Supplementary Materials for QGym: Scalable Simulation and Benchmarking of Queuing Network Controllers

Anonymous Author(s)
Affiliation
Address
email

1 Code release

Our code is available at: https://github.com/namkoong-lab/QGym

2 Additional Benchmark Result

We provide the benchmark results for reentrant-2 [exponential] and reentrant-2 [hyperexpenential] queuing systems in Table 1 and 2.

Table 1: Reentrant-2[Exp]

Network	$c\mu$	MW	MP	FP	PPO	PPO BC	PPO WC
2	26.01 ± 0.00	17.45 ± 0.00	24.5 ± 0.00	16.7 ± 0.00	9.04E+3 $\pm$ 41.13	39.16 ± 0.37	$\mathbf{13.72 \pm 0.22}$
3	26.27 ± 0.00	26.65 ± 0.00	30.0 ± 0.00	61.7 ± 0.00	1.82E+4 $\pm$ 37.23	48.59 ± 0.56	$\mathbf{22.09 \pm 0.29}$
4	27.62 ± 0.00	34.10 ± 0.00	41.3 ± 0.00	74.6 ± 0.00	1.77E+4 $\pm$ 52.30	79.20 ± 1.06	$\mathbf{29.90 \pm 0.47}$
5	44.82 ± 0.00	40.34 ± 0.00	49.0 ± 0.00	85.6 ± 0.00	2.58E+4 $\pm$ 63.54	91.65 ± 1.37	$\mathbf{38.01 \pm 0.52}$
6	$\mathbf{54.48 \pm 0.00}$	46.63 ± 0.00	58.8 ± 0.00	92.5 ± 0.00	4.02E+4 $\pm$ 349.64	526.57 ± 8.74	46.80 ± 0.47
7	70.25 ± 0.00	77.93 ± 0.00	68.0 ± 0.00	102.1 ± 0.00	5.78E+4 $\pm$ 116.92	352.02 ± 6.68	$\mathbf{55.51 \pm 0.57}$
8	70.32 ± 0.00	72.96 ± 0.00	77.7 ± 0.00	103.3 ± 0.00	4.79E+4 $\pm$ 208.70	1332.68 ± 7.82	$\mathbf{63.15 \pm 0.70}$
9	$\mathbf{65.80 \pm 0.00}$	77.34 ± 0.00	84.3 ± 0.00	106.7 ± 0.00	6.54E+4 $\pm$ 491.49	1574.86 ± 9.34	70.30 ± 0.86
10	81.35 ± 0.00	$\mathbf{82.00 \pm 0.00}$	92.7 ± 0.00	120.9 ± 0.00	8.11E+4 $\pm$ 355.34	1876.54 ± 89.20	80.36 ± 0.79

Table 2: Reentrant-2[Hyper]

Net	$c\mu$	MW	MP	FP	PPO	PPO BC	PPO WC
2	39.41 ± 1.37	39.06 ± 1.89	58.63 ± 2.26	39.75 ± 7.29	9.75E+3 $\pm$ 59.58	66.87 ± 1.11	$\mathbf{30.67 \pm 0.83}$
3	52.37 ± 2.14	57.89 ± 2.69	67.14 ± 2.94	55.52 ± 9.38	2.05E+4 $\pm$ 132.63	482.64 ± 17.98	$\mathbf{45.66 \pm 0.84}$
4	70.92 ± 2.68	76.80 ± 3.68	92.96 ± 4.01	72.09 ± 14.37	1.93E+4 $\pm$ 62.07	149.23 ± 3.17	$\mathbf{61.09 \pm 1.30}$
5	81.48 ± 2.66	90.45 ± 3.92	109.03 ± 5.35	84.17 ± 18.17	2.56E+4 $\pm$ 84.23	371.65 ± 10.81	$\mathbf{77.98 \pm 1.73}$
6	104.32 ± 5.07	103.80 ± 4.32	123.60 ± 5.25	99.51 ± 15.48	6.71E+4 $\pm$ 362.68	1363.93 ± 20.21	$\mathbf{93.84 \pm 1.26}$
7	116.61 ± 5.96	117.87 ± 5.06	135.58 ± 5.99	118.91 ± 15.67	6.54E+4 $\pm$ 214.22	2317.88 ± 15.54	$\mathbf{110.48 \pm 1.63}$

Submitted to 38th Conference on Neural Information Processing Systems (NeurIPS 2024). Do not distribute.

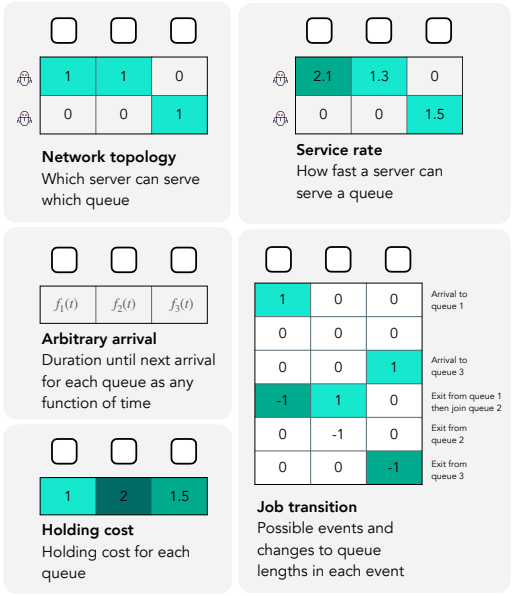

Figure 1: Ingredients of defining a queuing system. Our framework provides intuitive and flexible interface for users to define complex queuing systems.

3 Additional Simulator Design Details

We present a queuing system testing framework. The main goals of the framework are: 1. Provide benchmarks for queuing algorithms 2. Easy to test and deploy with an OpenAI Gym Interface 3. Allow easy configuration of new custom queuing systems with a large degree of freedom

Our testing framework allows the following user interactions with intuitive interface

- Defining a new queueing system or using one provided by our benchmark.
- Defining a policy that takes in observations and output queue priority prediction
- Simulating queueing system trajectories with selected policies

We will detail each of these components of our framework below

3.1 Define queuing system

Ingredients of a queueing system Our framework allows for flexible definition of queuing systems in a straightforward interface. A queuing system can be defined with the following descriptions:

- **Network matrix**: a binary matrix that specifies which server can serve which queue
- **Network transition matrix**: what happens when a server finishes serving a job
- **Service rate matrix**: a matrix that specifies how fast a server can serve a queue. Time of service is drawn from a distribution specified by user using service rate matrix as parameter
- **Arrival rate of queues**: User can define arbitrary arrival pattern for queues as a Python function that inputs time and outputs time until next arrival. This feature allows simulation of time-varying arrivals. User can define arrival rate as a random distribution
- **Queue holding cost**: holding cost per unit of time for each job in each queue
- **Server pool**: We also allow user to define each class of server as a pool of server. When having many servers with the same characteristics, instead of creating many separate servers and inflating the size of network matrix, we allow users to specify a server pool number for each server class. This mechanism allows simulation of large-scale system without slowing the simulation.

Users can define these elements of a queuing sytems in a `.yaml` file and a `.py` file.

3.2 Defining a policy

User can define a policy as a function that takes in queue length as observation and output a matrix that represents the policy's prediction of service priority. The matrix has the the same shape as network matrix (# server $\times$ #queues) that assigns priority to each server-queue pair. A policy can be either a static policy that decide priority based on observation with heuristics or contain a neural model to be trained.

3.3 Simulator design

The simulator environment is structured as OpenAI Gym environment. We follow the design of the OpenAI Gym so that users can easily train and test a variety of reinforcement learning algorithms. Each simulation trajectory consists of a sequence of steps (defined in OpenAI Gym `step` format). For each simulation trajectory, the simulator maintain a number of information as states.

3.3.1 Simulator features

We highlight some important features of our simulator below:

- **Job-level tracking** The simulator tracks the states on the job level. We track service time for each job in a queue. At each step, we allocate to decide which job is being served on an individual job level. This mechanism makes it possible for multiple servers to serve a single queue and allows the modeling of parallel server systems.
- **Event-based simulation** Our simulator is event-based. Each step corresponds to one event: arrival in a queue or one job finishes being served. Prior works designed simulators with fixed time-interval for each step. In comparison, we can simulate trajectories with more uneven event intervals with higher speed by reducing wasting steps on intervals without any event. We also allow more precise time keeping.
- **Batch simulation** Our simulator allows simulation of multiple runs in parallel. Our parallelization implementation allow users to leverage accelerators like GPUs to accelerate simulations.

3.3.2 States

In each trajectory, the simulator keeps track a number of variables as states

Based on the elements of queue systems defined above, the simulator also has the capability of drawing a new service duration for a job and arrival duration for a queue. In addition, during a simulation run, the environment keeps track of

- **Service time** Time until service finishes for a job
- **Arrival time** Time until next arrival occurs for a queue
- **Queue length** Length of each queue

At each step, the simulator updates service time and arrival time based on the event duration of the step. The simulator also has the capability to generate service time and arrival time for new jobs based on user specification of the queuing system. The simulator also updates queue lengths at each step based on the event occurred during the step.

3.3.3 An event-based simulation step

At each step, the simulator takes in action represented by the service priority matrix and returns the updated states in OpenAI Gym `step` function format. To simulate a step and obtain the output of the `step function`, our simulator decides an event that occurs based on the following procedure

- **Converting service priority to action matrix** The step function takes in service priority prediction from the policy. The priority matrix can be a float matrix. The `step` function converts this priority matrix into an action matrix that specifies which servers should serve which queues. Users can customize how the assignment is done. Default implementation provides linear assignment, softmax, and Sinkhorn assignment.

- **Job-server allocation** The action matrix pairs server and queues. Our simulator then assigns each job in in a queue to servers that the action matrix decides to serve the queue through an `allocator` function. User can customized how the allocation is performed. The default allocator implementation selects the fastest serving servers and pair them with jobs with shortest service time remaining.
- **Select event** Based on the remaining service time for each job and remaining arrival time for each queue, the simulator decides the closest next event to be either (1) a job finishes being served or (2) a new job arrives for a queue.
- **Update states** Based on the event occurred, the simulator updates the states correspondingly. If a job finishes being served, the simulator removes the job from the queue. If the transition matrix specifies that the job in one queue goes to another queue after being served, an new job is created for the queue that the job transitions into. If a new arrival occurs, the simulator creates a new job for the queue and generate the new arrival time until next arrival in the queue. Finally, the simulator deducts the event duration from all service times and arrival times.

4 Additional Experiment Details

4.1 Computational Resources

We run all our experments on an AMD EPYC 7513 32-Core Processor.

Checklist

The checklist follows the references. Please read the checklist guidelines carefully for information on how to answer these questions. For each question, change the default **[TODO]** to [Yes] , [No] , or [NA] . You are strongly encouraged to include a **justification to your answer**, either by referencing the appropriate section of your paper or providing a brief inline description. For example:

- Did you include the license to the code and datasets? [Yes] See Section **??**.
- Did you include the license to the code and datasets? [No] The code and the data are proprietary.
- Did you include the license to the code and datasets? [NA]

Please do not modify the questions and only use the provided macros for your answers. Note that the Checklist section does not count towards the page limit. In your paper, please delete this instructions block and only keep the Checklist section heading above along with the questions/answers below.

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] We described our contribution of a new simulating and benchmarking framework for queuing systems, which we detailed in the paper.
   (b) Did you describe the limitations of your work? [Yes] We discussed the limitation of the policies benchmarked and stated that there are plenty of rooms for improvement in section 4.
   (c) Did you discuss any potential negative societal impacts of your work? [NA] Our queuing simulation framework doesn't pose potential negative societal impact.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes] We have read and followed the guidelines.
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [NA] We do not include any theoretical result.
   (b) Did you include complete proofs of all theoretical results? [NA] We do not include any theoretical result.
3. If you ran experiments (e.g. for benchmarks)...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] We provide url to our code release in abstract, which readers can easily use our simulator to benchmark provided methods and additional methods.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] We describe our experimental setting in Section 4 of our paper.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] We provided standard deviation in all tables in our paper.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] We described computational setup in supplementary materials section 4.1
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [NA] We do not use existing assets.
   (b) Did you mention the license of the assets? [NA] We do not use existing assets.
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] We provide URL to our code release in abstract.
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [NA] We do not obtain data from other people.
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [NA] We do not collect these information.

5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [NA] We do not use human subjects
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [NA] We do not use human subjects
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [NA] We do not use human subjects