TauSSA: Simulating Markovian Queueing Networks with Tau Leaping

Matthew Sheldon, Giuliano Casale
Department of Computing
Imperial College London, UK
{ms4520,gcasale}@ic.ac.uk

ABSTRACT

In this paper, we present TauSSA, a discrete-event simulation tool for stochastic queueing networks integrated in the LINE solver. TauSSA combines Gillespie's stochastic simulation algorithm with *tau leaping*, a methodology for optimistic simulation acceleration. Although tau leaping is frequently used in chemical reaction network simulation, it has so far found limited application in queueing theory.

TauSSA offers one of the very first attempts to make this method broadly applicable to analyze extended queueing network models, which include class switching, fork-join, and non-exponential service and arrival distributions. We conceptualize various strategies for handling ordering and illegal states in tau leaping that arise specifically within queueing network models, and compare their performance through numerical experiments. Our main finding is that strategies that sort events based on the network topological order incur a better trade-off between speedup and approximation error.

Keywords

Queueing Networks, Tau Leaping, Simulation

1. INTRODUCTION

Queueing network simulation is routinely used for performance analysis in a variety of domains, from computer and communication systems to healthcare, manufacturing, and logistics [9, 2]. Although flexibility and generality are well-understood reasons for preferring simulation over analytical models, speed of simulation also remains an important dimension also for a simulation tool, given that this is often mentioned as a limiting factor of the approach.

In this paper, we therefore propose TauSSA, a tool for Markovian queueing network simulation that aims at accelerating model evaluation through the tau leaping method [11]. Tau leaping is a technique that accelerates simulation by optimistically processing several events simultaneously. As some of the processed events may be spurious, the technique coarsely approximates the trajectory of the real system, offering a tuneable trade-off between accuracy and the degree by which the simulation is accelerated.

TauSSA is made available to the community as part of the LINE solver suite[1] [5]. LINE offers a modeling language together with a collection of analytical and simulation based tools for solving stochastic networks, primarily to support queueing network analysis studies. Although a native simulator exists within LINE that offers Gillespie's stochastic simulation algorithm (SSA), this does not offer any acceleration means other than independent replication of the simulations. TauSSA addresses this and it is now available within LINE with the same license (BSD-3). Input models can be either specified through LINE's native specification language or via JMT's XML input model format [1].

Tau leaping builds upon the classic SSA method, which may be summarized as follows. Let x be the current system state and let $a_j(x)$ be the rate of the jth state transition out of state x, with $a_0(x) = \sum_j a_j(x)$. Then, with probability $a_j(x)/a_0(x)$ the next state is obtained after firing transition j after time τ, where $\tau \sim Exp(a_0(x))$. We refer to the discrete time instants at which state changes as epochs. Typical state transitions in queueing networks include job departures, job arrivals, and phase change rates. The state variables include, among others, the number of jobs at each node, possibly differentiated according to their class or current service phase.

Tau leaping differs from the SSA algorithm in the way it determines the epochs and the associated state updates. The method uses a deterministic, pre-determined, step size τ to advance time. The time τ is also called the *tau leap*. At each epoch, tau leaping calculates before advancing to the next epoch the state changes to be applied to the state variable by estimating the number of events n_j^τ that have occured within the preceeding time-interval of length τ. Note that n_j^τ depends on the transition j that fires, hence this number refers to identical event types. The realizations of the random variable n_j^τ are obtained by sampling a discrete random variable, often using a Poisson distribution. For the purposes of TauSSA, feasible state changes counted within n_j^τ include a departure, an arrival, or a phase change to a non-exponential service or arrival process, across all the nodes within the model.

An implicit assumption of the tau leaping method is for the system to admit a representation as a Markov process, therefore restricting its applicability to systems that can be modelled within this formalism. A notable limitation is the inability to model heavy-tailed distributions, such as Pareto or lognormal. However, Markovian approximations to these distributions across a finite range may be obtained by phase-type distributions [10].

[1] `http://line-solver.sf.net`

Another issue arising from the application of tau leaping is that applying to a node a given number of events n_j^τ (e.g., successive departures) may result in the node entering an invalid state (e.g., a negative population) requiring strategies to return the simulation to a valid state. While these issues may be simple to address in simple cases, in more complex scenarios some degrees of freedom exists on the choice of mechanism to recover the feasible state. For example, in a network of stations with finite capacity buffers, if the n_j^τ events influence the position of a collection of multiclass jobs, the final state may have a different job mix depending on which jobs are dropped, and which ones are left in the system, after the recovery action. As such, we propose and examine a number of strategies to address these contingencies, studying their interplay with both simulation accuracy and speed.

The rest of the paper is organized as follows. Section 2 positions TauSSA with respect to the state-of-the-art. Section 3 overviews the software architecture of the tool. A discussion of the challenges posed by the application of tau leaping to queueing networks is given in Section 4, alongside a set of strategies to address them. Section 5 compares the proposed tau leaping strategies by means of numerical experiments. Lastly, Section 6 gives conclusions.

2. PRIOR WORK

Existing discrete-event simulation tools for queueing networks, such as the JSIMgraph tool [1] or Omnet++[2], mostly differ from TauSSA in being centered on ad-hoc, general purpose, queueing simulation. Besides increased generality in cases such as heavy-tailed distributions, this simulation style does not assume specialist knowledge in Markovian processes and is therefore fairly widespread in industry. However, it has worse space complexity than SSA-based solutions, as the latter do not need to store memory about the system other than the current state. Furthermore, within such tools, common ways to accelerate the simulation include parallel and distributed algorithms, within which pessimistic and optimistic algorithms may be adopted. Such methods are not incompatible with techniques such as SSA, as forms of parallel SSA execution also exist [7].

Other notable approaches to queueing network simulation include tools exposing numerical methods based on ordinary differential equations (ODEs) [12] and techniques for perfect simulation based on coupling from the past [3]. ODE-based methods are typically applied to fluid and mean-field approximations of stochastic networks. They provide accurate approximations to large-scale systems and further enable the use of time-varying rates that ease the integration of empirical trace data with queueing analysis. At the same time, such methods can incur non-negligible numerical difficulties in presence of non-smooth or stiff differential ODEs, leaving margin for uncertainty on the robustness of the solution and posing challenges in the selection of delicate parameters such as the integration tolerances and step sizes.

Coupling from the past (CFTP) offers the ability to obtain samples for the equilibrium distribution of a Markov process therefore avoiding the bias introduced by transient periods at simulation start. The method requires the evaluation of the coupling of all the possible feasible trajectories obtained for all possible choices of the initial point. This provides an intrinsic limitation, as it is not uncommon for queueing systems to have an unbounded, or combinatorially large, number of initial points. While analytical bounding schemes may be applied to obviate to this issue [3], these complicate the extension of the tool to features not handled by the bounds as these would need to be re-derived.

[2] `https://omnetpp.org`

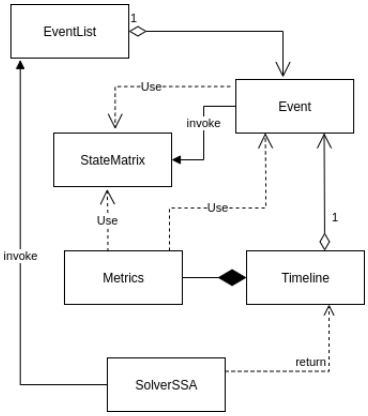


Figure 1: TauSSA internals

3. TAUSSA TOOL

3.1 TauSSA features

TauSSA supports a number of node types used to specify the queueing network, including multi-server queues, delay stations, fork-join nodes, class-switching nodes, and routers to specify arbitrary probabilistic split and join points in the network topology in addition to the basic routing probabilities out of the other nodes. Scheduling policies include first-come first-served, last-come first-served, processor sharing, and service-in-random order. As tau leaping requires the system to be Markovian, arrival and service processes include phase-type distributions such as Exponential or Erlang, but also more general Markovian Arrival Processes (MAPs). The latter are Markov-modulated point processes which generalize phase-type distributions such as the Erlang, but that also allow capturing of non-renewal processes where correlations exist among the inter-arrival times of events.

Relevant model metrics include queue length, utilization, response/residence time, and throughput. The user is able to use the MSER-5 and R-5 methods to determine the transient state of the simulation as well [6, 13, 1].

3.2 TauSSA implementation

The TauSSA simulator is developed fully in Java and consists of a few key data structures and components, as illustrated in Figure 1. The *SolverSSA* class, which is the top-level class exposed within the LINE solver suite, works as a user interface for both tau leaping and standard SSA simulations. This works as a bridge between the user and the model components, handles the simulation configuration, and calls each epoch. An object called *EventList* has the primary responsibility of keeping track of various events, calculating the rates, triggering events, and updating the time of the system. The *StateMatrix* object is tasked with the system state, including the number and type of jobs at each node,

their ordering, and tracking the phases of non-exponential service and arrival processes. The *Timeline* tracks the time and type of each event, as well as the past and mean values of each metric. The latter is also returned when the simulation terminates.

Since TauSSA exclusively simulates memoryless events, it features a number of differences in comparison to more traditional simulations. For example, no pending event list is necessary and the state can be updated as soon as the next event has been determined. Deciding the next event to fire, and when, can be significantly faster with the SSA algorithm. In addition to this, TauSSA claims a 100x improvement in SSA simulation time over LINE's existing SSA solver, in certain cases. When tau leaping is applied, solving time can be reduced by a further 20-50%.

4. TAU LEAPING METHOD

4.1 Algorithm

As mentioned, tau leaping differs from the traditional SSA algorithm for using a pre-determined step size τ to transition between epochs. Its pseudocode may be summarized as follows.

1. The system begins in an initial state, x_0 at time 0.
2. For each possible transition j out of the current state, characterized by vector e_j as the difference between the next state and the current one, all transition rates $a_j(x)$ are evaluated.
3. For each possible transition j, a Poisson random variable is calculated $n_j^\tau \sim Poisson(a_j(x), \tau)$.
4. Update t to equal $t + \tau$, and x based on a given state update strategy, the number of repetitions n_j^τ, and the nature of the event e_j.
5. Return to step 2 until the simulation ends.

An issue with tau leaping in queueing networks is that it may request the system to enter an illegal state. For example, 5 departures may be requested from a queue that only has 3 jobs. This issue has already been discussed in the literature, with solutions such as Binomial and Modified Poisson tau leaping [4]. This problem is more extensive in queueing networks than in chemical reaction models, as states where the buffer is exceeded compound to the issue of departures exceeding the available population. Given the differences between queueing networks and chemical reaction systems, we propose in the next section novel ways to handle invalid states within a epoch.

4.2 State strategies: handling invalid states

We now describe the state strategies available in TauSSA, which are experimentally compared in later sections.

Cutoff strategy. A simple way to implement this has been termed the *Cutoff* method, in which the final state is set to $max(0, min(x + d, c))$, where x is the initial state, d is the leap amount (positive or negative) and c is the capacity of the queue. The shortcoming of this approach is that it occasionally might result in an higher occurrence of states near 0 and the maximum capacity, yielding some inaccuracies in metrics such as utilization.

TimeWarp strategy. *TimeWarp* is a simplified implementation of a commonly used method in simulation known as time warping. In this case, the system will reject any tau leap that results in an illegal state. However, this can be quite demanding computationally. To bound overheads, the current TauSSA implementation has a maximum of W consecutive time warps in a row. If another illegal state is generated after the first W, the system applies the cutoff method. This is intended to prevent the system from remaining in a state where a legal tau leap is very unlikely, such as situations where several queues are near empty despite high departure rates. In the experiments reported in this paper, we have used $W = 2$.

TauTimeWarp strategy. *TauTimeWarp* is an extension of the *TimeWarp* method that allows the τ to halve with each successive time warp. The minimum value of τ is set at 0.0001, while W remains equal to 2. This method is one of the most computational demanding among the proposed techniques.

TwoTimes strategy. Another option is to first calculate the total number of event counts, and then run through each event twice. In the first pass, only events that do not bring into an invalid state are processed. Any invalid departures or arrivals can then be processed only on the second iteration. This further approximates the generated state changes without any reruns or constraint violations. A pseudo-code of the method is given next, to be executed at each epoch:

1. For each event e_j, find all event counts n_j^τ using the tau leaping algorithm.
2. For each event in the event list e_j, calculate the maximum number of event repetitions m_j, and the difference in state e_j.
3. For each event e_j, apply it $k_j = min(n_j^\tau, m_j)$ times. This is the first pass.
4. For each event e_j, apply it again $min(n_j^\tau - k_j, m_j)$ times. This is the second pass.

4.3 Ordering strategies: handling invalid sequences

The performance and accuracy of a queueing network simulated with tau leaping depends on the order in which each event is applied. For example, imagine a network with a source followed by a queue with a low-capacity buffer. If departure events from the source are processed first, then it is likely that not all jobs will not be able to enter the queue, resulting in a high drop rate measure. If departures from the queue are processed first, then more spare capacity will be available for source jobs to enter, resulting in a lower drop rate.

Therefore, it is necessary to define the order in which events in the event list will be processed. In this case, processing involves calculating the event counts n_j^τ, and then triggering the event. In what follows, we define a number of alternative strategies that may be pursued to apply specific orderings upon evolving a queueing network state with tau leaping. Each of these strategies constructs an ordered event list, and then TauSSA generates the number of event applications n_j^τ, applies these according to the state strategy, and then proceeds to the next event in the list.

RandomEvent strategy. The *RandomEvent* method involves triggering each event j at random order at each

epoch. Since an event is bound to a specific node and job class or phase, the state update operation is uniquely defined. An advantage of this method is that all events will be selected evenly across the ordering. This is the preferred solution, in many cases, for simultaneous events in a discrete-event simulation.

RandomEventFixed strategy. The *RandomEventFixed* method involves first shuffling the event list at time 0, and then triggering the events in the same order at each epoch. For each transition j, n_j^τ identical firings of that transition are consecutively applied. In this case, as in *RandomEvent*, the way that events were entered, and the topology of the queue, have no impact on the system. This strategy aims to imitate RandomEvent across simulations with multiple runs, without the need to sort at each iteration. A significant downside is that the simulation will be highly sensitive to the original ordering chosen at time 0. This random ordering can create bias in individual runs, but this decreases over multiple independent replications.

DirectedGraph strategy. Queueing networks can be represented as directed graphs, and a topological sorting of this is a possible ordering for events. The *DirectedGraph* method uses this to ensure that arrivals will generally be processed before departures. Under this condition, neglecting any cycles, each node will start with the maximum number of jobs it will have during the iteration, and will therefore be able to depart as many jobs as possible. This can help prevent negative states, but will create an upward bias in queue length. In cases of an open network, sources are considered first. In closed networks in TauSSA, each class has a reference station, which are considered first. The root of these is randomly chosen. Finding an ordering from the directed graph is done with a variation of Kahn's algorithm [8].

1. Consider an empty queue q, a list of all active events in, and an empty list out.
2. Loop through all events in in, and enqueue any departure events at a source or a reference station in q, and remove them from in.
3. Dequeue an event e from q, and add it to out.
4. Create a list of all nodes, n, that receive jobs outputted from the node corresponding to event e.
5. Loop through in, and enqueue all events corresponding to a node in n.
6. If q is not empty, return to step 3.

DirectedCycle strategy. The *DirectedGraph* method may experience difficulties in queueing networks with finite capacities, and cyclic networks. Under this method, a queue with a capacity will always receive jobs before it processes them. The *DirectedCycle* method solves this issue by cycling through the starting node on each iteration. During each iteration, the first event will be moved to the end of the event list. Therefore, departures will still be processed in order, but there will be chances at each node for jobs to depart before new jobs arrive.

A pseudocode of the method is as follows:

1. Find an ordering to the events using the directed graph method, and insert them into a list l.
2. Apply the tau leaping algorithm to determine the following event(s) and the number of applications.
3. Move the first event in the ordering to the end of l.
4. Check if the number of samples or time limit of the simulation has been exceeded. If so, end the simulation. Otherwise, return to step 2.

5. EXPERIMENTAL EVALUATION

We have validated TauSSA using two experiments, comparing state and ordering strategies, as well as the sensitivity of the results to varying τ values. These experiments are discussed in the next subsections.

5.1 State and ordering strategies

In this experiment, we compare the performance and accuracy of the proposed state and ordering strategies. The validation features all possible state-order strategy pairs, e.g., DirectedCycle/Cutoff, DirectedCycle/TwoTimes, etc. The accuracy of each pair is analyzed using 500 randomly generated model parameterizations, each with 5 M/M/1 queues, 1 open class, random topologies, and random service/arrival rates. Service and arrival rates range from 1 to 50, and any model with an unstable queue is discarded and re-generated. The τ parameter is determined using one of three different parameterizations: (i) $\tau = k/\max_j(a_j)$, $k \in \{2.0, 2.1, ...2.5\}$; (ii) $\tau = k/\text{avg}_j(a_j)$, $k \in \{0.5, 0.6, ...1.5\}$; (iii) $\tau = k/\min_j(a_j)$, $k \in \{0.1, 0.15, 0.2\}$. The execution is stopped upon either hitting simulation time $t = 10000$, or 10^7 samples, or a timeout of 1 second.

Results are given in Figure 2. Accuracy is measured by the Mean Absolute Percentage Error (MAPE) of the simulated queue lengths at all nodes, averaged across experiments. The figure indicates that the DirectedCycle/Cutoff systematically dominates the other techniques, offering a trade-off of about 21% MAPE in return for a simulation time reduction that is 67.2% of the standard SSA. As such, it is chosen as the default technique adopted in TauSSA. The TimeWarp-based methods are not included in the figure, as they are significantly worse than the other methods in terms of accuracy and speed. Various tradeoffs of MAPE and accuracy can be obtained, as shown in Figure 2. However, our analysis reveals that an unguided choice of method to compute τ may fundamentally compromise the quality of tau leaping execution, for example $\tau = k/\max_j(a_j)$ is found to underperform, even compared to standard SSA. We find instead that $\tau = 1.5/\text{avg}_j(a_j)$ with DirectedCycle/Cutoff performs best across the considered experiments.

To further examine the optimal recommendation, the relative simulation time in comparison to a normal SSA solution is included in Table 1. The table shows the MAPE of each configuration, including the TimeWarp-based methods methods. It is clear that the performance of these techniques is dominated by the Cutoff method. It is interesting to observe, that while it would be natural to conclude that DirectedCycle is superior to other methods by considering order sequencing that are closer to the real sample path, doing so by favouring the interleaving of arrival and departure events, this is reductive since DirectedGraph has a similar approach but underperforms. As such, it is only the mutual combination of DirectedCycle and Cutoff that provides an optimal performance.

Figure 2: MAPE vs Simulation Time Ratio under different τ formulas. Markers +:$k/\max(a_j)$, o: $k/\text{avg}(a_j)$, x: $k/\min(a_j)$

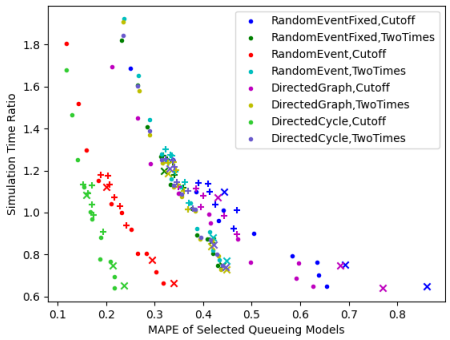
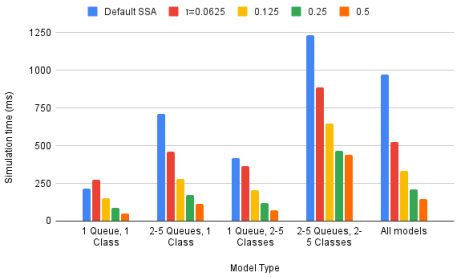


Figure 3: Simulation Time (ms) under different values of τ, by model type



5.2 Sensitivity analysis

Figure 3 describes the performance of TauSSA for different values of $\tau \in \{0.0625, 0.125, 0.25, 0.5, 1\}$ and also includes a comparison against the original SSA algorithm. In this experiment, 17 pre-selected sample models with varying topologies are used to determine the computation time under different values of τ. Our goal is to illustrate convergence of the simulator to exact for decreasing τ values. In particular, we use two strategies to illustrate the point, namely RandomEventFixed and TwoTimes. The experiment is done by trials spanning 30 runs, with 5 warm-ups, and a simulation time of $t = 10000$.

The results show that under the current implementation of TauSSA, tau leaping displays a performance that depends significantly on the type of model studied. In particular: simple model of isolated queueing systems with low event rates gain little to no time savings from tau leaping; conversely, more complex networks, with higher total event rates have been seen to achieve better savings.

According to Table 1, we further observe that a doubling of τ roughly corresponds to a 33% decrease in solving time at lower values of τ. Speed gains from higher values of τ are often offset by decreases in accuracy. Thus, a suitable tuning of the τ value should be chosen to strike the desired balance. For example, a few trial runs may be used to obtain a sensible assignment of τ.

6. CONCLUSION

Table 1: Evaluation of state and ordering strategies. The last column is the simulation time ratio, i.e., the percentage of time saved compared to standard SSA simulation.

Ordering Strategy	State Strategy	MAPE	$\frac{T_{tau}}{T_{ssa}}$
DirectedCycle	Cutoff	0.218	67.2%
RandomEvent	Cutoff	0.323	70.5%
RandomEvent	TauTimeWarp	0.414	193%
RandomFixed	TwoTimes	0.438	78.8%
RandomEvent	TwoTimes	0.440	81.0%
DirectedCycle	TwoTimes	0.444	78.9%
DirectedGraph	TwoTimes	0.444	77.2%
RandomEvent	TimeWarp	0.503	174%
DirectedGraph	Cutoff	0.673	67.3%
RandomFixed	Cutoff	0.675	70.0%
RandomFixed	TauTimeWarp	0.850	205%
DirectedCycle	TauTimeWarp	1.06	196%
DirectedCycle	TimeWarp	2.16	156%
DirectedGraph	TauTimeWarp	2.37	200%
RandomFixed	TimeWarp	2.90	206%
DirectedGraph	TimeWarp	17.2	160%

In this paper, we have proposed TauSSA, a queueing network simulator based on tau leaping, as well as a brief analysis of the simulation configuration and tau leap value selection. Our results indicate that TauSSA can significantly accelerate simulations under appropriate event ordering and selection of the tau leap value τ. These initial findings show that no all implementations of tau leaping perform equally well, and care should be taken to select an appropriate strategy to address invalid states and event ordering issues. We further find that the two issues appear to be intertwined, as there is significant reduction in accuracy when particular combinations of state and ordering pairs are chosen. Overall, our initial numerical study suggests good performance for simple strategies that correct states by forcedly imposing feasibility through cutoffs and that process events in cyclic order.

Future work may explore various lines of research. First, automated methods to determine the optimal τ value may be investigated, for example based on machine learning techniques. Moreover, advanced features features such as fork-join, priorities, temporal dependent in service or arrivals, or finite capacity regions have not been investigated in this initial study and offer novel outlets for possible investigation.

The TauSSA tool is part of the LINE solver[3], released with the same license (BSD-3).

7. REFERENCES

[1] Marco Bertoli, Giuliano Casale, and Giuseppe Serazzi. The JMT simulator for performance evaluation of non-product-form queueing networks. In *Proc. of ANSS*, pp. 3–10, IEEE, 2007.

[2] Gunter Bolch, Stefan Greiner, Hermann de Meer, and Kishor S. Trivedi. *Queueing Networks and Markov Chains*. John Wiley & Sons, 2006.

[3] Anne Bouillard, Ana Bušić, and Christelle Rovetta. Perfect sampling for multiclass closed queueing networks. In *Proc. of QEST*, pp. 263–278, 2015, Springer-Verlag.

[4] Yang Cao, Daniel T Gillespie, and Linda R Petzold. Avoiding negative populations in explicit Poisson tau-leaping. *The Journal of chemical physics*, 123(5):054104–8, 2005.

[3] `https://github.com/imperial-qore/line-solver`
`https://github.com/imperial-qore/line-solver-java`

[5] Giuliano Casale. Integrated performance evaluation of extended queueing network models with line. In *Proc. of WSC*, pp. 2377–2388. IEEE Press, 2020.

[6] George S Fishman. Statistical analysis for queueing simulations. *Management science*, 20(3):363–369, 1973.

[7] Arthur P. Goldberg, et al. Exact parallelization of the stochastic simulation algorithm for scalable simulation of large biochemical networks, 2020. *http://arxiv.orgabs2005.05295*

[8] A. B. Kahn. Topological sorting of large networks. *Commun. ACM*, 5(11):558–562, November 1962.

[9] Averill M. Law and W. David Kelton. *Simulation Modeling and Analysis*. McGraw-Hill, 3 edition, 2000.

[10] M. F. Neuts. *Structured Stochastic Matrices of M/G/1 Type and Their Applications* Marcel Dekker, NY, 1989.

[11] Muruhan Rathinam, et al. Consistency and stability of tau-leaping schemes for chemical reaction systems. *Multiscale Modeling & Simulation*, 4(3):867–895, 2005.

[12] Johan Ruuskanen, et al. Improving the mean-field fluid model of processor sharing queueing networks for dynamic performance models in cloud computing. *Perf. Eval.*, 102231, 2021.

[13] K.P White, M.J Cobb, and S.C Spratt. A comparison of five steady-state truncation heuristics for simulation. In *Proc. of WSC*, vol. 1, pp. 755–760, IEEE, 2000.