A Hybrid Reinforcement Learning Approach to Autonomic Resource Allocation

Gerald Tesauro, Nicholas K. Jong, Rajarshi Das and Mohamed N. Bennani

(Slides based on the above paper published at ICAC, pp. 65-73, 2006 IEEE International Conference on Autonomic Computing, 2006.

MAHMOUD AWAD

CS 895 - Autonomic Computing - Fall 2010
Dr. Daniel A. Menasce
George Mason University

Presentation Outline

- Introduction
  - Reinforcement Learning (Rl)
  - State-Action-Reward-State-Action (SARSA)
- Objective
- Hybrid Reinforcement Learning Approach
- Prototype Results
- Conclusions

Introduction

Reinforcement Learning (RL)

- Trial-and-Error learning method in dynamic non-deterministic environments where an agent senses the current state (s) of the environment and chooses an action (a) based on reward (r) (a scalar measure of value for performing action **a** in state **s**).
- The agent must ultimately try every state in the environment's state space.
- In essence, the agent learns management policies, which are mappings from environment states to actions.
- Ultimate objective of agent is to maximize the total reward in the long run (i.e., value). Reference (6) page 8 discusses **reward function versus value function**.

Introduction

Reinforcement Learning (RL)

- **Advantages:**

1. RL does not require an explicit model of either the computing system being managed or of the external process that generates workload or traffic.
2. Takes into account the dynamic phenomena of the environment in future calculations of reward and state.

- **Disadvantages:**

1. Poor scalability in large state spaces especially when it uses a lookup table to store a separate value for every possible state-action pair where the size of such a table increases exponentially with the number of state variables.
2. Poor performance if training is done online for two reasons:
    1. Initial policy may be inaccurate in case of lack of domain knowledge.
    2. Exploring difference actions is randomized and is considered costly in live systems.

Introduction

State-Action-Reward-State-Action (SARSA) RL algorithm

- Start with an environment state **s** and an initial "fixed" policy **π**.
- SARSA algorithm tries to learn the **value function**:
  - Q(s,a) (s: state, a: action) estimating the agent's long-range expected value starting in state **s**, taking initial action **a** and then using policy to choose subsequent actions

$$\Delta Q(s_t, a_t) = \alpha(t)\,[r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

(s_t, a_t):	Initial state and action at time t.
r_t:	immediate reward at time t.
(s_{t+1}, a_{t+1}) :	next state and next action at time (t + 1).
γ:	Constant – is a "discount parameter" between 0 and 1 expressing the present value of expected future reward.
$\alpha(t)$:	"learning rate" parameter, which decays to zero asymptotically to ensure convergence.

Introduction

Convergence proof of TD (Temporal Difference) learning

- Given enough training samples, RL can converge to the correct value function V^{π} associated with any fixed policy π, and that the new policy whose behavior greedily maximizes V^{π} is guaranteed to improve upon the original policy π .

OBJECTIVE

- Combine Reinforcement Learning (RL) with queuing models for a hybrid approach to server allocation decision making.
- RL trains offline on existing data sets to avoid poor performance in live online learning.
- To overcome the scalability issue with RL that uses lookup tables, SARSA is used instead to train the RL module. The nonlinear function approximator uses neural networks.
- Function approximators provide a mechanism for generalizing training experience across states, so that it is no longer necessary to visit every state in the state space.
- Prototype: data center dynamic server allocation among multiple web applications to maximize SLA payments.

OBJECTIVE

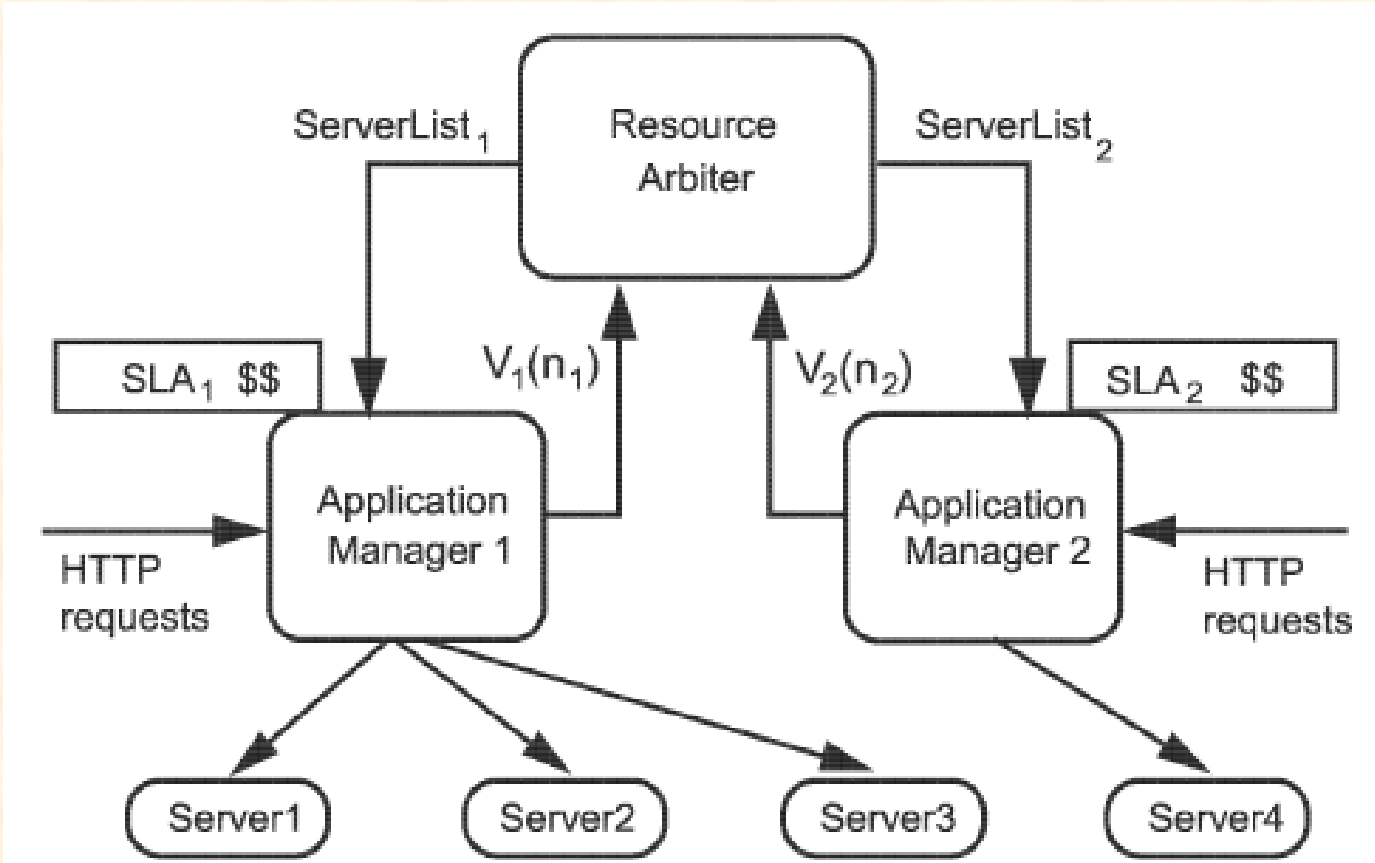

Fig. 1. Data center architecture.

Hybrid Reinforcement Learning Approach

- Two components:
  1. Nonlinear function approximator trained in batch mode on a recorded dataset.
  2. Initial queuing model policy.

Hybrid Reinforcement Learning Approach

Batch learning

- Input: recorded sequence of (T + 1) observations $\{ (S_t, a_t, r_t), 0 <= t <= T\}$ produced by an arbitrary management policy, where (s_t, a_t, r_t) are the observed state, action and immediate reward at time t.

Hybrid Reinforcement Learning Approach

- Batch learning: SARSA Implementation

Algorithm 1 Compute Q

1: Initialize Q to a random neural network
2: **repeat**
3: $SSE \leftarrow 0$ {sum squared error}
4: **for all** t such that $0 \leq t < T$ **do**
5: $target \leftarrow r_t + \gamma Q(s_{t+1}, a_{t+1})$
6: $error \leftarrow target - Q(s_t, a_t)$
7: $SSE \leftarrow SSE + error \cdot error$
8: Train $Q(s_t, a_t)$ towards $target$
9: **end for**
10: **until** CONVERGED(SSE)

- Train Q: uses the standard back-propagation algorithm, which adjusts each weight in the neural network in proportion to its error gradient.

Hybrid Reinforcement Learning Approach

• For the function Q(s,a) , the authors chose:
s = current mean arrival rate λ of page requests.
a = n : number of servers.
$Q(\lambda, n)$

• To guarantee an acceptable performance level of the initial policy, a heuristic was used that would make Q linearly dependent on demand per server (λ/n).

• In order to account for server switching or load rebalancing, a delay factors is incorporated into the state decision at time t by considering the allocation decision at t-1. i.e., input is $Q(\lambda_t, n_{t-1}, n_t)$

•SARSA discount parameter $\gamma = 0.5$

Hybrid Reinforcement Learning Approach

Initial Queuing Model Policies

- Needed to supply the RL algorithm with a reasonable initial policy.
- Two different types of queuing networks are used to model arrival and departure of requests:

1. Open network (infinite population).
2. Closed network (Finite population each alternating between the think state and the submitted state).

Hybrid Reinforcement Learning Approach

Initial Queuing Model Policies

1. Open network (infinite population):
   - If **overall demand is λ** and **number of servers is n**, then we model the application with n identical parallel open networks, each with one server and **demand level of λ/n.**
   - Uses the parallel M/M/1 queuing formulation to model the mean response time characteristics of an application.
   - Mean **response time** R with **service rate μ** and **arrival rate λ**

     $R=1/(\mu-\lambda)$ and $R_{t+1}=1/(\mu-(\lambda_t/n_{t+1}))$ ->

     $\mu=(1/R_t)+(\lambda_t/n_t)$

     μ is sensitive to variations to R_t because of garbage collection in Java, therefore a smoothing parameter is used (in the range of 0.1 – 0.5).

Hybrid Reinforcement Learning Approach

Initial Queuing Model Policies

2. Closed network (Finite population each alternating between the think state and the submitted state)
   - Same as open network except that overall demand is replaced by Number of customers M. Given n servers, each server will handle M/n customers (requests are assigned using round-robin method).
   - **Service rate μ:**

     $\mu = (1/R_t) + \lambda_t$

     μ is sensitive to variations to R_t because of garbage collection in Java, therefore a smoothing parameter is used (in the range of 0.1 – 0.5)

Prototype Results

Performance Without Switching Delays

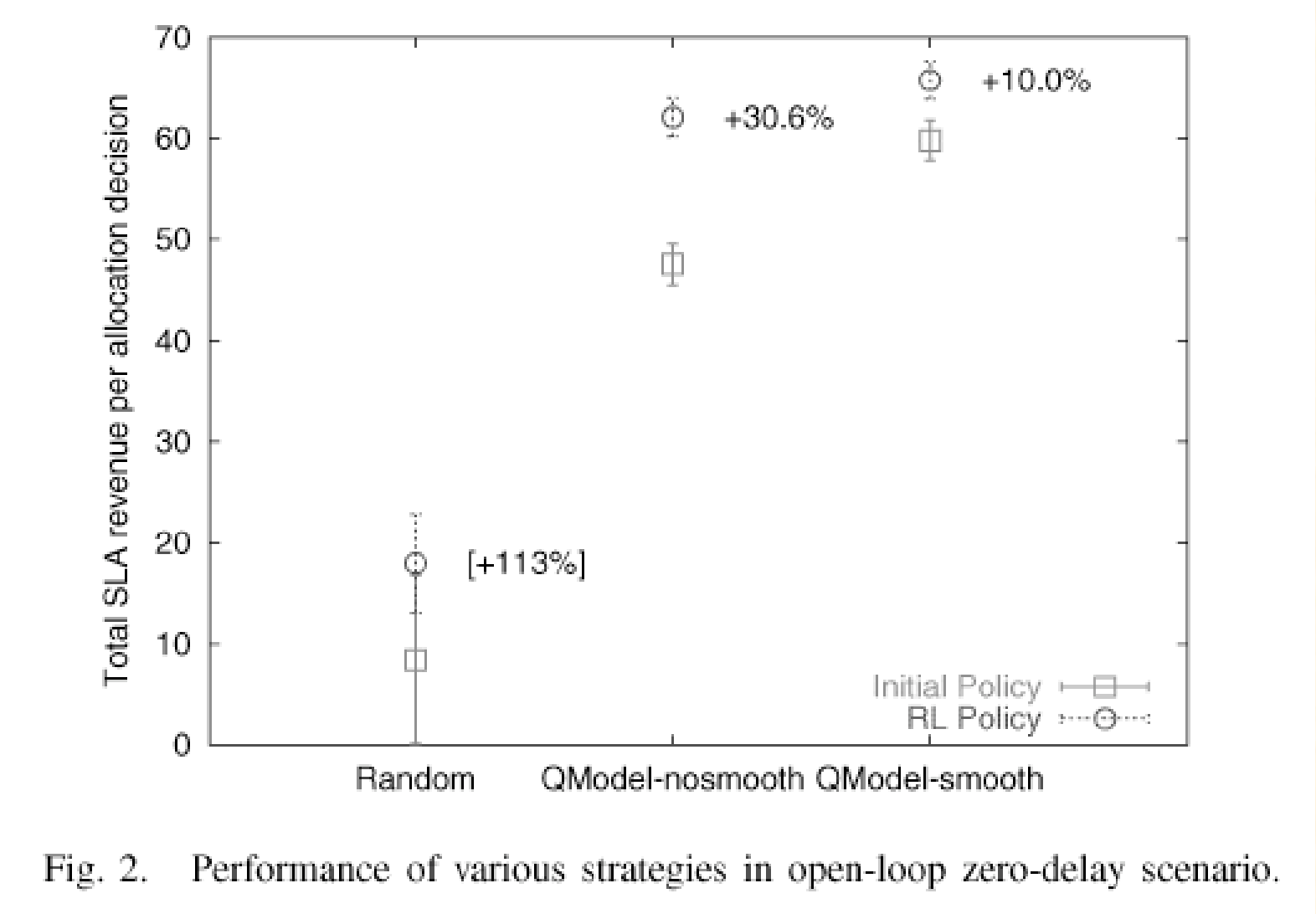

Fig. 2. Performance of various strategies in open-loop zero-delay scenario.

Prototype Results

Performance Without Switching Delays

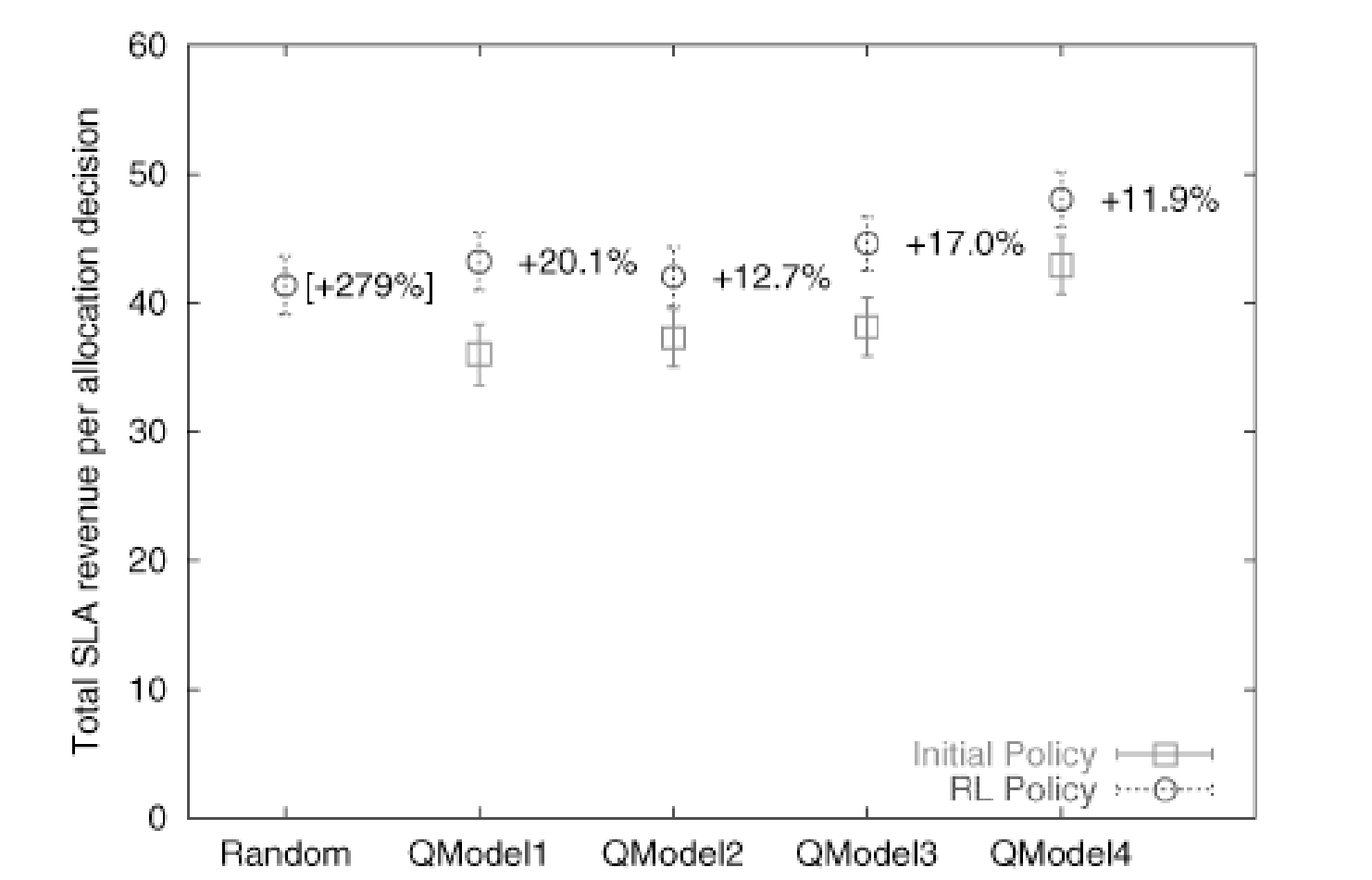

Fig. 3. Performance of various strategies in closed-loop zero-delay scenario. (The random policy performance lies off the scale at -23.0.)

Prototype Results

Performance Without Switching Delays

- QModel1: No smoothing parameter (for think time and Java garbage collection) for the **Service rate μ.**
- QModel2: Smoothing parameter = 0.1, and predicts future utility instead of immediate utility.
- Qmodel3: Uses the parallel M/M/1 model used in the open- loop network scenario (just for comparison).
- QModel4: Smoothing parameter = 0.1.

Prototype Results

Performance With Switching Delays

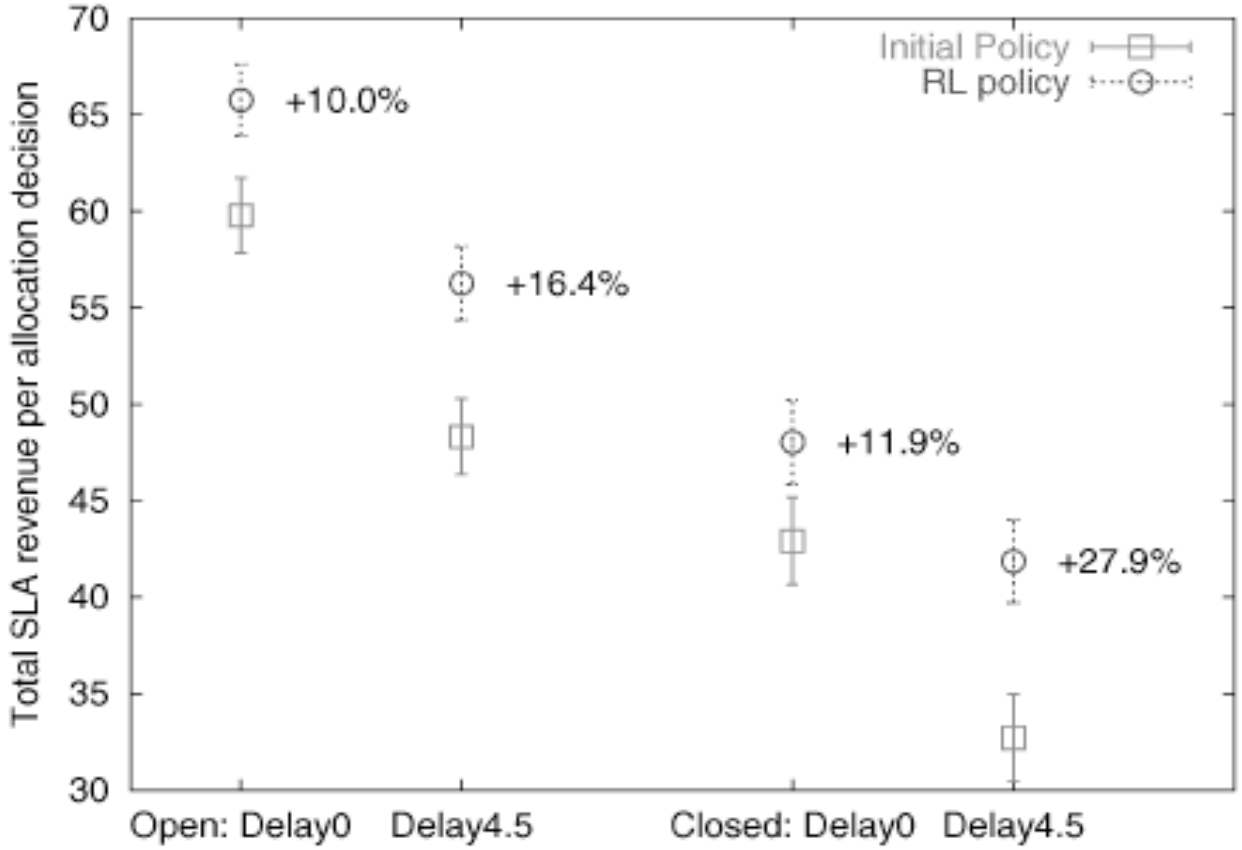

Fig. 5. Comparison of delay=4.5 sec with delay=0 results in open-loop and closed-loop scenarios.

Conclusions

- Hybrid learning method for resource valuation estimates, combining reinforcement learning and model-based policies.
- Hybrid RL approach could be applied in many performance management applications:
  - Dynamic allocation of other types of resources, e.g., bandwidth, memory, CPU slices, threads, LPARs, etc.
  - Performance-based online tuning of system control parameters, such as web server parameters, OS parameters, database parameters, etc.
  - Simultaneous management to multiple criteria (e.g. performance and availability), as long as the rewards pertaining to each criterion are on an equivalent numerical scale.