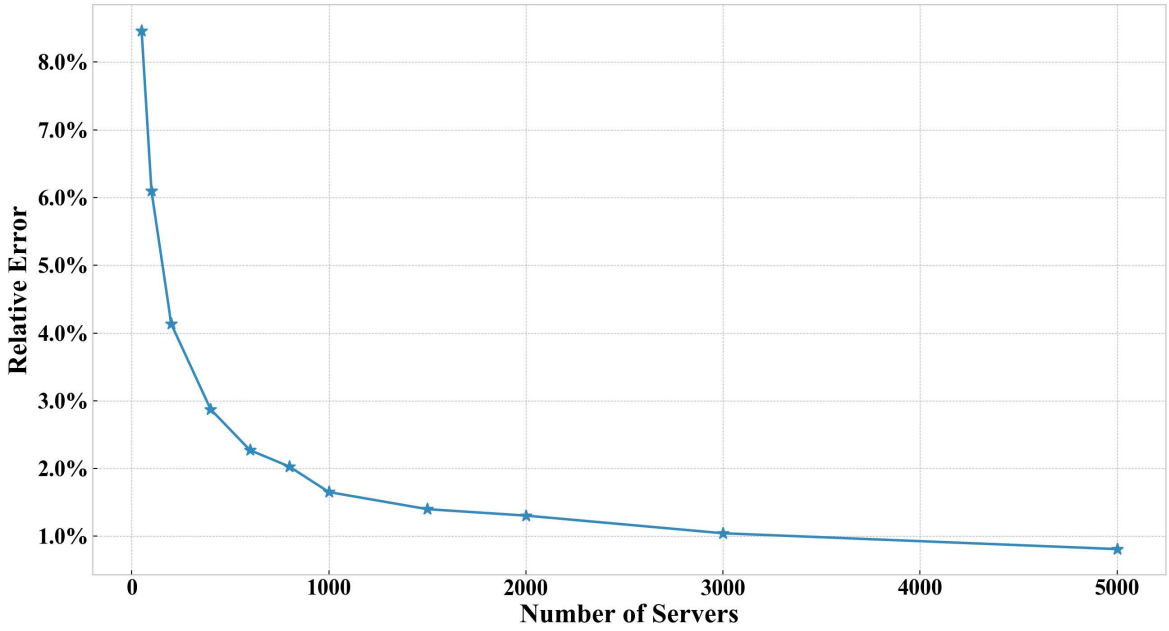

Figure 4: Relative errors.

To further evaluate the performance of the approximation algorithms, we plot the queue length distribution of a node in the network of $n = 100$, $m = 1000$ under DES and Algorithm 1. As shown in Figure 5, the queue length distribution obtained by Algorithm 1 is close to the result of DES.

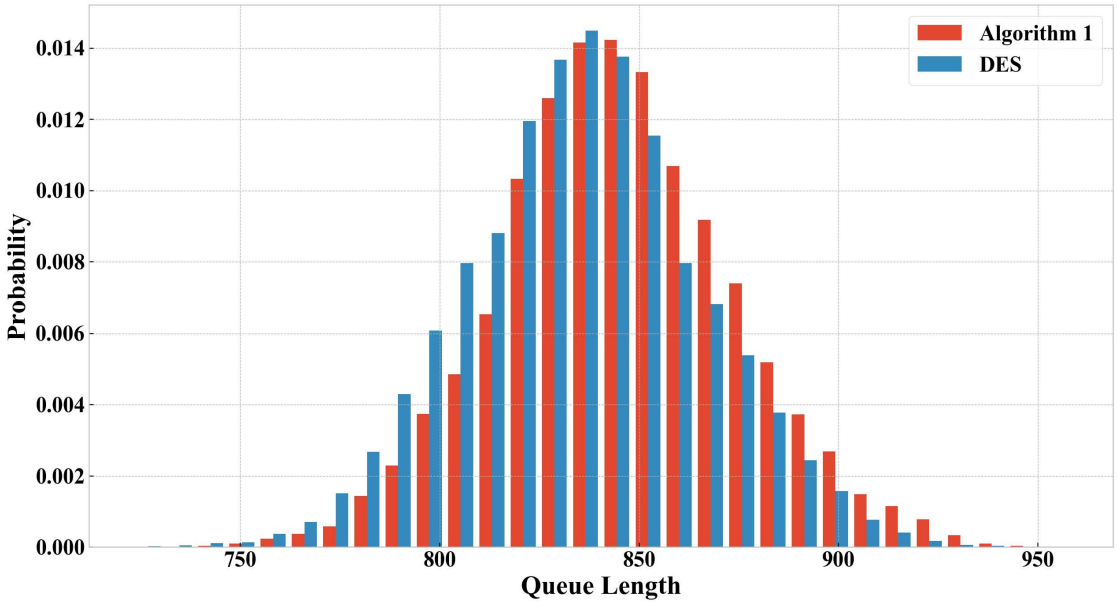

Figure 5: Queue length distribution.

6 CONCLUSION

We propose a parallel fast approximation framework for simulating large-scale Markovian queueing networks in this paper. We prove that the framework is not just a trade-off between accuracy and efficiency. With the appropriate choice of time interval, simulation speed and accuracy can be guaranteed simultaneously. The computational complexity analysis demonstrates that our method is much more efficient for large-scale networks, compared with DES. We provide preliminary theoretical analysis on the asymptotic performance of the proposed algorithms and show that the relative error converges to 0 as the system scales up. The experimental results show that our algorithm can be 60 times faster than the-state-of-the-art DES tools for queueing networks and the approximation performance is acceptable with appropriate time intervals. Although the proposed framework is already faster than DES for large-scale networks, it can be improved with more efficient parallel implementation modes in future work, since the time spent transmitting data and assigning tasks accounts for a large proportion of the total run-time currently.

ACKNOWLEDGMENTS

This research is supported by the National Natural Science Foundation of China [Grants 72091211 and 72161160340].

REFERENCES

Banks, J., J. S. Carson, B. L. Nelson, and D. M. Nicol. 2000. *Discrete-Event System Simulation*. 3rd ed. Upper Saddle River, New Jersey: Prentice-Hall, Inc.

Bryant, R. E. 1977. *Simulation of Packet Communication Architecture Computer Systems*. M.S. thesis, Computer Science Laboratory, Massachusetts Institute of Technology, Cambridge, Massachusetts. https://apps.dtic.mil/sti/pdfs/ADA048290.pdf, accessed 17^{th} June 2023.

Chandy, K., and J. Misra. 1979. "Distributed Simulation: A Case Study in Design and Verification of Distributed Programs". *IEEE Transactions on Software Engineering* SE-5(5):440–452.

Fujimoto, R. M. 1990. "Parallel Discrete Event Simulation". *Communications of the ACM* 33(10):30–53.

Fujimoto, R. M. 1993. "Parallel discrete event simulation: Will the field survive?". *ORSA Journal on Computing* 5(3):213–230.

Fujimoto, R. M., R. Bagrodia, R. E. Bryant, K. M. Chandy, D. Jefferson, J. Misra, D. Nicol, and B. Unger. 2017. "Parallel Discrete Event Simulation: The Making of A Field". In *Proceedings of the 2017 Winter Simulation Conference*, edited

by W. K. V. Chan, A. D'Ambrogio, G. Zacharewicz, N. Mustafee, G. Wainer, and E. Page, 262–291. Piscataway, New Jersey: IEEE.

Jefferson, D. R. 1985. "Virtual Time". *ACM Transactions on Programming Languages and Systems (TOPLAS)* 7(3):404–425.

Law, A. M., and W. D. Kelton. 2000. *Simulation Modeling & Analysis*. 3rd ed. New York: McGraw-Hill, Inc.

Leiserson, C. E., R. L. Rivest, T. H. Cormen, and C. Stein. 1994. *Introduction to Algorithms*. Cambridge, MA: MIT Press.

Malik, A. W., A. J. Park, and R. M. Fujimoto. 2010. "An Optimistic Parallel Simulation Protocol for Cloud Computing Environments". *SCS M&S Magazine* 4(1-9):61.

Nicol, D., and S. Riffe. 1990. "A Conservative Approach to Parallelizing the Sharks World Simulation". In *Proceedings of the 1990 Winter Simulation Conference*, edited by O. Balci, R. Sadowski, and R. Nance, 186–190. Piscataway, New Jersey: IEEE.

Palmer, G. I., V. A. Knight, P. R. Harper, and A. L. Hawa. 2019. "Ciw: An Open-Source Discrete Event Simulation Library". *Journal of Simulation* 13(1):68–82.

Pienta, R. S., and R. M. Fujimoto. 2013. "On the Parallel Simulation of Scale-Free Networks". In *Proceedings of the 1st ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, edited by M. L. Loper and G. A. Wainer, SIGSIM PADS '13, 179–188. New York, NY, USA: ACM.

Wang, T., and L. J. Hong. 2023. "Large-Scale Inventory Optimization: A Recurrent Neural Networks–Inspired Simulation Approach". *INFORMS Journal on Computing* 35(1):196–215.

Yan, A., and W.-B. Gong. 1999. "Time-Driven Fluid Simulation for High-Speed Networks". *IEEE Transactions on Information Theory* 45(5):1588–1599.

AUTHOR BIOGRAPHIES

TAN WANG is a PhD candidate in the School of Data Science at Fudan University in Shanghai, China. He received his bachelor's and master's degrees both from Shanghai Jiao Tong University. His research interests include simulation optimization, stochastic modeling, and machine learning. His email address is dwang19@fudan.edu.cn.

YINGDA SONG is an Associate Professor in Antai College of Economics and Management at Shanghai Jiao Tong University in Shanghai, China. His research interests include financial engineering, fintech, stochastic models and simulation. His email address is songyd@sjtu.edu.cn.

L. JEFF HONG is Fudan Distinguished Professor and Hongyi Chair Professor with joint appointment at School of Management and School of Data Science at Fudan University in Shanghai, China. His research interests include stochastic simulation, stochastic optimization, risk management and supply chain management. He is currently the associate editor-in-chief of *Journal of Operations Research Society of China*, the simulation area editor of *Operations Research*, an associate editor of *Management Science* and *ACM Transactions on Modeling and Computer Simulation*. His email address is hong_liu@fudan.edu.cn.