Learning Queuing Networks by Recurrent Neural Networks

Giulio Garbi, Emilio Incerto and Mirco Tribastone

IMT School for Advanced Studies Lucca

Lucca, Italy

giulio.garbi@imtlucca.it

ICPE 2020 Virtual Conference

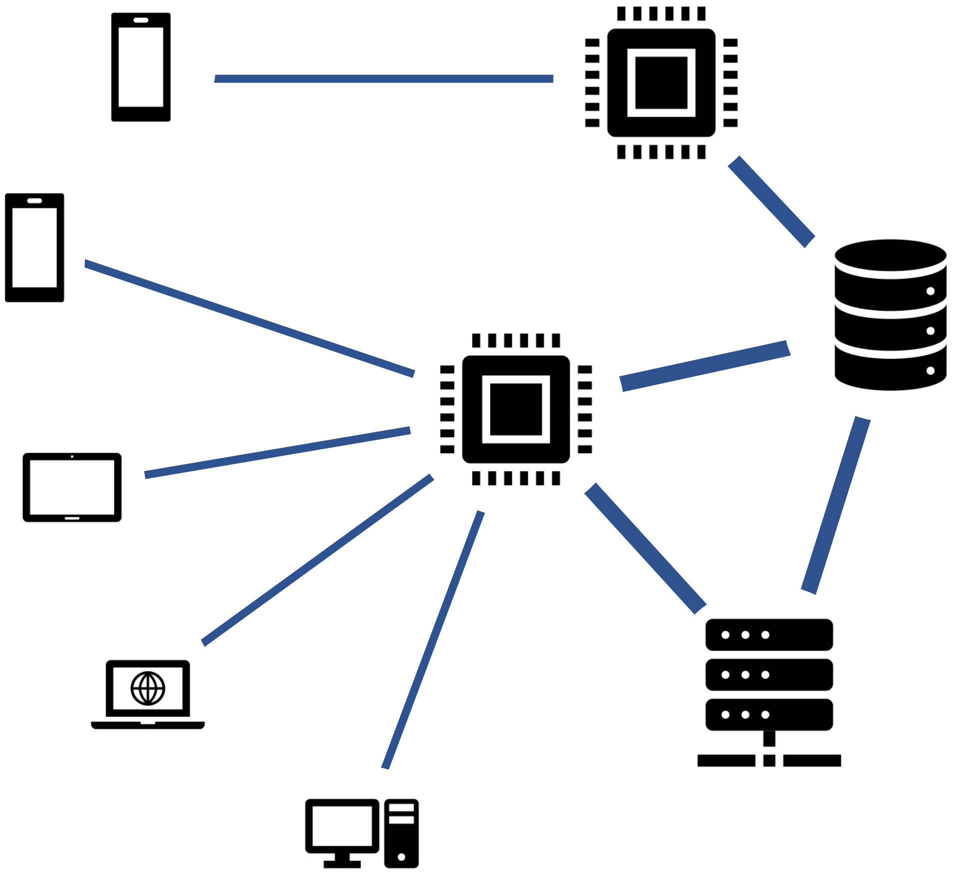
April 20—24, 2020

Motivation

- Performance means revenue
  - «*We are not the fastest retail site on the internet today*» [Walmart, 2012]
  - «*[…] page speed will be a ranking factor for mobile searches.*» [Google]

➔ It's worth investing in system performance. How?

Motivation



- Question: where to invest?
- Performance estimation:
  - Profiling: easy, does not predict
  - Modeling: needs expert and continuous update, predictions

Motivation: our vision

- If we had a model, we could try all possible choices, forecast and choose the best option.

➔ Automate model generation!!!

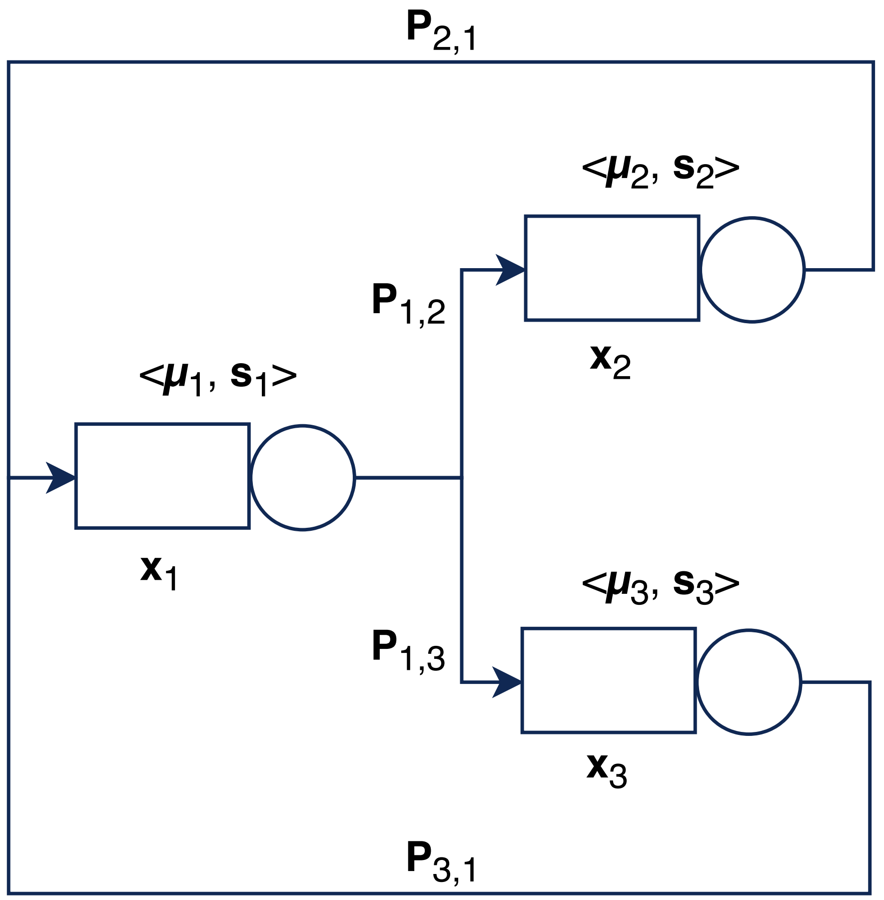
Our Main Contribution

- Direct association between:
  - Model: Fluid Approximation of Closed Queuing Networks
  - Automation: Recurrent Neural Networks

- Automatic generation of models from data

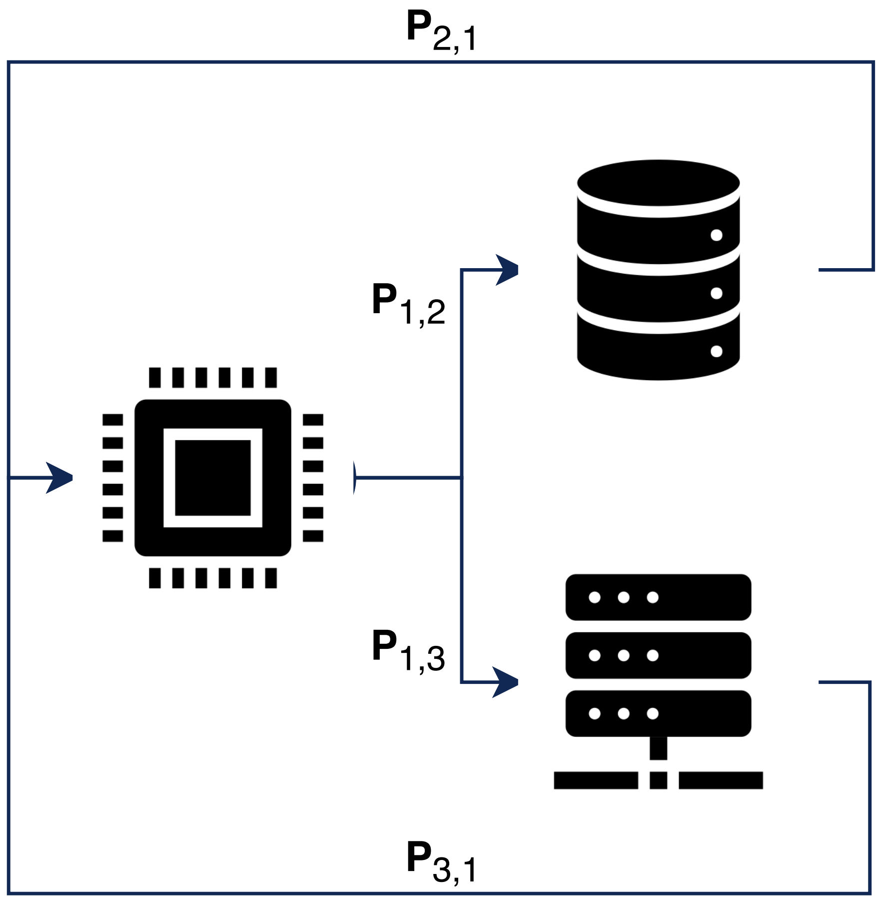
Model: Queuing Networks

- Model that represent contention of resources by clients
- Clients ask for work to station (resources)
- Stations have a maximum concurrency level, and a speed
- Clients once served ask another resource according to routing matrix

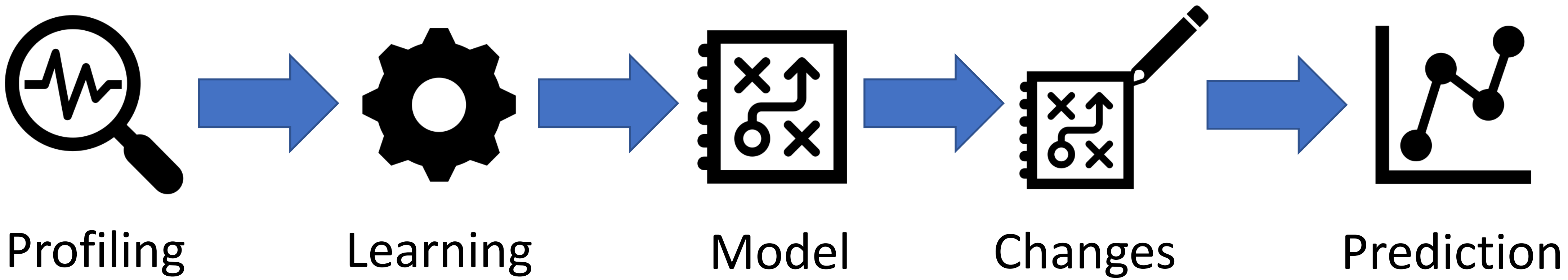


Model of a system

- Resource ➔ hardware
- Routing matrix ➔ program code
- Clients ➔ program instances

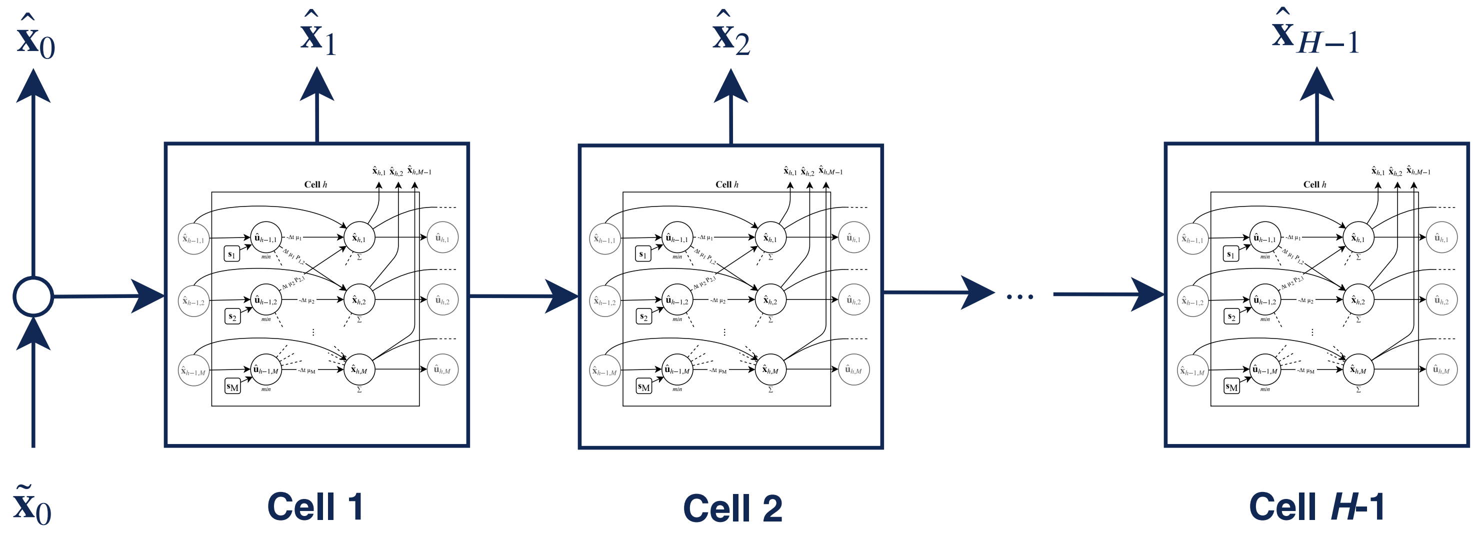


How our procedure works



Recurrent Neural Networks

- Recurrent neural networks (RNN) work with sequences (e.g. time series)
- We will encode the model as a RNN with a custom structure.

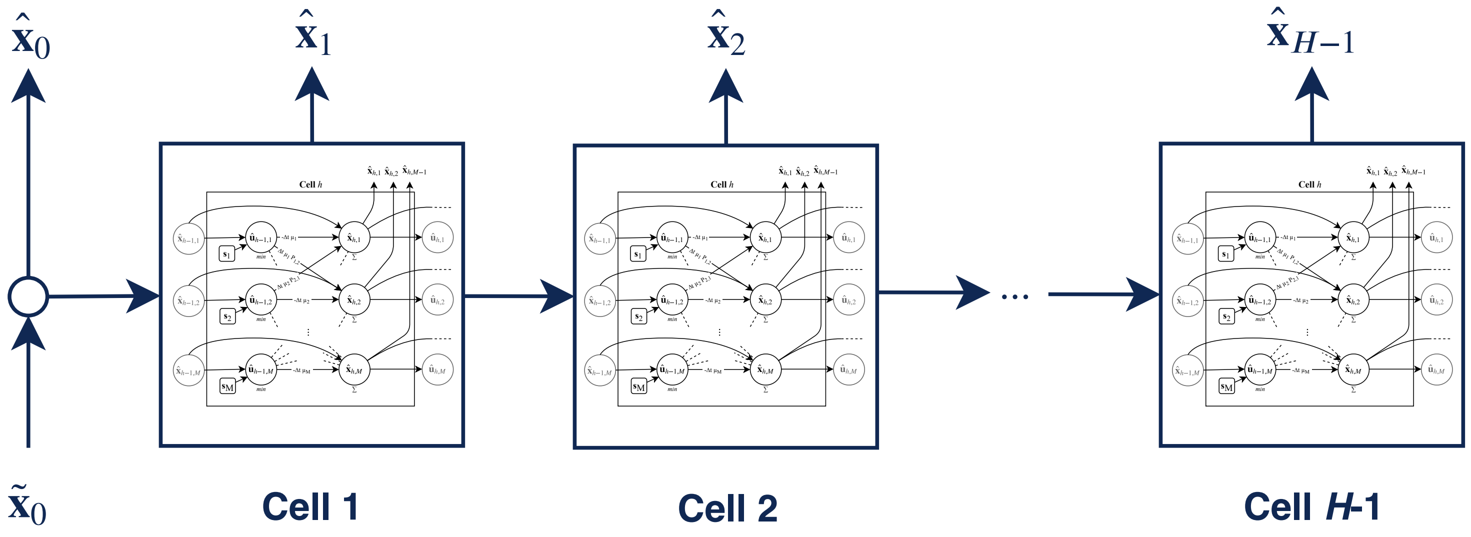


Recurrent Neural Networks

- The system parameters are directly encoded in the RNN cell

➔Learned model explains the system! (Explainable Neural Network)

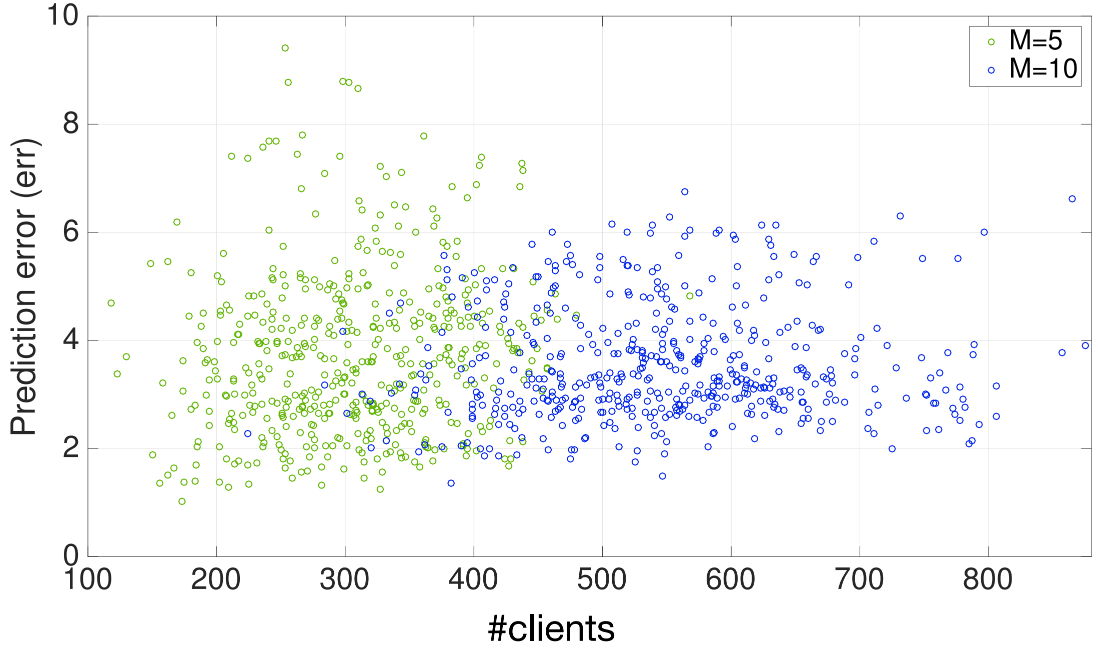
- We can modify the system afterwards to do prediction!



Synthetic case studies: setting

- 10 random systems: five with M=5 stations, five with M=10 stations
- Concurrency levels between 15 and 30
- Service rate between 4 and 30 clients/time unit
- 100 traces, each one being an average of 500 executions, with [0, 40 M] clients
- Learning time: 74 min for M = 5 and 86 min for M = 10
- Error function: % clients wrongly placed

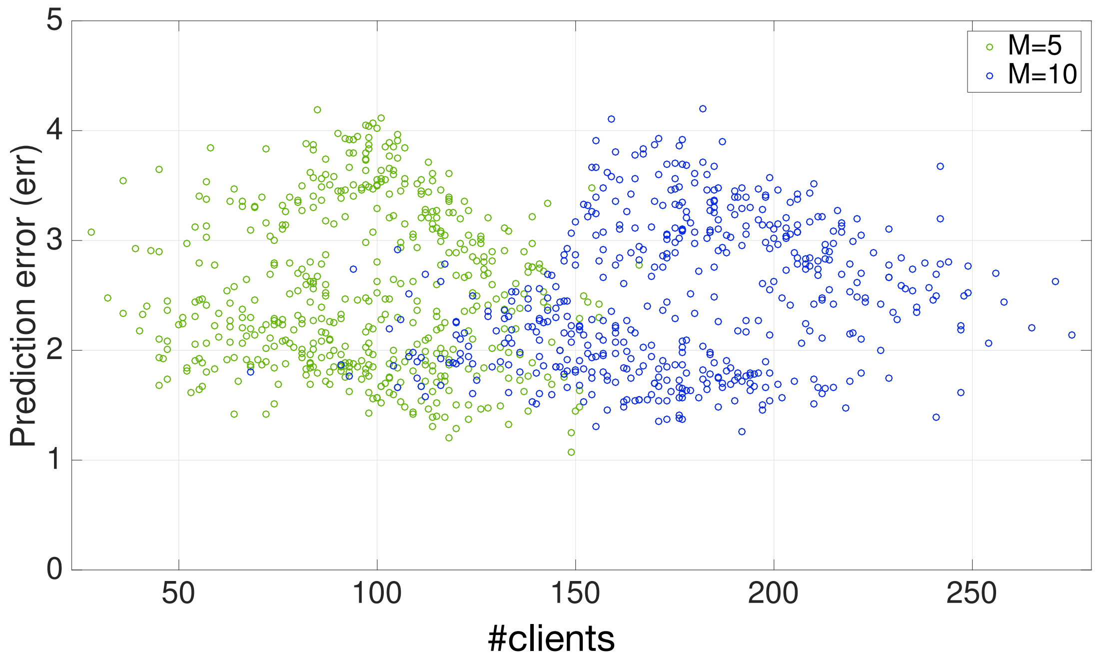
Synthetic case studies: prediction with different #clients



No significant difference among network size and number of clients.

➔ Good predictive power among different conditions

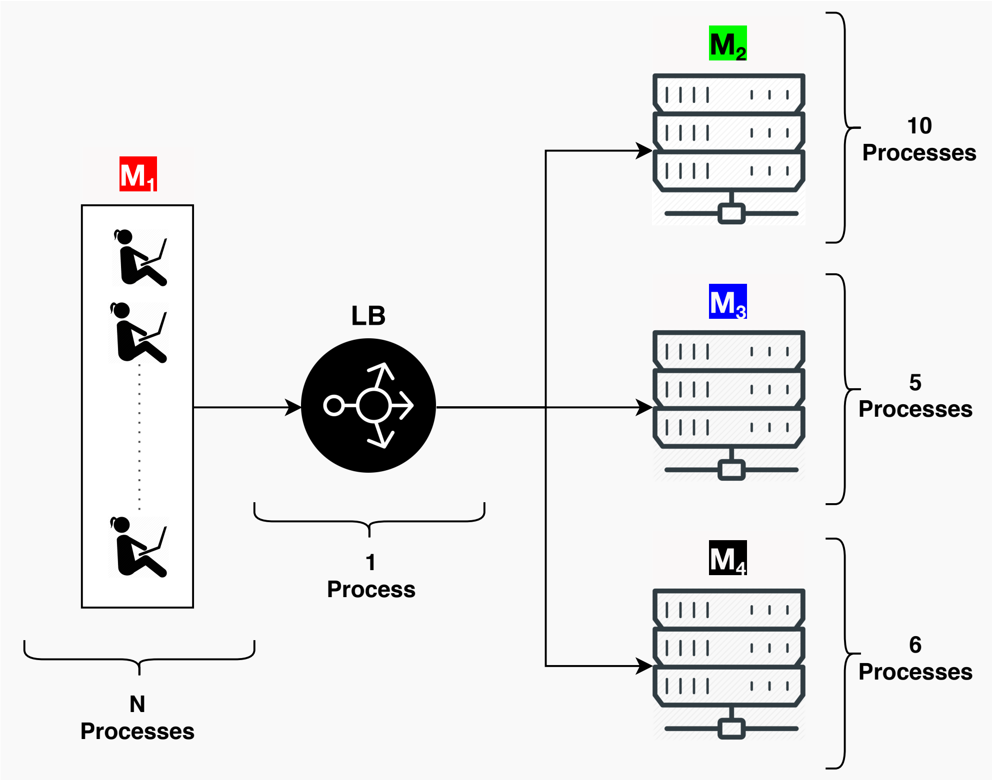
Synthetic case studies: prediction with different concurrency levels



Increased concurrency as to resolve the bottleneck

➔ Learning outcome resilient to changes in part of the network

Real case study: setting

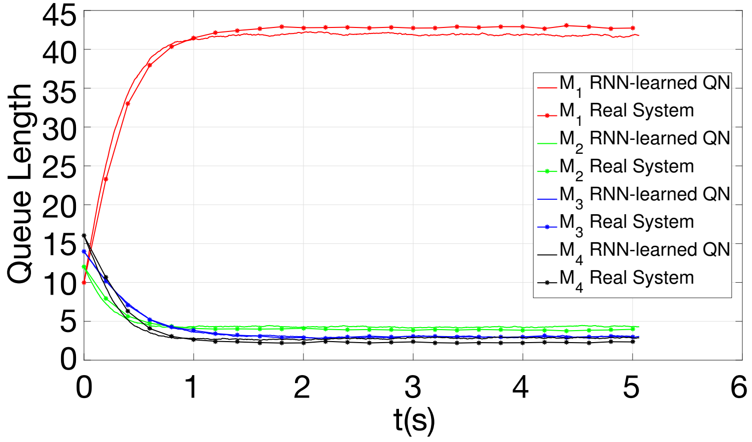


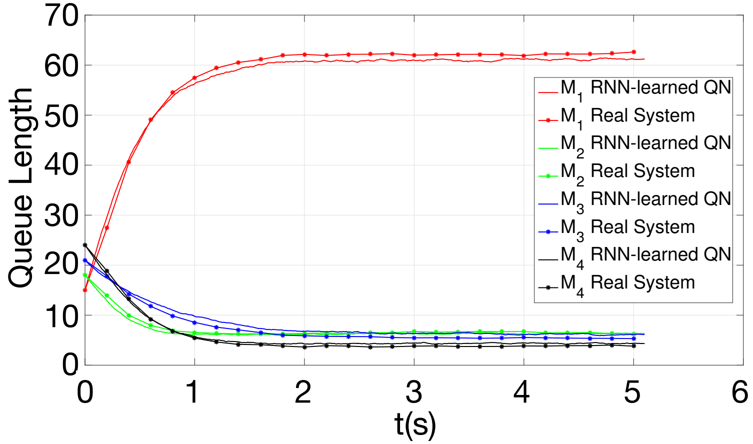
- node.js web application, replicated 3 times
- Python script simulates N clients
- Learning time: 27 min for N=26

Real case study: prediction with different #clients

N = 52

err = 6.46%



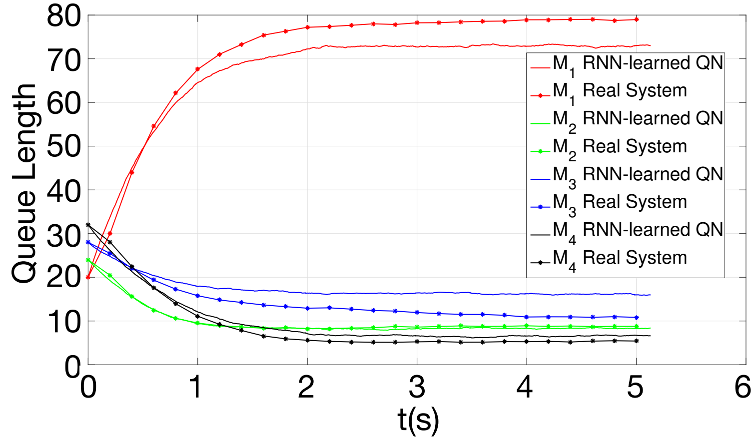


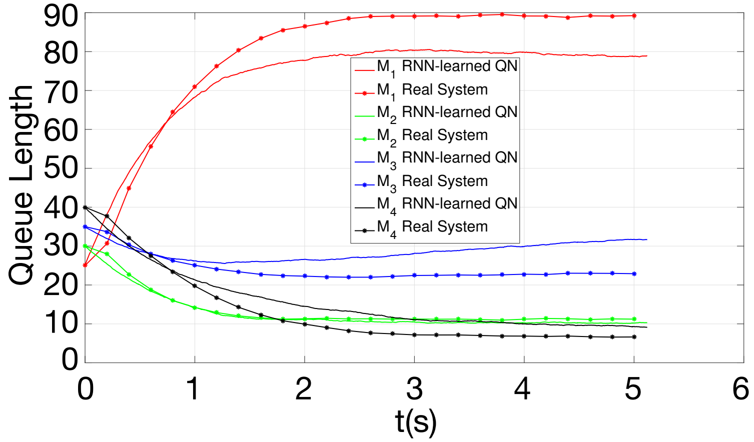
N = 78

err = 5.03%

N = 104

err = 6.45%





N = 130

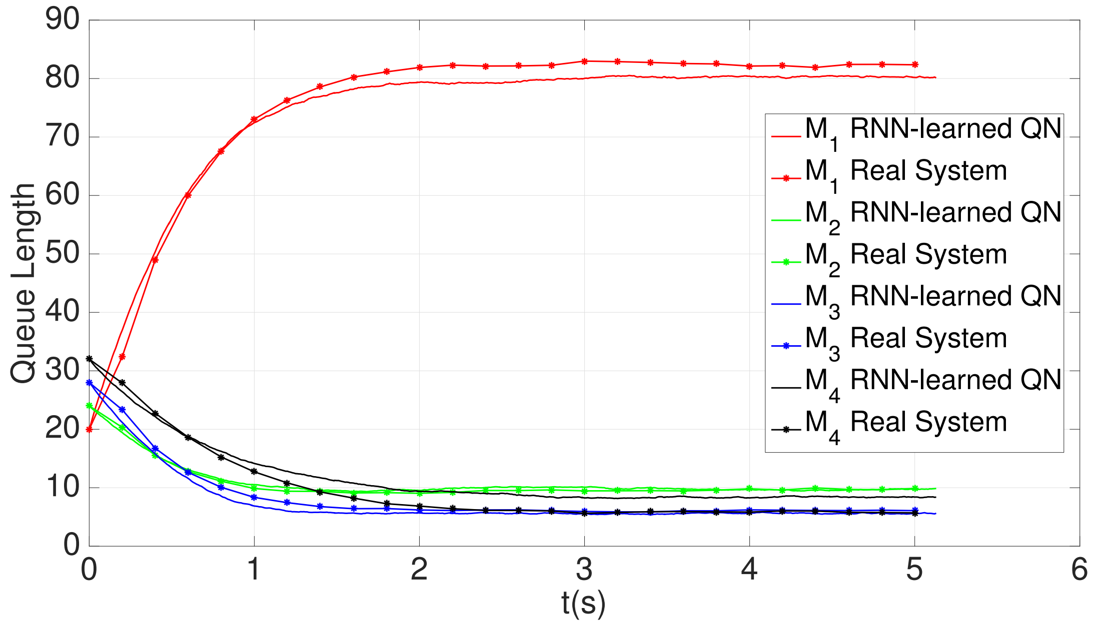
err = 9.05%

M_3 is the bottleneck, and this affects the UX. We need to solve it…

Real case study: prediction with different structure

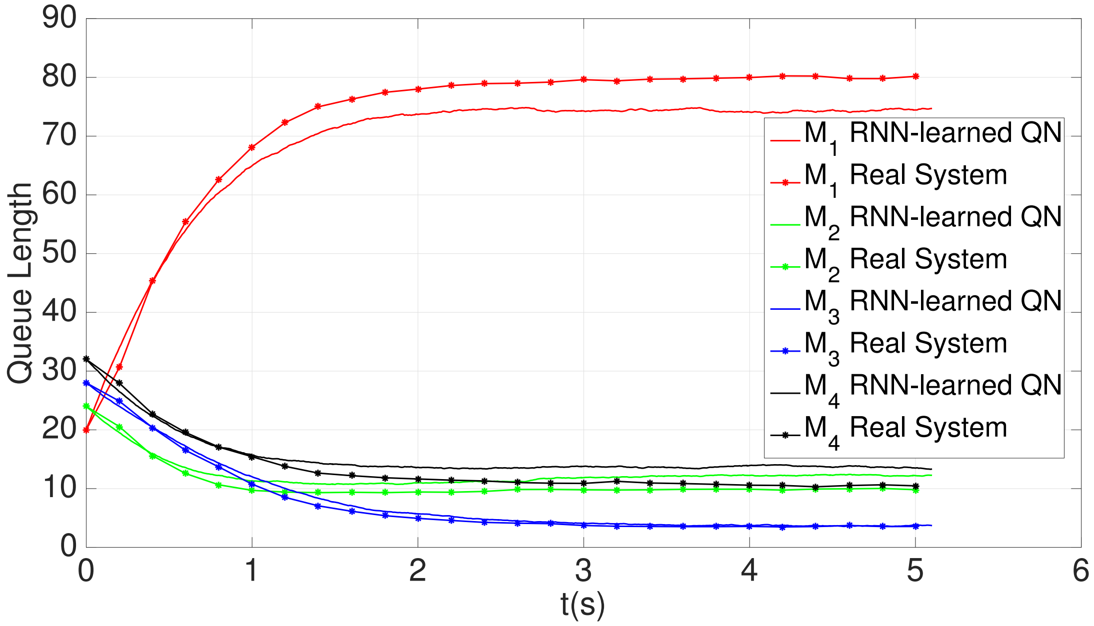
...by increasing the concurrency level of M_3

err: 5.98%



...by changing the LB scheduling policy

err: 6.10%



Bottleneck solved. Nice results also on a real HW+SW system.

Limits

- Many traces required to learn the system.
- System must be observed at high frequency.
- Layered systems currently not supported.
- Resilient to limited changes, not extensive ones.

Related work

- Performance models from code (e.g. PerfPlotter, not predictive)
- Modelling black-box systems (e.g. Siegmund et al., tree-structured models)
- Program-driven generation of models (e.g. Hrischuk et al., distributed components that communicate via RPC)
- Estimation of service demands in QN through several techniques (we estimate service demands and routing matrix)

Conclusions

- We provided a method to estimate QN parameters using a RNN that converges on feasible parameters.
- With the estimated parameters, it is possible to estimate the evolution of the system using a population different from the one used during learning or when doing structural modifications.
- We want to apply the technique to more complex systems (e.g LQN,multiclass), use other learning methodologies (e.g. neural ODEs) and improve the accuracy of the results

Thank you!