Adv Syst Sci Appl 2018; **4**:1–12
Published online at http://ijassa.ipu.ru/index.php/ijassa/article/view/660

Tuning hybrid distributed storage system digital twins by Reinforcement Learning

Vladislav Belavin[1*], Kenenbek Arzymatov[1], Maksim Karpov[1], Andrey Nevolin[2]
Andrey Sapronov[1,3], Andrey Ustyuzhanin[1,4]

[1]*National Research University Higher School of Economics, Myasnitskaya 20, 101000, Moscow, Russia*
[2]*YADRO, Rochdelskaya 15/13, 123022, Moscow, Russia*
[3]*Joint Institute for Nuclear Research, Joliot-Curie 6, Dubna, Moscow Region, 141980, Russia*
[4]*Moscow Institute of Physics and Technology, Institutskiy per. 9, Dolgoprudny, Moscow Reg., 141700, Russia*

Received November 11, 2018; Revised December 12, 2018; Published December 31, 2018

Abstract: In this paper we develop a concept of data-driven control of distributed storage systems digital twin. The design of the digital twin is supported by an optimal control strategy created by Reinforcement Learning technique. In the proposed approach we consider a combination of a trained neural network that tunes the parameters of a discrete event-driven storage system simulator to gain the best resemblance to the actual system. The proposed method has several benefits compared to conventional approaches providing trade-offs between the simplicity of customization, the ability to infer nontrivial patterns from the real system represented by the simulation model and interpretable behavior. We consider different optimization metrics and demonstrate the viability of the approach using a toy training system until physical system becomes available.

Keywords: Storage area network, digital twin, simulation, machine learning, reinforcement learning

1. INTRODUCTION

Digital twins [1, 2] are used to simulate a behavior of complex systems. The distributed storage systems are built of multiple components, many of which have measurable parameters altogether providing valuable diagnostic information. The task of storage system simulation converges to building its digital twin with a highly accurate description of components behavior and its interaction. Such approach allows for prediction of the components failure and studying the failure outcomes.

In this paper we consider the problem of building a credible digital twin of the distributed storage system on the assumption that we have a model-based simulator and data from the real system. It is also assumed that the simulator has a set of parameters that can be changed during the runtime thus affecting following simulation states. To solve this problem, we propose an approach that combines conventional computer simulation with machine learning. The first one is usually based on a mathematical model devised by an expert and the second one infers dependencies from the observed data. The control over these parameters is delegated to a neural network trained on real data using Reinforcement Learning (RL) techniques. In the case of distributed storage systems the following use case scenario can be applied. The real-time prediction of possible failures of the real system with the digital twin which is continuously provided with information from the real system: load and environment variables (temperature, pressure, etc.) collected from sensors.

One of the limitations of the simulators based on domain knowledge is the rigidity of the underlying mathematical model, resulting in a limited ability to reproduce unexpected patterns that might be observed in real data. Such inability to imitate real system behavior implies the necessity for a revision of the mathematical model over and over again. This cycle of trial and error can take a lot of time and human resources. Besides, simulators are usually ignorant of the appearance of new data, i.e. cannot adapt to newly observed patterns in the data due to, for example, the degeneracy of some components or/and changes in external conditions. On the other hand, simulation approaches in machine learning, for instance, generative adversarial networks [3], require only data to reproduce behavior of the real system. The indisputable advantage of the GAN approach is its simplicity. However at least two significant problems are arising. Firstly, without any underlying model, it is hard to interpret the neural network[4]. Secondly, it might be inaccurate in operating regions that have not been presented in the dataset.

We can achieve a synergy between two inherently different methods by loosening constraints on the simulator and delegating the control over parameters to the neural network. In this case, the simplification of the simulator model provides more freedom for the neural network, simultaneously regularizing it and making interpretation of the hybrid model much more manageable than for the single neural network.

Our approach has following key differences in comparison with existing solutions:

1. a novel idea of recovery of the functional dependency of control parameters, instead of point estimation;
2. a fusion of the discrete event simulator with the neural network;
3. the usage deep Reinforcement Learning for a such task;

In this paper, we test the viability of such an approach.

2. RELATED WORKS

The task of storage area network simulation has been approached in several publications. The majority of works are focused on the discrete event simulation methods. The researches CODES project uses a highly parallel simulation to explore the design of exascale storage architectures and distributed data-intensive science facilities [5]. The general purpose network simulator ns-3 [6] allows a detailed modelling of a storage system from the view of interconnected nodes with different functionality. Another general approach to network-like structure simulation is the OMNeT++ framework [7]. Another work presents a SANSim tool [8], a simulation compliant with the fiber channel technology often used in contemporary SAN architectures. More simulation methods descriptions and studies dedicated to SAN system modelling can be found in [9, 10, 11, 12]. From this overview we can see that the most popular approach to the storage area network modelling is the deterministic method based on discrete event sequences. Below we going to discuss an extension to this method by coupling it with the dynamic parameter tuning by means of a neural network trained with real data sample.

Initially, the topic of simulation parameters optimization using Reinforcement Learning was discussed in [13]. In this work, the discrete-valued parameters are optimized to construct a matrix of optimal actions for each state. Our approach features a solution for real-valued parameters with the usage of deep Reinforcement Learning which primarily became possible due to advances in GPU computing technology. Our crucial difference from the mentioned research is the derivation of parameter dynamics during the whole simulation cycle instead of just optimal initial parameters.

Another way to find optimal parameters in the presence of real data is a validation of the simulator for each set of parameters output[14, 15]. In this setup for each set of parameters for simulated data and real data the hypothesis is constructed:

$$\mathbb{H}_0 : \text{ model output is the same as system output}$$

$$\text{vs}$$

$$\mathbb{H}_1 : \text{ model output is not the same as system output}$$

The choice of an appropriate statistic for outputs comparison as well as the selection of a statistical test depends on a particular problem. However, this methodology can only give an estimation of a single parameter point.

3. SIMULATOR

3.1. Simulated system

TATLIN (Figure 3.1) is a PCIe-centric distributed storage system designed by the YADRO Company [16]. It consists of hardware computational tools and a special software designed for storing and transmitting large amounts of data.

Due to the unified access protocol(Hybrid Unified Storage), this system supports a wide range of disk interfaces: NVMe/SAS SSD and SAS/NL-SAS/SATA HDD.

The platform consists of three major building blocks:

- The PCIe fabric controller is a PCI Express bus that provides interconnection between storage controllers and storage media;
- The storage controllers, the computer servers that provide access to storage media to end users;
- The drive enclosures generally known as JBODs (just a bunch of disks).

To ensure data integrity in TATLIN, the YADRO uses customizable data protection technology based on Reed-Solomon codes with minimal redundancy instead of conventional RAID. However, writing data on disks using this technology requires a lot of computational resources and can take considerable amount of time. To meet this problem and reduce CPU load the YADRO proposed a concept of hardware deduplication acceleration with non-volatile cache (NVRAM) built into the storage controllers.

3.2. Description of the simulator

The GoTatlin framework (Figure 3.2) is designed to simulate the TATLIN distributed storage system. It has been implemented using the Go programming language [17].

The storage system simulation is based on a discrete-event simulation model where the system's evolution in time is presented by a discrete sequence of changes in its state. Each change in the system state occurs at a distinct point of time due to some triggering event.

3.3. Output data of the simulator

We construct a vector of features from the output of the simulator. The features are summarized below.

- storage controllers features:
  - traffic $[\mathrm{GBs}^{-1}]$;
  - load $[\%]$;
- storage media features:
  - used space $[\mathrm{GB}]$;
  - average read/write speed $[\mathrm{GBs}^{-1}]$;
- general TATLIN features:

Fig. 3.1. DSS TATLIN.

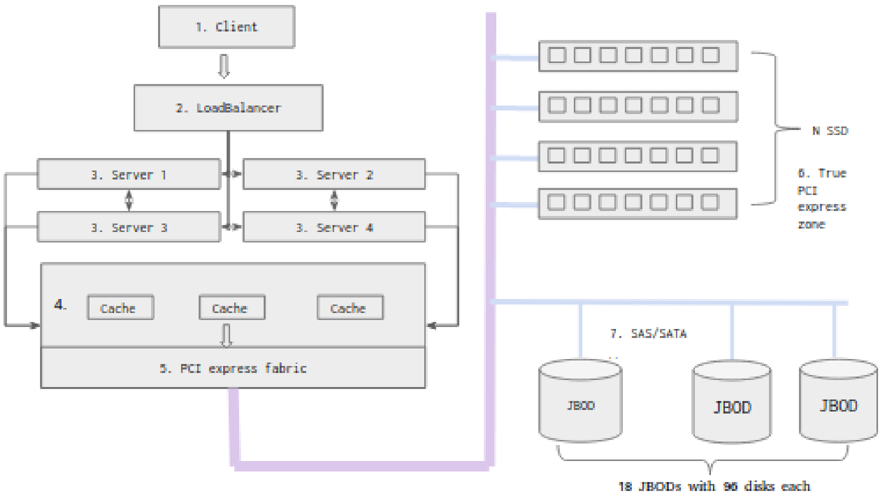


Fig. 3.2. GoTatlin architecture view.

- amount of data transferred in read/write mode [GB];
- number of read/write requests per unit of time $[\mathrm{ms}^{-1}]$;
- read/write requests response time [ms];
- read/write requests processing time [ms];

3.4. Control parameters

A full list of the control parameters of the simulator is the following:

- storage media:
  - *read* – read speed $[\mathrm{GBs}^{-1}]$;
  - *write* – write speed $[\mathrm{GBs}^{-1}]$;
- drive enclosures:
  - *speed* – computational power [GFLOPS];
- storage controllers:
  - *speed* – computational power [GFLOPS];
- PCIe fabric controller:

  - *speed* – computational power [GFLOPS];
- physical network links:
  - *bandwidth* [GBs^{-1}];
  - *latency* [ms^{-1}].

As one might imagine, the real system should have much more parameters, which represent the system's behaviour (L1/L2 cache sizes, RAM parameters, etc.), than the simulated one has. But we intentionally reduced the number of parameters implying that these variables represent most 'effectively' the real system. On the one hand, changing those parameters helps to relax rigidity of the simulator, and on the other hand, these effective parameters can't be derived from the physical system and should be calibrated by the algorithm explained in the following chapter. In the rest of the paper, we control those effective parameters unless stated otherwise.

4. CONTROL SYSTEM

The core part of the control system is a neural network which analyses the output of the simulator at the time t and generates a new set of control parameters that will be used for simulation until next moment of time $t + \Delta t$. Δt is an external parameter of the simulator fixed a priori.

It should be noted that Δt is a hyper-parameter of the simulation control system and can be chosen arbitrarily. We set Δt to 3s a priori because in the real TATLIN we are expecting the delay between successive log outputs to be approximately $1 - 10$s.

General outlines of the simulation and training pipelines are presented at the Figures 4.3a and 4.3b respectively.

4.1. Reinforcement Learning

In Reinforcement Learning [18] the following terminology is commonly used:

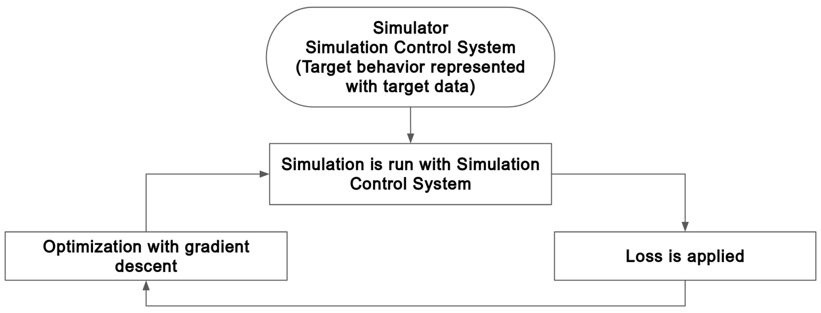
- environment (s_t) is an observed system (in our case, the system is a simulator);
- state is a current status of the simulation represented by output data;
- goal is a desired state of the simulation;
- agent is a system that takes actions (a_t) in the environment and changes its state;
- reward (r_t) is a scalar value that shows how close is our system's state to the goal state.

In our case, goal state of the simulator is a state in which its output fully resembles the real storage system's output. We will discuss the construction of the reward function in more details in Section 4.2.

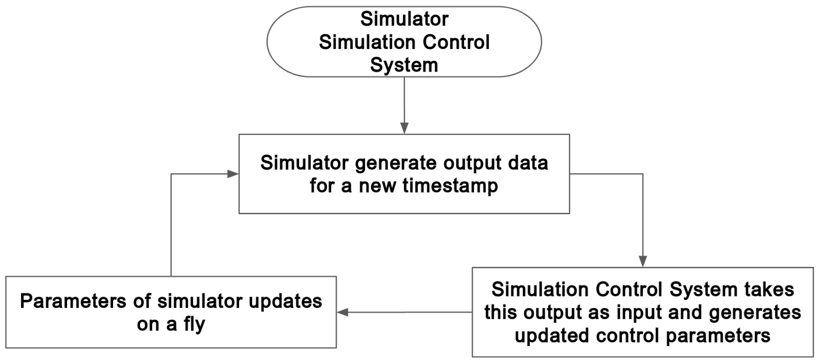
The most common approach is to assume that the environment is modelled as a Markov decision process. In this case our model can be described with 4-tuples $(S, A, P(\cdot,\cdot, R(\cdot,\cdot)))$, where

- $S = \{s_i\}_{i=0}^{N}$ is a set of all possible states of the simulator;
- $A = \{a_i\}_{i=0}^{K}$ is a set of all possible actions of the agent;
- $P(s_{t+1}|s_t, a_t)$ is a probability of transition from state s_t at time t to state s_{t+1} at the next moment of time $t + 1$ under action a_t;
- $r(s_{t+1}, s_t) = r(s_t, a_t)$ is a reward after transition from state s_t to state s_{t+1};

The most successful approach that use markovian assumption about environment is the so-called Q-learning [19] and its successor in deep learning domain Deep Q Network (DQN) [20]. In DQN method the neural network generates probability distribution over all possible actions in the current state of environment $\pi_\theta(a|s)$ and the next action is sampled from it ($a_t \sim \pi_\theta(s)$).


Simulator
Simulation Control System
(Target behavior represented with target data)
Simulation is run with Simulation Control System
Optimization with gradient descent
Loss is applied

(a) Proposed Simulation Control System pipeline.


Simulator
Simulation Control System
Simulator generate output data for a new timestamp
Parameters of simulator updates on a fly
Simulation Control System takes this output as input and generates updated control parameters

(b) The training pipeline.

Fig. 4.3. Simulation Control System training pipeline.

The maximization of agent rewards in all possible states of the system can be represented as the maximization of the following objective:

$$\mathcal{L}(\pi_\theta) = \mathbb{E}_{s,a}\left[r(s,a)\right] = \int\limits_S r(s) \int\limits_A \pi_\theta(s) ds da$$

In the case of neural networks, optimization can be easily performed with Monte-Carlo evaluation of this mathematical expectation with following gradient descent and backward propagation of errors:

$$\nabla_\theta \mathcal{L}(\pi_\theta) = \mathbb{E}_{s,a}\left[\nabla_\theta \pi_\theta(s) r(s,a)\right]$$

However, this naive approach has an extremely slow convergence rate. This happens due to a large number of control parameters in continuous real-valued space. To deal with this problem we decided to use DDPG (Deep Deterministic Policy Gradients) [21], an algorithm that was designed to tackle the continuous control problems. The authors of DDPG propose to predict not just a distribution over all possible actions that agent can make but a specific action. To endorse exploration of state-action space they suggest adding random noise sampled from Ornstein–Uhlenbeck process. In our realization the Ornstein–Uhlenbeck noise was sampled this way:

$$n_{t+1} = n_t + \theta(n_0 - n_t) + \sigma N(0, \sigma),$$

where θ defines a momentum of regression toward the mean(n_0), $N(0,1)$ is normal random variable, σ is a variance of the process. The action predicted by the neural network is consequently modified by addition of the noise:

$$\hat{a}_t = a_t + n_t.$$

This choice of the noise provides an adequate level of "curiosity" due to its random nature. Furthermore, autocorrelation property of this process ensures smooth transitions of actions thus making it more suitable for control tasks and additionally reducing variance during training.

4.2. *Metrics*

We implemented various metrics to compare output data from different simulation runs:

- cosine similarity of spectra or wavelet spectra;
- mean square error;
- Kolmogorov distance (KD), Wasserstein distance (W), Cramér–von Mises distance (CvM) between distributions of two sets of output parameters.

Our goal is to find a reward function that would be sensitive to differences in the control parameters, but smooth and robust to small perturbations. These requirements are important for the convergence of neural network training.

We tested metrics on simulated data with different fixed parameters. It was done without complex analysis but with simple hand-selection based on plots of metric dependencies. We propose following loss function that combines several metrics.

Numeric coefficients were chosen so that all values in loss function would have about the same order of magnitude and similar contribution.

$$R_t = \frac{|S_t^{ref} - S_t^{controlled}|}{10^7} + \frac{CvM(WV_{1,\ldots,t}^{ref}, V_{1,\ldots,t}^{controlled})}{20} + \frac{CvM(WQ_{1,\ldots,t}^{ref}, Q_{1,\ldots,t}^{controlled})}{2},$$

where

- S_t is a total occupied storage space at the time t.
  Occupied storage space is a cumulative and robust characteristic of the system. One drawback of this metric is that it accumulates simulation errors.
- $V_{1,\ldots,t}$ is an empirical distribution function of transferred data amount in "write" mode from $t' = 0$ to $t' = t$;
- $Q_{1,\ldots,t}$ is an empirical distribution function of write requests response time from $t' = 0$ to $t' = t$;
  These two metrics represent dynamical behavior of the system.

Then we can easily define reward function as a temporal difference between losses at consequent moments of time, i.e.

$$r_t = R_{t-1} - R_t.$$

5. TRAINING

5.1. *Experiments*

For the moment, the data from the real system is not yet available for the optimization of simulation. Due to this, the experiment is conducted in the following way: we take two configurations of the system (two sets of initial control parameters), the first configuration is assumed to be the target system that we want to be able to properly simulate the behavior of. The second configuration is used for initialization of the neural network. We initialize all layers randomly and for the last layer, which represents parameters of the simulator, we

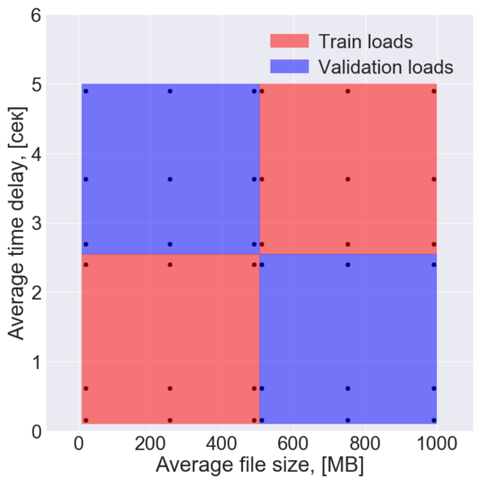


Fig. 5.4. Load parameters sampling areas. Loads for training set are sampled from red area, loads for validation set are sampled from blue area. Black dots are sampled points.

zero the weights and set biases of layer to parameters from the second configuration. On the assumption that in the real system the optimal parameters could be far away from the initial parameters we deliberately set parameters of the second configuration to be very different from the target system.

Also, we made one more simplification. Our configurations differ only in the following sets of parameters: bandwidth of physical network links and the computational power of storage controllers. These parameters have the highest impact on the behavior on the system in comparison with other parameters like latency or computation power of drive enclosures. Another reason for reduction of parameter space is an exponentially increasing complexity of optimization problems. Overall, neural network generates 5 parameters: four parameters account for computational power of four storage controllers and the fifth one accounts for bandwidth(Figure 5.5a).

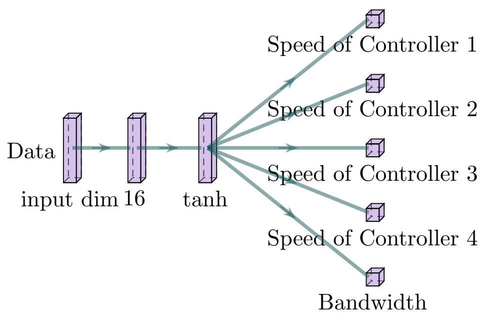
The system load is modelled as Poisson process with mean time between incoming files equal $\bar{\tau}$[s]. The file size is modelled with exponential distribution parameterized by average file size $\bar{s}$[Mb]. For tuning our algorithm we consider a finite set of possible load configurations: $L = S \times T = \{s_i\}_{i=1}^{N} \times \{\tau_j\}_{j=1}^{K}$.

The loads for training L_{train} and loads for validation L_{test} are sampled from uniform distributions on a 2D-grid (Figure 5.4). It is important to note that L_{train} and L_{test} distributions do not overlap and this allows us to test generalization of our model on unseen situations.

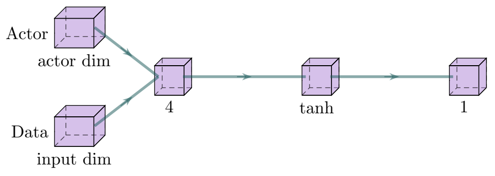
We chose quite simple architecture (Figures 5.5a and 5.5b), thus allowing us to avoid over-fitting and speed-up training. The PyTorch [22] framework was used. At each epoch of training the neural network is fed with data from 40 new and 40 old (experience replay) runs of simulation. Fed data is time differences between the outputs of simulator(Section 3.3) at the moment $t + \Delta t$ and t. Optimization is done with Adam with learning rate set to 10^{-2} and all other parameters left as default. Evaluation on validation set of loads is done every 20 epochs.

5.2. Results

We judge our improvement in simulation by looking at difference in occupied storage space ($|S_t^{ref} - S_t^{controlled}|$[B]). As can be noted, this metric is the first part of R_t definition(Section 4.2). It was chosen for comparison mostly due to simplicity of interpretability.



(a) Actor architecture

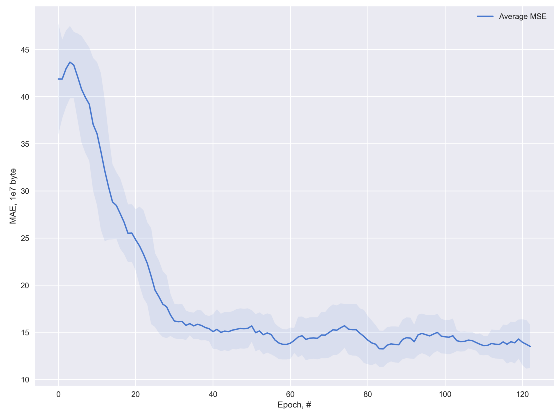


(b) Critic architecture

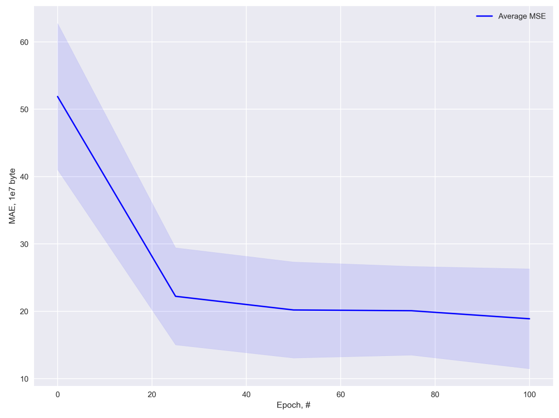
Fig. 5.5

The improvement of the metric on the training set of loads (Figure 5.6a) and validation set of loads (Figure 5.6b) have a similar pattern. Taking this into account, two conclusions may be derived:

- our agent (neural network) is able to tune the parameters of the simulation and make it similar to the target system;
- the trained neural network is robust and capable of generalizing on unseen situations.



(a)



(b)

Fig. 5.6. Averaged metric with confidence interval on training (left) and validation (right) set of loads. Lower is better.

At Figure 5.7a and Figure 5.7b one can see metric(R_t) dependence from simulation time and epoch number. As expected, the distance between the hybrid and target systems is increasing while simulation time is growing, i.e. we are observing the disorder between two processes. This disorder diminishes during training of neural network making system more and more compliant.

6. CONCLUSION

We demonstrated the viability of our approach with a fusion of conventional simulation techniques and RL-trained neural network on the problem of optimization of discrete event

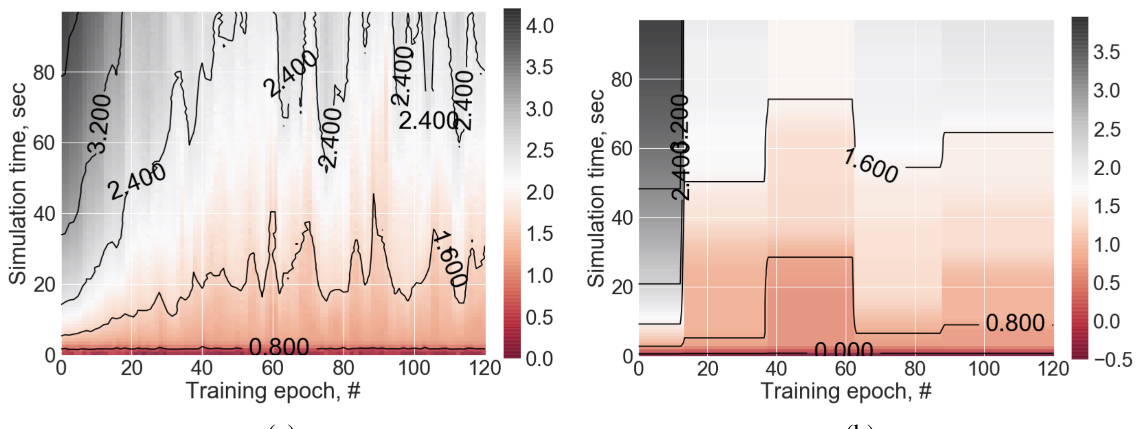


Fig. 5.7. Contour plot of logarithm of the metric as a function of simulation time (vertical axis) and epoch (horizontal axis) during training (left) and validation (right).

simulator of the distributed storage system. In this setup, a neural network is trained by the Reinforcement Learning algorithm to find an optimal control strategy for parameters of the simulator.

The hybrid simulator has a wide range of benefits in comparison with vanilla simulators: simplified customization of the simulator; ability to infer patterns from data. And in contrast with GANs, it has interpretable behavior and ability to generalize on partially unseen scenarios.

Further research may cover such steps as:

- increase complexity of the system under control by adding more control parameters;
- assess the sensitivity of the agent to the choice of reward functions;
- build and validate the approach on a realistic multiparametric data sample.

7. ACKNOWLEDGEMENTS

The research was carried out with the financial support of the Ministry of Science and Higher Education of Russian Federation within the framework of the Federal Target Program "Research and Development in Priority Areas of the Development of the Scientific and Technological Complex of Russia for 2014-2020". Unique identifier – RFMEFI58117X0023, agreement 14.581.21.0023 on 03.10.2017.

References

1. Tao, F., Cheng, J., Qi, Q., Zhang, M., Zhang, H., & Sui, F. (2018) Digital twin-driven product design, manufacturing and service with big data. *The International Journal of Advanced Manufacturing Technology*, **94**, 3563–3576, `https://dx.doi.org/10.1007/s00170-017-0233-1`.
2. Aaron Parrott, L. W. (2017) Industry 4.0 and the digital twin: Manufacturing meets its match. p. 110.
3. Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., & Bengio, Y. (2014) Generative Adversarial Networks. *ArXiv e-prints*.
4. Doshi-Velez, F. & Kim, B. (2017), Towards a rigorous science of interpretable machine learning.

5. Cope, J., Liu, N., Lang, S., Carns, P., Carothers, C., & Ross, R. H. (2011) Codes : Enabling co-design of multi-layer exascale storage architectures.
6. Riley, G. F. & Henderson, T. R. (2010) *The ns-3 Network Simulator*, 15–34. Springer Berlin Heidelberg.
7. Varga, A. & Hornig, R. (2008) An overview of the omnet++ simulation environment. In *Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops*, ICST, Brussels, Belgium, Belgium, 60:1–60:10, Simutools ’08, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).
8. Wang, C.-Y., Zhou, F., Zhu, Y.-L., Chong, C. T., Hou, B., & Xi, W.-Y. (2003) Simulation of fibre channel storage area network using sansim. In *The 11th IEEE International Conference on Networks, 2003. ICON2003.*, 349–354, https://dx.doi.org/10.1109/ICON.2003.1266215.
9. Molero, X., Silla, F., Santonja, V., & Duato, J. (2000) Modeling and simulation of storage area networks. In *Proceedings of the 8th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, Washington, DC, USA, 307–, MASCOTS ’00, IEEE Computer Society.
10. Molero, X., Silla, F., Santonja, V., & Duato, J. (2001) A tool for the design and evaluation of fibre channel storage area networks. In *Proceedings of the 34th Annual Simulation Symposium (SS01)*, Washington, DC, USA, 133–, SS ’01, IEEE Computer Society.
11. Perles, A., Molero, X., Marti, A., Santonja, V., & Serrano, J. J. (2001) Improving the execution of groups of simulations on a cluster of workstations and its application to storage area networks. In *Proceedings. 34th Annual Simulation Symposium*, 227–234, https://dx.doi.org/10.1109/SIMSYM.2001.922136.
12. Muknahallipatna, S., Miles, J., Hamann, J., & Johnson, H. (2010) Large fabric storage area networks: Fabric simulator development and preliminary performance analysis. *International Journal of Computers and Applications*, **32**, 167–180, https://dx.doi.org/10.1080/1206212X.2010.11441973.
13. Paternina-Arboleda, C. D., Montoya-Torres, J. R., & Fabregas-Ariza, A. (2008) Simulation-optimization using a reinforcement learning approach. In *2008 Winter Simulation Conference*, 1376–1383, https://dx.doi.org/10.1109/WSC.2008.4736213.
14. Sargent, R. (2011) Verification and validation of simulation models. *Engineering Management Review, IEEE*, **37**, 166 – 183, https://dx.doi.org/10.1109/WSC.2010.5679166.
15. S. Carson II, J. (2002) Model verification and validation. *Winter Simulation Conference Proceedings*, **1**, 52–58, https://dx.doi.org/10.1109/WSC.2002.1172868.
16. YADRO (2017), TATLIN hybrid storage. https://yadro.com/products/tatlin, [Online; accessed 29-October-2018].
17. Google (2012), Go programming language. https://golang.org, [Online; accessed 29-October-2018].
18. Sutton, R. S. & Barto, A. G. (1998) *Introduction to Reinforcement Learning*. Cambridge, MA, USA: MIT Press, 1st edn.
19. Watkins, C. J. C. H. & Dayan, P. (1992) Q-learning. *Machine Learning*, **8**, 279–292, https://dx.doi.org/10.1007/BF00992698.
20. Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. A. (2013) Playing atari with deep reinforcement learning. *CoRR*, **abs/1312.5602**.
21. Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., & Riedmiller, M. (2014) Deterministic policy gradient algorithms. In *Proceedings of the 31st International*

Conference on International Conference on Machine Learning - Volume 32, I–387–I–395, ICML'14, JMLR.org.

22. Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., & Lerer, A. (2017) Automatic differentiation in pytorch.