Factored Reinforcement Learning for Auto-scaling in Tandem Queues

Thomas Tournaire, Yue Jin, Armen Aghasaryan, Hind Castel-Taleb, Emmanuel Hyon

Factored Reinforcement Learning for Auto-scaling in Tandem Queues

Thomas Tournaire
Nokia Bell Labs France, SAMOVAR
Paris-Saclay, France
thomas.tournaire@nokia.com

Yue Jin
Nokia Bell Labs France
Paris-Saclay, France
yue1.jin@nokia-bell-labs.com

Armen Aghasaryan
Nokia Bell Labs France
Paris-Saclay, France
armen.aghasaryan@nokia-bell-labs.com

Hind Castel-Taleb
SAMOVAR, Telecom SudParis
Evry, France
hind.castel@telecom-sudparis.eu

Emmanuel Hyon
UPL, Université Paris Nanterre
Nanterre, France
emmanuel.hyon@parisnanterre.fr

***Abstract*—As today's networking systems utilises more virtualisation, efficient auto-scaling of resources becomes increasingly critical for controlling both the performance and energy consumption. In this paper, we study the techniques to learn the optimal auto-scaling policies in a distributed network when parts of the system dynamics are unknown. Reinforcement Learning methods have been applied to solve auto-scaling problems. However they can run into computational and convergence issues as the problem scale grows. On the other hand, distributed networks have relational structures with local dependencies between physical and virtual resources. We can exploit these structures to overcome the convergence issues by using a factored representation of the system.**

We consider a distributed network in the form of a tandem queue composed of two nodes. The objective of the auto-scaling problem is to find policies that have a good trade-off between quality of service (QoS) and operating costs. We develop a factored Reinforcement Learning algorithm, named FMDP online, to find the optimal auto-scaling policies. We evaluate our algorithm with a simulated environment. We compare it with existing Reinforcement Learning methods and show its relevance in terms of policy efficiency and convergence speed.

***Index Terms*—Factored MDP, Reinforcement Learning, Auto-scaling, Queuing Systems, N tier architecture**

I. Introduction

With the proliferation of new services (e-health, vehicular networks, video streaming,...) network operators face two main challenges : QoS (Quality of Service) guarantee and control of operating costs, such as energy consumption. Resource auto-scaling [1] technique is a very efficient solution for adapting resource to a varying service demand by activating and deactivating virtual resources according to the workload [2]. This enables the control of both the performance and operating costs.

Markov Decision Process (MDP) has been widely used to model resource management problems including auto-scaling problems. Different algorithms exist to find the optimal management policy, when the underlying transition probabilities are known [21]. When the system's statistics is not known, Reinforcement Learning (RL) techniques can be employed to learn the optimal policy. Many recent papers present RL solutions for networking problems: [8] for auto-scaling in cloud, [11] for 5G network slicing with RL and [13], [18] for slicing with Deep RL, to quote just a few. Although MDP/RL methods hold great promise for learning adaptive control policies, they can still suffer from slow convergence caused by '*curse of dimensionality*'. There is a great interest to study techniques that limit the dimensionality while offering more guarantees of accuracy and greater explainability.

In this paper we propose **FMDP online**, a factored **RL** algorithm that can overcome these issues by expressing the problem in a compact form. This provides the learning agent with the relational structure of the environment. Indeed, we believe that in distributed network environments such as $5G$ slicing architectures or queuing networks, there exist local dependencies between physical nodes. Events in a physical node have direct impacts only on its neighbours and affect other nodes indirectly in most cases. In this context *Factored MDP* [3] framework takes advantage of local dependencies structure in the environment, accelerates convergence and overcomes the so-called 'curse of dimensionality' in MDP solutions. We use the Factored MDP framework to model the auto-scaling problem in a tandem queue with two nodes. The objective of the auto-scaling problem is to find policies that have a good trade-off between quality of service and operating costs and minimise the total discounted costs.

Main contributions: We propose a factored representation of a tandem queuing system and devise a factored Reinforcement Learning method; we evaluate it and show a gain comparing to existing model-free and model-based RL methods, in terms of policy efficiency and convergence speed. Although we focus on auto-scaling techniques in this work, we strongly believe that factored approaches are applicable in large distributed networks (5G slicing, etc).

Paper structure: We briefly present the related works and background methodologies in sections II and III, respectively. We describe in section IV the multi-tier architecture modelled as a tandem queue and the auto-scaling problem modelled

as a Markov Decision Process. We describe the factored representation of the tandem queue environment and factored Reinforcement Learning method in V. We show experimental results and comparison between assessed algorithms in section VI. Finally we discuss our results in the conclusion and provide comments about further research issues.

II. Related Works

Our work is related to a number of research topics: Queuing Models and Markov Decision Process in resource management, Reinforcement Learning solution for cloud resource allocation and factorisation in Reinforcement Learning.

A. Queuing Models and Markov Decision Process for Resource Management

Resource management in queuing models is frequently used to represent the auto-scaling techniques involving activations and deactivations of virtual resources. A server farm represented by multi-server queuing systems [16] is one existing model. The research on optimal auto-scaling policies has led to a large body of relevant literature and a wide variety of solution methods. Dynamic control and especially Markov Decision Process appear to be the most direct method. Numerous works have been devoted to compute optimal policy in multi-server queue models with MDP (see [22] and references therein). However, Markov Decision Process framework requires a complete knowledge of the model (queuing statistics such as arrival or service distributions etc.) which might not be available in real systems.

B. Reinforcement Learning for Resource Allocation

A very comprehensive survey about Reinforcement Learning techniques for auto-scaling in the cloud can be found in [8]. Many works for RL application on cloud resource allocation problems are discussed and different taxonomies about the resolution techniques, the criteria to optimise as well as the type of problem are presented.

A model-free Reinforcement Learning technique with Q-Learning for autonomic resource allocation in the cloud was proposed by Dutreilh and al. in [7]. In this work an agent has to control the number of resources and optimise a cost function that takes into account virtual machines costs and SLA. Moreover, [9] proposes Q-learning to derive auto-scaling policies in the cloud to minimise delays and number of activated resources.

In [8] only two methods are classified as model-based techniques. These two works estimate the transition probabilities and solve the problem with MDP algorithms. However the representation of planning problems as MDPs often cause issues as the size of the state spaces grows large. In general, the state space of planning problems can be described in terms of a set of domain features. The state spaces grow exponentially in the number of features and result in the so-called Bellman's "curse of dimensionality". In order to resolve this difficulty, both the specification of an MDP—in particular, the specification of system dynamics and a reward function—and the computational methods used to solve MDPs must be reworked. This leads to the our focus in the next section: the factored representation of MDPs and the factored Reinforcement Learning.

C. Factored Reinforcement Learning

Factored MDP framework provides the learning agent with local dependencies between state variables, expressed with Dynamic Bayesian Networks (DBNs). The aim of factored representation is to represent more compactly the environment dynamics. This brings several benefits: smaller memory implementation for large-scale systems and higher speed of convergence.

Since the seminal work of Boutilier [3], several studies have devised factored MDP/RL algorithms. We give an overview of existing factored Reinforcement Learning approaches. Most of the works consider graphical representations of trees [14] or a DBN [6]. These two objects allow to represent relations between state variables and conditional probabilities in a compact form. However these works often show applications with binary feature variables in the state space. In this setting, [15] extends the current framework with multi-valued feature variables.

On the other hand some works have been investigating factored form of classic MDP algorithms. This includes factored policy iteration [12] and factored value iteration [20]. For model-based Reinforcement Learning techniques, Kearns and al. present in [10] a factored implementation of the E_3 MDP-online algorithm. Still, the very few number of applications with factored solutions in the literature call for further research in this topic and for comparison with existing model-free and model-based algorithms. To the best of our knowledge, there hasn't been works that apply factored RL approaches on networking systems. This is the focus of our work in this paper.

III. Background

A. Reinforcement Learning

Reinforcement Learning [19] involves an agent taking actions in an environment to maximise some cumulative reward by trials and errors. Markov Decision Process [17] is the theoretical model behind this framework. It allows to describe the interactions between the agent and its environment in terms of states, actions, and rewards.

The state space $\mathcal{S}$ represents how the agent perceive the environment. The action space $\mathcal{A}$ is the set of available decisions of the agent. We denote by $\mathcal{R}(s, a)$ the reward obtained after taking action a in state s. The environment behaves stochastically and $P\left(s'|(s, a)\right)$ denotes the probability to move in state s' after taking action a in state s.

The aim of the agent is to maximise the total reward received during an episode. For this aim, the agent needs to determine the best policy π. The return of an episode of length

T is defined as the expected sum of the discounted rewards when following a policy π:

$$V^{\pi}(s) = \mathrm{E}^{\pi}\Big(\sum_{k=0}^{T} \gamma^t \mathcal{R}(s_k, \pi(s_k)) \mid s_0 = s\Big).$$

We also define the action-value function (also called Q-function):

$$Q_{\pi}(s,a) = \mathcal{R}(s,a) + \gamma \sum_{s'} P(s'|(s,a)) V^{\pi}(s').$$

We have $V_{\pi}(s) = \arg\max_a Q_{\pi}(s,a)$ and $\pi^* = \arg\max_{\pi} V_{\pi}$.

We define the true model $\mathcal{M} = \{P, \mathcal{R}\}$ as the dynamics of the environment. In Reinforcement Learning scenarios, such model is unknown and learning is needed to overcome this lack of information. Thus two frameworks exist: model-free learning in which agent updates its policy or value function directly from experience; and model-based learning in which agent tries to approximate the model $\tilde{\mathcal{M}} = \{\tilde{P}, \tilde{\mathcal{R}}\}$ for offline planning.

B. Factored MDP framework

Factored MDP (fMDP) [3] is a feature-based representation of MDP framework. Formally, the environment state can be characterised by a finite set of random variables $s = \{s_1, \ldots, s_N\}$, each with a finite domain $Val(s_i)$. It uses dynamic Bayesian networks (DBNs) to represent the transition probabilities associated with a specific action a. The Bayesian network nodes correspond to local state variables s_i and is partitioned into two sets: the state of the system before the action is performed, s_i and the state after the action is executed, s'_i. Edges between these two set of nodes represent direct probabilistic influence among the corresponding variables under the action a. For a DBN under action a we can extract the conditional probability distributions (CPDs) of each variable s_i at time $t+1$, denoted $P_i(s'_i|Pa(s_i), a)$, where $Pa(s_i)$ represents the parents variables of s_i at time t. To express the global conditional distribution $P(s'|s,a)$ as a product of local conditional probabilities, we need to ensure there are no synchronic arcs at time $t+1$.

Proposition 1 ([15]). *If $\forall s'_i, s'_j \in \mathcal{S}', s'_i \perp\!\!\!\perp s'_j | \mathcal{S}$, then*

$$P(s'|s,a) = \prod_i P_i(s'_i|Pa(s_i), a)$$

C. MDP resolution

We consider dynamic programming methods when the system's dynamics is known, i.e. when the agent has access to P and $\mathcal{R}$.

1) Value Iteration: The Value Iteration algorithm [17] uses the Bellman equations to update the value function V. It starts with V_0 and updates at iteration t the value function $V(s)$ for all state s with backprojection formula under matricial form:

$$V_{t+1} \leftarrow \max_a(r^a + \gamma P^a V_t)$$

The Bellman operator $\mathcal{T}$ is a max-norm contraction such that:

$$\mathcal{T}V := \max_a(r^a + \gamma P^a V) \text{ and } V_{t+1} = \mathcal{T}V_t$$

2) Approximate Value Iteration with Linear Approximation: The Value Iteration algorithm also exist for linear approximation of the value function. Consider the value function V to be a linear combination of K basis functions h_k where $K << |\mathcal{S}|$. We define $\mathcal{H}$ the $|\mathcal{S}| \times K$ basis function matrix. By substituting V_t by $\mathcal{H}w_t$, we can rewrite Bellman equations:

$$\mathcal{H}w_{t+1} \leftarrow \max_a(r^a + \gamma P^a \mathcal{H} w_t)$$

However the right-hand-side (r.h.s.) might not be contained in the image space of $\mathcal{H}$. Hence, we need to project the r.h.s. on the image space with a projector $\mathcal{G}$:

$$w_{t+1} \leftarrow \mathcal{G}\left[\max_a(r^a + \gamma P^a \mathcal{H} w_t)\right]$$

3) Factored Value Iteration: Factored Value Iteration [20] presents a factored version of the Value Iteration with convergence guarantees. We first have linear decomposition of reward function and value function.

$$\mathcal{R}(s,a) = \sum_{j=1}^{J} \mathcal{R}_j(s[Z_j], a) \ , \ V(s) = \sum_{k=1}^{K} h_k(s[C_k]) \cdot w_k$$

where $Z_j \subseteq \mathcal{S}$ and $C_k \subseteq \mathcal{S}$ are sets of local variables. Here $s[Z_j]$ and $s[C_k]$ corresponds respectively to the values of local variables in Z_j and C_k for state s.

The idea of factored value iteration is to replace Bellman's equation with a factored version, by considering factored local transitions P_i and linear value function approximation. Recall that $V_{t+1} = \sum_{k=1}^{K} h_k(s[C_k]) \cdot w_k^{t+1}$.

$$V_{t+1} = \mathcal{G}\max_a \left[\sum_{j=1}^{J} \mathcal{R}_j(s[Z_j], a)\right] + \gamma \sum_{k=1}^{K} \sum_{s'[C_k] \in \mathcal{S}[C_k]} \Big(\prod_{i \in C_k} P_i(s'_i|Pa(s_i), a)\Big) h_k(s'[C_k]) w_k^t$$

By considering the backprojection matrix $\mathcal{B}$ such that:

$$\mathcal{B}^a_{s,k} = \sum_{y[C_k] \in \mathcal{S}[C_k]} \Big(\prod_{i \in C_k} P_i(y_i|x[\Gamma_i], a)\Big) h_k(y[C_k])$$

in matricial computation we can update the weights of the value function by:

$$w^{t+1} = G\max_a \Big[r^a + \gamma \mathcal{B}^a w^t\Big]$$

where G is the matrix form of projection operator $\mathcal{G}$.

Also the authors [20] propose in their work a *sampling* method to work on a subset of the original state space $\hat{\mathcal{S}} \subseteq \mathcal{S}$ where $|\hat{\mathcal{S}}| = \text{poly}(N)$.

D. RL resolution

When the agent does not have access to P and $\mathcal{R}$, we require RL algorithms. We recall standard RL algorithms that will be used for comparison with our proposal.

1) Model-free Q-Learning: The agent does not have any model of the dynamics and updates its state-value function Q by interactions with the environment. For a given observation of the system s it chooses an action a with an ϵ-greedy policy then observes new state s' and reward r. It updates Q from its experience $< s, a, r, s' >$ with the Q-learning update: $Q(s,a) \leftarrow Q(s,a) + \alpha[r(s,a) + \gamma \max_{a'} Q(s',a') - Q(s,a)]$.

2) Dyna-Q or buffer replay RL: Dyna architectures [19] aims to improve model-free methods by adding a supplementary planning phase where the agent can update more often its state-value function from experiences. It stores experiences in a replay memory $\mathcal{D}_i = \{< s, a, r, s' >_1, < s, a, r, s' >_2, \cdots, < s, a, r, s' >_i\}$ where the agent can select tuples to additionally update Q.

3) State-transitional model-based RL: In state-transitional model-based methods, the agent learns a standard approximated model of the environment dynamics $\tilde{\mathcal{P}}, \tilde{\mathcal{R}}$. It updates the dynamics and reward functions/tables with the buffer memory $\mathcal{D}_i$ and uses a planning method on the model learned so far to decide the actions. Planning can be done in two different ways here: using dynamic programming algorithms such as Value Iteration [17] or using its model to generate new samples and update in a model-free fashion.

IV. Multi-tier Model

This section presents the behaviour of our three-tier architecture model. We describe the queuing system as well as the transition probabilities and the costs.

A. Model Description

We concentrate on a small segment of a multi-tier network with two physical nodes in tandem (Figure 1). Each node is represented by a multi server queue (or a buffer, where jobs wait for a service) and servers (or Virtual Machines: VMs) which can be activated or deactivated by a controller. We assume that each node i has a finite capacity B_i, i.e. the maximum number of requests either waiting for a service or in service. We define K_i the maximum number of usable VMs in node i. We must have at least one machine activated; $K_i \geq 1$. All virtual machines (VMs) in a given node are homogeneous, and the service rates can be modelled by an exponential distribution with rate μ_i for node i ($i = 1, 2$).

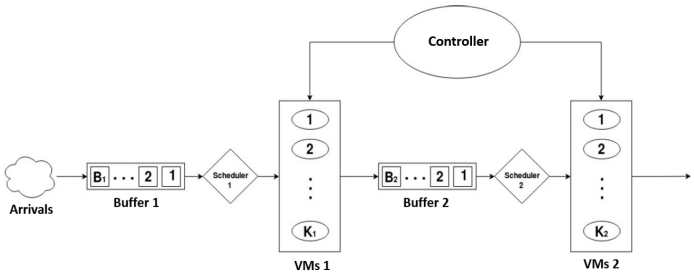


Fig. 1: Tandem queue representation of the three-tier architecture

We suppose that requests arrive in the system only in node 1 and the arrivals follow a Poisson process with parameter λ. When a request arrives in node 1 and finds B_1 customers in node 1, it is lost. Otherwise, it waits in the queue until a server becomes free. After being served in node 1 the request enters in node 2, unless the second queue is full in which case the request is lost. Once the request finishes its service in node 2, it leaves the system.

At each transition epoch, a single controller manages the number of activated VM in each node and can decide to turn on or turn off virtual machines or to do nothing. Only one VM can be deactivated or activated in each node each time.

B. Semi Markov Decision Process Description

1) System's dynamics: The system state includes the current number of requests in node 1 and node 2, denoted by m_1 and m_2 respectively, as well as the number of active servers in node 1 and node 2, denoted by k_1 and k_2 respectively. Thus, the state space is defined by $\mathcal{S}$ with $s = (m1, m2, k1, k2) \in \mathcal{S}$ such that $0 \leq m1 \leq B1$, $0 \leq m2 \leq B2$, $1 \leq k_1 \leq K_1$, and $1 \leq k_2 \leq K_2$.

We denote by a_i the action available in node i (with $i \in \{1, 2\}$). It can take three values: $\mathbf{1}$ if we activate one VM; $-\mathbf{1}$ if we deactivate one VM; and $\mathbf{0}$ if the number is left unchanged. Any action taken by the controller is the couple of actions in each of the nodes. Hence, the action space is $\mathcal{A}$ where $a = (a_1, a_2) \in \mathcal{A}$ with $a_i \in \{-\mathbf{1}, \mathbf{0}, \mathbf{1}\}$.

We describe now the transitions. We consider here that the controller can observe the system just after any change in the state and reacts. The actions are instantaneous. When the number of active servers is k_i at node i and we take the action a_i, the number of active servers changes to $N(k_i + a_i) = \min\{\max\{1, k_i + a_i\}, K_i\}$ After an action, the system evolves until the next transition occurs.

So, at state $s = (m1, m2, k1, k2)$, after triggering action $a = (a_1, a_2)$, the possible transitions are:
An arrival in queue 1: we move, with rate λ, to:

$$s'_1 = \big(\min(m_1 + 1, B1), m_2, N(k_1 + a_1), N(k_2 + a_2)\big)\,.$$

A departure from queue 1 (and consequently an arrival in queue 2): we move, with rate $\mu_1 \min(m_1, N(k_1 + a_1))$ to:

$$s'_2 = \big(\max(m_1 - 1, 0), \min(m_2 + 1, B_2), N(k_1 + a_1), N(k_2 + a_2)\big).$$

A departure from queue 2: we move, with rate $\mu_2 \min(m_2, N(k_2 + a_2))$, to:

$$s'_3 = (m_1, \max(m_2 - 1, 0), N(k_1 + a_1), N(k_2 + a_2))\,.$$

We define the transition rate of state-action pair (s, a) by

$$\Lambda(s,a) = \lambda + \mu_1 \min\{m_1, N(k_1 + a_1)\} + \mu_2 \min\{m_2, N(k_2 + a_2)\}\,.$$

2) System's costs: We consider a continuous time discounted model [17] with the discount factor γ. We define the costs that represent the trade-off between quality of services (*QoS*) and energy consumption. There are *instantaneous costs* that are charged once an action is taken: the activation cost of

a VM in node i, C_{A_i}, the deactivation cost in node i, C_{D_i} and the cost of rejecting a request in node i, C_{R_i} when the buffer is full and there is an arrival. There are *accumulated costs* that accumulate over time: the cost per time unit of using a VM in node i, C_{S_i} and the cost per time unit of holding a request in node i, C_{H_i}. Formally, after taking decision $a = (a_1, a_2)$ in state $s = (m_1, m_2, k_1, k_2)$, the accumulated cost c_i at node i equals:

$$c_i(s,a) = N_i(k_i + a_i) \cdot C_{S_i} + m_i \cdot C_{H_i}$$

and the instantaneous cost at node i is equal to:

$$\begin{aligned} h_i(s,a) &= C_{A_i} \cdot \mathbb{1}_{\{a_i=1\}} + C_{D_i} \cdot \mathbb{1}_{\{a_i=-1\}} \\ &\quad + \frac{\lambda}{\Lambda(s,a)+\gamma} C_{R_i} \mathbb{1}_{\{m_i=B_i\}} \text{ if } i = 1; \end{aligned}$$

$$\begin{aligned} h_i(s,a) &= C_{A_i} \cdot \mathbb{1}_{\{a_i=1\}} + C_{D_i} \cdot \mathbb{1}_{\{a_i=-1\}} \\ &\quad + \frac{\min\{m_1, N(k_1+a_1)\} \cdot \mu_1}{\Lambda(s,a)+\gamma} C_{R_i} \mathbb{1}_{\{m_i=B_i\}} \\ &\quad \text{if } i = 2\,. \end{aligned}$$

3) Objective function: We define a Markov Deterministic stationary policy π as a mapping $\pi : \mathcal{S} \to \mathcal{A}$. Based on the discounted model in [17], we define the stage cost function $\mathcal{R}$ as:

$$\mathcal{R}(s,a) = \frac{1}{\Lambda(s,a)+\gamma} \left[c_1(s,a) + c_2(s,a) + h_1(s,a) + h_2(s,a)\right].$$

We search the best policy π to minimise the expected discounted reward. The objective function is

$$V^*(s) = \min_{\pi} \mathrm{E}^{\pi} \left[\sum_{k=0}^{\infty} \exp^{-\gamma t_k} \mathcal{R}(s_k, \pi(s_k)) \mid s_0 = s \right], \tag{1}$$

where t_k is the epoch and s_k is the state of the kth transition.

4) Uniformisation: Most Reinforcement Learning models deal with discrete time scenario. In order to handle this control model with an RL approach we uniformise the continuous time model, following the process of [17]. We refer to uniformised objects by symbols augmented with a bar $-$. We define the uniformisation rate as $\bar{\Lambda} = \lambda + K_1 \cdot \mu_1 + K_2 \cdot \mu_2$.

$$\bar{\Lambda} \times \bar{P}(s'|s,a) = \begin{cases} \lambda & \text{if } s' = s'_1 \\ \mu_1 \min\{m_1, N(k_1+a_1)\} & \text{if } s' = s'_2 \\ \mu_2 \min\{m_2, N(k_2+a_2)\} & \text{if } s' = s'_3 \\ \left(\bar{\Lambda} - \Lambda(s,a)\right) & \text{when } s' = s \\ 0 & \text{otherwise}\,. \end{cases}$$

For the stage costs they are now given in the discounted model by:

$$\bar{\mathcal{R}}(s,a) = \frac{\Lambda(s,a)+\gamma}{\bar{\Lambda}+\gamma} \mathcal{R}(s,a)\,.$$

Then the Bellman Equation of such a model is:

$$V^*(s) = \min_{a \in \mathcal{A}} \Big(\bar{\mathcal{R}}(s,a) + \frac{\bar{\Lambda}}{\bar{\Lambda}+\gamma} \sum_{s'} \bar{P}(s'|(s,a)) V^*(s') \Big)\,.$$

Note that this equation is never solved in R.L. methods but we solve it in our numerical experiments to assess the accuracy of the algorithms. Indeed its solution is the theoretical optimal value.

C. Simulated SMDP environment for RL agent

We are dealing with a RL model therefore the uniformised model of Section IV is not known but only experienced by the controller. Indeed, the controller does not have any information about queuing statistics (arrival rate, etc.), therefore does not know the dynamics of the system. The environment has state space $\mathcal{S}$ and action space $\mathcal{A}$ similarly to the SMDP model. When the agent interacts with the environment, the SMDP model is simulated. It will behave by returning a state, sampled according to the transition probabilities $\bar{P}$, and return the costs $\bar{\mathcal{R}}(s,a)$.

V. Factored-transitional model-based RL

In this section we first present the factored representation of the tandem queue environment regarding state space, dynamics and reward. Next we give the methodology of our factored Reinforcement Learning method and its parameterisation.

A. Factored model representation

Understanding structural properties of the tandem queue system is key to help the agent learn faster the auto-scaling policy. We model the tandem queue environment with a factored approach to bring this knowledge to the learning agent. If we currently try to use factorisation on the current state space representation, then we will have dependencies within the state at time $t+1$, which will violate Proposition 1. In the tandem queue system the actions a does not affect the structure of the Bayesian network. Yet all the DBNs in our environment have the same relational structure. We define relations between state variables and build a dictionary of child-parent variables, displayed in Table I.

Child variables at time t+1	Parents variables at time t
m_1	m_1, k_1
m_2	m_1, k_1, m_2, k_2
k_1	k_1
k_2	k_2

TABLE I: Child-parent architecture

In the current MDP model of tandem queue, the conditional independence assumption (Proposition 1) is violated, rending the factored decomposition impossible. In other words, the factored form of the probability distribution $P(s'|s,a) = \prod_{i=1}^{N} P_i(s'_i|Pa(s_i),a)$ is not possible because the two events, departure from node 1 ($m_1 \to m_1 - 1$) and arrival in node 2 ($m_2 \to m_2 + 1$), are the same, thus interdependent at time $t+1$. In details, $p(s'|s,a) \neq p(m'_1|m_1,k_1,a)p(m'_2|m_2,k_2)p(k'_1|k_1,a)p(k'_2|k_2,a)$ when a customer goes from node 1 to node 2. Hence we did slight modification of the MDP model to build the factored model.

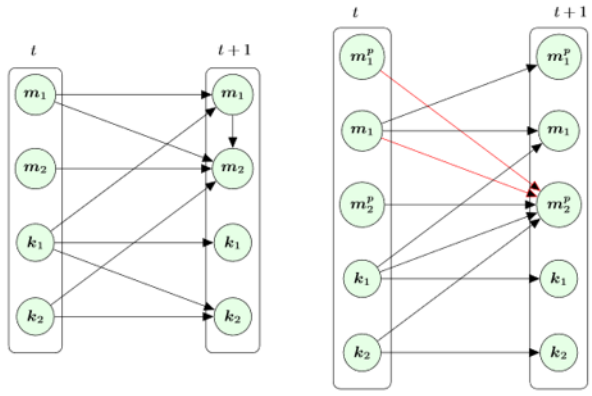


Fig. 2: Dynamic Bayesian Networks (l.h.s. for initial state space ; r.h.s. for augmented factored state space)

1) Augmented state space: To overcome the violated assumption, we extended the state space to break the dependency between the variables m_1 and m_2 at time $t+1$. This is done while considering variables m_1 and m_2 at previous time step $t-1$, denoted m_1^p, m_2^p and the current variable m_1 at time t. We now consider the state at time t: $s = (m_1^p, m_1, m_2^p, k_1, k_2)$. We draw the DBN associated to the new representation of the state space in Figure 2.

2) Factored system's dynamics: We define the factored transition probabilities P_i for each local state variable s_i. First we have deterministic local transitions for servers variables k_1 and k_2 since these values only depend on the agent's decision a. Therefore, we will move from local states k_1 to $N(k_1+a_1)$ and from k_2 to $N(k_2+a_2)$ with probability 1.

We have the same factored transition for local variable m_1^p since m_1^p at time $t+1$ simply retrieves the value of local variable m_1 at time t. Therefore, the randomness in the system is only integrated in the factored transitions of local variables m_1 and m_2^p, which will represent possible events (arrival in node 1, departure from node 1 to node 2 and departure from node 2).

Now the full probability distribution can be written under the factored form.

$$\begin{aligned} P(s|s',a) &= P(m_1^{p'}|Pa(m_1^p),a)\cdot P(m_1'|Pa(m_1),a)\cdot \\ &\quad P(m_2^{p'}|Pa(m_2^p),a)\cdot P(k_2'|Pa(k_2),a)\cdot \\ &\quad P(k_1'|Pa(k_1),a) \end{aligned}$$

3) Factored system's costs: The reward function is also linearly decomposed. We decide to represent two features, one for each node i. We have $\mathcal{R}(s,a) = \mathcal{R}_1(m_1,k_1,k_2,a) + \mathcal{R}_2(m_1,m_2,k_1,k_2)$ where:

$$\begin{aligned} \mathcal{R}_1(m_1,k_1,k_2) &= (m_1 C_{H_1} + (k_1+a_1)C_{S_1})/\bar{\Lambda} \\ &\quad + C_{R_1}\lambda\frac{\Lambda(s,a)+\gamma}{\bar{\Lambda}+\gamma} \\ &\quad + C_{A_1}\frac{\Lambda(s,a)+\gamma}{\bar{\Lambda}+\gamma}1_{(a_1=1)} \\ &\quad + C_{D_1}\frac{\Lambda(s,a)+\gamma}{\bar{\Lambda}+\gamma}1_{(a_1=-1)} \end{aligned}$$

$$\begin{aligned} \mathcal{R}_2(m_1,m_2,k_1,k_2) &= (m_2 C_{H_2} + (k_2+a_2)C_{S_2})/\bar{\Lambda} \\ &\quad + C_{R_2}k_1\mu_1\frac{\Lambda(s,a)+\gamma}{\bar{\Lambda}+\gamma} \\ &\quad + C_{A_2}\frac{\Lambda(s,a)+\gamma}{\bar{\Lambda}+\gamma}1_{(a_2=1)} \\ &\quad + C_{D_2}\frac{\Lambda(s,a)+\gamma}{\bar{\Lambda}+\gamma}1_{(a_2=-1)} \end{aligned}$$

B. Factored MDP online algorithm

1) Methodology: As opposed to state-transitional model-based methods, we now assume that the agent has a supplementary knowledge. It understands the relational structure of the environment: how state variables are related to each other. The dynamics are now represented by DBNs and local conditional distributions $P_i(s_i'|Pa(s_i),a)$. However the agent does not know the statistics and needs to interact with the environment to learn these statistics for planning. We assume that the reward function is known and only dynamics P_i have to be updated. We devise **FMDP online** (Algorithm 1) that uses methodologies from MDP online algorithms and the Factored Value Iteration (FVI) [20] for the planning phase.

The method starts by assuming that the initial system is in an absorbing state, i.e. we initialise all local CPTs as identity matrices [4]. This is fair assumption at the beginning since the agent does not know how the environment behaves. At the end of episode e, it updates its local CPTs from its experiences. The update is done by counting occurrences, i.e. :

$$P(s_i'|Pa(s_i),a) = \frac{n(s_i',Pa(s_i),a)}{n(Pa(s_i),a)}$$

where $n(\cdot)$ denotes the number of occurrences and tuple $(s_i',Pa(s_i),a) = (s'[i],s[Pa(s_i)],a$.

Since the agent might not be confident in the approximated model $\tilde{\mathcal{M}}$ during first episodes, we only run FVI algorithm for a few number of iterations. In practice, we do e planning iterations to update the weights w of the value function V.

Also for the FVI algorithm we have a linear decomposition of the value function. Thus we need to define the basis functions which is very tough to adjust for good results. Last, we need to choose a projector for the factored Bellman update to compute the weights w.

2) Feature selection for linear approximation of the value function: We describe our choices for basis functions and projection operator. We choose least-squares (L2-)projection for the projection operator $\mathcal{G}$, according to [20]. Least-squares fitting is very often applied for projecting value functions. In this case, the linear weights w are chosen so that it minimises the least-squares error $w = \text{argmin}_w||Hw - v||_2^2$.

This corresponds to the linear projection $G = H^+$ (i.e., $w = H^+v$) where $H^+ = H^T(HH^T)^{-1}$ is the Moore-Penrose pseudoinverse.

For basis functions selection we decided to select several sub-parts of the reward function. We have chosen 8 basis

functions to combine as much as possible the relations between the local variables and the impact of each local variables so it will be integrated in the value function.

Basis H	Functions
$h_1(m_1^p)$	$(C_{h_1} \cdot m_1^p)/\bar{\Lambda}$
$h_2(m_2^p)$	$(C_{h_2} \cdot m_2^p)/\bar{\Lambda}$
$h_3(k_1)$	$(C_{s_1} \cdot k_1)/\bar{\Lambda}$
$h_4(k_2)$	$(C_{s_2} \cdot k_2)/\bar{\Lambda}$
$h_5(m_1^p, k_1, k_2)$	$\lambda C_{r_1} \cdot 1_{(m_1^p = B_1)}/\bar{\Lambda}$
$h_6(m_1^p, m_2^p, k_1, k_2)$	$\min(m_1^p, k_1)\mu_1 C_{r_2} \cdot 1_{(m_2^p = B_2)}/\bar{\Lambda}$
$h_7(m_1^p, k_1)$	$\big((C_{h_1} \cdot m_1)^2 + C_{s_1} \cdot k_1\big)/\bar{\Lambda}$
$h_8(m_2^p, k_2)$	$\big((C_{h_1} \cdot m_1)^2 + C_{s_1} \cdot k_1\big)/\bar{\Lambda}$

TABLE II: Basis functions selection for factored value iteration

Algorithm 1: Factored MDP online algorithm

Input: Basis functions H_i, Local CPTs $P_i = \mathbf{I}$, $V^0 = Hw^0, \pi^0$

Output: π^*, V^*

Data: Statistics unknown, DBNs known

for $e \in Episode$ **do**
  for $i \in Iterations$ **do**
    Select state $s \in \mathcal{S}$
    Take action $a \in \mathcal{A}$ with epsilon-greedy policy.
    Observe s' and reward $r(s,a)$ and collect tuple $< s, a, r, s' >$
  for $j = 1, .., n$ **do**
    Update local transitions $P(s'_j | Pa(s_j), a)$
  w^e, π^e = **Factored Value Iteration** planning with e update iterations

VI. Numerical Results

We present in this section the comparison between our method and state-of-the-art Reinforcement Learning algorithms on the tandem queue system.

A. Environments and Simulation Parameters

1) Gym environments: OpenAI Gym [5] is a toolkit for developing and comparing Reinforcement Learning algorithms. We developed a python simulator in the Gym environment to simulate the tandem queue model.

2) Cloud parameters: We first describe cloud simulations parameters. We consider two cloud scenarios of different scales $C_1 : \{B_1 = B_2 = 10, K_1 = K_2 = 3, \mu_1 = \mu_2 = 2, \lambda = 8\}$ and $C_2 : \{B_1 = B_2 = 15, K_1 = K_2 = 5, \mu_1 = \mu_2 = 2, \lambda = 15\}$. For all these scenarios the costs parameters are the same in the two nodes: $\{C_a = 1, C_d = 1, C_s = 2, C_h = 2, C_r = 10\}$.

3) Algorithms comparison: We compare our **FMDP online** method with state-of-the-art RL algorithms. We choose one model-free RL method: **Q-Learning**, one buffer-based RL method: **Dyna Q** and finally the conventional model-based RL method: **MDP online**. The MDP online algorithm works similarly to FMDP online. However it works directly with the global conditional distribution $P(s'|s,a)$. Therefore it can suffer from heavy update computations because of the matrix size in large-scale systems. This is one issue our method can resolve since it only updates small size matrices (local conditional probabilities P_i), and does not require excessive amount of memory for problems in large-scale systems. We also recall that in the FVI algorithm we can use a sampling method that considers only a subset $\mathcal{S}' \subseteq \mathcal{S}$ of the whole state space.

4) Simulation and algorithms parameters: We detail here the simulation and algorithms parameters. Learning phase runs for 300 episodes of length 10000 iterations. One iteration corresponds to a transition from state s with action a to state s' with reward r. The agent runs epsilon-greedy policy over the whole learning process, with initial epsilon set to $\epsilon = 1$. We decrease epsilon value after each episode with a decay rate $\epsilon_{dec} = 0.99$. The discount rate is set to $\gamma = 0.9$.

B. Comparison Criteria between Algorithms

We compare the different algorithms during the learning phase. Performance measurements during learning are represented by a learning curve. We look at running time of algorithms and the average reward obtained after each episode to compare the goodness of the RL methods. We note that RL algorithms should also be assessed offline, e.g. with Monte-Carlo offline policy evaluation. For this purpose we need to convert the factored policy of the augmented state space for fair comparison with the classical policy. This is currently being investigated as part of further research. Henceforth we only display the comparison during the learning process.

C. Results

We display the average reward over learning episodes for first cloud scenario C_1 in Figure 3. We observe that our method **FMDP online** converges much faster than model-free Q-Learning and Dyna Q techniques. Its convergence is on par with the MDP online techniques. The model-based techniques require much less interactions with the environment to find optimal solution.

Fig. 3: Average reward per episode over learning episodes in cloud C_1

We display numerical results in Figure 4 for the second cloud scenario C_2. The learning process lasted for 100 episodes and the epsilon decay rate was smaller $\epsilon_{dec} = 0.95$. The same conclusions as in scenario C_1 are drawn. The number of iterations required to reach an average reward of 2.05 is impressively less for model-based RL techniques compared to model-free.

In Figure 5, we show the comparison of running time between the MDP online method and our FMDP online method for the two scenarios. In the first scenario, our method has a similar running time to the MDP online method, as shown in 5(a). As the problem scale grows, our method has a lower running time in the second scenario, as shown in 5(b). This demonstrates an advantage of factored solution in execution time with similar policy quality as the problem scale grows.

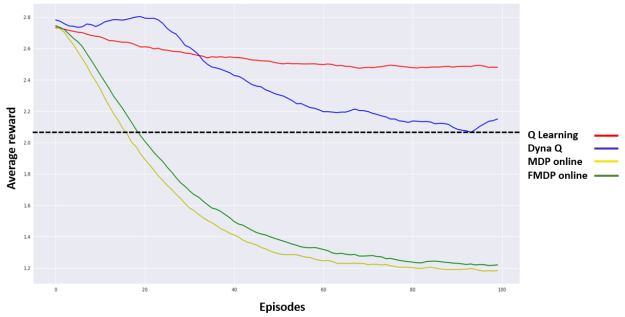
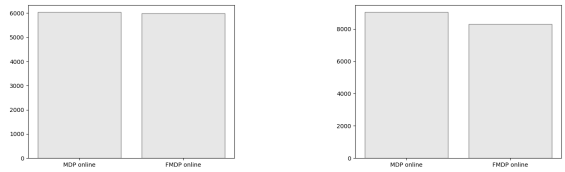


Fig. 4: Average reward per episode over learning episodes in cloud C_2



(a) Running times in seconds for scenario C_1

(b) Running times in seconds for scenario C_2

Fig. 5: Barplots showing running time of model-based RL algorithms

Last we consider a large scale cloud scenario $B = 50$ $k = 10$ with state space considering 250000 states. In this setting, it's not feasible for our computational setup to hold in memory the representations of the full transition matrix (25000x25000) and state values. Thus under tabular forms, only the FMDP online method is feasible, since it only requires lower dimensional arrays to represent local conditional distributions and uses sampling technique.

VII. Discussion and Conclusion

We present **FMDP online**, a factored Reinforcement Learning algorithm for optimal resource management in this work. We demonstrate that if we integrate local relations between state variables in the representation of the environment, it can help to overcome the "curse of dimensionality" issues in standard RL methods.

We have shown the performance improvement obtained by the proposed method in a tandem queue system. It presents a good trade-off between policy quality and computation time. The model-free techniques such as Q-Learning or Dyna architectures may require too many iterations to converge. The model-based algorithms may require fewer iterations to converge. But they suffer greatly on large-scale systems simply due to an exponential increase of the state space. Working with huge matrix representation for the transition probability leads to long computations for the updates and low confidence in the approximated dynamics. Moreover as the state space grows, it may become infeasible to hold the transition matrix in memory. On the contrary, the factored approach have smaller size matrices for the local conditional distributions and can be quickly updated. The sampling technique of factored value iteration also allows to reduce the complexity by considering a subset of the whole state space.

Nevertheless this work requires further investigations. The first one is about evaluating learned policy offline with Monte-Carlo evaluations where we need to convert factored policy for fair comparison. Secondly we would like to extend this work to larger scale network models such as network of queues. Lastly we plan to evaluate FMDP online on very large scale cloud scenarios with deep Reinforcement Learning methods.

References

[1] Amazon. AWS Auto Scaling, 2018.

[2] A. Benoit, L. Lefèvre, A.-C. Orgerie, and I. Rais. Reducing the energy consumption of large scale computing systems through combined shutdown policies with multiple constraints. Int. J. High Perform. Comput. Appl., 32(1):176–188, 2018.

[3] C. Boutilier, R. Dearden, and M. Goldszmidt. Stochastic dynamic programming with factored representations. Artif. Intell., 121:49–107, 2000.

[4] R. I. Brafman and M. Tennenholtz. R-max - A general polynomial time algorithm for near-optimal Reinforcement Learning. J. Mach. Learn. Res., 3(null):213–231, Mar. 2003.

[5] G. Brockman and al. OpenAI GYM. arXiv:1606.01540, 2016.

[6] T. Degris, O. Sigaud, and P.-H. Wuillemin. Learning the structure of factored Markov Decision Processes in reinforcement learning problems. 23^{rd} International Conference on Machine Learning, 2006.

[7] X. Dutreilh and et al. Using Reinforcement Learning for Autonomic Resource allocation in clouds: Towards a fully automated workflow. ICAS, 2011.

[8] Y. Gari, D. A. Monge, E. Pacini, C. Mateos, and C. Garino. Reinforcement Learning-based application autoscaling in the cloud: A survey. Engineering Applications of Artificial Intelligence, 102, 2021.

[9] Y. Jin and al. Testing a Q-learning Approach for Derivation of Scaling Policies in Cloud-based Applications. In ICIN, 2018.

[10] M. Kearns and D. Koller. Efficient reinforcement learning in factored MDPs. In Proceedings of the 16th International Joint Conference on Artificial Intelligence - Volume 2, IJCAI'99, page 740–747, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[11] G. Kibalya, J. Serrat, J.-L. Gorricho, R. Pasquini, H. Yao, and P. Zhang. A reinforcement learning based approach for 5G network slicing across multiple domains. In 2019 15th International Conference on Network and Service Management (CNSM), pages 1–5, 2019.

[12] D. Koller and R. Parr. Policy Iteration for factored MDPs. page 326–334, 2000.

[13] J. Koo, V. B. Mendiratta, M. R. Rahman, and A. Walid. Deep reinforcement learning for network slicing with heterogeneous resource requirements and time varying traffic dynamics. In 2019 15th International

Conference on Network and Service Management (CNSM), pages 1–5, 2019.

[14] O. Kozlova. Hierarchical and factored reinforcement learning. PhD thesis, 2010.

[15] J.-C. Magnan. Représentations graphiques de fonctions et processus décisionnels Markoviens factorisés . Phd thesis, Université Pierre et Marie Curie - Paris VI, Feb. 2016.

[16] I. Mitrani. Managing performance and power consumption in a server farm. Annals of Operations Research, 202:121–134, 2013.

[17] M. Puterman. Markov Decision Processes: Discrete Stochastic Dynamic Programming. Wiley, 1994.

[18] V. Sciancalepore, X. Costa-Perez, and A. Banchs. Rl-nsb: Reinforcement learning-based 5g network slice broker. IEEE/ACM Transactions on Networking, 27(4):1543–1557, 2019.

[19] R. S. Sutton and A. G. Barto. Reinforcement Learning: An Introduction. The MIT press, 2015.

[20] I. Szita and A. Lörincz. Factored value iteration converges. CoRR, abs/0801.2069, 2008.

[21] T. Tournaire, H. Castel-Taleb, and E. Hyon. Optimal control policies for resource allocation in the cloud: comparison between markov decision process and heuristic approaches. CoRR, abs/2104.14879, 2021.

[22] T. Tournaire, H. Castel-Taleb, and E. Hyon. Reinforcement learning with model-based approaches for dynamic resource allocation in a tandem queue. In 26th International Conference on Analytical and Stochastic Modelling Techniques and Applications, In press, 2021.