Reinforcement Learning for Run-Time Performance Management

Gabriele Russo Russo
University of Rome Tor Vergata, Italy

March–April 2024

Agenda

- Run-time performance management
  - Why?
  - Examples
  - Key challenges
- Introduction to Reinforcement Learning
  - Markov Decision Processes
  - Tabular RL
  - Deep RL
  - Policy methods

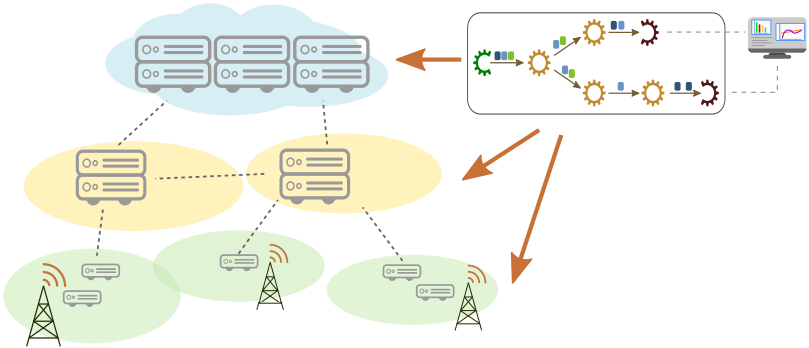
Scope

Run-time Performance Management

Performance of …???

Scenario

Distributed systems and applications in heterogeneous computing environments with Quality-of-Service requirements



e.g., microservice-based apps, data processing pipelines

Scenario (2)

Distributed systems and applications in heterogeneous computing environments with Quality-of-Service requirements

- This is the scenario I will mostly refer to
- But the techniques we discuss are not specific to this class of systems!

Performance Management

- Almost every computing/network system is expected to guarantee a desired level of performance
  - a desired Quality-of-Service level, in general
- Requirements impact system design, implementation, deployment, …
  - e.g., architectural choices oriented by performance requirements
  - e.g., capacity planning for system deployment
- A lot of work already done before the system is “on”
- More challenges to come over time!

Uncertainty

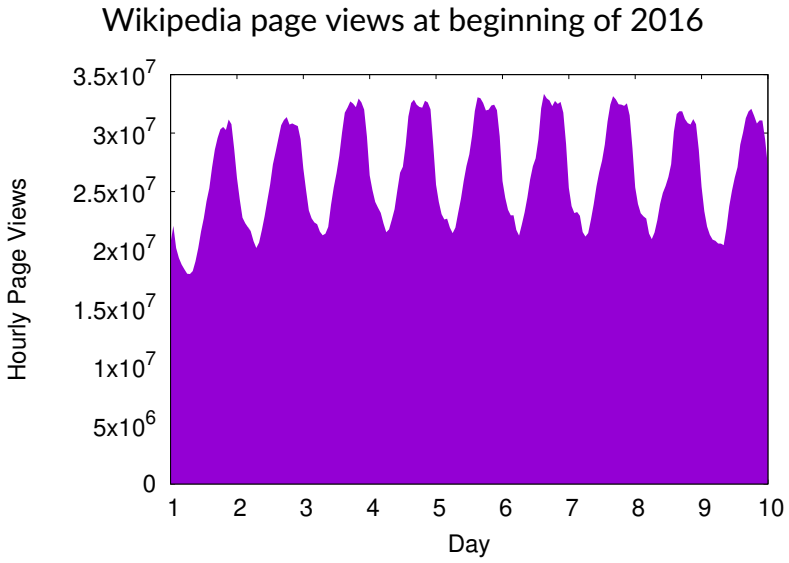
- We would like systems to consistently deliver the expected level of performance during their operation
- However, designing and developing systems to meet this goal is very difficult
- A fundamental challenge ahead: uncertainty
- (Partial or complete) lack of knowledge regarding elements of the system and the environment at design time
- Unpredictable situations (both internal to the system and external) which the system needs to deal with at run-time

Uncertainty in Modern Computing Environments

- Working conditions likely changing over time
  - Workloads
  - Network conditions (e.g., due to congestion)
  - Performance of (virtualized) multi-tenant computing resources
  - Security attacks
  - Variable monetary costs for on-demand resources
  - Intermittent energy supplies (e.g., solar power)
  - User mobility
- Black-box specifications of applications
- Heterogeneous computing and network resources
- Humans in the loop

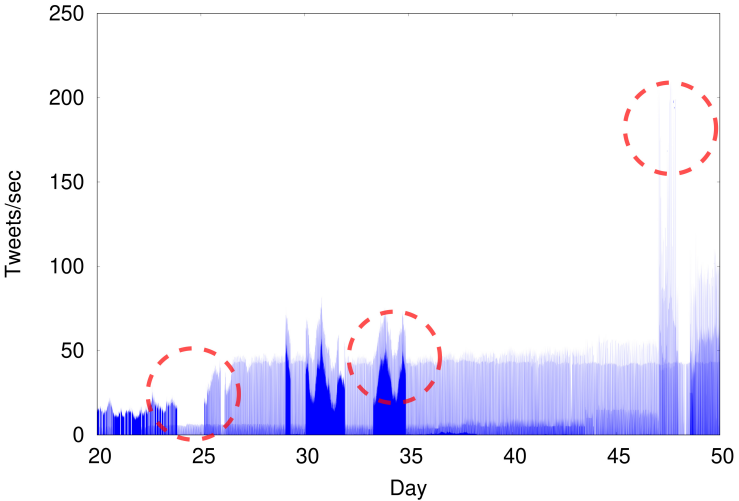
- How to develop automated performance management solutions to cope with uncertainty?

Example: Workload Variability



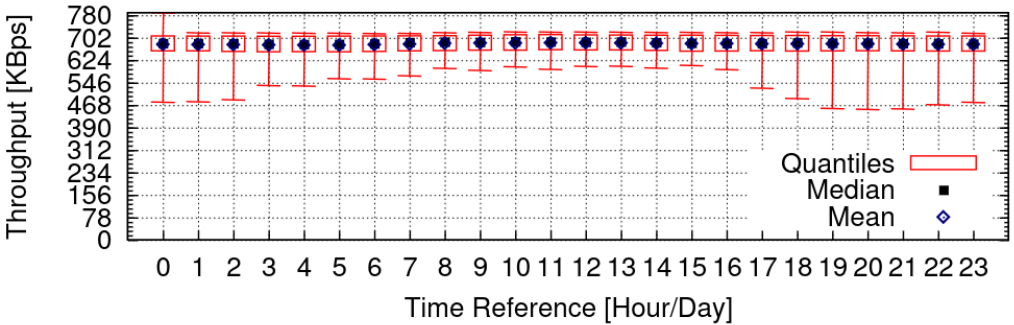
Example: Workload Variability (2)

Tweets about “COVID” at beginning of 2020



Example: Cloud Performance Variability

Throughput of Amazon S3 (measurements from 2009)



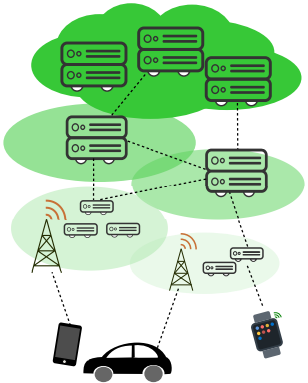
Iosup et al., "On the performance variability of production cloud services" (2011):
`https://ieeexplore.ieee.org/abstract/document/8102951`

Run-Time Adaptation

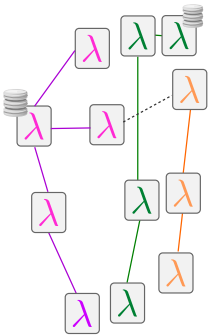
- How can computing systems deal with changes at run-time?
- Need to adapt to changing working conditions
  - Migrating components
  - Provisioning more/less resources
  - …

Self-Adaptation

- Adaptation cannot be driven and controlled by humans alone…
- Scale and complexity of modern applications and infrastructures
- Applications must be able to self-adapt!

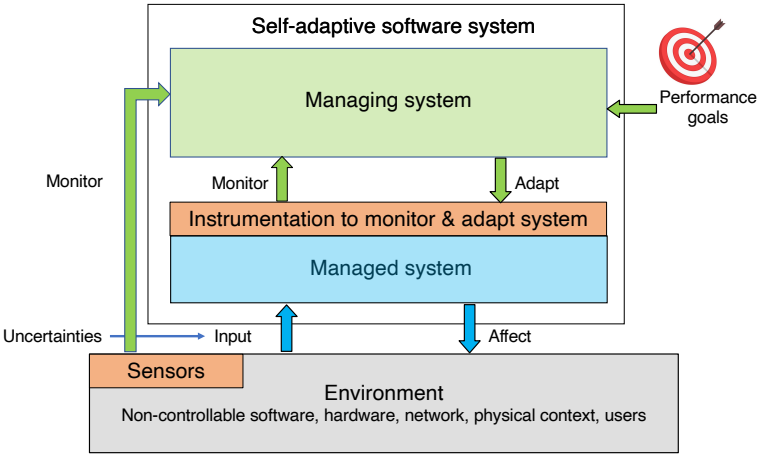


Computing Infrastructure



Applications

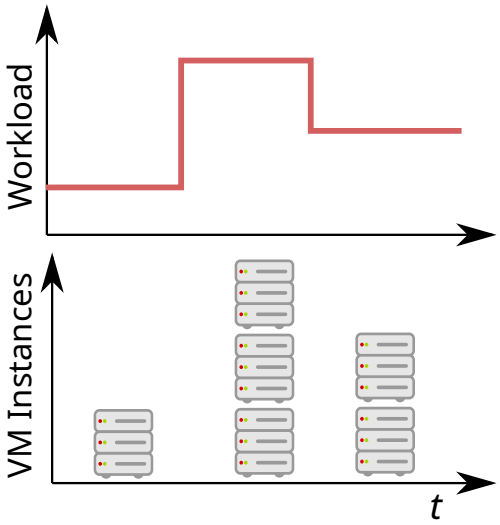
Self-Adaptive System



Reference book: “An Introduction to Self-Adaptive Systems” [Weyns 2020]

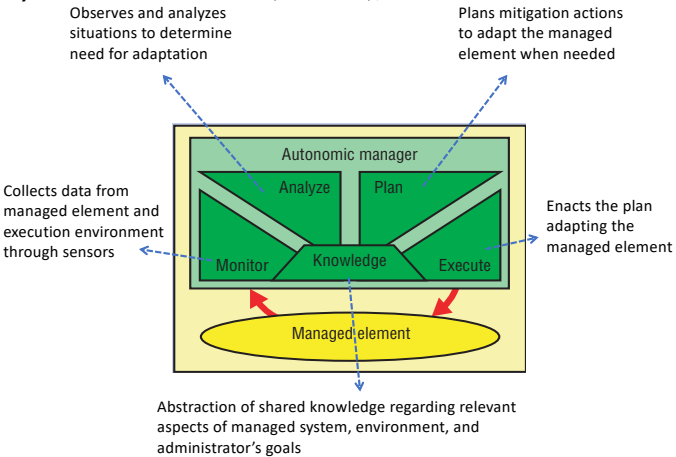
Example: VM Auto-scaling

- Consider a simple web application
- We can acquire virtual machines (VMs) from a cloud provider to deploy the application server(s)
- We can elastically acquire/release VMs at any time based on current demand
- e.g., “whenever the average CPU utilization of the VMs exceeds 70%, create a new VM”



Reference Architecture for Self-Adaptive Systems

- Monitor-Analyze-Plan-Execute (MAPE), with shared Knowledge (MAPE-K)



Seminal paper by [Kephart and Chess 2003]

MAPE: Monitor & Analyze Phases

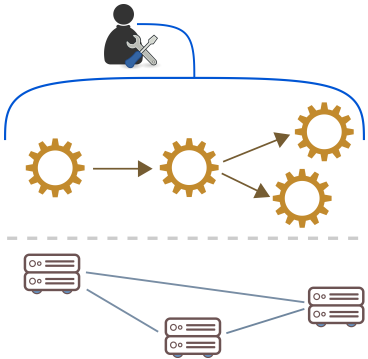
- Monitor: main design options
  - *When*: continuously, on demand
  - *What*: resources, workload, application performance, …
  - *How*: architecture (centralized vs. decentralized), methodology (active vs. passive)
  - *Where* to store monitored data and how (e.g., some pre-processing)
- Analyze: main design options
  - *When*: event- or time-triggered
  - *How*: reactive vs. proactive
    - Reactive: in reaction to events that have already occurred (e.g., scale-out to react to workload increase)
    - Proactive: based on prediction so to plan adaptation actions in advance (e.g., scale-out before workload increase effectively occurs)

MAPE: Plan Phase

- The most studied MAPE phase
- A variety of methodologies
  - Queueing theory
  - Optimization theory
  - Meta-heuristics
  - Control theory
  - Machine learning (including reinforcement learning)

Architecture of the Managing System

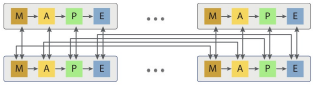
- How to design the control architecture for distributed applications?
- First idea: Centralized MAPE
  - All MAPE components in the same node
  - Global view of the system ✓
  - Lack of scalability and resiliency X



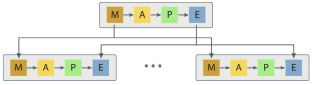
Architecture of the Managing System

- Alternative approach: decentralized MAPE
  - Several patterns to decentralize and distribute MAPE phases
  - e.g., master-worker, fully decentralized, and hierarchical [Weyns et al. 2013]
  - No clear winner, depending on system, environment and application features and requirements

Master-worker

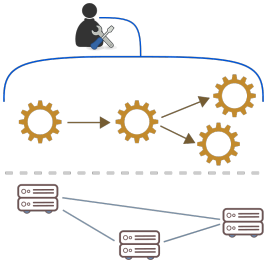


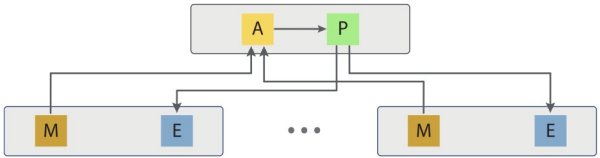
Fully decentralized (e.g., coordinated)



Hierarchical

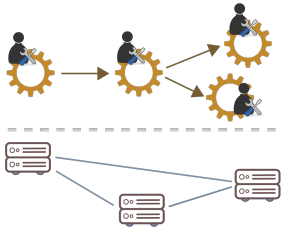
Decentralized MAPE: Master-Worker

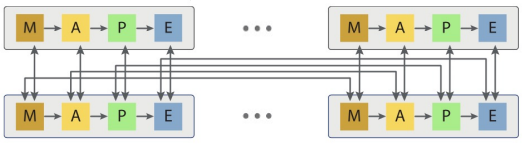




- Decentralize M and E on workers, keep A and P centralized on master
- Global view on master ✓
- Communication overhead, bottleneck risk and single point of failure on master X

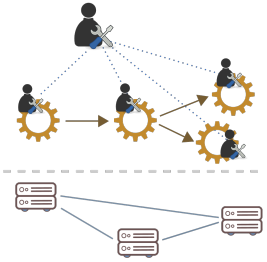
Decentralized MAPE: Fully Decentralized

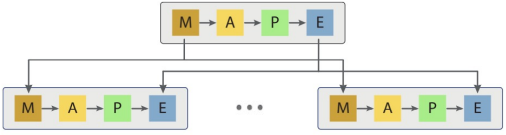




- Multiple control loops, each one in charge of some part of the controlled system, possibly coordinated through interaction (or no interaction at all)
- Cloud/edge applications: distinct control loop per application component
- Improved scalability ✓
- More difficult to take joint adaptation decisions X

Decentralized MAPE: Hierarchical





- Multiple MAPE loops, which can operate at different time scales and with separation of concerns
- Top-level MAPE can achieve global goals ✓
- Non-trivial to identify multiple levels of control X
- e.g., for Cloud/Edge applications:
  - Top-level MAPE: entire application
  - Bottom-level MAPE: application component

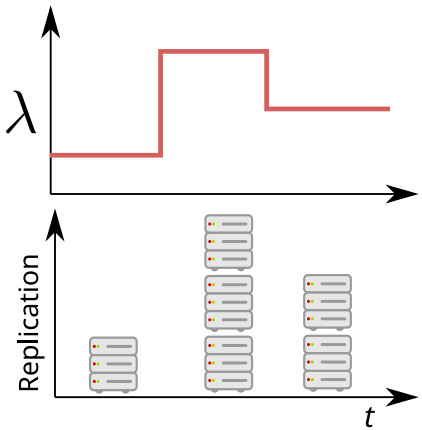
Introduction to Reinforcement Learning

Planning Adaptation Actions

- We focus on the Analyze and Plan phases of the MAPE loop
- How to identify and plan the most suitable adaptation actions?

Example: Auto-scaling

- Automated provisioning of VMs/containers/threads at run-time
- Conflicting performance and cost indices:
  - Performance (e.g., response time)
  - Monetary cost of allocated resources
  - Reconfiguration overhead
- Traditional solution: threshold-based policies



Threshold-based Policies

- Resource allocation varies according to a set of rules
- Rule = condition + action
- Conditions defined in terms of thresholds
- Model-free approach

if $x_1 > H_1$ and/or $x_2 > H_2$ and/or ... for D_H seconds
    scale-out(N)
if $x_1 < L_1$ and/or $x_2 < L_2$ and/or ... for D_L seconds
    scale-in(N)

Threshold-based Policies (2)

- Conditions may involve one or more metrics
  - e.g., CPU utilization, used memory
- For each metric, multiple conditions (hence, thresholds) can be defined
- Usually upper (or, high) and lower (or, low) thresholds are used
  - e.g., for CPU utilization, upper threshold 75% and lower 20%
- Conditions may also require thresholds to be exceeded for a certain amount of time before triggering an action

Example of Rules

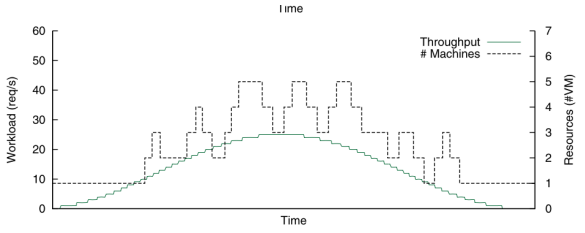
- If CPU utilization > 70% for at least 1 minute, add one VM
- If CPU utilization < 30% for at least 5 minutes, terminate one VM
- If avg. response time > 100ms for at least 30s, increase CPU frequency by 10%

Issues with Thresholds

- Threshold-based policies are easy to implement and execute
- But defining suitable rules is not trivial!
- Which metrics? System vs application-oriented
  - System metrics may work across different apps
  - Application metrics directly mapped onto QoS requirements
- Which thresholds? Tuning required!

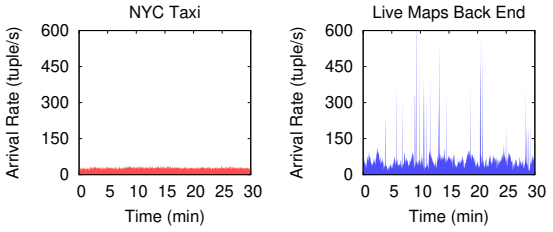
Issues with Thresholds: Oscillation

- If thresholds are too close, frequent oscillations may occur
- Oscillations negatively impact performance
  - System oscillates between under- and over-provisioning
  - Scaling may introduce additional overhead too
- Possible workaround: cooldown (or, calm) period
  - Auto-scaler inhibited for a short period after every scaling action



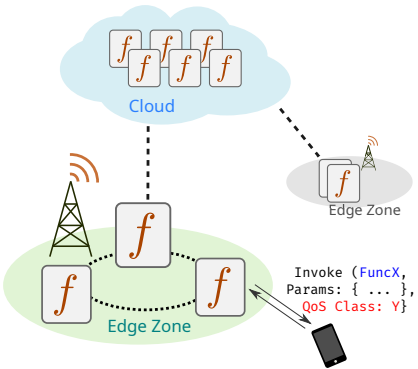
Issues with Thresholds: Bursts

- Scaling conditions are usually evaluated over short-medium time periods (e.g., seconds, minutes)
- Not all workloads can be characterized looking at average metrics over such time windows
- These workloads lead to similar average utilization over 1-minute windows, but they are profoundly different due to bursts [Russo Russo et al. 2021]



Example: Edge-Cloud Computation Offloading

- Serverless functions invoked on Edge nodes
- Nodes can offload execution on neighboring nodes or to the Cloud
- Must guarantee maximum response time for functions
- Different resource cost for execution
- Traditional approach: greedy heuristics

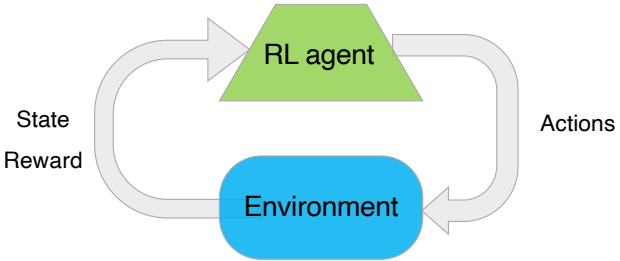


Reinforcement Learning

- Supervised learning
- Unsupervised learning
- **Reinforcement learning**
  - Goal-directed learning
  - Learning from interaction with an environment, rather than from examples

- Learning from interaction is probably the first idea to occur when thinking to the nature of learning

Reinforcement Learning: Overview



- Sequential decision-making
- Agent interacts with an environment
  - Agent perceives the state of the environment
  - Agent performs actions
- Feedback in the form of reward (and new states)
- Goal: maximizing cumulated reward over the long run

Reinforcement Learning: Key Ideas

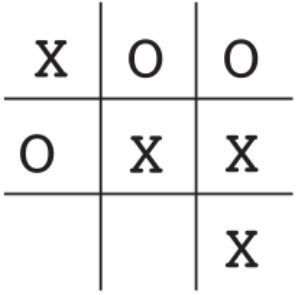
- The learner is not told which actions to take (or which actions are the best)
- The agent must discover which actions yield the most reward by trying them (trial-and-error)
- Often, actions may affect not only the immediate reward but also the next situation and, through that, all subsequent rewards (delayed rewards)
- These two characteristics are the most important distinguishing features of RL

A Bit of History

- Modern RL emerged around 1980's as two research trends intertwined
  - Learning from trial and error (originated in the psychology of animal learning)
  - Optimal control of stochastic systems (usually through dynamic programming)
- Consistent advancements through 80's and 90's
- New wave of popularity (and many new applications) with deep RL, after 2018

Example: Tic-Tac-Toe

- State: representation of the board (3x3 matrix)
- Actions: available cells to mark
- Reward: 1 for a winning move, 0 otherwise



Example: AlphaZero by DeepMind

- Software able to play Go, Chess and Shogi
  - Board games with huge number of legal positions (i.e., state space)
  - Number of legal board positions in Go approximately $2 \times 10^{1}70$, far greater than the number of atoms in the observable universe
- Trained via self-play and advanced deep RL techniques
- Superhuman level of play with 24-hour training
- First presented in 2017; in 2019 MuZero, generalization to play Atari games and other board games without prior rule knowledge

Paper: `https://arxiv.org/abs/1712.01815`

Example: AlphaDev by DeepMind

- Announced in 2023[1]
- RL used to develop new C++ sorting algorithm, now accepted in the standard library
- 70% faster on short sequences (2-3 items), 1.7% faster on long sequences
- State: instructions generated so far and state of the CPU
- Actions: assembly instructions to add
- Reward: based on sorting correctness and efficiency

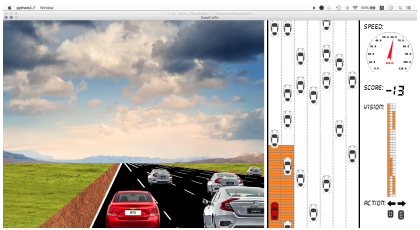
[1] https://www.deepmind.com/blog/alphadev-discovers-faster-sorting-algorithms

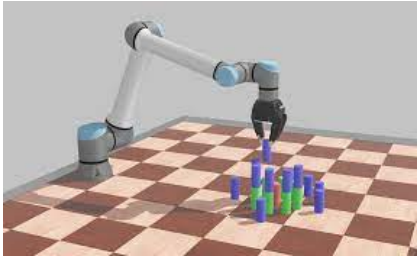
Example: deep RL agent playing Breakout

https://www.youtube.com/watch?v=TmPfTpjtdgg

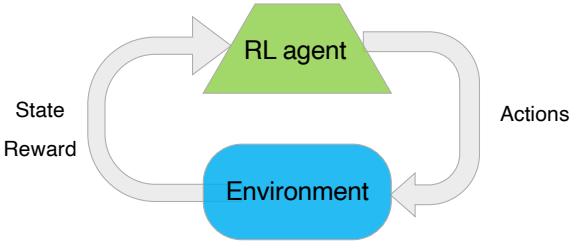
Other Examples

- Autonomous vehicles
- Robot control
- Trading
- Autonomous network and computer systems
- Videogames
- …





Reinforcement Learning



- Agent, environment, actions, state, rewards, …
- Modeling depends on the specific task
  - e.g., autonomous car uses different state information compared to chess player
- The problem tackled by RL is formally described as a Markov Decision Process (MDP)
- A mathematical framework to model sequential decision making, in situations where outcomes are partly random

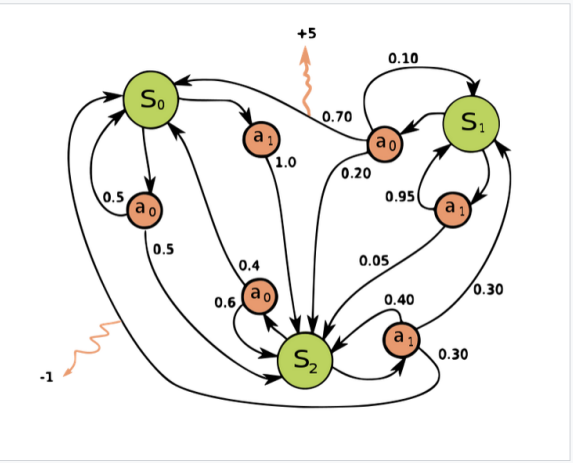
Markov Decision Process (MDP)

- Extension of discrete-time Markov chains
  - "A stochastic model describing a sequence of possible events, occurring at discrete time steps, in which the probability of each event depends only on the state attained in the previous event."
- At each time step t, the process is in some state s_t
- The agent chooses an action a_t among those available in state s_t
  - e.g., robot observes current position and decides direction to move; some directions might be blocked by obstacles

Markov Decision Process (2)

- Following a_t, the process moves to (random) state s_{t+1}
  - e.g., autonomous drone chooses an action to reduce altitude; actual outcome may depend on (unpredictable) wind speed
- Agent receives a reward r_t (or, equivalently, pays a cost)
  - e.g., robot may get a reward for reaching its final destination
  - e.g., chess player rewarded at the end of a match

Example



Example of a simple MDP with three states (green circles) and two actions (orange circles), with two rewards (orange arrows).

Markov Decision Process (3)

What defines an MDP?

- $\mathcal{S}$: a (finite) set of states
- $\mathcal{A}$: a (finite) set of actions
- p: state transition probabilities

$$p(s'|s,a) = P[s_{t+1} = s'|s_t = s, a_t = a]$$

- r: reward function
  1. $r(s,a) = E[r_t|s_t = s, a_t = a]$
  2. $r(s,a,s') = E[r_t|s_t = s, a_t = a, s_{t+1} = s'] \longrightarrow r(s,a) = \sum_{s'} p(s'|s,a) r(s,a,s')$

Markov Property

"The future is independent of the past given the present"

Definition

A state s_t is Markov if and only if

$$P[s_{t+1}|s_1, \ldots, s_t] = P[s_{t+1}|s_t]$$

- The state captures all relevant information from the history
- i.e., the state is a sufficient statistic of the future

References I

- Kephart, J.O. and D.M. Chess (2003). "The Vision of Autonomic Computing". In: *IEEE Computer* 36.1, pp. 41–50. DOI: 10.1109/MC.2003.1160055.
- Russo Russo, G. et al. (2021). "MEAD: Model-Based Vertical Auto-Scaling for Data Stream Processing". In: *Proceedings of 21th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing, CCGRID '21, Virtual Event, May 10-13, 2021*, pp. 314–323. DOI: 10.1109/CCGrid51090.2021.00041.
- Weyns, D. (2020). *An Introduction to Self-adaptive Systems: A Contemporary Software Engineering Perspective*. Hoboken, NJ, USA: Wiley-IEEE Computer Society Press.
- Weyns, D. et al. (2013). "On Patterns for Decentralized Control in Self-Adaptive Systems". In: *Software Engineering for Self-Adaptive Systems II*. Vol. 7475. LNCS. Springer.