EasyChair Preprint

№ 6355

Continuous Self-Adaptation of Control Policies in Automatic Cloud Management

Wlodzimierz Funika, Paweł Koperek and Jacek Kitowski

August 23, 2021

Continuous Self-Adaptation of Control Policies in Automatic Cloud Management

Włodzimierz Funika[1[0000−0003−3321−7348]], Paweł Koperek[1[0000−0003−3613−2390]], and Jacek Kitowski[1,2[0000−0003−3902−8310]]

[1] AGH-UST, Faculty of Computer Science, Electronics and Telecommunication, Institute of Computer Science, al. Mickiewicza 30, 30-059, Kraków, Poland
[2] AGH, ACC CYFRONET AGH, ul. Nawojki 11, 30-950, Kraków, Poland

email:funika@agh.edu.pl, pkoperek@gmail.com, kito@agh.edu.pl

Abstract. Deep Reinforcement Learning has been recently a very active field of research. The policies generated with use of that class of training algorithms are flexible and thus have many practical applications. In this paper we present the results of our attempt to use the recent advancements in Reinforcement Learning to automate the management of resources in a compute cloud environment. We describe a new approach to self-adaptation of autonomous management, which uses a digital clone of the managed infrastructure to continuously update the control policy. We present the architecture of our system and discuss the results of evaluation which includes autonomous management of a sample application deployed to Amazon Web Services cloud. We also provide the details of training of the management policy using the Proximal Policy Optimization algorithm. Finally, we discuss the feasibility to extend the presented approach to further scenarios.

Keywords: Computing clouds · Autonomous control · Digital Twin · Deep Reinforcement Learning.

1 Introduction

In the last few years, computing clouds have gained wide-spread adoption. Almost every newly created software utilizes resources which are available through a cloud-like interface. On the one hand this approach allowed to greatly improve the development time, on the other hand it also posed a number of challenges. One of the more prominent ones is the optimization of costs, especially when working with Infrastructure-as-a-Service (*IaaS*) environments. Since the resources (Virtual Machines - VMs) are charged usually based on how long they are being used, in order to limit the costs one needs to reduce the usage time. Unfortunately, this is a non-trivial task, especially given that there might be special constraints imposed by Service Level Agreements (*SLAs*).

In the recent years we could also observe a lot of progress being made in the field of Reinforcement Learning (*RL*) [19]. Initially, the algorithms which

are part of that domain, were only perceived as applicable to relatively simple problems. It was assumed that the controlled environment could be observed with the use of only a few metrics and there could not be too many actions to execute. Fortunately, combining RL with the Deep Learning techniques allowed to mitigate those limitations and reach new state-of-the-art results [14,15,18]. The main advantage of the mentioned methods is the ability to learn through observing and interacting with an environment which is similar to or the same as the one the agent is going to operate in. Using such an approach allowed to achieve results surpassing the performance of humans.

There are also first attempts to utilize Deep Reinforcement Learning (*DRL*) in the context of autonomous cloud management. These systems share one common flaw: their policies are able to make good decisions only in situations, which they were exposed to in the prior training. Without external intervention there is no way to update the policy after deployment. One might argue that an obvious solution to this problem would be to continuously train the policy while it is in control of the cloud environment, in other words: use an *online* policy training algorithm. Unfortunately, there is one significant disadvantage to this approach. Due to the nature of the training process, the new versions of the policy might not make decisions as good as the current policy. Making constant changes introduces a risk that the update might trigger applying potentially disastrous changes into the managed environment. To avoid such a situation, the performance of a new version of the policy needs to be verified prior to its deployment. One way of doing this is to compare the reward achieved by the old and new policies within a tightly controlled environment. A good example is a simulation, where the conditions: time flow, workload, available resources are provided equally and the decisions coming from the policies are the only major difference. Another advantage of such an approach is that it introduces a mechanism which allows the policy to become closer and closer suited to the environment it controls. New information is constantly being added to the representation of the policy (e.g. in the case of DNN - to the neural network weights).

An approach, which also utilizes a simulated copy of the managed resources, called the *Digital Twin* or the *Virtual Twin* has been employed in industrial and manufacturing systems for over a decade [12]. In this paper we present an experimental monitoring and management system, which to the best of our knowledge, is a first attempt to apply the concept of a digital twin to cloud resources management. It is an extension of our previous research [5] which demonstrated how DRL techniques can be used to control cloud application's resources. It uses the newly acquired data to continuously re-train the control policy and then compare this with the currently used version. This allows the management system to respond to a potentially changing workload while addressing the issues described above. This paper's contribution includes a novel architecture of an autonomous management system which utilizes a continuous policy improvement loop, initial policy training procedure, implementation of the described concepts available as an Open Source project [6], experiments and analysis of their results.

The paper is organized as follows: in Section 2 we overview related work, Section 3 describes the system's architecture and Section 4 explains the policy training procedure. Section 5 discusses the design of the experiment, description of the environment it was executed in and evaluates the results of the experiment. Section 6 summarizes our research and outlines further work.

2 Related Work

Reinforcement Learning can be applied in the field of cloud resource autoscaling in various ways [8], e.g. to create a policy which changes the number of acquired resources (typically VMs) or a policy which assigns a computational task to a specific resource (typically VMs to physical servers).

In [16] authors aim to create a cloud resource scheduling framework, which uses the Deep Q-network (DQN) algorithm. The autonomous agent is assigning virtual machines, which execute computational tasks, to a set of physical servers. Its objective is to minimize both the submitted task execution timespan and the energy consumption of resources. The approach has been verified using a simulated experiment, in which the proposed approach has been compared to random, round robin and multi-objective particle swarm optimization allocation algorithms. The policy created using the DQN algorithm was able to find near-optimal allocation, what suggests that the presented approach can be considered as an efficient resource allocation and task scheduling strategy. A similar approach is used in [3]. In this case, however, the objective of the DQN-trained policy was to choose an assignment policy (e.g. first fit) for incoming VM placement requests. Authors performed a number of simulation experiments where they compared the proposed approach with traditional assignment heuristics. That analysis showed the effectiveness of the DRL-based approach, especially in the context of handling workloads with major fluctuations. In [20] a resource provisioning framework based on the concept of monitoring-analysis-planning-execution (*MAPE*) loop is introduced. It consists of two loops: the first one is responsible for provisioning resources from an IaaS provider and uses DRL techniques; the second loop is coordinating cloud services which use the provisioned resources. Using both loops allows to control the number of used VMs while reducing the waste caused by incorrectly predicting the specific task resource consumption. The approach has been verified using a simulated experiment which demonstrated its ability to increase utilization, decrease the total cost while avoiding SLA violations.

The mentioned papers suggest that autonomous control achieved by using DRL techniques can render good results. Unfortunately such conclusions are confirmed only by results of simulations. This raises a concern, whether the discussed approach can be applied to more complex, real-world infrastructures. In our previous work [5] we demonstrated how such a task could be addressed and presented a proof-of-concept of an autonomous resource provisioning system. That system used a policy, created by a DRL training algorithm, to control resources utilized by a sample application deployed to Amazon Web Services cloud

[1]. In this paper we are extending this approach. We introduce a simulated copy of the managed resources (a so called *digital twin*) which is used to continuously improve the initially deployed policy.

The idea of using a virtual clone of a physical object or system is not new. It has been first proposed in 2003 [10] and since then applied primarily to manufacturing processes, aviation and healthcare [2]. The digital copy can be a source of information for production optimization, predictive maintenance, cost optimization and physical resource management. To the best of our knowledge, the presented system is the first attempt to apply this technique together with DRL to cloud resources management. In this context, a set of simulated resources becomes the *digitial twin* of the application infrastructure deployed to a public cloud. Due to the fact that the source environment is also digital and can be examined through a set of well defined APIs, its replication is relatively easy. The simulated behavior of the managed resources can be made quite accurate as they are governed by complex, yet precise and deterministic rules. By using a simulation we can create a safe environment in which on the one hand the training process can be performed safely (a policy making disastrous changes would not be copied to the real environment) and in a short amount of time (e.g. thanks to a speed up in the time flow).

3 Digital Twin System Architecture

The architecture of the described system consists of two loops. The first one is the feedback loop of the subsystem which embeds the control policy in the real cloud environment. The second one is formed of the components used to continuously update the policy, including the simulated copy of the controlled resources (the *digital twin*). The components of both loops are shown in Fig. 1.

The *first feedback loop* starts with collecting measurements about the resources which take part in executing the jobs. Each of them is configured to start reporting relevant measurements as soon as the resource becomes online. The measurements often differ in their nature what influences how often their values are provided, e.g. the amount of free RAM and CPU usage is reported every 10 seconds while the virtual machine (VM) count - once per minute. To simplify the implementation of collecting of those raw measurements, we introduced the Graphite monitoring tool [9]. Graphite aggregates all the collected values into a single interval to create a consistent snapshot of the environment. This interval in our case is set to one minute.

Next, the measurements are passed to the SAMM monitoring and management system [7]. SAMM enables experimenting with new approaches to management automation. It allows to easily add support for new types of resources, relevant metrics, to integrate new algorithms and technologies, and to observe their impact on the observed system. In our use case, SAMM is used to combine other elements of the system together. First, it periodically polls measurements which portray the current state of the system (e.g. the average CPU usage in the computation cluster, amount of used memory etc). Then, SAMM aggregates the

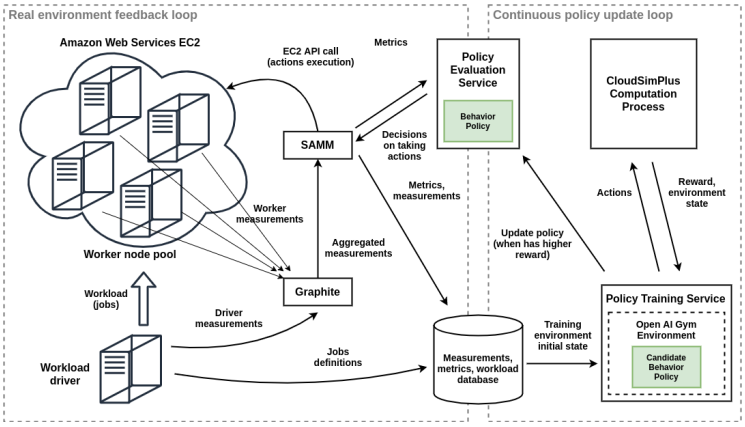

Fig. 1: Components of the real cloud environment under discussion. On the left side there is the environment-policy feedback loop. On the right side, the continuous policy update loop. Arrows denote interactions between the components.

measurements into metrics used by the decision policy. Finally, it communicates with the *Policy Evaluation Service*: provides the current state of the system in a form of metric values and retrieves decisions. The decisions are then executed through a cloud vendor API (e.g. Amazon Web Services API).

The *Policy Evaluation Service* provides decisions on how to change the allocation of resources based on the results of evaluation of the observed system state. The decisions are made according to the policy trained with the use of the PPO [18] algorithm. The results of the evaluation may include *starting a new small, medium or large VM* (deficient resources are used to handle the workload under the current system state), *removing resources - shutting down a small, medium, large VM* (excessive resources are used given the current state of the system), *doing nothing* (a proper amount of resources is allocated). One should remember that implementing the change is always subject to environment constraints. Not always it is possible to immediately execute an action. We might need to wait for a while because the system is in a *warm-up* or *cool-down* (a period of inactivity to allow to stabilize the metrics after the previous action has been executed), the previous request might still be being fulfilled, the request failed and needs to be retried in some time. In order to be able to train a policy which can cope with such limitations, the mentioned factors need to be involved in the simulation used for training.

For the described system we make a few assumptions about the workload under management:

- processing is organized into many independent tasks,
- the number of tasks which are yet to be executed can be monitored,
- the tasks which have been interrupted before their termination (e.g. in case the processing VMs are shutdown) are rescheduled,

- the tasks are considered idempotent, i.e. executing them multiple times does not change the end result,
- information about the currently executed tasks (e.g. schedule time, resources usage) needs to be available to the management system,
- resources administering the workload (e.g. accepting the input requests) are exempt from automatic management to prevent the workload from being accidentally terminated.

Fulfilling the monitoring requirements may require introducing *extensions* to the software which generates the workloads and *instrumenting the resources* which are used to create tasks. In our case, the workload driver has been enhanced with the capability to store relevant workload information in a database.

The *second loop* highlighted in Fig. 1 is responsible for continuously updating the policy. This part of the autonomous management system is responsible for ensuring that the decisions implemented in the real environment are made by the policy which has been retrained with the use of the most recent data. This loop starts with the *Policy Evaluation Service* which hosts the currently used *Behavior Policy*. Actions taken by the policy are implemented in the cloud environment and thus are observable in the measurements and metrics (e.g. in the number of used CPU cores) recorded in the database. The content of the database is then used by the *Policy Training Service*. It periodically retrieves a set of the most recently processed tasks and the specification of the resources which were available when those tasks were being executed. This allows the service to configure the simulated environment in which a *Candidate Behavior Policy* is being trained. Once the training is over, the driver compares the reward from the simulation and that from the real environment. If the former reward is greater, the candidate policy replaces the currently used one. The simulator has been implemented following the results of our prior research [4].

The policy is trained according to the procedure described in the next section.

4 Policy Training

The policy has been implemented as a neural network. We experimented with different architectures of the neural network used as a decision policy. The best results have been obtained with the use of the *long-short term memory (LSTM)* [11] architecture. LSTM is a type of recurrent neural network, which means it passes the output of a layer back to its input. This makes it well-suited to process data in form of sequences, as it has access to the previously made decisions. A basic building block in the LSTM networks is usually described as a *cell*. In our case the network consisted of 128 cells. For training we have used the *Proximal Policy Optimization* (PPO) [18] algorithm with parameters as shown in Table 1.

To avoid the cold start problem, we have trained the initial version of the policy in the above described simulator. As the workload we have used a set of 1551 jobs. The jobs have been organized into 21 batches (10 batches of 100 and 11 batches of 50 jobs) submitted at 8 minute intervals. Every job requested 360

Parameter name	**Parameter value**	**Parameter name**	**Parameter value**
Value function coefficient	0.0005	Lambda	0.97
Gamma	0.99	Training timesteps	250000
Clipping factor	0.2	Learning rate	0.0003
Batch size	250	Simulator speedup	60

Table 1: Policy training process - parameters.

seconds on a single CPU core. The single job has been added 30 minutes after the final batch. This ensured that there would always be a cool-down period of time at the end. We chose such a workload because on the one hand it was small enough so that it allowed to conduct a full simulation in a short amount of time and on the other hand it was comprehensive enough to allow the policy to gather some valuable experience about batch processing applications. In our experiment we present a scenario where such an application is being automatically managed. The discussed approach is not limited to batch processing, though. If a different type of workload needs to be controlled, the policy can be adjusted by training it with the use of a different workload.

The agent objective was defined as minimizing the overall cost of resources, which has been expressed as maximizing the following reward function:

$$F(T_S, T_M, T_L, T_Q) = -\sum_{x \in V} (T_x \cdot C_x) - T_Q \cdot C_Q \tag{1}$$

where:

- $F(T_S, T_M, T_L, T_Q)$ is the negative cost of resources used for processing,
- V denotes a set of possible VM sizes. In our experiments it includes S, M or L which represent *small*, *medium* or *large* VMs, accordingly,
- T_x denote the number of hours of running VMs of size x,
- C_x is the hourly cost of running a machine of size x. In our case $C_S = \$0.2$, $C_M = \$0.4$ and $C_L = \$0.8$,
- T_Q – the hours spent by tasks waiting for execution.
- C_Q – the hourly penalty for missing SLA targers when a task is waiting for execution. The cost 0.036 is accrued for every second of a delay between submitting task for execution and actual execution. There were no limitations on the waiting time or the waiting queue size.

5 Experiment

To evaluate our approach to the autonomous cloud resources control, we have conducted an experiment with the use of resources of a publicly available cloud environment. The overall objective was to quantify the impact of the continuous training loop on the management process. First, we ran the sample application 10 times and managed it using the *initial* version of control policy. Next, we

ran the same sample application 10 more times but managed it with the use of a policy which was being continuously updated. Afterwards, we compared the average resource costs and computation times. Finally, we analyzed how the update process influenced the decisions made by the policy.

As a sample workload, we have used the *pytorch-dnn-evolution* tool [17]. This is a tool which attempts to discover an optimal structure of a Deep Neural Network (*DNN*) to solve a given problem (e.g. categorize images in a given set) using a co-evolutionary algorithm. In our setup, the evolution process was configured to search an optimal DNN architecture for recognizing the handwritten digits from the MNIST dataset [13]. Using *pytorch-dnn-evolution* also has its drawbacks. The workload is CPU-intensive and very irregular. The number of evaluated individuals can greatly change in subsequent evolution iterations, what makes it hard to choose the proper amount of resources. On the other hand, we have verified that such a workload met all the conditions outlined in Section 4, which enabled using a dynamic scaling approach.

The experiment was carried out with the use of the Amazon Web Services [1] infrastructure. The sample application has been using Virtual Machines (VMs) of three types: *large* (2 core CPU and 8 GB of RAM), *xlarge* (4 core CPU and 16 GB of RAM) and *2xlarge* (8 core CPU and 32 GB of RAM). Each run started with 1 virtual machine of each type already provisioned and ran until all the scheduled tasks were completed. We did not allow the autonomous management policy to remove all VMs of a given type to avoid situations in which the progress would have stalled completely. This would force us to predict whether the policy is going to recover from such a state, which is a variant of the halting problem. Shutting down all the virtual machines is also undesirable in a production environment, therefore we have decided to exclude this possibility from our tests. It is worth noting, however, that it is technically possible to configure the presented system to allow the disposing of all of the provisioned resources. All VMs were running in the same region (US North Virginia) and the same availability zone to avoid introducing any additional network latency. The components SAMM, Graphite and the workload driver have been running on a separate VM.

Workload run	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Resources cost, initial policy (USD)	7.86	8.10	8.07	7.82	8.43	8.15	8.29	8.04	8.25	7.41
Resources cost, policy updates (USD)	7.35	6.63	7.04	6.62	6.64	6.70	7.05	6.63	6.57	6.89
Workload time, initial policy (min.)	322	316	282	308	257	249	235	260	248	345
Workload time, policy updates (min.)	344	300	295	272	272	312	300	315	275	311

Table 2: Raw measurements of subsequent workload runs.

Table 2 presents raw observations of the total resources cost and the time required to process all of the jobs for a given workload run. When the policy remained unchanged, the average cost of resources was equal to 8.04 USD (stan-

dard deviation of 0.29). That value decreased to 6.81 USD (standard deviation of 0.26) when the policy updates were activated. That can be interpreted as a 15.3% cost reduction. We have attempted to confirm that result with the use of the one-tail t-Student test, however we noticed that one of the value sets did not meet the near-normal criterion according to Shapiro-Wilk test. However, the observed cost averages show a strong difference while manifesting their low standard deviations. We reason that updating the control policy while a workload is being executed, rendered superior results.

It is also worth noting that the lowering of the resource cost seemed to increase the workload time (on average by 6.17%, from 282.20 to 299.6 minutes). The dynamically changed policy on the average showed slower overall execution, which shows that the policy traded off the execution time for the desirable cost reduction. We have analyzed a number of factors, which potentially might have affected the execution time, to confirm that the observed cost reduction resulted from introducing the changes in control policy.

- **The number of the fully executed jobs** (not interrupted by a VM termination) within a workload run. In both cases those numbers were very similar and thus would not affect the results.
- **Average computation time required for a single job.** We have noticed that the average amount of time required to finish a single job has been shorter by 0.98 seconds in the case of continuous policy updates. That can be caused by the workload driver generating jobs which require less computations to finalize. When considered in the context of a whole workload run, that factor could account for a reduction in the total used resource time by 184.69 minutes or in other words a reduction of $0.13 of monetary costs. Given that the difference between the averages of raw observations in the two considered scenarios (with continuous policy updates and without them) is equal to $1.23, we can estimate that at least $1.1 of the cost reduction can be attributed to the continuous policy updates. This allows to sustain the claim that policy updates reduce the overall monetary cost.
- **Environment factors.** Factors like network latency, VM start-up time, etc. had the same impact on both approaches (using a static policy and using that with continuous updates). These factors' effect has been reduced by running each variant of the policy multiple times. We also assume that given the small differences in the total used resource time measurements (e.g. 11221 minutes on average with a standard deviation of 403.98 minutes for runs without policy updates) the impact of the environment factors on the results of our experiment is very limited.

The summarized results of the conducted experiment are shown in Table 3.

To demonstrate the effects of continuous re-training and updating the policy, in Fig. 2 we present a single run of the *initial policy* versus a run of one of the policy versions obtained after a few iterations of the continuous training. The initial policy seems to focus on multiple changes to the count of small VMs. The *second* policy is more aggressive in the resource allocation: it launches multiple

Measured value	Without updates	With updates
Average resource cost (USD)	8.04 ($\sigma = 0.29$)	**6.81** ($\sigma = 0.26$)
Average workload time (min.)	**282.20** ($\sigma = 37.95$)	299.60 ($\sigma = 22.71$)
Average used resource time (min.)	11221	**9504**
Average single job time (seconds)	41.87	40.77
Average number of fully executed jobs	11422	11424

Table 3: Results of managing the sample application with and without the continuous policy updates. All values are averages over 10 runs.

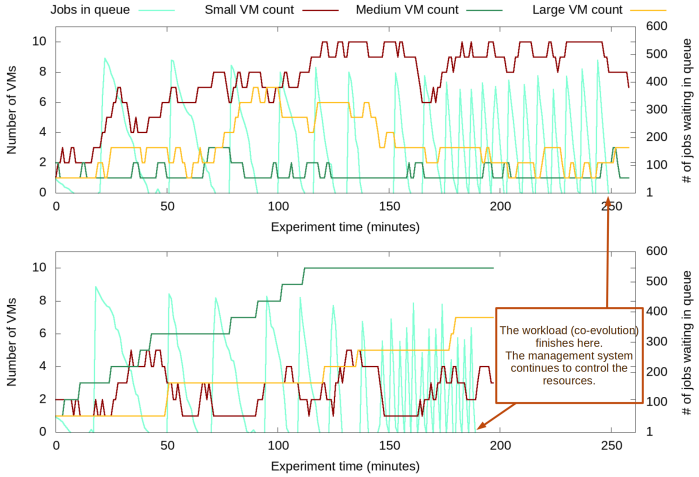

Fig. 2: Number of VMs running while the environment was being managed automatically. Top: initial policy, bottom: updated policy.

medium and large VMs, however the changes in VM counts seem to be less frequent. This results in a shorter total calculation time (189 minutes instead of 249 minutes in the top chart). The number of the executed jobs and the average job execution times are very similar in both cases: 11424 and 41.76 for the initial policy vs. 11389 jobs and 41.34 seconds for the updated policy. This allows to conclude that the observed differences were primarily driven by the change in policy. As presented in the example above, the continuous retraining process is capable of introducing the versions of policy, which manifest significantly different behaviors, what translates to an optimization of costs.

6 Conclusions and Further Work

In this paper we have presented the use of the *digital twin* approach for the autonomous management of cloud resources. We created a digital, simulated version of an existing environment and used it to reduce the monetary cost of

running it. We explained the architecture of a novel management system based on that idea and discussed its implementation which is based on the SAMM monitoring and management system. Finally, we have conducted an experiment to verify the presented approach, which empirically demonstrated the benefits of the used management method. We were able to reduce the average cost of resources from $8.04 to $6.83 (by 15.3%). The management policy was being updated using the new information coming from the managed environment, which allowed to respond to the situations to which it not been exposed before.

The continuous policy update loop proved to be an effective way of dynamically adjusting the management policy. At the same time this approach also has its disadvantages. Training an RL policy requires quite a significant amount of time. This means that, depending on the pace of changes to the actual workload, the update procedure might not be able to respond fast enough to changes in the workload or environment. Extending the management system with the continuous training loop increases its complexity and adds more parameters that need to be tuned. These parameters need to be tuned very carefully, otherwise one risks creating a policy which e.g. ignores historical data and only focuses on the most recent observations.

Our on-going work is focused on extending the presented approach. We plan to investigate the influence of different variants of the continuous training setup on the performance of the policy. We are working on introducing a parallelism of simulation, which would support more frequent re-training. We believe that neural network models which are used as the control policy can be further optimized, e.g. by using a bi-directional LSTM layer.

Acknowledgements. The research presented in this paper was supported by the funds assigned to AGH University of Science and Technology by the Polish Ministry of Education and Science. The experiments have been carried out on the PL-Grid infrastructure resources of ACC Cyfronet AGH and on the Amazon Web Services Elastic Compute Cloud.

References

1. Amazon Web Services Elastic Compute Cloud. `https://aws.amazon.com/ec2/` (2020), accessed: 2020-11-30
2. Barricelli, B.R., Casiraghi, E., Fogli, D.: A survey on digital twin: Definitions, characteristics, applications, and design implications. IEEE Access **7**, 167653–167671 (2019). https://doi.org/10.1109/ACCESS.2019.2953499, `https://doi.org/10.1109/ACCESS.2019.2953499`
3. Caviglione, L., Gaggero, M., Paolucci, M., Ronco, R.: Deep reinforcement learning for multi-objective placement of virtual machines in cloud datacenters. Soft Computing (2020). https://doi.org/10.1007/s00500-020-05462-x, `https://doi.org/10.1007/s00500-020-05462-x`
4. Funika, W., Koperek, P.: Evaluating the use of policy gradient optimization approach for automatic cloud resource provisioning. In: Wyrzykowski, R., Deelman, E., Dongarra, J., Karczewski, K. (eds.) Parallel Processing and Applied Mathematics. pp. 467–478. LNCS 12043, Springer International Publishing (2020)

5. Funika, W., Koperek, P., Kitowski, J.: Automatic management of cloud applications with use of proximal policy optimization. In: Krzhizhanovskaya, V.V., Závodszky, G., Lees, M.H., Dongarra, J.J., Sloot, P.M.A., Brissos, S., Teixeira, J. (eds.) Computational Science – ICCS 2020. pp. 73–87. Springer International Publishing, Cham (2020)
6. Funika, W., Koperek, P.: Trainloop driver. `https://gitlab.com/pkoperek/trainloop-driver` (2020), accessed: 2021-04-30
7. Funika, W., Kupisz, M., Koperek, P.: Towards autonomic semantic-based management of distributed applications. Computer Science **11**(0), pp. 51–64 (2010)
8. Garí, Y., Monge, D.A., Pacini, E., Mateos, C., Garino, C.G.: Reinforcement learning-based application autoscaling in the cloud: A survey (2020)
9. Graphite Project. `https://graphiteapp.org/` (2011), accessed: 2020-11-28
10. Grieves, M.: Digital twin: manufacturing excellence through virtual factory replication. White paper **1**, 1–7 (2014)
11. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Computation **9**(8), 1735–1780 (1997)
12. Jones, D., Snider, C., Nassehi, A., Yon, J., Hicks, B.: Characterising the digital twin: A systematic literature review. CIRP Journal of Manufacturing Science and Technology **29**, 36–52 (2020). https://doi.org/https://doi.org/10.1016/j.cirpj.2020.02.002, `https://www.sciencedirect.com/science/article/pii/S1755581720300110`
13. LeCun, Y., Cortes, C.: MNIST handwritten digit database. http://yann.lecun.com/exdb/mnist/ (2010), `http://yann.lecun.com/exdb/mnist/`
14. Mnih, V., Badia, A.P., Mirza, M., Graves, A., Harley, T., Lillicrap, T.P., Silver, D., Kavukcuoglu, K.: Asynchronous methods for deep reinforcement learning. In: Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48. p. 1928–1937. ICML'16, JMLR.org (2016)
15. Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M.: Playing atari with deep reinforcement learning (2013)
16. Peng, Z., Lin, J., Cui, D., Li, Q., He, J.: A multi-objective trade-off framework for cloud resource scheduling based on the deep q-network algorithm. Clust. Comput. **23**(4), 2753–2767 (2020). https://doi.org/10.1007/s10586-019-03042-9, `https://doi.org/10.1007/s10586-019-03042-9`
17. PyTorch DNN Evolution. `https://gitlab.com/pkoperek/pytorch-dnn-evolution` (2018), accessed: 2020-12-01
18. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms. CoRR **abs/1707.06347** (2017), `http://arxiv.org/abs/1707.06347`
19. Sutton, R.S.: Temporal Credit Assignment in Reinforcement Learning. Ph.D. thesis, University of Massachusetts Amherst (1984)
20. Zong, Q., Zheng, X., Wei, Y., Sun, H.: A deep reinforcement learning based resource autonomic provisioning approach for cloud services. In: Gao, H., Wang, X., Iqbal, M., Yin, Y., Yin, J., Gu, N. (eds.) Collaborative Computing: Networking, Applications and Worksharing. pp. 132–153. Springer International Publishing, Cham (2021)