An Online Planning Agent to Optimize the Policy of Resources Management

Aditya Shrivastava[1] , Aksha Thakkar[1] ,Vipul Chudasama[1]

Department of Computer Science and Engineering,
Institute of Technology, Nirma University
{17bit014,17bit003,vipul.chudasama }@nirmauni.ac.in

Abstract. Reinforcement Learning based systems have received a lot of attention in various domains in recent years. In such domains, an autonomous agent which learns from environment to provide a solution. Resource scheduling is considered as research challenge where such autonomous agent optimize the solutions. This work is presented as an investigation on the effectiveness of various algorithms which drives actions associated with autonomous agent. We give a detailed contention between three differing algorithms–Q-learning, Dyna-Q and Deep-Q-Network, given the task of effectively allocating the resources in an online basis. Amongst the mentioned algorithms, the Q-learning and Deep-Q-Network, which are model free algorithms have remained in wide use for planning. However, this paper focuses on highlighting the effectiveness of lesser known model-based algorithm Dyna-Q. The experiment results show agent-based policy derived by the Dyna-Q algorithm provides optimize resource scheduling for current environment.

Keywords: Dyna-Q Framework · Resource Allocation · Reinforcement Learning · Model-based Learning

1 Introduction

Consummate management of resources has been imperative for operating high performance computing tasks. In cloud computing applications users make requests to the service providers to utilize multiple resources. Subsequently, systematic resource management has to be done to make efficient use of those resources. Scheduling tasks in cloud computing using traditional methods which are in use today have been meticulously tested and implemented [7, 9]. Resource management has also been essential to reduce energy consumption in data centers [8] and predicting relay selection [10]. However the policies in use at present are generally manually defined keeping in mind the nature and working of any particular environment. Automating these decisions to allocate resources without without any human governance for various applications is a difficult task.

Reinforcement learning is turning out be the most promising and optimistic field of research in the recent years. It pans out a very resolute and distinctive approach for making decisions in complex environments which seem implausible by other Machine Learning paradigms. An agent receives a reward on the basis of what actions it takes and learns what decisions to make by gaining experience from the defined environment. Reinforcement Learning has been used for many applications like a network traffic signal control system in a simulated environment [1]. Latest breakthroughs in game-playing agents in AlphaGo and Atari games have substantiated that Deep Reinforcement Learning is capable of solving complex decision making problems in complicated environments [16][14]. Reinforcement Learning has proved to outperform conventional approaches in Resource Management in computer clusters [13] and in cooling datacenters [4]. Other applications include automating robotics [12] , web system auto-configuration [2] and so on. These applications involve obscure and continuously changing environments which need adaptive and automated solutions.

Dyna-Q and Deep Q-Network are the extension of Q-Learning method in Reinforcement Learning. Dyna-Q is a simple yet vigorous algorithm which compounds Q-Learning and Q-planning. Here *planning* refers to the generation of simulated experiences other than real experience which is used to improve the efficiency of our model. This way of improving value functions and policies is referred to as *indirect reinforcement learning*. We can see some reasons that this method could enhance the existing approach :

1. In environments where we have to improve the policy using insubstantial experience Dyna-Q helps to attain results with better efficiency.
2. Dyna-Q speeds up the learning procedure and takes less number of iterations to achieve maximum amount of reward.
3. There are other variants of Dyna-Q algorithm like $Dyna - Q^+$ which has an added exploration bonus and it is found to perform better.

One of the basic examples illustrating the working of Dyna-Q is Dyna-Maze[17]. Dyna-Q has shown good results in applications like Dialogue Policy Learning [15] and path planning for robots [19]. On the other hand Deep Q-Network(DQN), a deep reinforcement learning approach uses a neural network instead of the Q-table to approximate the Q-value functions. DQN is used to scale the Q-Learning approach to complex environments i.e having large number of states and actions. DQN has become common in reinforcement learning applications as it is used in most of the models for game playing agents. These algorithms can help to get better results for the resource management problem as they provide better decision making techniques.

In the subsequent sections we discern the architecture of the above mentioned techniques, their application in a simple exemplary model and results. Section 2 shows the related work in this area which was used as an inspiration for this paper. In Section 3 we understand more about these algorithms. In Section 4, we dilineate the methodology of our experiments. In Section 5, we test them on sample data and observe results in Section 6, finally we ponder upon the results and conclude our hypotheses.

2 Related Work

Many industries and cloud computing applications require allocation of certain jobs without compromising the resource limitation. Reinforcement Learning was first used for such a problem in a space shuttle payload processing problem(SSPP) for NASA to minimize the the total duration of the final schedule [20]. The temporal difference methods used in this paper were observed to perform better than the standard heuristic approaches. This paper was the first to explore the potential of reinforcement learning algorithms to solve scheduling task. Our paper is particularly inspired from Hongzi Mao's paper [13] on Resource Management with Deep Reinforcement Learning. Noteworthy results were observed as a reinforcement learning model using deep neural networks performed better than the standard approaches like Shortest Job First and Tetris. The reinforcement learning model used in this paper was based on Monte Carlo methods. Findings by an IBM research team also provide support on feasibility of such methods[18]. This paper demonstrated some positive results of a new hybrid Reinforcement Learning method which combines strength of both reinforcement learning and model-based policies for resource valuation estimates. Also insights from other papers having similar outlook to this problem [3, 11] and papers on cluster scheduling [6] and task scheduling [5] provided motivation for this paper.

3 Reinforcement Learning

Here we first explain the general setting of a Reinforcement Learning problem and how it evaluates the problem in a basic Q-Learning setting. Then we shine some light on how indirect Reinforcement Learning methods like Dyna-Q will augment the Direct Reinforcement Learning methods. Lastly we will see how DQN replaces the Q-table in a Q-Learning problem.

3.1 Problem Setting

In a standard reinforcement learning problem there are two main elements, an active decision making agent and the environment. The agent interacts with the environment and tries to achieve a goal with the maximum reward possible. Figure 1 shows a labelled diagram of a general Reinforcement learning problem setting. Consider a sequence of discrete time steps t = 0,1,2,3,... At every time step t the agent arrives at a *state* in the environment, $s_t \ \epsilon \ S$ and based on the policy selects an action, $a_t \ \epsilon A(S_t)$ from the environment. Here S is the set of possible states and $A(S_t)$ is the set of actions available in state s_t . After completing the *action* the agent receives a reward $r_{t+1} \ \epsilon R$ where R is the set of rewards and then it moves on to the state s_{t+1} . The agents maps states to the probabilities of selecting the actions. This is known as the policy of the environment and it is generally denoted by π_t , where $\pi_t(a|s)$ is the probability of selecting action a if the state is s. Our ultimate goal is to choose the states which lead to obtain the maximum rewards. However for continuing tasks generally we discount the rewards to get the cumulative discounted return. This is represented as $\sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$, where $0 \leq \gamma \leq 1$. Here γ is the discount rate. Hence our goal would be to maximize the expected cumulative discounted reward. [17]

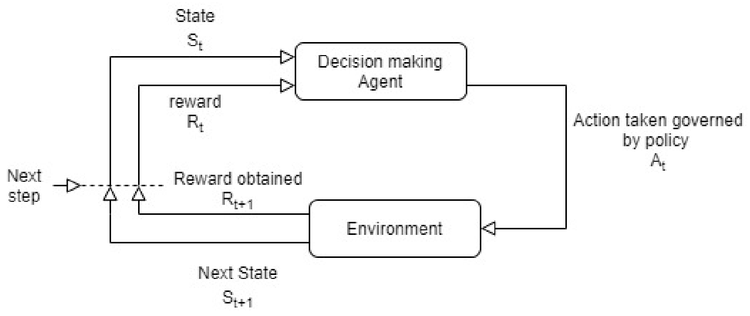


Fig. 1: General Reinforcement Learning problem setting

Q Learning is a model-free off policy reinforcement learning algorithm. Consequently it does not have a transition probability function associated with the Markov Decision Process and it does not have a policy function. Q-Learning performs actions to learn policies that maximise the total reward. The optimal value function or the q-function is approximated using experiences gained from the environment. So for a finite Markov Decision Process Q-Learning can find the optimal action-selection policy. A q-table is maintained to store the q-values obtained after every episode. This table would help to get the best action $a_t$to perform in a state s_t using the q-values. Let us see the update rule for $Q(s_t, a_t)$ i.e the q-value of action a_t at state s_t to approximate the q-function :

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)]$$

Here α is the learning rate which basically decides how much importance should be given to new value as compared to the old value and γ being the discounting rate to discount future rewards. These updates are iterated many times to get optimal q-values. Also for taking action we have two options : to explore the environment or to exploit by selecting the action with max value of those actions. This trade-off is decided by ϵ, where $0 \leq \epsilon \leq 1$. Once the agent gains enough experience the algorithm converges if the hyper parameters are selected carefully. Hence Q-learning handles the transitions and rewards which are stochastic.

3.2 Dyna-Q

Pseudocode of Dyna-Q algorithm

```
Initialize $Q(s, a)$ and model $M(s, a)$ for all the states $s \epsilon S$ and actions $a \epsilon A$ arbitrarily
Here S is the set of states, A is the set of actions at a particular state, Q is the set of Q-value pairs and M is the set of values of the model
while $Q$ is not converged(for a certain number of episodes) do
    #Direct Reinforcement Learning starts
    $s \leftarrow$ current(non terminal) state
    $a \leftarrow \epsilon - greedy(S, Q)$
    Get reward $r \leftarrow R(s, a)$ and new state $s' \epsilon S$ by executing action $a$
    $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_a Q(s', a) - Q(s, a)]$
    #Direct Reinforcement Learning ends
    #Model Learning starts
    $M(s, a) \leftarrow (s', r)$
    #Model Learning ends
    #Indirect Reinforcement Learning starts
    for $i$ in range $1, \ldots, n$ do
        $s \leftarrow$ random previously observed state
        $a \leftarrow$ random action previously taken in $s$
        $(s', r) \leftarrow M(s, a)$
        $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_a Q(s', a) - Q(s, a)]$
    #Indirect Reinforcement Learning ends
return $Q$
```

The Dyna-Q method is a combination of learning through real experiences and using a world model to generate hypothetical experience i.e planning. This combination of direct and indirect reinforcement learning along with model learning helps us to get better results. Figure 2 demonstrates the working of a generalized Dyna-Q architecture.

Along with direct updates using the update rule specified above in equation 1. The experience also help to build a model of the environment. Assuming a deterministic environment along with the q-values $Q(s, a)$ we also maintain a model of the environment $M(s, a)$. This model helps to generate the simulated experiences and search control selects the initial states and actions for the model to start with. The simulated experiences perform n number of planning steps where the Q-values are updated iteratively using the model. On performing this procedure on a number of episodes Dyna-Q converges faster than the basic Q-Learning problem. While this algorithm converges faster it should be taken care of that number of epochs are limited as this algorithm takes up a lot of computing power. The algorithm shown below would make this clear.

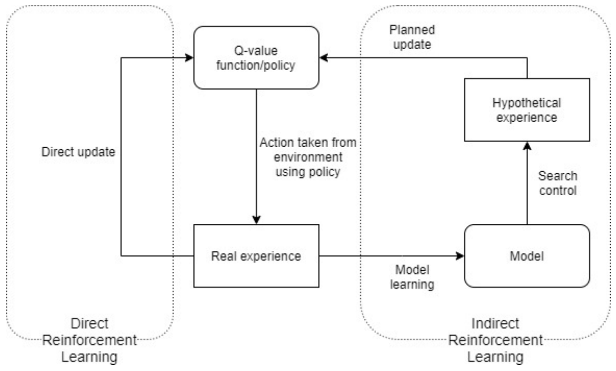


Fig. 2: General Reinforcement Learning problem setting

3.3 Deep Q-networks

Q-Learning is a powerful algorithm but what happens when the environment has a large number of states and actions? The q- values stored in the q-table will take a lot of memory space and the algorithm will take longer to run too. To solve this problem we use neural networks replacing the q-table to approximate the Q-function. The selection of actions is decided by the Q-network. The loss function for the neural network is defined as the mean squared error between the predicted and the target Q-values. The equation below shows the cost function used in Deep Q-networks.

$Cost = [Q(s,a;\theta) - (r + \gamma \max_a Q(s',a;\theta))]^2$

Here θ denotes the weights of the neural network. Also $Q(s,a;\theta)$ represents the predicted Q-value and $Q(s',a;\theta)$ represents the target Q-value. Using this equation we train the network to learn Q-values. As the target is unstable and variable in the Reinforcement Learning problem generally a separate network with similar architecture is used which estimates the target. This helps to stabilise the training. This is how the Q-learning problem is scaled using Deep Q-networks for large-scale applications.

4 Problem Description

In this section, we present the formulation of the problem of job scheduling as a reinforcement learning based on the algorithms delineated above. We show how to represent each these algorithms in a way suitable for Job Scheduling. Finally, we provide our approach and solution adhering to the apparatus prescribed above.

4.1 Proposed Model

Initially we account a cluster that can contain jobs demanding at most d different resource types, for instance CPU, memory etc. These jobs arrive and make a request for the resources in a sequential manner at different timestamps. The job scheduler chooses from a pool of k jobs. Here initially, the pool shall be empty and scheduler shall start to accommodate the jobs in the pool as they arrive until a total of k jobs are present in the pool. The rest of the jobs are queued according to their timestamps as they arrive. Once any particular job is completed, the scheduler scans and chooses one or more of the waiting jobs for scheduling depending upon the availability in the pool after each timestep. We assume here that each job is aware of the resources that it needs to utilize. Here, the resource indication is each job j is given by $r^j = (r_1^j, r_2^j, r_3^j, ..., r_d^j)$. For the convenience and evaluation purposes, note that scheduler allocates the job in a non-preemptive way, the duration of the job represented by T_j is known beforehand and we treat cluster as single collection of resources instead of discrete resources fragmented across the system. While these aspects form an important portion when considering the practical scenario, we allow to derogate them to investigate and capture insights about the essential elements and provides a non-trivial setting. This can help capture the essential elements of multi-resource scheduling and provide a non-trivial setting to study the effectiveness of RL methods in this domain.

4.2 Scenario

State-space First we begin by providing the description for the state that our scheduler shall encounter. Note that here, the RL agent shall be the job scheduler which carries out the selection-action of most seemingly optimal job and allocates the resources to it. Thus, in all, the state-space from the job scheduler perspective may include the following:

1. Current jobs with allocated resources.
2. The jobs in the pool waiting to be scheduled.

Here the jobs pool shall be initially represented by a vector comprised of job vectors each of length d–depicting values of each resources. Ideally, the vector containing vectors (no. of resources) can have indefinitely large number of vectors. However, as mentioned earlier it can become impractical for the agent to choose from such large number of resources. Therefore, the agent shall maintain only the first k jobs. These k jobs shall either be chosen from directly in case of the algorithms Q-learning and Dyna-Q and shall be given as an input to the neural network in the case of Deep-Q-Network.

Action-space After every time step, the agent is given the task of choosing the best job from the input space of k jobs. Thus, the full action space set of an agent shall be characterised by $\{\phi, 1, 2, ..., k\}$. Here, any element, for instance 3, means "schedule the 3^{rd} job from the pool." and that ϕ signifies void selection of jobs meaning no job shall be selected. The scheduler examines the state after every time-step and gets to choose from the input space. Once any job from the input space is selected, the scheduler stays frozen until the time one of the scheduled process gets finished.

Rewards We craft the rewards that can efficiently guide the agent to learn the optimal solution. In our experiments the rewards for allocating a particular job would be the weighted sum of the values of the stipulated resources. (Please refer 4.2 for more on how to calculate the weighted sum). Thus, for every state the agent shall be given the task to maximize the reward for which the optimal solution would be to select the resources that represent the highest priority in terms of highest rewards or weighted sum. Please note that the agent's job is bot to maximize the immediate reward but the cumulative reward from the start till the time that the jobs are allocated and the agent does not receive any reward for intermediate decisions during a time-step or for any void action chosen.

Algorithms It shall be impractical to give the vector representing the resources for a specific job as an input to one specific input neuron. Therefore, to counter this problem, we allow for a weighted sum of the resources to be computed. This means that we shall separately maintain a weight vector represented by,

$$W = \{w_1, w_2, w_3, ..., w_d\} \tag{1}$$

In this way every job shall obtain its weighted sum value that shall account for the final value of the form that can be presented in the action space of the agent. We describe the complete process as below:

$$y = W^\top X \tag{2}$$

$$y = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ \vdots \\ w_d \end{bmatrix} \begin{bmatrix} r_1^1 & r_2^1 & r_3^1 & ... & r_d^1 \\ r_1^2 & r_2^2 & r_3^2 & ... & r_d^2 \\ r_1^3 & r_2^3 & r_3^3 & ... & r_d^3 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ r_1^k & r_2^k & r_3^k & ... & r_d^k \end{bmatrix} \tag{3}$$

$$y = \begin{bmatrix} w_1 \cdot r_1^1 & w_2 \cdot r_2^1 & w_3 \cdot r_3^1 & ... & w_d \cdot r_d^1 \\ w_1 \cdot r_1^2 & w_2 \cdot r_2^2 & w_3 \cdot r_3^2 & ... & w_d \cdot r_d^2 \\ w_1 \cdot r_1^3 & w_2 \cdot r_2^3 & w_3 \cdot r_3^3 & ... & w_d \cdot r_d^3 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ w_1 \cdot r_1^k & w_2 \cdot r_2^k & w_3 \cdot r_3^k & ... & w_d \cdot r_d^k \end{bmatrix} \tag{4}$$

$$y = \begin{bmatrix} j^1 \\ j^2 \\ j^3 \\ \vdots \\ j^k \end{bmatrix} \tag{5}$$

Here, eqn. 2 describes the operation for eliciting weighted sum for all of the respective jobs. W here is a d dimension vector and therefore its transpose–$W^\top$ is of dimension $1 \times d$ and is multiplied with the jobs vector–X of dimension $d \times k$. Here d represent the types of resources and k represent the number of jobs. And finally we get a vector of length k representing the effective value of each of the jobs. And this is the entity that shall be given as an action space for the user.

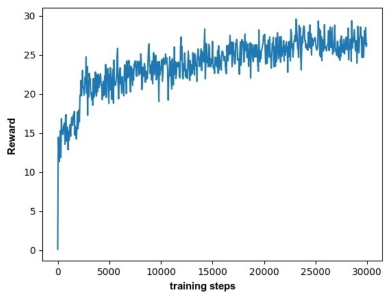
This pre-processed vector shall be an action space of three agents adopting three different respective algorithms whose contention we explore here– Q-learning, Dyna-Q and Deep-Q-Network.

5 Experiments & Discussion

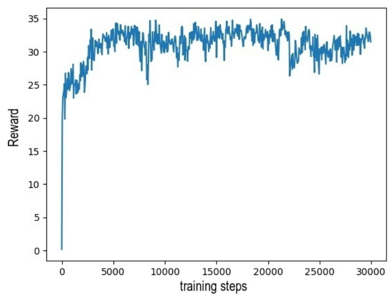
We simulate the environment discussed in 4 and train our RL agents using the three algorithms described in 3 namely–Q-learning, Dyna-Q and Deep-Q-Network. Amongst the three methods discussed, our primary advocacy lies with the recently proposed Dyna-Q method. The other two methods namely Q-Learning and Deep-Q network are the well established and experimented with methods and therefore we choose them for validation of the proposed Dyna-Q method. We run the experiments for 30,000 training steps and in this section we reflect on the behavior of our agents over the course of its learning to maximize its reward. It can be observed from the Fig. 3a that the agent learns gradually when trained using Q-learning procedure. This suggests that conventional model-free based learning demands significant amount of time to train well. It is only after 2500 epochs that the agent's learning really starts to bump up. The q-learning agent is seen to have consistently exploring and being affected by these exploring decisions. The Q-learning agent while exploring is faced by lesser rewards multiple times from 7500 to 12500 steps. However, it not until 20000 steps that the q-learning agent counters the highest reward and stays consistent since then. Thus for its application in job scheduling the Q-learning can induce skepticism because of its inability to learn the optimal solution quickly.

Next, we explored the Dyna-Q procedure under identical circumstances. The Dyna-Q method learns to maximize its rewards in its early phase itself. Here the agent takes an action for a few steps and while waiting for the reward and shifting to the next phase, it simulates the identical environment to perform virtual actions and meanwhile also learns from it. This enables the agent, as observed from the figures to learn to make the optimal choices approximately 10 times faster than its predecessor–Q-learning method. For instance, the optimal reward of 30 units is reached by a Q-learning agent at 25000th training step. Whereas, for the case of Dyna-Q, it reaches this mark at just around 3000 steps. Such high yielding capacity of the Dyna-Q algorithm can be attributed to its ability to simulate a virtual environment in parallel. Employing this algorithm instead of Q-learning algorithm can help in marginally overcoming the problem of latency.

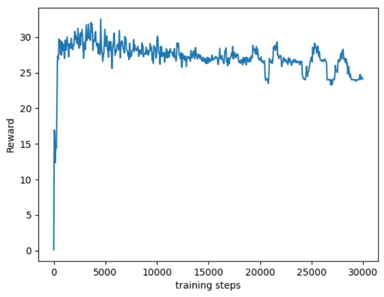
Lastly, we employed Deep-Q-Network for the allocation of resources. In this case, the policy of the algorithm was represented as neural network. The neural network shall take the input of the collection of resources as described in 4.2. And the neural network shall output the probability distribution over the action space and then the action with the highest probability shall be chosen and performed. And as shown in the figure 3c the Deep-Q-Network algorithm is the fastest one to learn. However, during the later stages, the algorithm's performance degrades. This means that the agent trained with Deep-Q-Network faces difficulty when it encounters unprecedented states. The figure 3d shows its cost to be consistently reducing and simultaneously the agent loses its optimal performance (fig. 3c). This behavior may introduce skepticism for employing the algorithms in real systems.



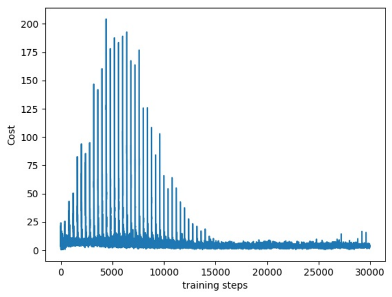
(a) Q-learning



(b) Dyna-Q



(c) Deep-Q-Network



(d) Deep-Q-Network cost

Fig. 3: The above figure shows the contention between the performances of the three approaches considered. The vertical axis represents reward and horizontal axis represents training steps. Since the third architecture opted for contention is Deep-Q-Network which consists of a neural network, we also include the graph showing improvement in learning cost of Deep-Q agent.

Thus, in general sense, considering the overall performance of agents with three different algorithms, we suggest that Dyna-Q is the most robust and reliable algorithm for the resource allocation technique. This technique can help in marginally overcoming the problem of both latency in reaching optimality that is faced by mainstream Q-learning algorithm and low efficacy displayed Deep-Q-Network when faced with high variance. Also the some works have known to derogate Deep-Q-Network to applied to real systems. This is because the computational complexity that it involves. Thus, considering various perspectives we suggest that Dyna-Q seems the most optimal algorithm for scheduling resources in cloud on an online basis. Also, this algorithm can be boosted even further, simulating the environment for virtual learning in an online fashion.

6 Conclusion

The need of managing the resources efficiently has been increasing with the new advancements in Cloud Computing and many other fields. However making it autonomous has remained a challenge. In this work we attempt to shed light on this challenging issue. The results achieved above provide conclusive evidence on how a Reinforcement Learning approach called Dyna-Q can prove to be a rather plausible alternative for resource management. This paper conducts experiments in a resource management problem setting where we introduce and compare the Reinforcement Learning methods. The Dyna-Q algorithm discussed here provides better latency and less difficulty in achieving optimal performance in a practical environment and offer a viable solution.

However, we spot a few further directions for improvements in this approach. The agent takes prolonged time in learning appropriate actions corresponding to the states. Dyna-Q faces the same problem. However, when it comes to Dyna-Q, we see significantly reduced times as compared to the other two approaches. Dyna-Q framework is fundamentally unique about its scalability in a way that it can distribute its learned progress across multiple systems and environments. However, its scalability within a single system yet remains questionable.

References

1. Arel, I., Liu, C., Urbanik, T., Kohls, A.G.: Reinforcement learning-based multi-agent system for network traffic signal control. IET Intelligent Transport Systems **4**(2), 128–135 (2010)
2. Bu, X., Rao, J., Xu, C.Z.: A reinforcement learning approach to online web systems auto-configuration. In: Proceedings of the 2009 29th IEEE International Conference on Distributed Computing Systems. p. 2–11. ICDCS ’09, IEEE Computer Society, USA (2009). https://doi.org/10.1109/ICDCS.2009.76, https://doi.org/10.1109/ICDCS.2009.76
3. Dutreilh, X., Kirgizov, S., Melekhova, O., Malenfant, J., Rivierre, N., Truck, I.: Using reinforcement learning for autonomic resource allocation in clouds: Towards a fully automated workflow (2011)
4. Evans, R., Gao, J.: Deepmind ai reduces google data centre cooling bill by 40%, https://deepmind.com/blog/article/deepmind-ai-reduces-google-data-centre-cooling-bill-40
5. Gawali, M.B., Shinde, S.K.: Task scheduling and resource allocation in cloud computing using a heuristic approach. J. Cloud Comput. **7**(1) (Dec 2018). https://doi.org/10.1186/s13677-018-0105-8, https://doi.org/10.1186/s13677-018-0105-8

6. Grandl, R., Ananthanarayanan, G., Kandula, S., Rao, S., Akella, A.: Multi-resource packing for cluster schedulers. SIGCOMM Comput. Commun. Rev. **44**(4), 455–466 (Aug 2014). https://doi.org/10.1145/2740070.2626334, https://doi.org/10.1145/2740070.2626334
7. Hameed, K., Ali, A., Jabbar, M., Junaid, M., Haider, A., Naqvi, M.: Resource management in operating systems-a survey of scheduling algorithms (09 2016)
8. Heller, B., Seetharaman, S., Mahadevan, P., Yiakoumis, Y., Sharma, P., Banerjee, S., McKeown, N.: Elastictree: Saving energy in data center networks. pp. 249–264 (01 2010)
9. Huang, Y.F., Chao, B.W.: A priority-based resource allocation strategy in distributed computing networks. Journal of Systems and Software **58**(3), 221 – 233 (2001). https://doi.org/https://doi.org/10.1016/S0164-1212(01)00040-1, http://www.sciencedirect.com/science/article/pii/S0164121201000401
10. Jiang, J., Das, R., Ananthanarayanan, G., Chou, P.A., Padmanabhan, V., Sekar, V., Dominique, E., Goliszewski, M., Kukoleca, D., Vafin, R., et al.: Via: Improving internet telephony call quality using predictive relay selection. In: Proceedings of the 2016 ACM SIGCOMM Conference. pp. 286–299 (2016)
11. Karthiban, K., Raj, J.S.: An efficient green computing fair resource allocation in cloud computing using modified deep reinforcement learning algorithm (2020)
12. Kober, J., Bagnell, J.A., Peters, J.: Reinforcement learning in robotics: A survey. The International Journal of Robotics Research **32**, 1238 – 1274 (2012)
13. Mao, H., Alizadeh, M., Menache, I., Kandula, S.: Resource management with deep reinforcement learning. In: Proceedings of the 15th ACM Workshop on Hot Topics in Networks. p. 50–56. HotNets '16, Association for Computing Machinery, New York, NY, USA (2016). https://doi.org/10.1145/3005745.3005750, https://doi.org/10.1145/3005745.3005750
14. Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M.: Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602 (2013)
15. Peng, B., Li, X., Gao, J., Liu, J., Wong, K.F.: Deep Dyna-Q: Integrating planning for task-completion dialogue policy learning. In: Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). pp. 2182–2192. Association for Computational Linguistics, Melbourne, Australia (Jul 2018). https://doi.org/10.18653/v1/P18-1203, https://www.aclweb.org/anthology/P18-1203
16. Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., Hassabis, D.: Mastering the game of Go with deep neural networks and tree search. Nature **529**(7587), 484–489 (jan 2016). https://doi.org/10.1038/nature16961
17. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. The MIT Press, second edn. (2018), http://incompleteideas.net/book/the-book-2nd.html
18. Tesauro, G., Jong, N.K., Das, R., Bennani, M.N.: A hybrid reinforcement learning approach to autonomic resource allocation. In: 2006 IEEE International Conference on Autonomic Computing. pp. 65–73 (2006)
19. Viet, H., An, S., Chung, T.: Extended dyna-q algorithm for path planning of mobile robots. J. Meas. Sci. Instrum. **2**(3), 283–287 (2011)
20. Zhang, W., Dietterich, T.G.: A reinforcement learning approach to job-shop scheduling. In: IJCAI. vol. 95, pp. 1114–1120. Citeseer (1995)