Autonomic Container Elasticity through Reinforcement Learning

Fabiana Rossi

Department of Civil Engineering and Computer Science Engineering
University of Rome Tor Vergata, Italy

Exploiting Elasticity

- **New scenario**: *IoT edge/fog computing*

- **New software architectures**: *container-based architectures*

- **Common requirement**: *elasticity*

Goal

To adapt at run-time the deployment of container-based applications jointly consider horizontal and vertical scalability.

Solutions proposed in literature:

- threshold-based policies;
- optimization (mainly ILP);
- heuristics;
- control theory;
- time series analysis;
- Reinforcement Learning.

Solutions proposed in literature:

- threshold-based policies;
- optimization (mainly ILP);
- heuristics;
- control theory;
- time series analysis;
- Reinforcement Learning.

Limits of existing solutions:

- many works consider horizontal elasticity, few works consider vertical elasticity;
- lack of general approaches.

Solutions proposed in literature:

- threshold-based policies;
- optimization (mainly ILP);
- heuristics;
- control theory;
- time series analysis;
- Reinforcement Learning.

Limits of existing solutions:

- many works consider horizontal elasticity, few works consider vertical elasticity;
- lack of general approaches.

Main Contributions

Autonomic elasticity of container-based applications:

- Horizontal or Vertical
- Horizontal and Vertical

Reinforcement learning algorithms:

- Q-learning
- Dyna-Q
- Model-based

model-free

model-based

System Definition

- Container-based applications
- Per-application RL agent

- RL agent adapts:
  - number of containers
  - amount of resources assigned to each application

System Definition

- Container-based applications
- Per-application RL agent

- RL agent adapts:
  - number of containers
  - amount of resources assigned to each application

For each application, we define:

- $s = (k, u, c)$: application state
- k : number of containers
- u : CPU utilization (discretized)
- c : CPU quota assigned to each container
- a : action carried out in the state s
  - 5 Actions: a in $\{-1, 0, 1, -r, r\}$
  - 9 Actions: a in $\{-1, 0, 1\} \times \{-r, 0, r\}$

Immediate Cost Function

Immediate Cost associated to each triple *(s, a, s')*.

$$
\boxed{\begin{aligned} c(s, a, s') = {} & w_{\mathrm{rcf}} \cdot c_{\mathrm{rcf}} \\ & + w_{\mathrm{perf}} \cdot c_{\mathrm{perf}} \\ & + w_{\mathrm{res}} \cdot c_{\mathrm{res}} \end{aligned}}
$$

Immediate cost defined as the weighted sum of:

- reconfiguration cost;
- performance cost;
- resource cost.

Immediate Cost Function

Immediate Cost associated to each triple (*s*, *a*, *s'*).

$$c(s, a, s') = w_{\mathrm{rcf}} \cdot \boxed{c_{\mathrm{rcf}}} + w_{\mathrm{perf}} \cdot \boxed{c_{\mathrm{perf}}} + w_{\mathrm{res}} \cdot \boxed{c_{\mathrm{res}}}$$

c_{rcf}: Vertical scaling operations lead to application unavailability

c_{perf}: Penalty if the application response time violates the SLA

c_{res}: Resource cost, proportional to the amount of allocated resources (k and c)

Reinforcement Learning Algorithms

- $Q(s, a)$: estimate of long-term cost due to the execution of action a in s

Reinforcement Learning Algorithms

- *Q (s, a):* estimate of long-term cost due to the execution of action *a* in *s*
- **Q-learning**
  - uses *Q (s, a)* to choose the action to be performed in the state *s*
  - action selection policy: ε-greedy

Reinforcement Learning Algorithms

- *Q (s, a):* estimate of long-term cost due to the execution of action *a* in *s*
- **Q-learning**
  - uses *Q (s, a)* to choose the action to be performed in the state *s*
  - action selection policy: ε-greedy
  - estimates *Q(s, a)* from the experience:

$$Q\left(s_i, a_i\right) \leftarrow (1-\alpha) Q\left(s_i, a_i\right) + \alpha\left[c_i + \gamma \min_{a' \in \mathcal{A}\left(s_{i+1}\right)} Q\left(s_{i+1}, a'\right)\right]$$

Reinforcement Learning Algorithms

- *Q (s, a):* estimate of long-term cost due to the execution of action *a* in *s*
- **Q-learning**
  - uses *Q (s, a)* to choose the action to be performed in the state *s*
  - action selection policy: ε-greedy
  - estimates *Q(s, a)* from the experience:

$$Q\left(s_i, a_i\right) \leftarrow (1-\alpha) Q\left(s_i, a_i\right)+\alpha\left[c_i+\gamma \min_{a^{\prime} \in \mathcal{A}\left(s_{i+1}\right)} Q\left(s_{i+1}, a^{\prime}\right)\right]$$

- **Dyna-Q**
  - same features as Q-learning;

Reinforcement Learning Algorithms

- *Q (s, a):* estimate of long-term cost due to the execution of action *a* in *s*
- **Q-learning**
  - uses *Q (s, a)* to choose the action to be performed in the state *s*
  - action selection policy: ε-greedy
  - estimates *Q(s, a)* from the experience:

$$Q\left(s_i, a_i\right) \leftarrow (1-\alpha) Q\left(s_i, a_i\right)+\alpha\left[c_i+\gamma \min _{a^{\prime} \in \mathcal{A}\left(s_{i+1}\right)} Q\left(s_{i+1}, a^{\prime}\right)\right]$$

- **Dyna-Q**
  - same features as Q-learning;
  - mantains a model *(s,a)*$\longrightarrow$*(s', c)*;
  - uses the model to simulate the experience and update *Q(s, a)*.

Model-Based Reinforcement Learning

- selects the best action in terms of $Q(s, a)$

Model-Based Reinforcement Learning

- selects the best action in terms of *Q(s, a)*
- uses the Bellman equation to update Q (s, a):

$$Q(s,a) = \sum_{s' \in \mathcal{S}} p(s'|s,a) \left[c(s,a,s') + \gamma \min_{a' \in \mathcal{A}} Q(s',a') \right] \quad \begin{matrix} \forall s \in \mathcal{S}, \\ \forall a \in \mathcal{A}(s) \end{matrix}$$

Model-Based Reinforcement Learning

- selects the best action in terms of *Q(s, a)*
- uses the Bellman equation to update Q (s, a):

unknown

$$Q(s,a) = \sum_{s' \in \mathcal{S}} \boxed{p(s'|s,a)} \left[\boxed{c(s,a,s')} + \gamma \min_{a' \in \mathcal{A}} Q(s',a') \right] \quad \begin{array}{c} \forall s \in \mathcal{S}, \\ \forall a \in \mathcal{A}(s) \end{array}$$

Model-Based Reinforcement Learning

- selects the best action in terms of *Q(s, a)*
- uses the Bellman equation to update Q (s, a):

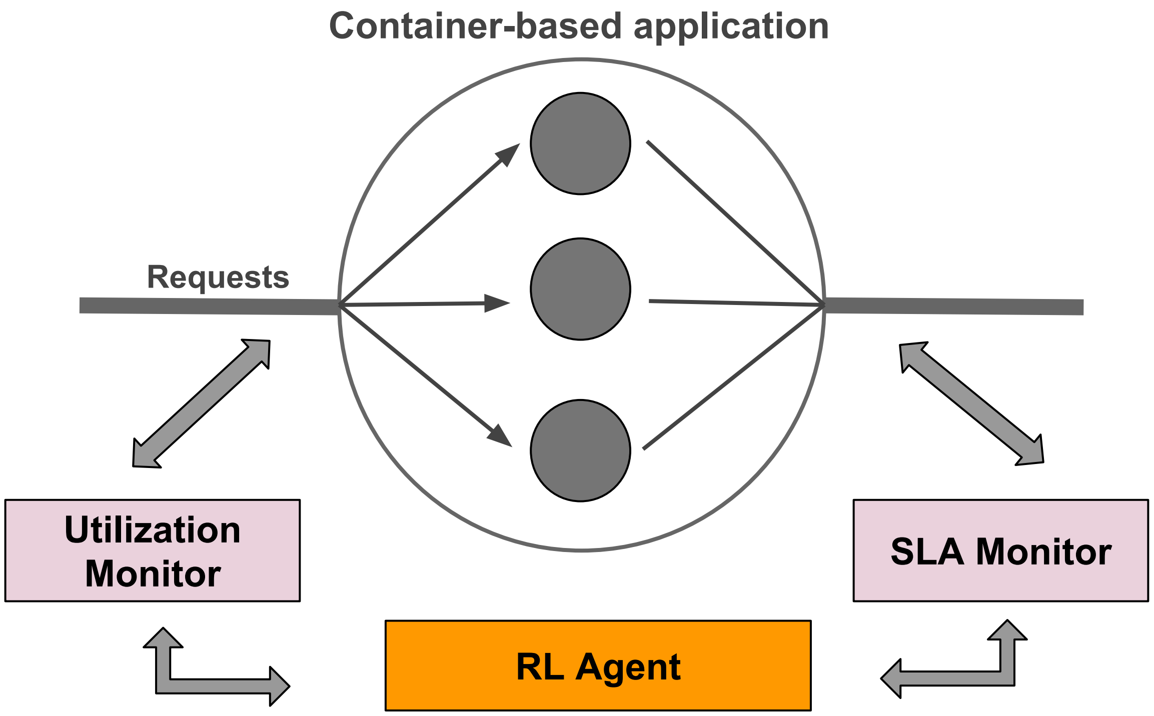
unknown

$$Q(s,a) = \sum_{s' \in \mathcal{S}} \boxed{p(s'|s,a)} \left[\boxed{c(s,a,s')} + \gamma \min_{a' \in \mathcal{A}} Q(s',a') \right] \quad \begin{matrix} \forall s \in \mathcal{S}, \\ \forall a \in \mathcal{A}(s) \end{matrix}$$

- **<u>Idea: to approximate unknown factors using experience</u>**

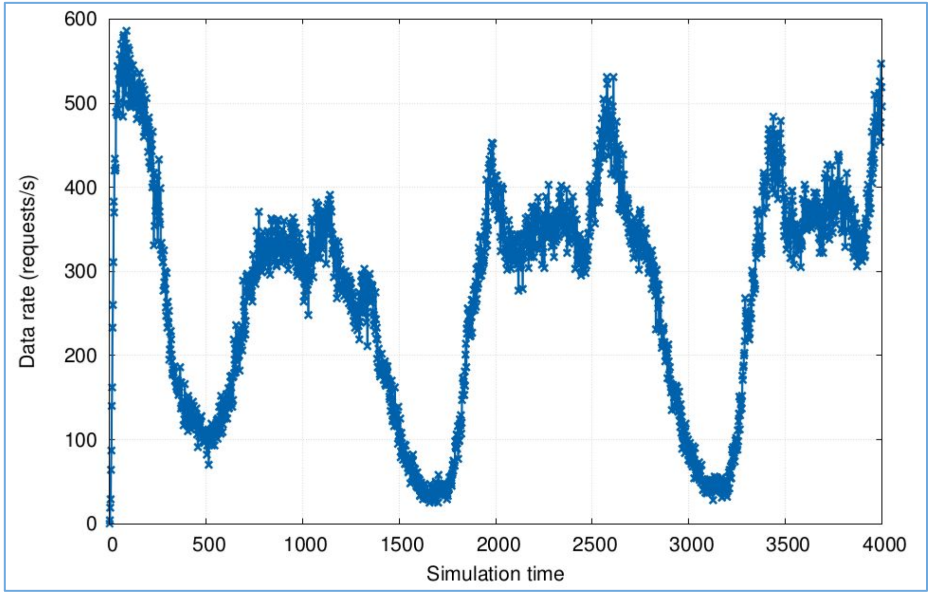
Evaluation

Container Deployment Simulator

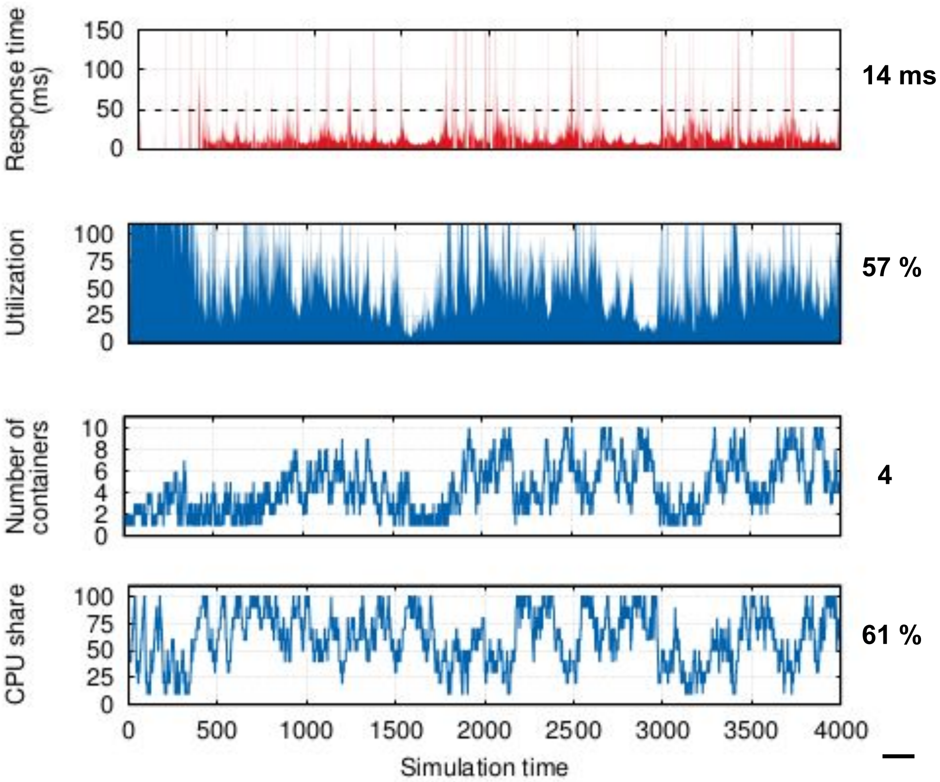


Experimental setting

$$c(s, a, s') = w_{\mathrm{rcf}} \cdot c_{\mathrm{rcf}}$$
$$+ \boxed{w_{\mathrm{perf}}} \cdot c_{\mathrm{perf}}$$
$$+ w_{\mathrm{res}} \cdot c_{\mathrm{res}}$$

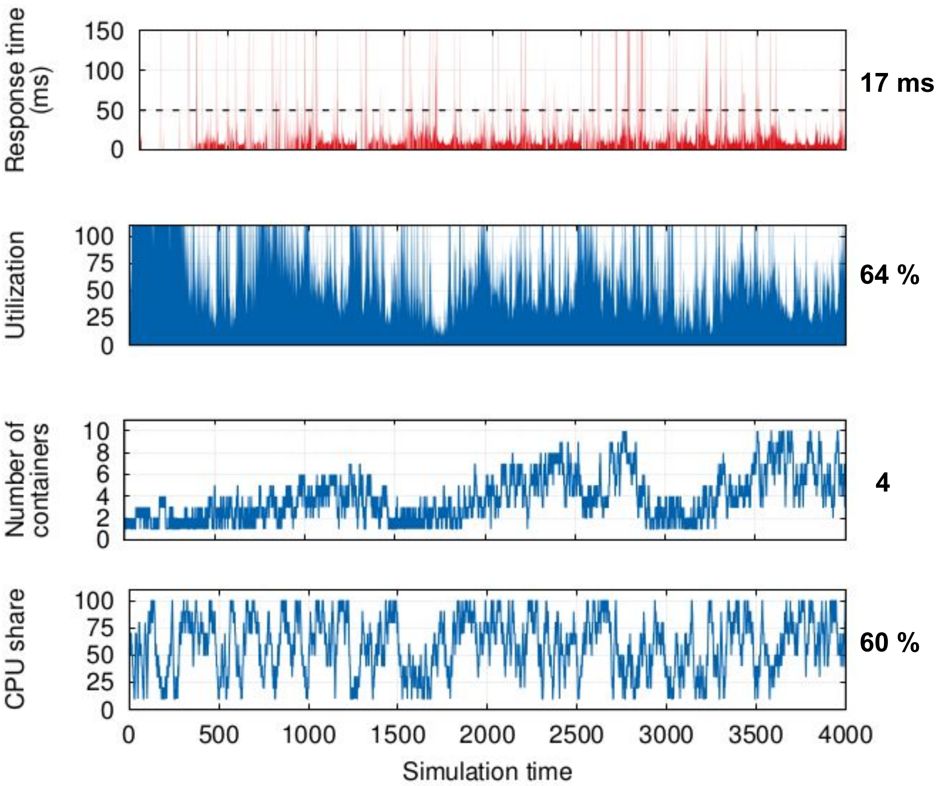


Q-learning
(5 actions)

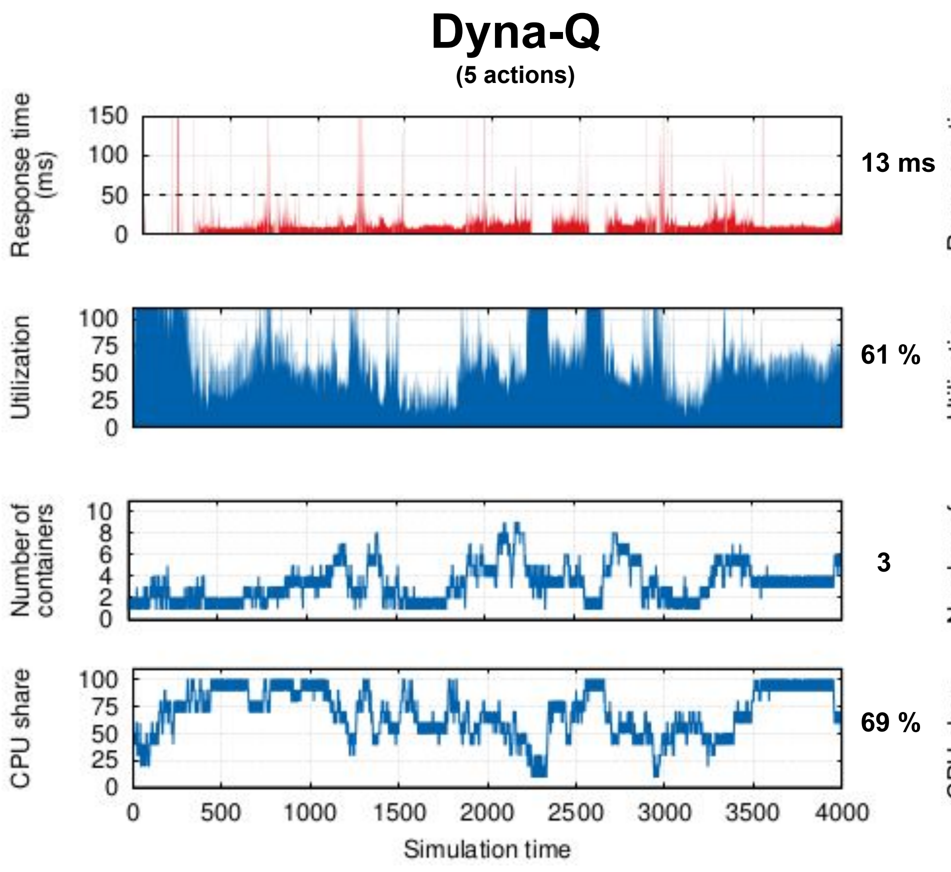


n. SLA violations: 18.52%

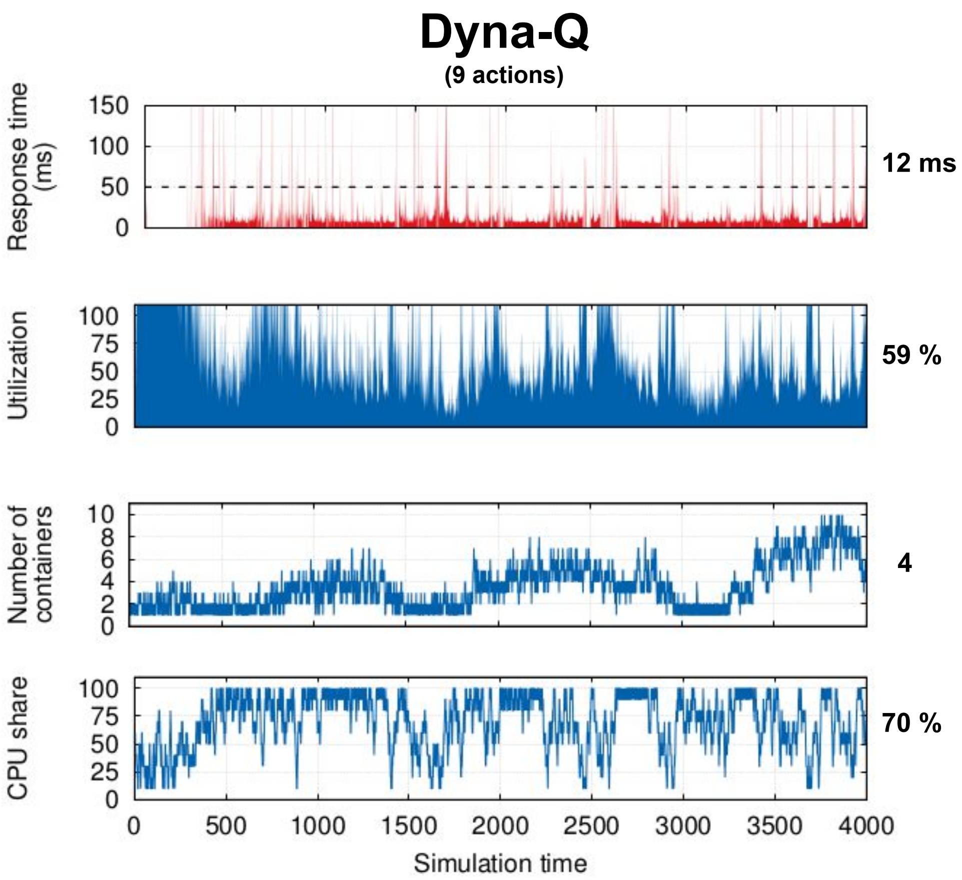
Q-learning
(9 actions)



n. SLA violations: 28.12%



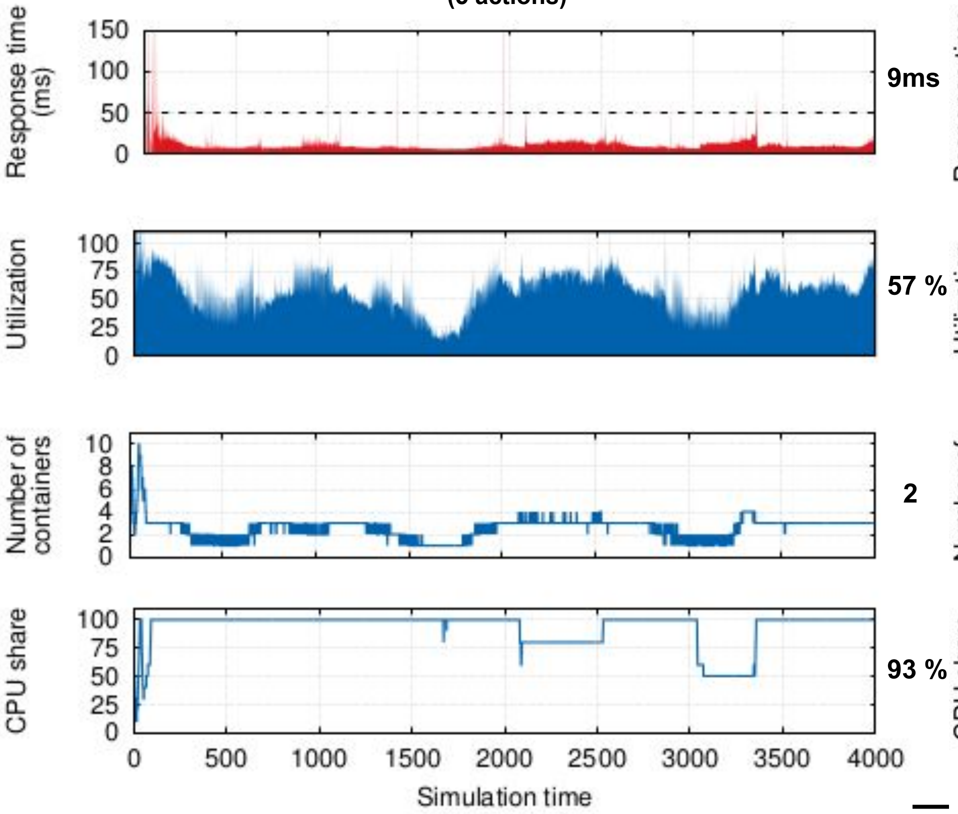
n. SLA violations: 17.92%



n. SLA violations: 21.54%

Model-based

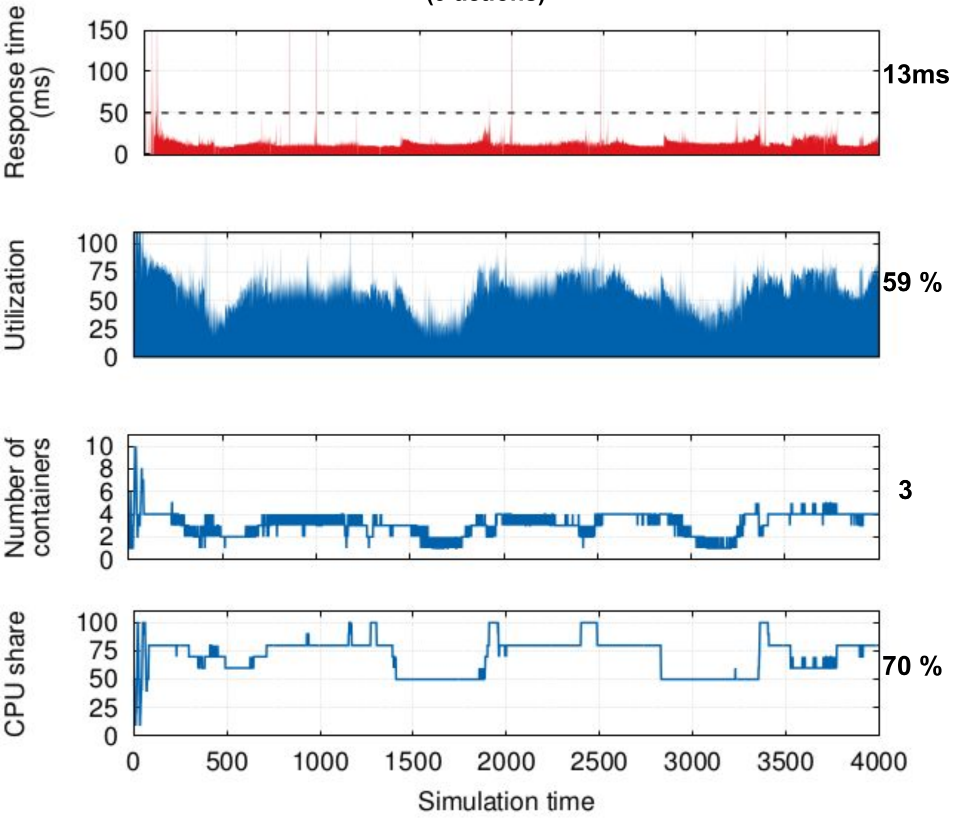
(5 actions)



n. SLA violations: 1.15%

Model-based

(9 actions)



n. SLA violations: 1.60%

Conclusion

- We designed RL-based solutions for the autonomic elasticity of containers
- We evaluated different techniques and system models
- We showed the benefits of model-based approaches
  - they reduce n. of SLA violations
  - they more quickly learn adaptation policies

Ongoing Work:

- To prototype the proposed solution (e.g., using Docker Swarm)
- To consider resource heterogeneity and their geo-distribution

Thank you!

Fabiana Rossi
f.rossi@ing.uniroma2.it

Simulation Configuration

- Application modeled as a M/D/*n* queue
  - *n*: number of containers
  - application rate $\mu = 200 \cdot c$ requests/s, where $c \in (0, 1]$ is the CPU share assigned to application;
- Target response time: 50 ms
- Max number of containers per-application: 10
- Discretization factors:
  - Utilization: 10%
  - CPU share: 10%
- Weights: w_rcf = 0.001, w_res = 0.019, w_perf = 0.98

Internet of Things

Empowering IoT Edge Computing

To fully attain the potential of edge computing for IoT, we need to address four concerns:

- hardware abstraction;
- programmability;
- interoperability;
- elasticity.

Empowering IoT Edge Computing

To fully attain the potential of edge computing for IoT, we need to address four concerns:

- hardware abstraction;
- programmability;
- interoperability;
- elasticity.

→ **Software containers**

Empowering IoT Edge Computing

To fully attain the potential of edge computing for IoT, we need to address four concerns:

- hardware abstraction;
- programmability;
- interoperability;
- elasticity.

Software containers

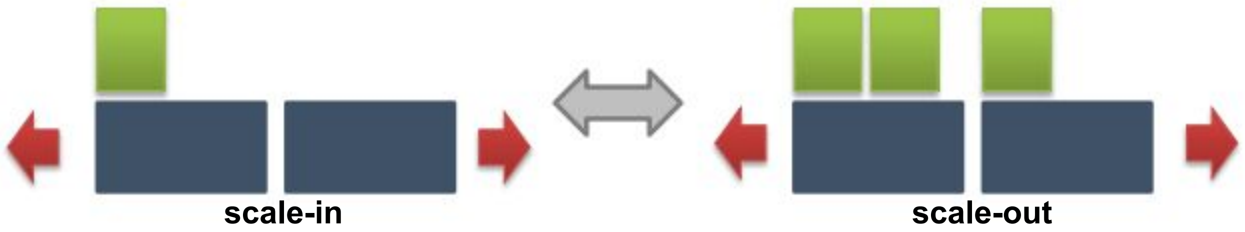
Elasticity

The possibility of cloud computing to provide resources on demand has encouraged the development of elastic applications.

Elasticity

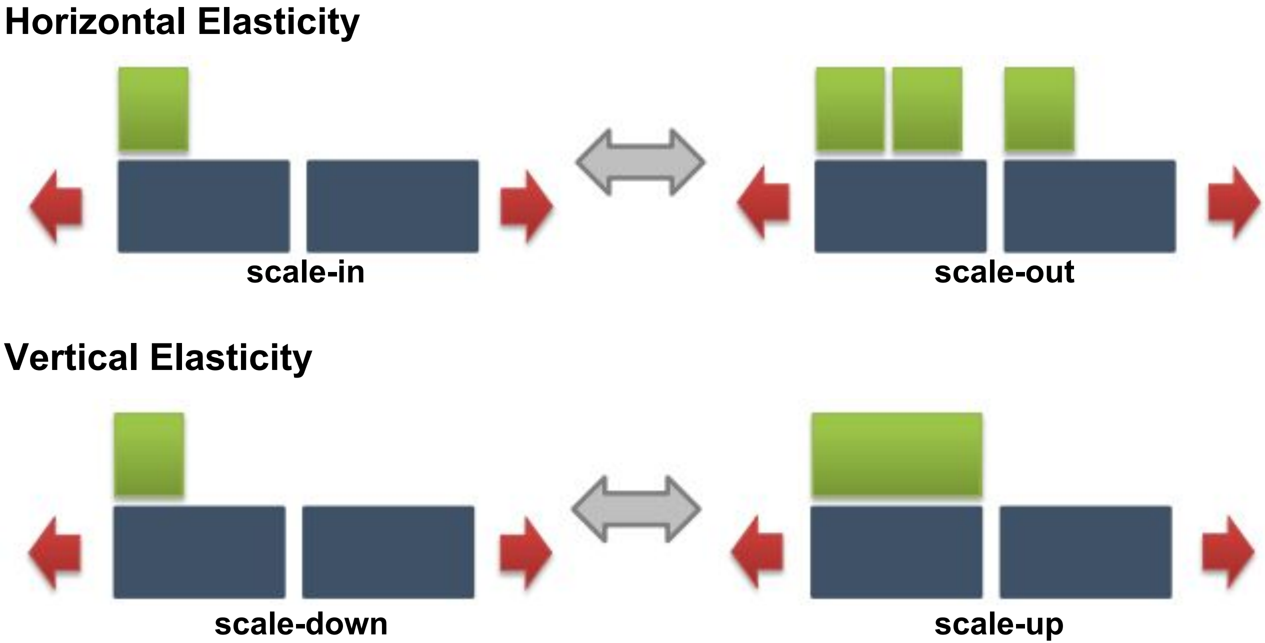
The possibility of cloud computing to provide resources on demand has encouraged the development of elastic applications.

Horizontal Elasticity



Elasticity

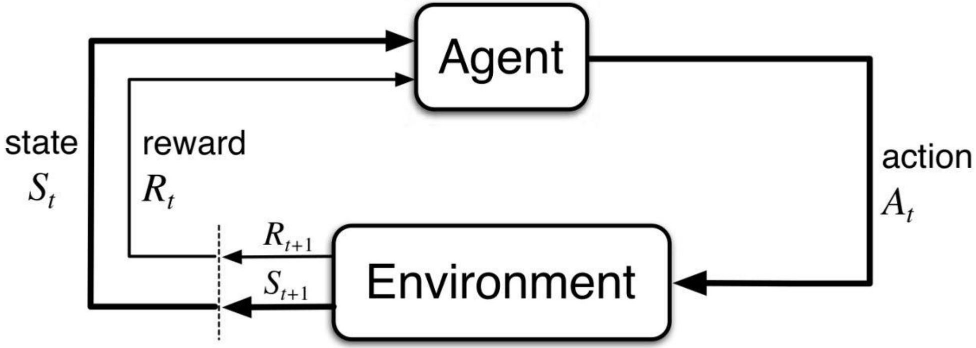
The possibility of cloud computing to provide resources on demand has encouraged the development of elastic applications.



Reinforcement Learning

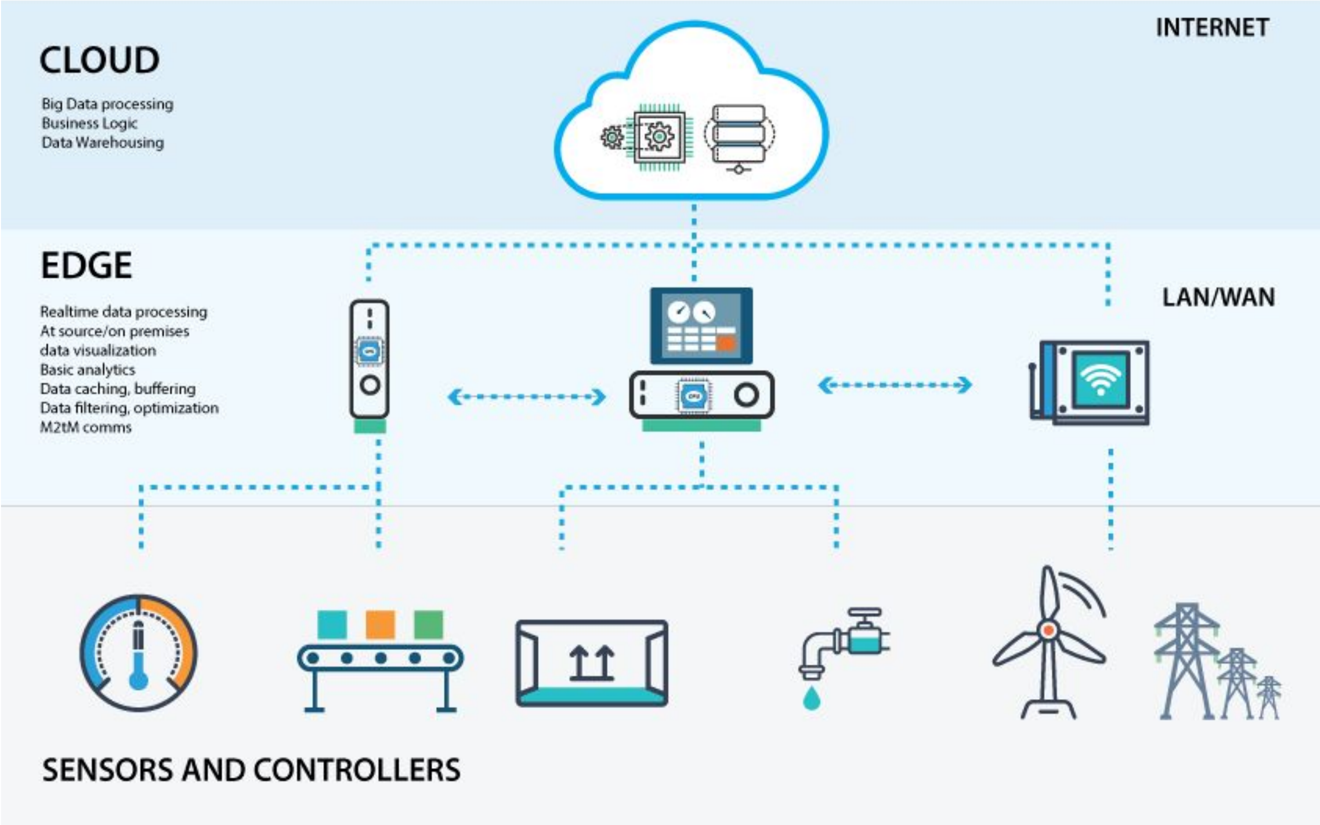
RL is a machine learning technique, where:

- the agent learns how to map situations to actions through the experience.

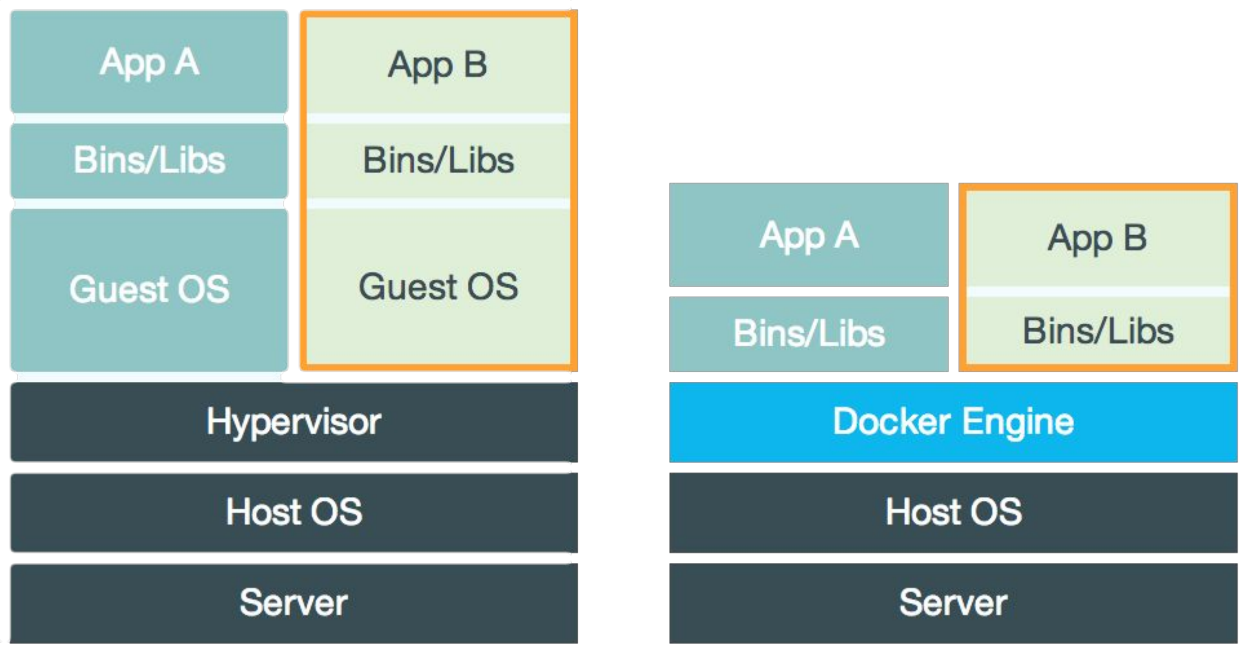


Sutton, R. S., and Barto, A. G. Reinforcement learning: An introduction. MIT Press, Cambridge, 1998.

Three-tier IoT Edge Computing Infrastructure



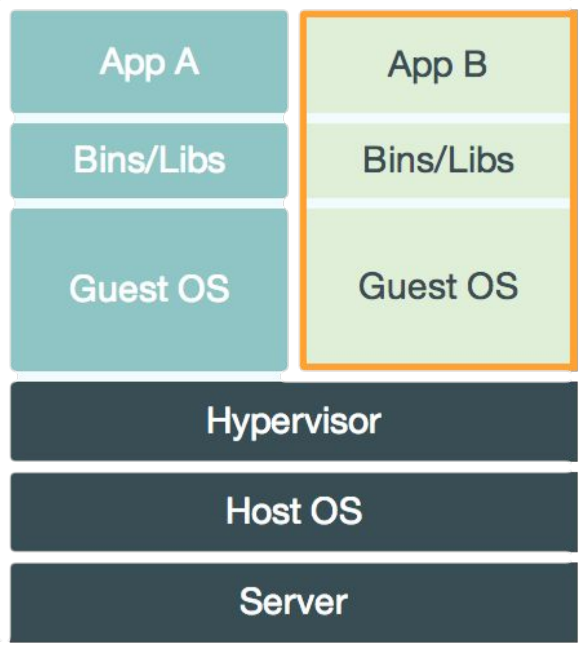
Virtualization techniques comparison



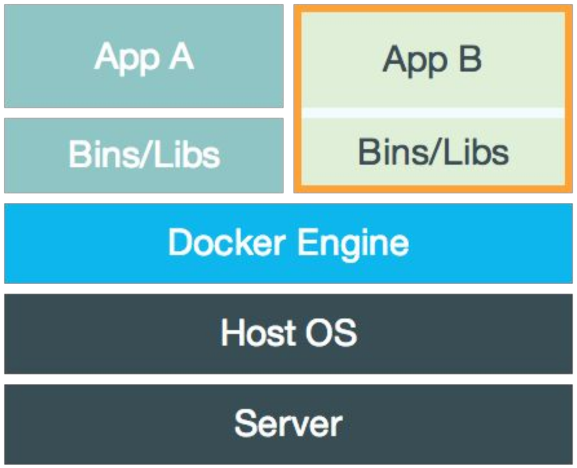
Virtual machines

Software containers

Virtualization techniques comparison



Virtual machines



Software containers

Known and Unknown Cost

To estimate *c(s, a, s')*, it is divided into two terms: **known** and **unknown cost**.

$$c(s, a, s') = \underbrace{\boxed{w_{\text{rcf}} \cdot c_{\text{rcf}} + w_{\text{res}} \cdot c_{\text{res}}}}_{\text{known cost}} + \underbrace{\boxed{w_{\text{perf}} \cdot c_{\text{perf}}}}_{\text{unknown cost}}$$

To **update** the **unknown cost estimate**:

$$\hat{c}_u(s') \leftarrow (1 - \alpha)\hat{c}_u(s') + \alpha c_u$$

Estimated Transition Probability

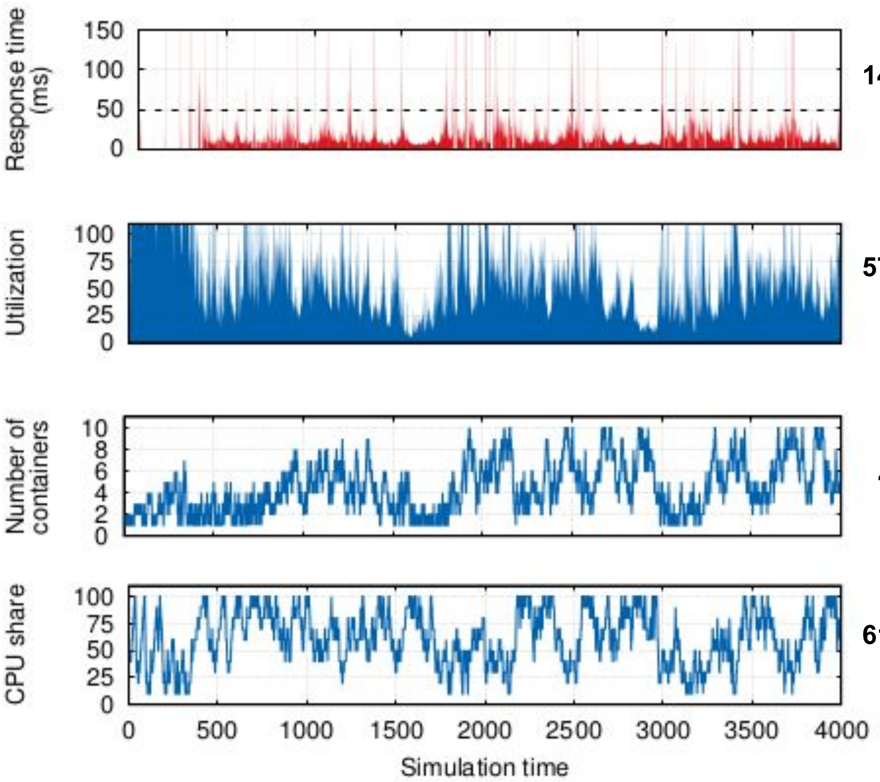
The **transition probability** is defined as

$$
\begin{aligned}
p(s'|s,a) &= P[s_{i+1} = (k', u', c')|s_i = (k, u, c), a_i = a] = \\
&= \begin{cases} P[u_{i+1} = u'|u_i = u] & k' = k + a_1 \wedge c' = c + a_2 \\ 0 & \text{otherwise} \end{cases}
\end{aligned}
$$

Given $u = j\bar{u}$ and $u' = j'\bar{u}$ at time instant *i*, the **estimated transition probability** is

$$
\widehat{P_{j,j'}} = \frac{n_{i,jj'}}{\sum_{l=0}^{L} n_{i,jl}}
$$

Q-learning
(5 actions)



14 ms

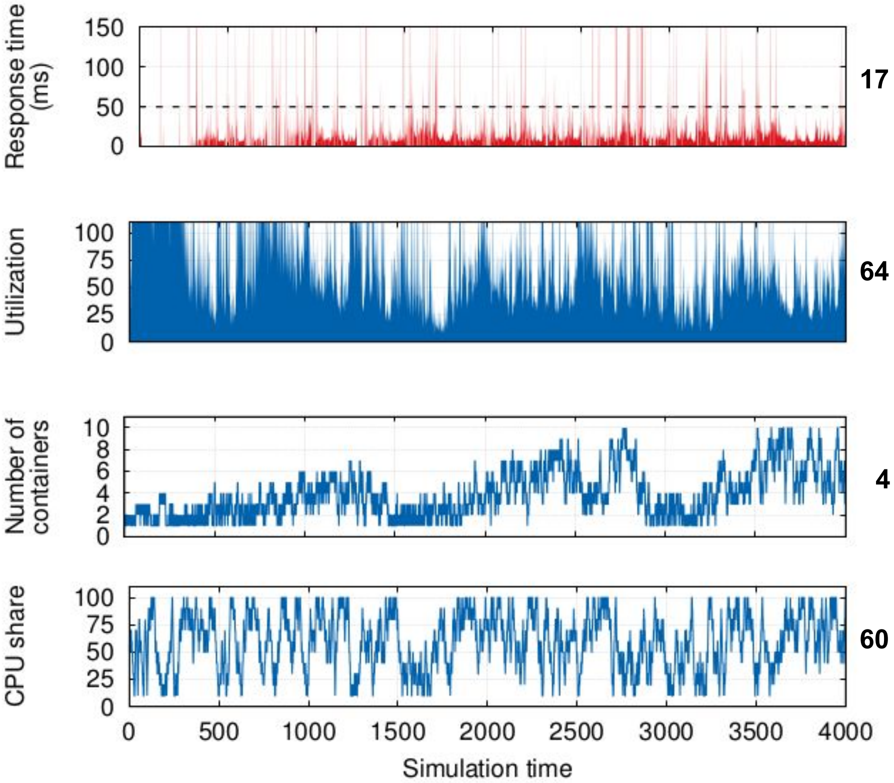
57 %

4

61 %

n. reconfigurations: 76.56%

Q-learning
(9 actions)



17 ms

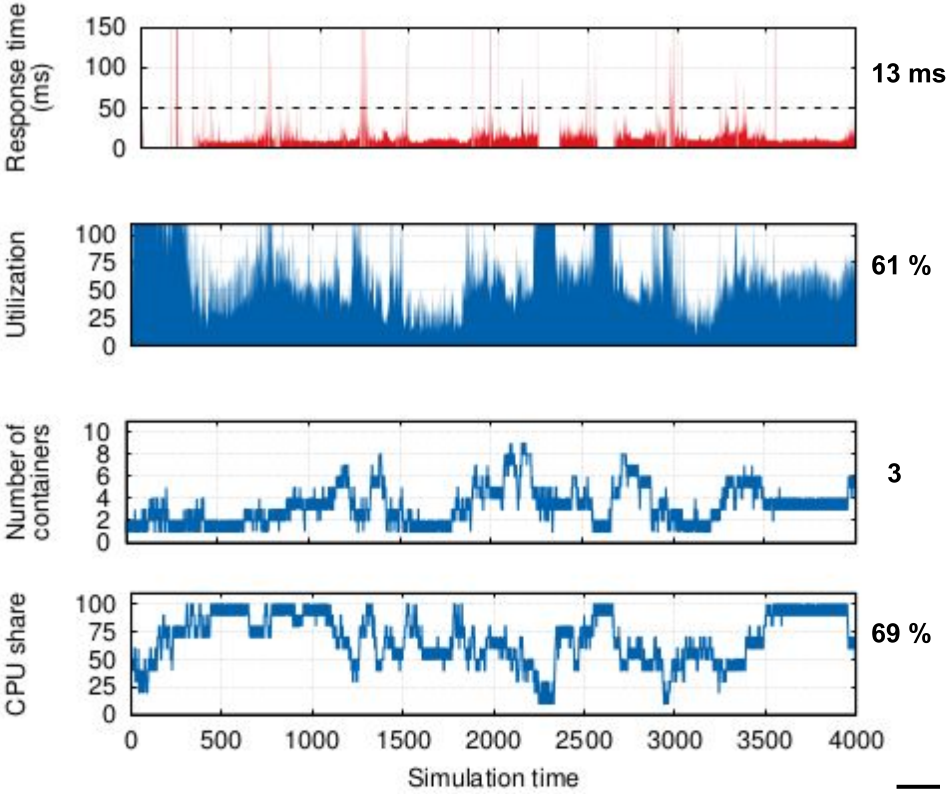
64 %

4

60 %

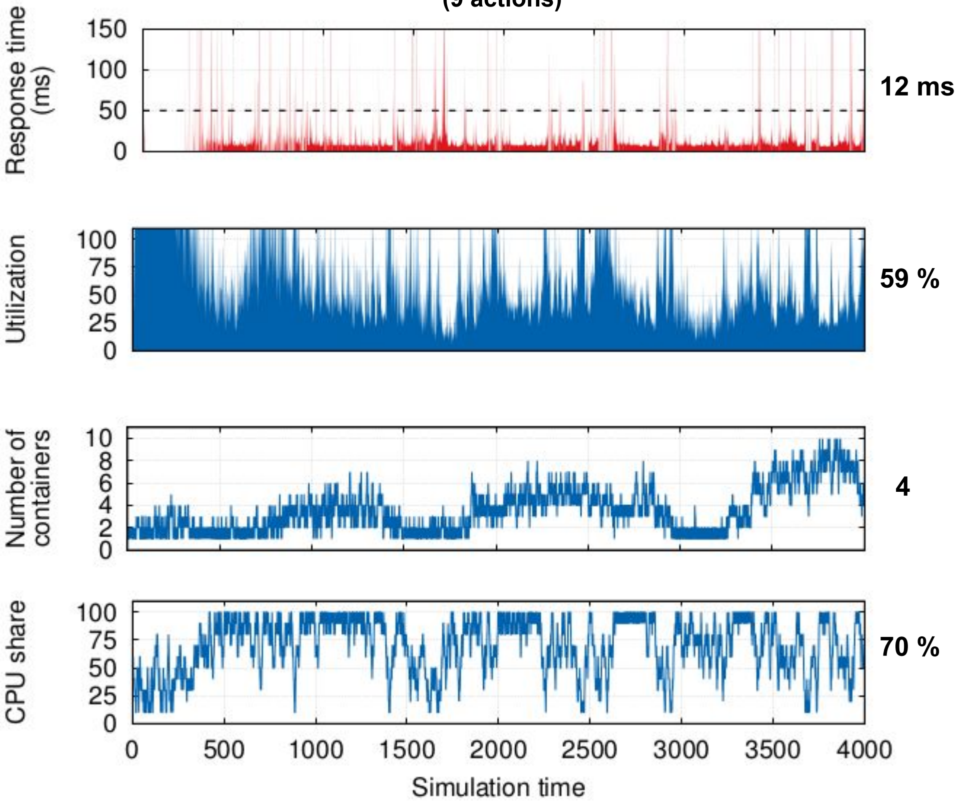
n. reconfigurations: 89.75%

Dyna-Q
(5 actions)



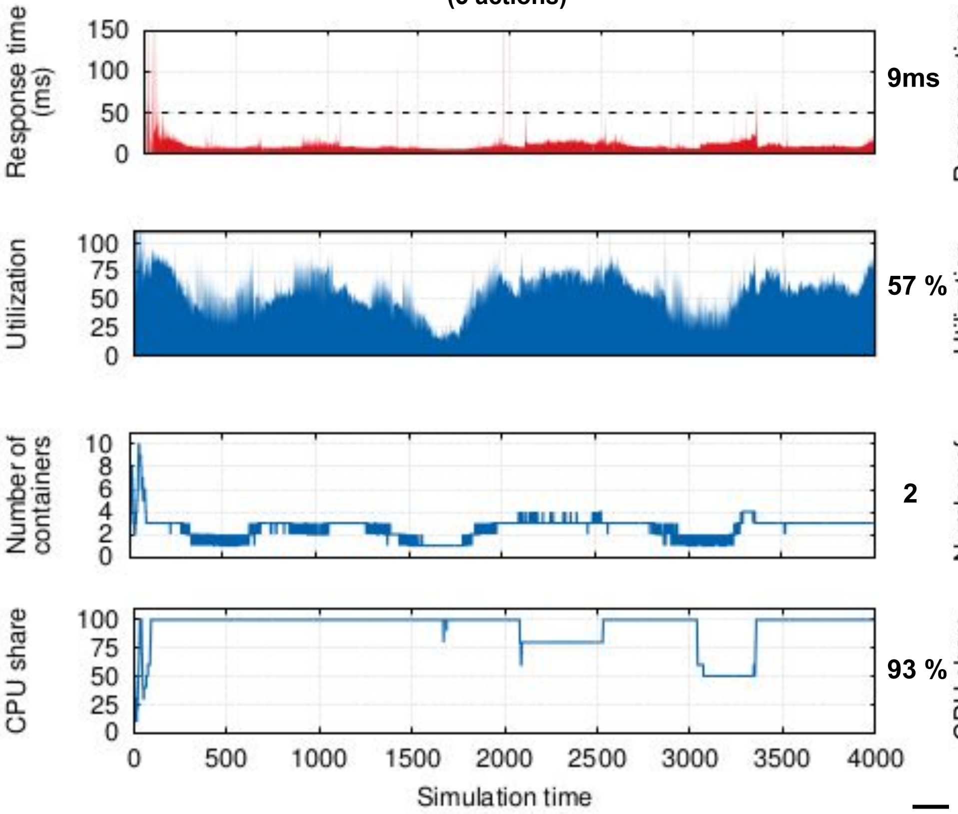
n. reconfigurations: 87.60%

Dyna-Q
(9 actions)



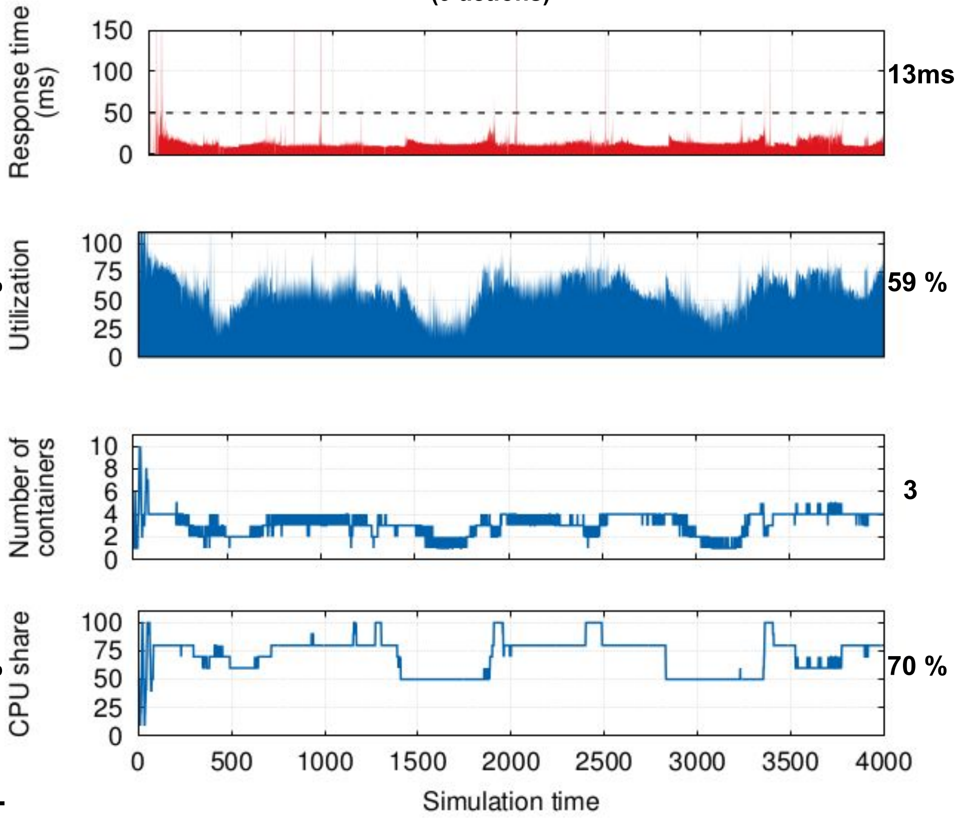
n. reconfigurations: 92.75%

Model-based
(5 actions)



n. reconfigurations: 34.09%

Model-based
(9 actions)



n. reconfigurations: 46.91%