A HYBRID REINFORCEMENT LEARNING APPROACH TO AUTONOMIC RESOURCE ALLOCATION

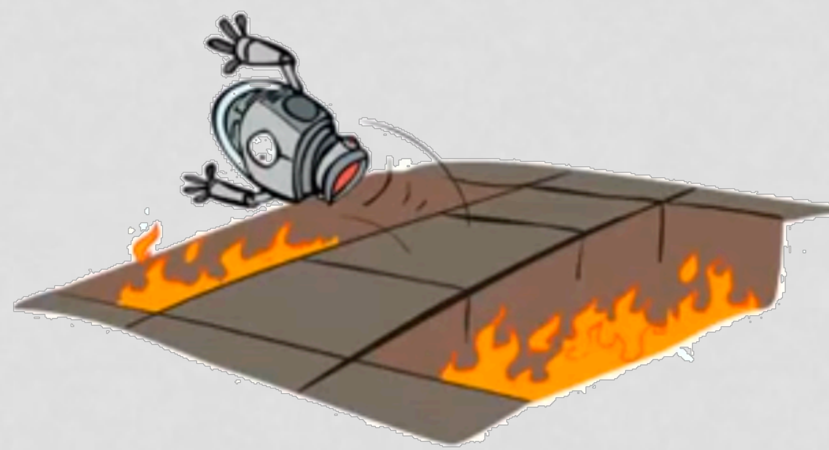
by Gerald Tesauro, Nicholas Jong, Rajarsa Das, Mohamed Bennani

presented by **ARDA GUMUSALAN**

BACKGROUND

REINFORCEMENT LEARNING - *OVERVIEW*

- Relies on the association between a goal and reward (penalty) value.
- Objective is to progressively find, by trial and error, the **policy** maximizing the rewards.
  - A **policy** is a mapping from states to actions.
- The reinforcement value permits the agent to modify its behavior.
- **Online learning**.



REINFORCEMENT LEARNING - *OVERVIEW*

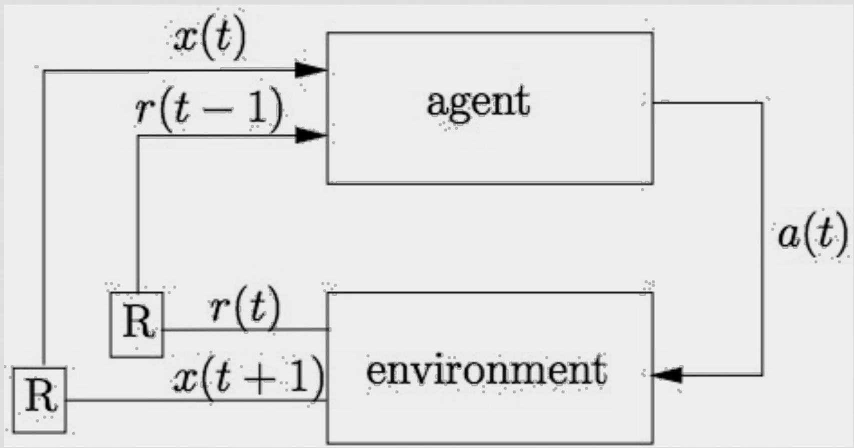
Reinforcement learning

- The received signal is usually either *positive*, *negative*, or *neutral*.

Supervised learning

- The received signal is the error : the difference between the actual and the reference output.

REINFORCEMENT LEARNING - *OVERVIEW*



- Consists of:
  - A set of environment states
  - A set of actions
  - Rules of transition between states
  - Rules to determine *reward*
  - Rules to observe the new state

SARSA

$$\Delta Q(s_t, a_t) = \alpha(t)[\underbrace{r_t + \gamma \underbrace{Q(s_{t+1}, a_{t+1})}_{\text{Do another action}} - \underbrace{Q(s_t, a_t)}_{\text{Old value}}}_{\text{Learned value}}]$$

- This particular paper uses SARSA algorithm.
  - s,a,r,s',a'
  - (s_t, a_t): original state
  - r_t: immediate observed reward at t.
  - s': next state
  - a': next action

- Q value is the sum of expected reward values of all future steps.
- α is a learning rate.
- γ is a discount parameter.
  - [0,1]
  - Trades off the importance of sooner versus later rewards.

SARSA

- SARSA is usually more cautious and takes longer time to find the optimal solution.
- Under these conditions SARSA is proven to converge to the optimal value.
  - SARSA typically is represented by using look up table.
  - The environment is represented as Markov Decision Process.
  - The decision needs to be made globally.

- **However, this work does not have any of these conditions.**

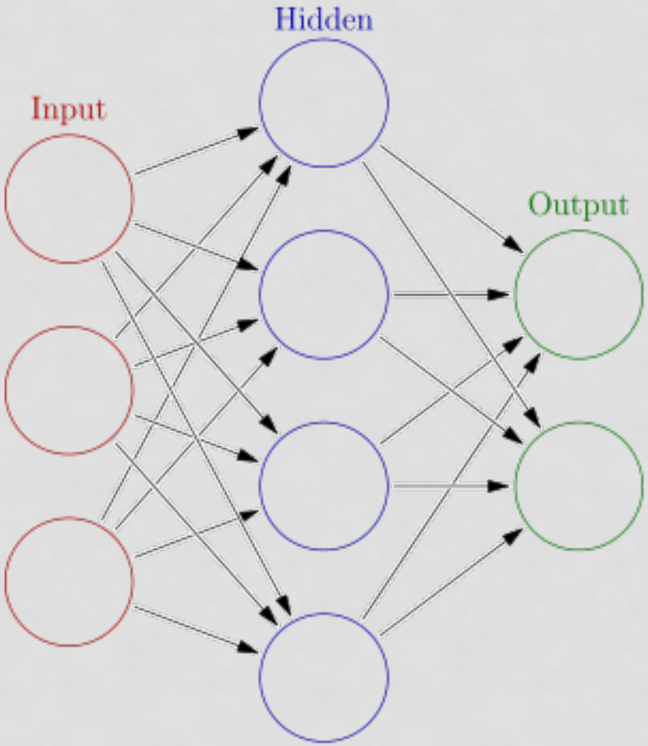
ACTION SELECTION

- Many approaches.
  - Including randomized approach.
- It mat not be reasonable to consider every option available at every instant in time.
  - There must be a way to constraint the options.
- More recent approach is neural network based action selection.

NONLINEAR APPROXIMATOR

- Approximator: resolves a possibly complicated function by simpler and easier to compute functions.
  - **What is our function now?**
- In this case our approximator is nonlinear.
- Many techniques:
  - Regression trees
  - Support Vector Machines
  - Wavelets
  - Splines
  - Neural Network

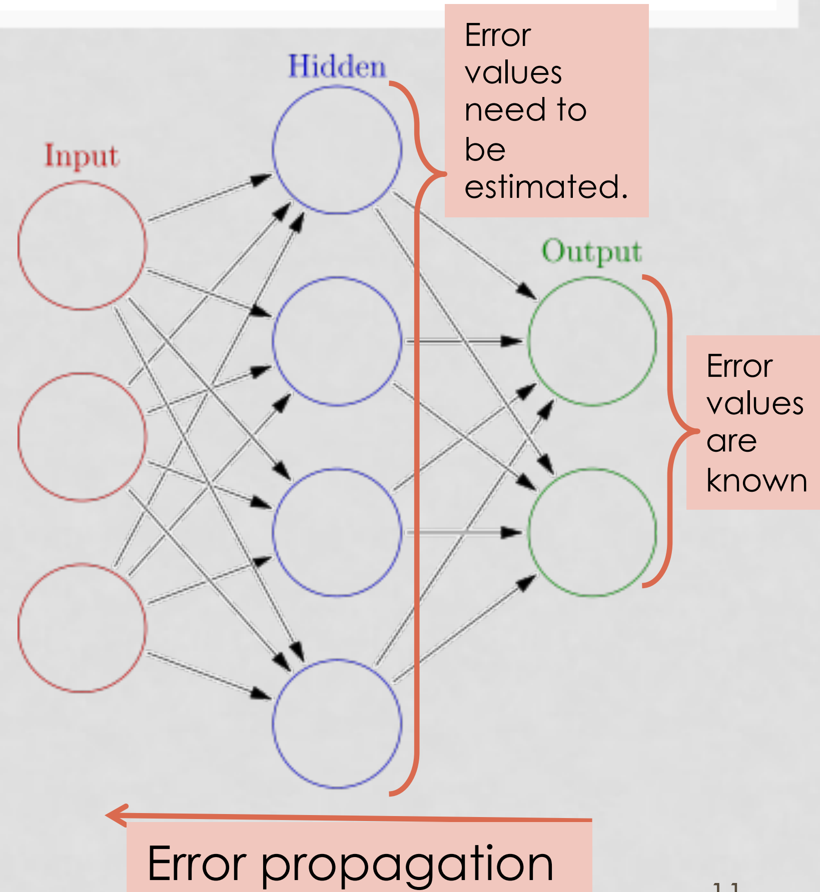
NEURAL NETWORK



- Actual computation is done at hidden layers.
- Weighted edges.
- Contains some form of learning rule.
  - Back-propagation is the one in our case.

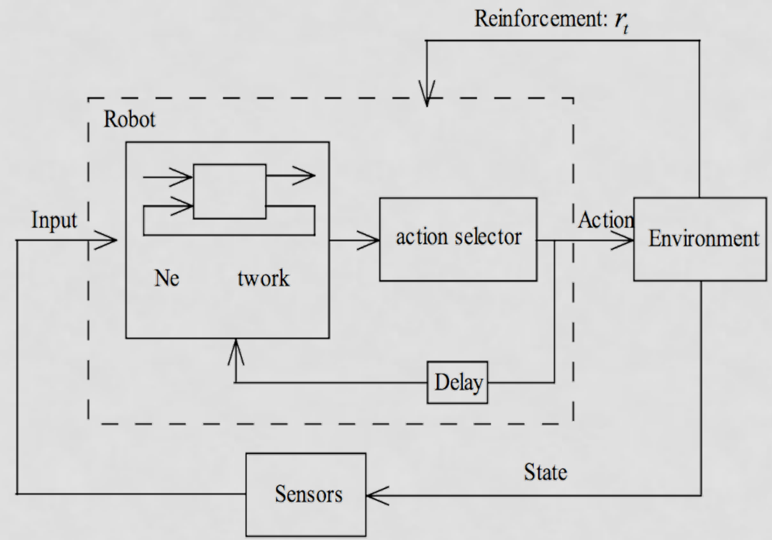
BACK-PROPAGATION

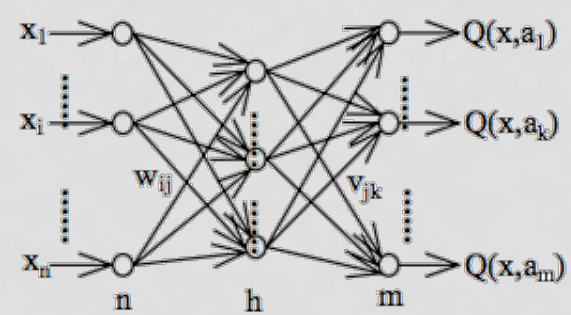
- When a neural network is initially presented with a pattern, it makes a random "guess" as obtain the intended output.
- It then sees the *error*.
- Based on the *error* calculated at the *output* nodes, the error value propagates towards *input* nodes.



NEURAL NETWORK AS ACTION SELECTOR

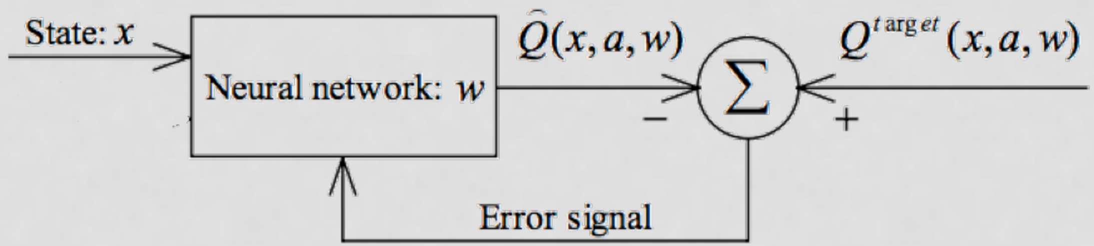
- Called **Reinforcement Learning Neural Network** (RLNN)
- NN has n input neurons, h hidden neurons, and m output neurons
  - Each m corresponds to an action value.
  - Input neurons correspond to the current state.





(Bing-Qiang, 2005)

RLNN TRAINING



- The training process is correlated for RL and NN.

(Bing-Qiang, 2005)

INTRODUCTION

MOTIVATION

- The current queuing-theoretic approach for Autonomic Systems makes use of explicit system performance models.
- However, the design and implementation of accurate performance models of complex systems are highly knowledge and labor intensive.
  - Enough data to create one may be not be available.

PROBLEM DESCRIPTION

- **Question**: Given the central goal of autonomic computing, is it possible to automatically learn high-quality management policies without an explicit system performance models?
- **Answer**: Yes.
  - The authors have shown the feasibility of using RL in their previous works.
  - RL can be used to learn resource valuation estimates which can be used to make high-quality server allocation decisions in a Data Center.

USING RL

- Advantage of RL:
  - No explicit model of computing system is needed.
  - No explicit model of workload or traffic generation is needed.
  - Can adapt to dynamic environments.
    - Can reflect the possibility that a current decision may have delayed consequences for future states.

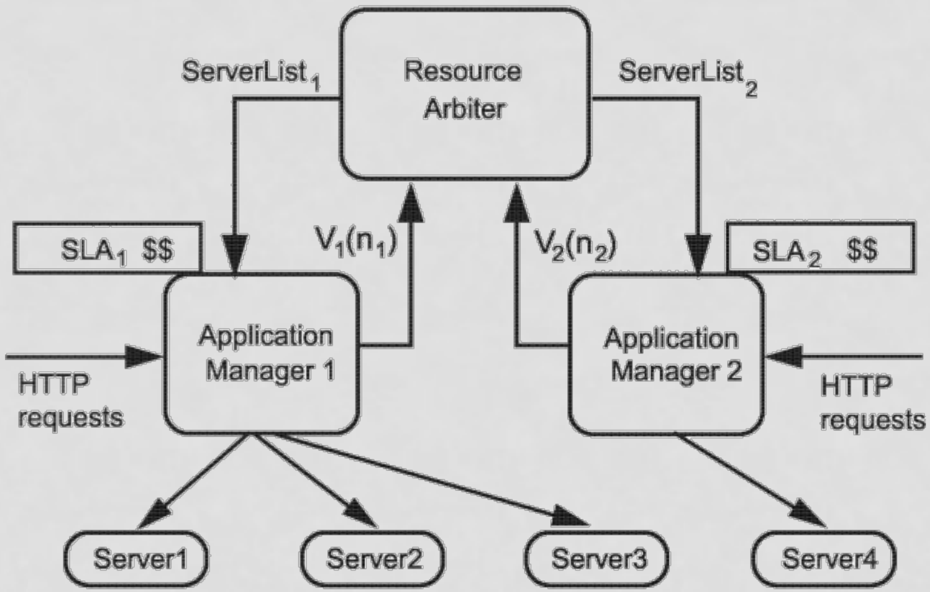
USING RL

- Disadvantage of RL:
  - Can suffer from poor scalability for large state spaces.
    - For each state an action pair, an entry is stored in a look up table.
    - Increases exponentially.
  - Poor live online training performance.
    - If there is no domain knowledge or a good heuristic.
    - Some of the *random* action trials can be too costly in real life.
    - **Your company might lose all of its customers before your system starts to perform well**.

SYSTEM MODEL

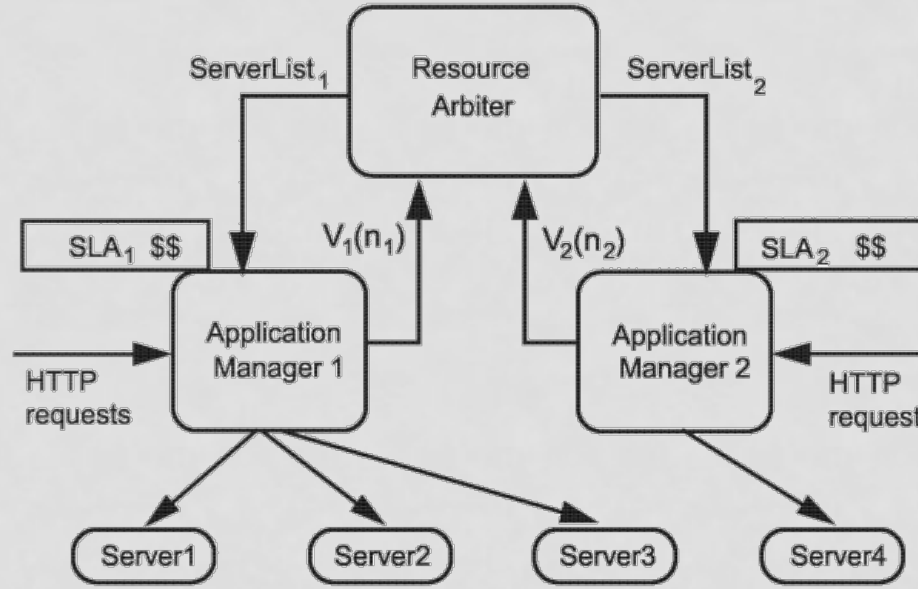
PROTOTYPE DATA CENTER

- Example data center:
  - Set of identical servers.
  - Multiple web applications.
  - How to dynamically allocate the servers among applications?



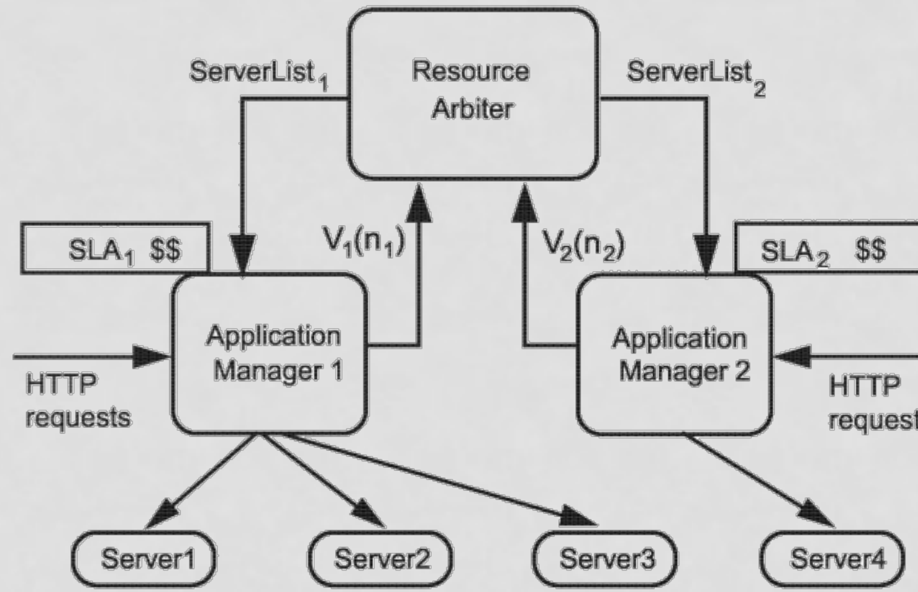
PROTOTYPE DATA CENTER

- Each application has its own *application manager.*
  - Performs optimization within the application.
    - Based on a **local** performance function called ***expected business value***.
  - Communicates with *resource arbiter* for the resource needs.



PROTOTYPE DATA CENTER

- $V_i(.)$ is the estimate of expected business value as a function of number of servers.
  - Expected revenue – SLA violation penalties.
- Arbiter maximizes the global utility and sends the list of assigned servers to each application manager.
- Decision is made once in every 5 seconds.



HYBRID RL

HYBRID RL - *OVERVIEW*

- Train the RL offline on data collected while an externally supplied initial policy makes the management decisions.
- It is proven that if enough training data is given, the policy created by RL will improve the original policy.
- Avoids the initial poor system performance that could occur while training.
  - Can be applied recursively: p, p', p'',…
  - *This also implies that the RL stops learning after being employed?*

HYBRID RL - *TRAINING*

- Training of RLNN can be done with either *batch* or *online* manner.
- In *batch* the weights are updated after presenting the entire training data (all inputs and targets).
  - The sample complexity might lead higher iterations.
  - The targets are non-stationary, as the approximator moves towards a set of targets, this causes the targets themselves to change.

HYBRID RL - *ALGORITHM*

Initialize Q to a random neural network
repeat
  $SSE \leftarrow 0$ {sum squared error}
  for all t such that $0 \leq t < T$ **do**
    $target \leftarrow r_t + \gamma Q(s_{t+1}, a_{t+1})$ — Current reward + reward of the new action
    $error \leftarrow target - Q(s_t, a_t)$ — Current reward + reward of the new action – old expected reward
    $SSE \leftarrow SSE + error \cdot error$
    Train $Q(s_t, a_t)$ towards $target$ — Back propagation
  end for
until CONVERGED(SSE)

$$\Delta Q(s_t, a_t) = \alpha(t)\underbrace{[r_t + \underbrace{\gamma Q(s_{t+1}, a_{t+1})}_{\text{Do another action}} - \underbrace{Q(s_t, a_t)}_{\text{Old value}}]}_{\text{Learned value}}$$

HYBRID RL - *ALGORITHM*

- Using Mean-Squared-Error does not give any theoretical guarantees for converging.

- Due to stochastic gradient nature of Sarsa/back-propagate, there is noise in the error measurements.
  - Gradient: how the error value changes with respect to weight values.
  - The convergence criterion must maintain sufficient history of prior SSE.

HYBRID RL – *STATE REPRESENTATION*

- The state s_t must be fully observable.
  - It is best to include as many sensor reads as possible.
    - However, it is problematic if RL cannot distinguish the most relevant ones.
- Previous work by the same authors has shown solely using current demand, λ, is a reasonable approximation.
- $(s,a) = (\lambda, n)$
  - Not enough.
  - Does not reflect the dynamic consequences of allocation decisions such as switching delays.
    - Initial unavailability of a reallocated server.
    - The cost of setting up the reallocated server.
    - **Big difference between going from 2 to 5 and 4 to 5**.

HYBRID RL – *STATE REPRESENTATION*

- Employ a delay aware representation.
  - Previous allocation decision is added to the new state.

- $(\lambda_t, n_{t-1}, n_t)$
  - As long as the delay does not affect more than one allocation interval (5 sec), this should work.

INITIAL QUEUING MODEL POLICIES

OPEN-LOOP QUEUING NETWORK MODEL

- Has external arrival and departure requests from an infinite population of costumers.
- HTTP requests are assigned via *round-robin* fashion.
- *n* servers with *n* independent and identical parallel open networks.
  - Each demand level is λ / n
- Apply M/M/1 queuing formulation in parallel.
  - Poisson arrival process.
  - Exponential service times.

OPEN-LOOP QUEUING NETWORK MODEL

- SLA requirement is a sigmoidal function of mean-response-time.
- M/M/1
  - μ is the service rate
  - λ is the arrival rate
  - R is the mean response time

$$R = \frac{1}{\mu - \lambda}$$

- For the next allocation time:
  - Predicted mean arrival rate is the same
    - $\lambda_{t+1} = \lambda_t$
  - The workload is divided equally among the servers
    - Each server has λ_t / n_{t+1}

Estimated response time

$$R_{t+1} = \frac{1}{\mu - \frac{\lambda_t}{n_{t+1}}}$$

OPEN-LOOP QUEUING NETWORK MODEL

- μ can be estimated by the same formula for the current allocation period.

$$\mu = \frac{1}{R_t} + \frac{\lambda_t}{n_{t+1}}$$

- However, quiet sensitive to the variations in R_t.
  - Use an exponential smoothing for μ before solving for R_{t+1}.
- Experiments shown that the error in prediction is less than 20%.

CLOSED-LOOP QUEUING NETWORK MODEL

- Has a finite population of customers, each alternating between think-state and submitted-state.
- HTTP requests are distributed via *round-robin* fashion.
- M customers.
- n independent parallel closed networks.
    - M/n customers per server.

CLOSED-LOOP QUEUING NETWORK MODEL

- Modeling *mean response time* is done by using ***Mean Value Analysis*** (MVA).
  - Two unknown model parameters:
    - Average think-time Z
    - Average service time at the server, $1/\mu$
- Using interactive response time law:
  - $M_{t+1} = M_t$
- M/M/1 model can be used to estimate $1/\mu$ a.k.a mean service time.

$$Z = \frac{M_t}{X_t} - R_t$$

X_t is the measured average throughput of the application

$$\mu = \frac{1}{R_t} + \lambda_t$$

CLOSED-LOOP QUEUING NETWORK MODEL

- Exponential smoothing is applied again on the estimates of Z and μ.
  - Accounts for finite sampling effects and Java garbage collection.

PERFORMANCE EVALUATION

SYSTEM PARAMETERS

- 8 identical servers.
- 3 applications.
  - T1 and T2 are both running a realistic simulation of an electronic trading platform, designed to benchmark web servers.
    - SLA requirement is a sigmoidal function of *mean-response-time*
    - Rage is [-150,50]
  - T3 is a non-web based, CPU intensive computation.
    - Monte Carlo simulations.
    - Batch workloads that can be paused and restarted.
    - SLA requirement is an increasing function of number of assigned servers.
    - Range is [-78,68]

SYSTEM PARAMETERS

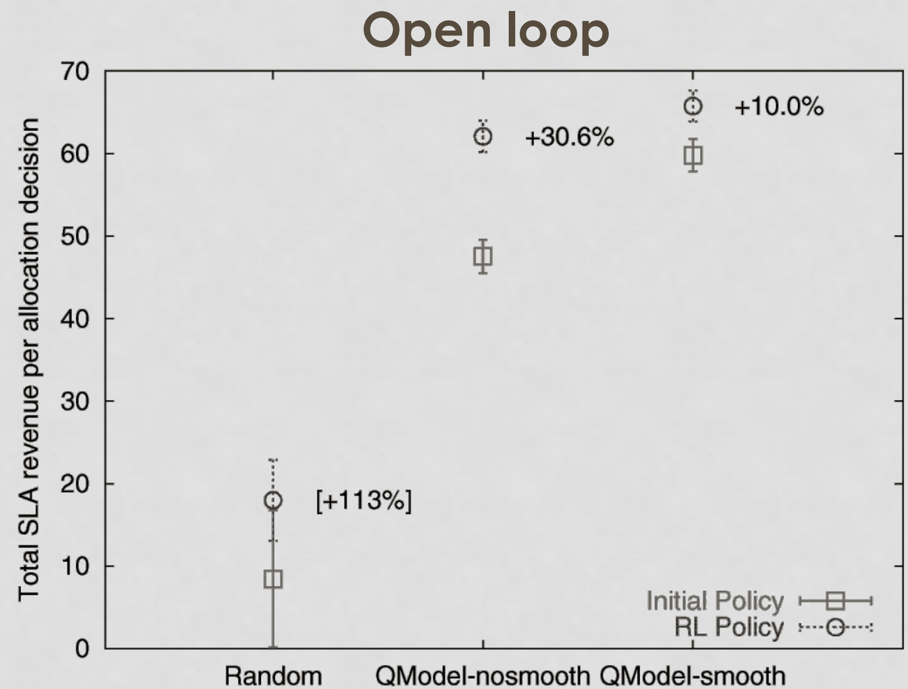
- Open loop generates Poisson HTTP requests with mean arrival rate between 10 and 400 requests/second.
- Closed loop generates 5 to 90 number of costumers.
  - Exponentially distributed think-state duration with fixed mean Z=0.17
- Stochastic bursty time-varying demand is embedded.

HYBRID RL PARAMETERS

- α , learning rate is 0.005
- γ , discount rate is 0.5
  - Observed that it took 10-20K iterations to converge.

PERFORMANCE RESULTS - *NO SWITCHING DELAY*

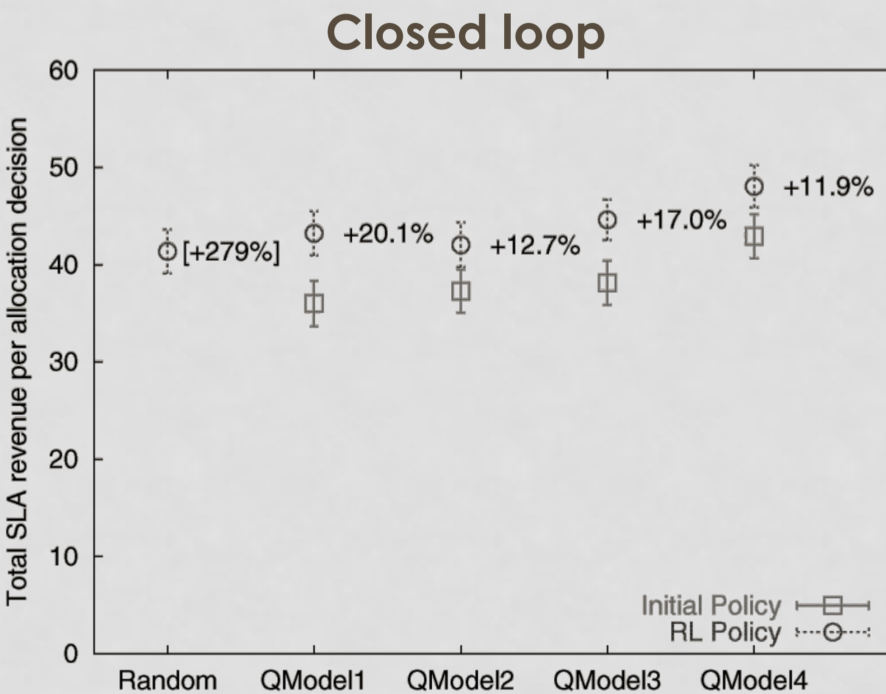
- Performance measure:
  - total SLA revenue per allocation decision summed over all three applications.
- x-axis:
  - **Random**: uniformly random server allocation.
  - **Qmodel-nosmooth**: Described queuing prediction model with no smoothing.
  - **Qmode-smooth**: Described queuing model with smoothed μ value.



- T-test is also performed.
  - 1% significance with P-value ≤ 10^{-6}

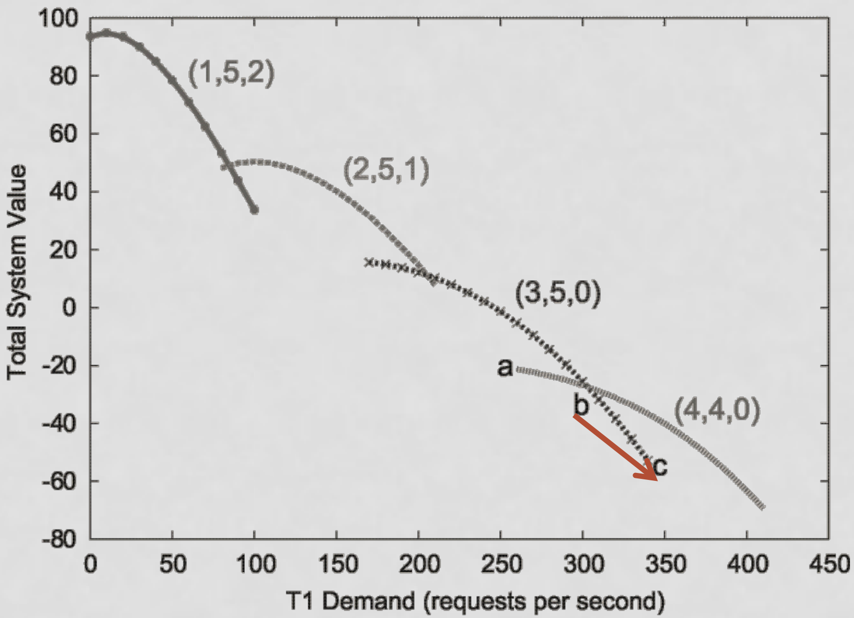
PERFORMANCE RESULTS – *NO SWITCHING DELAY*

- x-axis:
  - **Random**
  - **QModel1**: MVA approach with no smoothing.
  - **QModel2**: MVA approach with smoothing and utility estimation factor with 0.5 which leads to a predicting of cumulative future utility.
  - **QModel3:** same model used for open loop.
  - **QModel4:** MVA approach with smoothing and prediction for only immediate utility.



POLICY HYSTERESIS

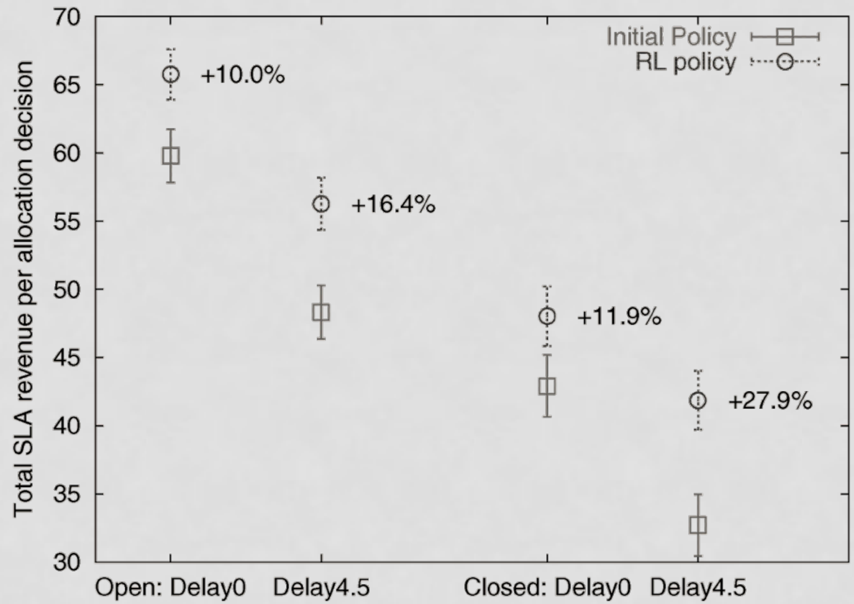
No switching delay



- **Shows the portion of learned joint value function.**
  - **Estimating total value summed over all three applications.**
  - **Demand is only for one Trade3 application.**

- The causes:
  - Unable to perform any work during the delay interval.
  - Transient period of suboptimal performance within the application.
    - Due to starting time of new processes.
  - Switching the server might be for a short term.
    - Needs to be switched back with increases the delay.
  - Thrashing:
    - Removing a server may lead to increase servers again for the same application so the server is immediately switched back.

PERFORMANCE RESULTS – *WITH SWITCHING DELAY*



- The amount of policy improvement of hybrid RL increases.

PERFORMANCE RESULTS – *WITH SWITCHING DELAY*

- The results are averaged over two T1 and T2.
- The mean number of servers assigned is less for RL than QM.
- Noticeably less server swaps using RL.
  - ≈ 50% reduction.
  - Due to **greater hysteresis** and **elimination of trashing**.

Mean change in number of assigned servers

Experiment	$< n_T >$	$< \delta n_T >$
Open-loop Delay=0 QM	2.27	0.578
Open-loop Delay=0 RL	2.04	0.464
Open-loop Delay=4.5 QM	2.31	0.581
Open-loop Delay=4.5 RL	1.86	0.269
Closed-loop Delay=0 QM	2.38	0.654
Closed-loop Delay=0 RL	2.24	0.486
Closed-loop Delay=4.5 QM	2.36	0.736
Closed-loop Delay=4.5 RL	1.95	0.331

Mean number of servers

PERFORMANCE RESULTS – *WITH SWITCHING DELAY*

- Reduction for RL nets for 4.5 second delay compared to 0 delay.

Mean change in number of assigned servers

Experiment	$< n_T >$	$< \delta n_T >$
Open-loop Delay=0 RL	2.04	0.464
Open-loop Delay=4.5 RL	1.86	0.269
Closed-loop Delay=0 RL	2.24	0.486
Closed-loop Delay=4.5 RL	1.95	0.331

Mean number of servers

CONCLUSION AND CRITIQUE

CONCLUSION

- Demonstrated the success of combining model-based policies and reinforcement learning of the policies.
- RL does not require an explicit system or traffic model.
- Delay aware representation naturally handles switching delays.
  - This lies outside of traditional steady-state queuing models.
- Wide range of applicability.

CRITIQUE

- Interesting and valuable work but:
- The continuous online learning is not embedded.
  - Only possible with offline training.
- Very badly explained, so many left out points.
  - Overall idea is given, almost no details.
  - I need to see more details on how they did it.
  - In hysteresis analysis, they should have put another graph with the delay constraint.
  - The simulation settings are distributed over 3 sections.
  - **Couple actual algorithm iteration must have been shown.**
  - **Meaningless pseudo code, applies to almost any case, needs to be more specific about the states and the actions.**
  - V function? Only described.
  - **What is the reward function?**
    - Is it the SLA calculation?