SANgo: a storage infrastructure simulator with reinforcement learning support

Kenenbek Arzymatov[1], **Andrey Sapronov**[1,2], **Vladislav Belavin**[1], **Leonid Gremyachikh**[1], **Maksim Karpov**[1], **Andrey Ustyuzhanin**[1,3], **Ivan Tchoub**[4], **and Artem Ikoev**[4]

[1]**National Research University Higher School of Economics, Myasnitskaya 20, 101000, Moscow, Russia**

[2]**Joint Institute for Nuclear Research, Joliot-Curie 6, Dubna, Moscow Region, 141980, Russia**

[3]**Moscow Institute of Physics and Technology, Institutskiy per. 9, Dolgoprudny, Moscow Region, 141700, Russia**

[4]**YADRO, Rochdelskaya 15/13, 123022, Moscow, Russia**

Corresponding author:
Maksim Karpov[1]

Email address: mekarpov@hse.ru

ABSTRACT

We introduce SANgo (Storage Area Network in the Go language) – a Go-based package for simulating the behavior of modern storage infrastructure. The software is based on the discrete-event modeling paradigm and captures the structure and dynamics of high-level storage system building blocks. The flexible structure of the package allows us to create a model of a real storage system with a configurable number of components. The granularity of the simulated system can be defined depending on the replicated patterns of actual system behavior. Accurate replication enables us to reach the primary goal of our simulator – to explore the stability boundaries of real storage systems. To meet this goal, SANgo offers a variety of interfaces for easy monitoring and tuning of the simulated model. These interfaces allow us to track the number of metrics of such components as storage controllers, network connections, and hard-drives. Other interfaces allow altering the parameter values of the simulated system effectively in real-time, thus providing the possibility for training a realistic digital twin using, for example, the reinforcement learning (RL) approach. One can train an RL model to reduce discrepancies between simulated and real SAN data. The external control algorithm can adjust the simulator parameters to make the difference as small as possible. SANgo supports the standard OpenAI gym interface; thus, the software can serve as a benchmark for comparison of different learning algorithms.

INTRODUCTION

A storage system is a critical part of any IT infrastructure. A significant effort is put into developing reliability techniques and failure protection schemes of storage area network systems (SAN), which is an example of a general concept of storage infrastructure. There are various means of achieving and increasing the system's reliability in terms of both accessibility and data preservation: replication of the system's physical components or software solutions such as cluster management and RAID technology. The requirements for reliability and scalability of the SAN system result in its complexity, and consequently, the SAN architecture becomes challenging to supervise. A dedicated computer simulation model or a digital twin can be created to observe the behavior of the storage infrastructure virtually. Such computer model granularity depends on the simulation tasks, which may be performance optimization, failure diagnostics/prediction, or any other data-driven method for improving the storage system's functionality.

There are three qualitatively different approaches to create a SAN simulator:

1. a purely physical model with detailed hardware/software processes;

2. a pure Machine Learning (ML) model, based on data from previous SAN operations;

3. a hybrid of the above – when a simplified SAN architecture is implemented in simulation, and its parameters are adjusted by a trained Deep Learning algorithm.

The first method gives the best results in simulation accuracy and has outstanding potential for extrapolation beyond the known operation domain. It predicts the system's behavior with a configuration that might not have been implemented yet. However, this approach requires significant expertise in the field of storage architecture and a deep understanding of the software stack used on-board the SAN. It also takes enormous resources to implement the model with all the physical and logical details of the system components when trying to achieve the required simulation quality.

The pure ML model is based on the data-driven approach, meaning that the model is trained on the data collected from the existing storage system. This method, on the contrary, may not require as much knowledge of the SAN structure and its operation principles. It may produce satisfactory accuracy of the simulated parameters, but lack the vital feature of scalability – the model is bonded with the data collected for particular SAN architecture and configuration and is unable to extrapolate beyond the scope of the training parameters.

The hybrid simulation technique is a good trade-off in terms of the quality and depth of the domain expertise needed. It requires a relatively simple model of the SAN architecture, implementing only basic components with their functionalities and logical associations. The components must have effective adjustable parameters with meanings similar to the real ones, for example, the CPU clock speed or link bandwidth. These parameters must be adjusted by a reinforcement learning (RL) algorithm to improve the simulation quality. Similarly to the second approach, the hybrid approach needs real data for training.

The development of SANgo was motivated by research aimed to create a monitoring tool able to diagnose the current SAN state and predict possible failures of its components. The diagnostic algorithms were designed to analyze the time series of different parameters collected during storage system operation. Since the algorithms are based on the ML paradigm, they needed plenty of training data, which, in this particular case, could only be obtained from a simulated environment. An essential requirement was the physical consistency of the synthetic data, and with reasonable efforts, it became plausible with the hybrid approach to the simulation.

Such a coupling of the simplified SAN simulator with runtime control by an RL agent required a dedicated study. With many deep learning technologies available, one has to select an approach and its configuration to obtain the best combination of simulation quality, training effort, and speed. The corresponding study was conducted and presented in (Karpov et al., 2018; Sapronov et al., 2018).

The simulator is developed to provide a user with the simplicity of configuration. The user needs to specify the following files: a file that sets the architecture of a storage array in an XML format, a file that defines the impact of external effects (temperature, humidity, atmospheric pressure, and vibration) and finally a set of functions that defines the behavioral logic of each component of a storage array.

There are several open-source storage system simulators available, for example, a software package "CODES project" (Cope et al., 2011; Mubarak et al., 2017) developed by a team of researchers from the Computer Science and Math department of Argonne National Laboratory and Rensselaer Polytechnic Institute (US, Illinois). The CODES simulator is based on the technologies of the Rensselaer's Optimistic Simulation System (ROSS), which allows the parallel execution of an event-driven system that can significantly decrease the runtime of the simulation. The main uses of CODES include large-scale storage systems, scientific distributed applications, parallel and high-performance computing systems with high-load input/output operations, and computational complexity. Another simulator, the C++ based ns3 framework (Riley and Henderson, 2010), is also popular among researchers. The general approach to network-like structure simulation is the OMNeT++ framework (Varga and Hornig, 2008). Another work presents a simulation compliant with the fiber channel technology often used in contemporary SAN architectures, developed as the SANSim tool (Wang et al., 2003). More simulation method descriptions and studies dedicated to SAN system modeling can be found in (Molero et al., 2000a, 2001; Perles et al., 2001; Molero et al., 2000b; Muknahallipatna et al., 2010).

SOFTWARE DESCRIPTION

SANgo is a modular framework for the Discrete Event Simulation (DES) of storage infrastructure. Its metadata description is given in Table 1. DES is a method of simulating the behavior and performance of a real-life process, facility, or system. DES models the system as a series of events (e.g., a beginning/end

99 of file writing, data block transfer or a start of TCP connection), which happens over time. The main
100 assumption of the DES paradigm is the invariability and consistency in the modeled system between the
101 events. It proves itself as a viable approach to effectively evaluating diverse sets of algorithms. More
102 information can be found in (Fishman, 1978).

Nr.	Code metadata	Description
C1	Current code version	v 0.8
C2	Permanent link to code/repository used for this code version	*https://github.com/HSE-LAMBDA/ sango*
C3	Code Ocean compute capsule	*https://codeocean.com/capsule/ 9707185*
C4	Legal Code License	GPL
C5	Code versioning system used	git
C6	Software code languages, tools, and services used	Golang 1.8+, Python 3.6, Jupyter Notebook
C7	Compilation requirements, operating environments & dependencies	pytorch 0.4.0+, OpenAI/gym, libzeromq-dev
C8	If available, link to developer documentation/- manual	
C9	Support email for questions	karzymatov@hse.ru

Table 1. Code metadata

103

104 SANgo provides the functionality to build a simple model for specific computing environments,
105 especially storage area networks. The primary site of interest is the exploration of the behavior of the
106 storage machine under stress testing or exploitation in the medium- or long-term, for observing failures of
107 its components.
108 In SANgo, each file to be written to the storage is modeled as an independent entity that has
109 corresponding attributes, such as name, size, and block size. The fundamental principle of the simulation
110 process is resource modeling and control. SANgo focuses on algorithms that have the following common
111 objective: assigning a set of tasks to a set of resources in a way that is optimal with respect to some
112 metric. The three basic types of resources are provided: CPU, network interface, and storage. Other types
113 are built upon these basic ones. For example, a storage controller, PCIe-fabric can use CPU type as its
114 basis; RAM, SSD, hard-drives, and JBOD (as a collection of hard-drives) use storage type to implement
115 additional functionality. An example of SAN architecture with these primary resources collected into
116 SAN components is shown in Figure 1.
117 The SANgo core library is written in the Go programming language. This choice was made because
118 Go can efficiently create lightweight threads, so-called goroutines. We use a goroutine as a representation
119 of the SAN component logic that a user wants to simulate. In other words, the user creates a function and
120 specifies a sequence of actions and/or behaviors that a resource should follow. Such functions are given in
121 Table 2. When sequences of 'behaviors' are specified, the simulation starts.

Simulation sequence

123 The general algorithm of the SANgo operation is shown in Figure 2. The entry point is the definition
124 of the input parameters: the storage system topology and component parameters and the definition of
125 the load scenario. The latter means the sequence of I/O requests from a virtual client that the simulated
126 SAN must process. Depending on the load scenario, a specific process (component activity) and mode
127 (asynchronous or synchronous) are chosen. Further, an option of component failure, either spontaneous or
128 planned, is processed. With or without this failure, the process of event simulation continues until the
129 load scenario is complete. During each simulation stage, metrics are collected from the components and
130 written out. These metrics describe the state of the simulated system and provide primary information
131 on its operation. Unlike other simulators, the framework allows online adjustments of the components
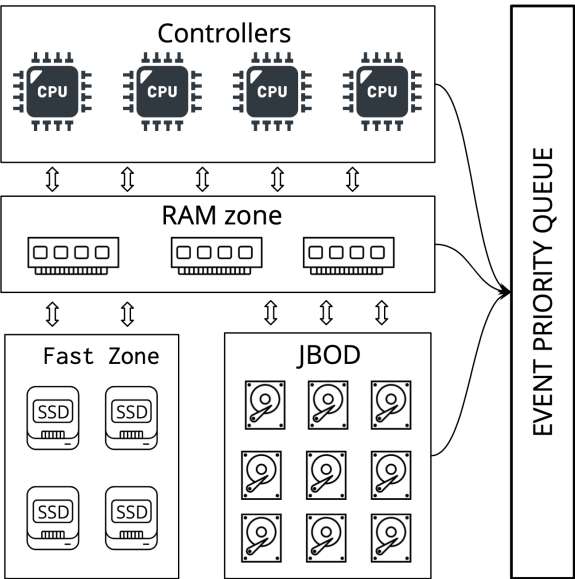
Figure 1. An example of SAN architecture implemented in the SANgo framework.

Component name	Behavior	Description
Contoller	`SendPacketSync`	Send a `packet` to the `dest` and wait until the end of transmission (two side communication)
	`SendPacketAsync`	Send a packet without waiting for the end of the transmission (two side communication)
	`BroadCastSendPacket`	Send a broadcasting message with a `packet` to all processes which listen to 'dest' address
	`DetachedSendPacket`	Send a packet and wait until the end of the transmission (one side communication)
	`ExecutePacket`	Process packet
	`Wait`	Turn on waiting mode
Hard-drive	`WriteAsync`	Write data blocks in a logical volume in an `async`/`sync` mode
	`WriteSync`	
	`ReadAsync`	Read data blocks from a logical volume in an `async`/`sync` mode
	`ReadSync`	

Table 2. Functions that define the simulated behaviors of SAN components

132 parameters. This feature makes possible coupling with an RL agent as described in Section "Illustrative
133 Examples".

Software functionalities

135 **Online configuration** The simulation models created within the SANgo framework are configurable
136 during runtime. It is an important feature, created intentionally, for coupling with a controlling agent and
137 implementing the hybrid simulation method. The details of such cooperation can be found in Section
138 "Illustrative Examples".

139 **Load generation** One of the main interests is how a storage array behaves under different loads. The
140 complete list of the fully measured output metrics is shown in Table 3.

141   **Output format**. During the runtime of the simulation, the program opens a file descriptor and saves
142 the simulated metrics in a JSON (Crockford, 2018) array format. Each item in the array is a key-value
143 representation of the resource objects (storage controllers, links, and hard-drives) created in the simulation.
144 Go allows users to change the encoding of each object by the format string stored under the "json" key in
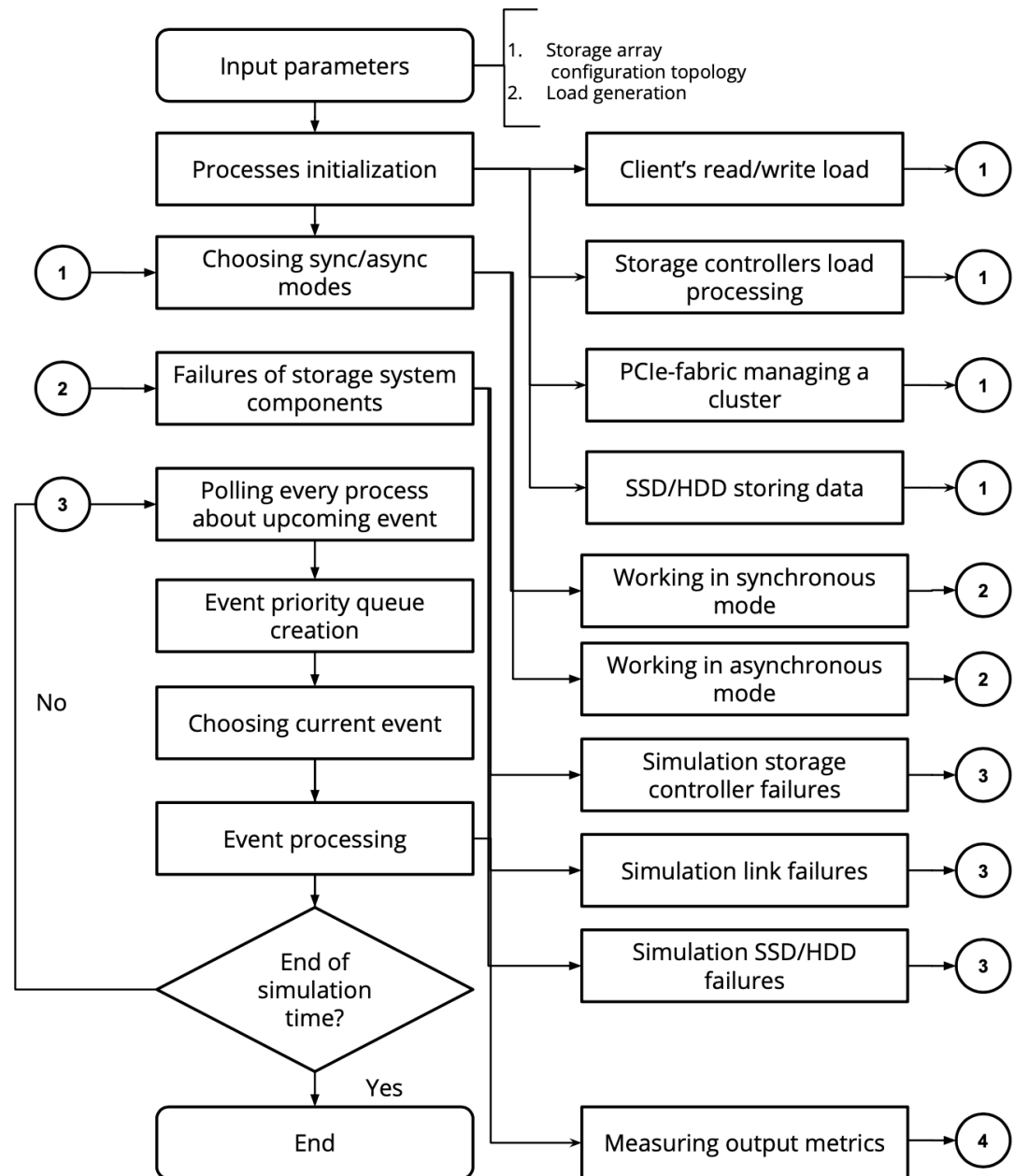
Figure 2. SANgo discrete event simulation workflow.

Parameters	Component name	Output name	Description
Inner	Storage controller	`cpu_usr`	User CPU utilization, %
		`cpu_sys`	System CPU utilization, %
		`cpu_idle`	CPU idle time, %
		`mem_used_MB`	Used RAM memory, `MB`
		`mem_free_MB`	Free RAM memory, `MB`
		`cpu_temp`	CPU temperature, $°C$
		`fan_speed`	Fan angular velocity, `Hz`
	Network interface	`RxKB`	Input traffic through interface, `MB`
		`TxKB`	Output traffic through interface, `MB`
	Hard drive	`total_cap_MB`	Raw capacity, MB
		`used_cap_MB`	Used capacity, MB
		`alloc_MB`	Memory used by system, MB
		`len_MB`	Requested volume size, MB
		`r_KBps`	Disk read/write speed, KB/s
		`w_KBps`	
		`r_ops`	Read/write operations per second, IOPS
		`w_ops`	
		`r_await_ms`	Average read/write request processing time, ms
		`w_await_ms`	
		`ccrm`	Number of denied requests, n
		`rrqm`	Number of read/write requests, n
		`wrqm`	
		`avgrq_sz`	Average size of request, n
		`avgqu_sz`	Average client queue size, n
		`MB_read`	Total amount of read/write data, MB
		`MB_wrtn`	
		`blk_read`	Number of read/write blocks with given size, n
		`blk_wrtn`	
	Each component	Health state	One of `OK`, `MISSED`, `BAD` or `LOST`
Outer	Atmospheric	`humidity` `atm_pressure` `temperature` `vibration`	Current values of humidity, atmosphere pressure, temperature, and vibration meausred inside storage system `%`, `kPa`, $°C$, `Hz` respectively

Table 3. Output characteristics

the resource field's tag. This language feature allows the user to additional mute parameters. Together with native struct embedding (inheritance), it is possible to adapt the output format needed for different hardware configurations easily.

Anomalies. For the simplified simulation, we chose to split the failure factors into two categories: internal and external. The former relates to the component operation mode, its load, and neighboring component conditions. The internal factors are better described by the RL counterpart of the SANgo, which takes into account operation history and composite state of the system. For the latter category, we consider four ways in which the external parameters could affect the failure of parts of the system, such as the impact of temperature, humidity, atmospheric pressure, and vibration level.

Temperature. There are multiple failure mechanisms in electronic components related to their temperature: electromigration (d'Heurle, 1971), high temperature stress migration (Aoyagi, 2005), thermal fatigue (Zhou and Hashida, 2002), mechanical stresses induced by differential thermal expansion of materials (Rabiei and Evans, 2000), the drift of parameters (frequency, current, voltage) of devices (Rabiei and Evans, 2000), solder joint failures (Yeh et al., 2002), ionic effects (Berg and Paulson, 1980),

increase in leakage current (Hamidi and Coquery, 1997) and bond-wire fatigue (Roesch and Jittinorasett, 2004) (Matsunaga and Uegai, 2006). The detailed review of electronics failure modes that are influenced by temperature is given in (Blanks, 1990). The temperature of individual simulated components we consider as an internal factor and, therefore, not implement in the SANgo code. Inclusion of all or even some of these mechanisms into simulation would require a very detailed simulation of the circuit boards and electronic parts of the SAN components.

In general, the Arrhenius model (Lakshminarayanan and Sriraam, 2014) is a good approximation for exact failure mechanisms, including electromigration, corrosion, and certain manufacturing defects when dealing with slowly changing ambient temperature. The model is derived from the observed dependence of chemical-reaction rates on temperature changes. We use the Arrhenius model solely to take into account the external factor of ambient temperature for the failure rate of the electronic components. According to this model, the reaction acceleration rate is given by:

$$K = A \exp \{E_a/k[1/T_{ref} - 1/T]\}, \tag{1}$$

where K is the resulting failure rate, A is a rate constant empirically derived, E_a is the activation energy (eV), k is the Boltzmann's constant ($8.6 \times 10^{-5} eV/K$), T is the ambient temperature (K) and T_{ref} is the component's reference temperature (K).

For non-electronic components (such as hard-drives), the failure rate is calculated based on fail-safe operation time T (Sankar et al., 2011). Using the maximum likelihood method, β (form factor), ζ (scale coefficient) the failure probability is estimated:

$$K_{\beta,\zeta} = \frac{\beta T^{\beta-1} \exp{-\frac{T}{\zeta}}^{\beta}}{\zeta^{\beta}}, \tag{2}$$

where $\beta > 0, \zeta > 0, T > 0$.

Humidity. *Atmospheric pressure*. *Vibration*. The impacts of these three external factors are modeled similarly. The predefined mappings from the environment factor value to failure rates for humidity, pressure, and vibration are taken from (Mitchel, 1996; Strom et al., 2007; Dutta and Barnard, 2017) correspondingly. At each timestamp t of the simulation, current values of `humidity`, `pressure`, and `vibration` are taken. Then the corresponding failure rates are calculated using the said relationships.

For all of the external factors the failure-triggering pseudo-code looks like:

```
WHILE simulation is going:
    READ current values of temperature, humidity,
                          pressure, vibration;
    CALCULATE failure rates imposed by:
                    f1 ← temperature;
                    f2 ← humidity;
                    f3 ← pressure;
                    f4 ← vibration;
    GENERATE a random number R from [0, 1) uniformly;
    IF R < (f1 + f2 + f3 + f4): CREATE breakdown;
    ELSE: nothing happens;
    WAIT until next time step;
```

ILLUSTRATIVE EXAMPLES

An example of the SANgo application for the simulation of a mid-range storage array with basic structure is described below. The simulated architecture was configured with parameters corresponding to the real SAN components parameters. Most underlying effects of the SAN functioning, such as the operating system and software logic, load distribution algorithms, and other more complex hardware details are not described by the simulator. In order to better approximate the behavior of the real-life system, SANgo was coupled with DeepController (DC) – an RL optimizer. The scheme of cooperation between SANgo and DC is shown in Figure 3.
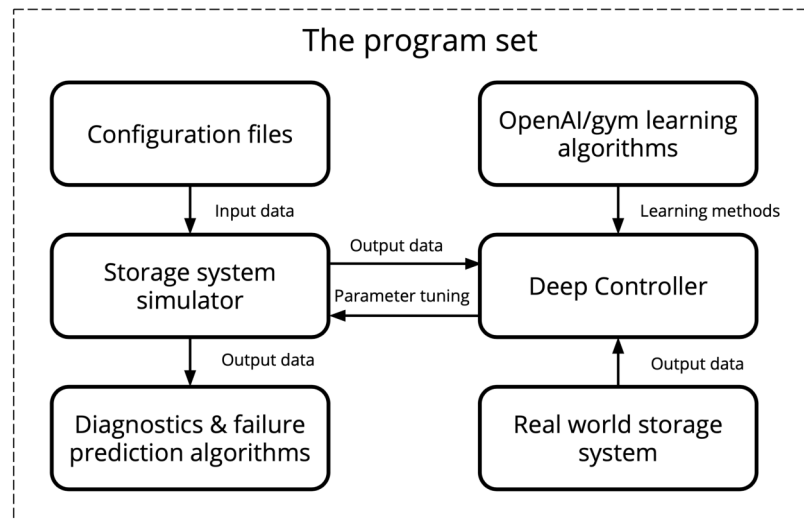
Figure 3. The hybrid simulation scheme using OpenAI/gym interfaces.

²⁰⁴ DC represents the RL and deep learning paradigm for controlling the simulator by tuning the effective
²⁰⁵ parameters of its components. The simulation process is split into two phases.

²⁰⁶ **Training phase** : DC takes as input the real SAN data (load and metrics), and the initial effective
²⁰⁷ parameters of SANgo. By varying the effective parameters, DC learns what needs to be done to
²⁰⁸ obtain a simulator state, similar to the real SAN under the same load conditions. The output for this
²⁰⁹ phase is a trained model with corresponding neural network (NN) weights.

²¹⁰ **Control phase** : during the actual simulation, DC takes as input the load and current effective parameters
²¹¹ and simulated metrics. The corrections to the effective SANgo parameters are given to the output
²¹² for every next simulation step.

²¹³ In this context, it is important to note that a special wrapper was implemented to create an OpenAI
²¹⁴ Gym environment (Brockman et al., 2016) using DC, SANgo, real SAN data, and initial effective
²¹⁵ parameters. Such an environment provides an advantageous opportunity to use multiple state-of-the-art
²¹⁶ RL-algorithm libraries, implemented in the OpenAI toolkit (Dhariwal et al., 2017; Hill et al., 2018;
²¹⁷ Kolesnikov, 2018). In particular, we used a DDPG model to train DC as one of the efficient approaches
²¹⁸ for Continuous Control problem (Lillicrap et al., 2015). Also, the obtained gym-environment could be
²¹⁹ used as a benchmark for other RL-algorithms, especially since there are not so many environments for
²²⁰ digital twins (Koch et al., 2019).

²²¹ The real SAN data were used to validate the behavior of the hybrid simulator. A sequence of load
²²² requests was generated on the real storage system prototype together with artificially induced failures of
²²³ one of the components: a storage controller, a network interface, or a storage device. The labeled metrics
²²⁴ of the healthy and broken systems were collected and used to train the hybrid simulator with the same
²²⁵ workload and failure scenarios. In the end, the simulated metrics are compared against the real ones. An
²²⁶ example of one of the metrics – the storage controller CPU load is shown in Figure 4.

²²⁷ One can observe a qualitative agreement between the simulated and the real data. The more component
²²⁸ parameters added, the closer the DC model distributions were to the real data.

²²⁹ For generating data in this scenario, the simulator was launched by the following command:

```
go run main.go -sim_run=210000 -platform=virt_setup.xml
        -num_jobs_config=num_jobs.json -packet=packet.json
        -controlling_mode=0 -num_jobs=1 -atm_dep=temp.json
        -atm_control=atm_control.json -file_amount_w=10
        -file_size_w=10GB -output=output.json
```

²³⁵ In this command snippet, several flags are used. They define the following behavior of the simulation.
²³⁶ It runs for 210000 seconds, writes 10 files, each with size 10GB, uses 1 job, a topology information
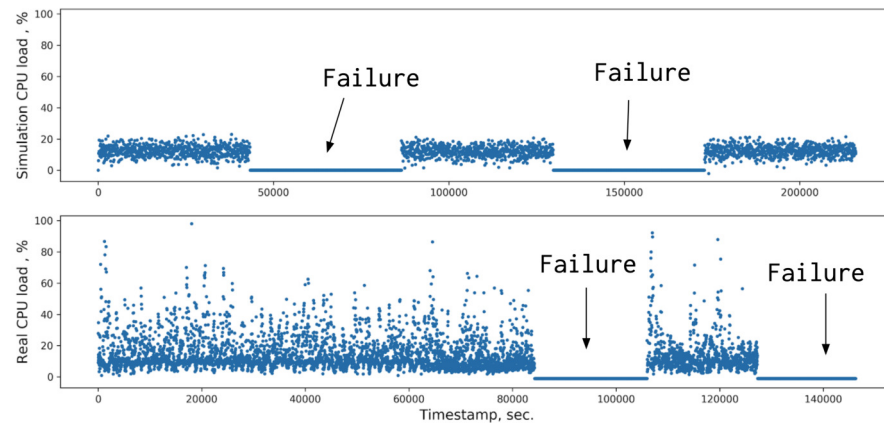
Figure 4. Comparison of the CPU load metrics between simulated (upper) and real data (lower). The periods marked 'Failure' correspond to a storage processor being offline.

237 is provided in `virt_setup.xml`, the external dependencies are defined in `atm_control.json`,
238 `packet.json` provides a packet block timing information, and finally, output data is written to the
239 `output.json` file. There are other flags available to configure the simulator functionality. A complete
240 list of execution options is given in Table 4.

241

242 The hardware parameters configuration is an essential step in the SANgo setup. For example, we used
243 empirically obtained parameters of the storage devices affecting the speed of writing and reading data
244 using Flexible I/O tester. We performed data transfer in different modes, such as reading, writing, random
245 reading, random writing, reading/writing, and random reading/writing. These modes were combined
246 with different data block sizes – from 2 Kb to 2^6 Gb to cover a full range of possible IO conditions. The
247 FIO tool allows user to directly measure some parameters, such as latency, transmission, and processing
248 time. Other parameters, such as rate and seek time, had to be calculated from the parameters that FIO
249 allowed to measure. An example of hard drive parameter values for 8 Kb block size, measured with such
250 an approach, is provided in Table 5.

251 IMPACT

252 The SANgo simulator helps researchers to address several issues in the area of SAN architecture devel-
253 opment. The primary purpose of the developed SAN digital twin (Sapronov et al., 2018), in which the
254 SANgo is a critical component, is to provide realistic data for the SAN optimization. In conditions where
255 the real data are scarce, for example, during the early stages of development, the synthetic data may help
256 to improve the system's stability, study its behavior under different load regimes and observe how the
257 system reacts to various failures of its components. Besides, synthetic data allows the development of
258 diagnostic and prediction tools for system malfunction based on data-driven algorithms available in the
259 Machine Learning domain (Hushchyn et al., 2019). The SAN architecture optimization and parameter
260 tuning is another possible application of such a digital twin.
261 For example, the SANgo code was used to produce synthetic data for training diagnostic and predictive
262 software. The software relies on data-driven algorithms to assess the state of the storage system, diagnose
263 its malfunction, and calculate the probability of failures in the near future. Using SANgo helps to
264 significantly reduce the amount of collected real SAN operation data, which is quite expensive in most
265 cases.
266 The application of SANgo software is somewhat limited outside the intended scope. The simulator
267 intentionally describes the storage infrastructure in low detail and, therefore, can be used only for
268 deterministic and approximate emulation of the SAN operation. However, the general approach of
269 combining the basic simulator with an RL agent is very scalable for other uses where one needs to
270 simulate a complex apparatus or a system with many loosely controlled parameters, but with available
271 real data collected during its operation.

Type	Flag's name	Type	Description
Common	`-sim_run`	`float`	Total simulation runtime
	`-disk_amount`	`int`	Amount of hard-drives
	`-num_jobs`	`int`	The analog of the `fio numjobs`
Input files	`-platform`	`string`	
	`-packet`	`string`	Data block's timing parameters
	`-atm_control`	`string`	A change of atmosphere parameters over time
	`-atm_dep`	`string`	Impact of atmosphere parameters on the system components
	`-client`	`string`	
	`-num_jobs_config`	`string`	Client's load & quota specification
Load	`-file_amount_w`	`int`	Number of files to be written
	`-file_size_w`	`float`	Write files size range
	`-load_range_w`	`float`	File write rate
	`-file_amount_r`	`int`	Number of files to be read
	`-file_size_r`	`float`	Read files size range
	`-load_range_r`	`float`	File read rate
Anomaly	`-anomaly_type`	`string`	Controller, link, disk type anomaly
	`-anomaly_amount`	`int`	Number of anomalies
	`-anomaly_time_range`	`float`	Anomaly ocurring rate
	`-anomaly_duration`	`float`	Duration of anomalies
DC[1] mode	`-controlling_mode`	`bool`	Enabling DC
	`-host`	`string`	Hostname
	`-port`	`int`	A communication endpoint
	`-protocol`	`tcp, udp`	SANgo-DC communication protocol
	`-delay`	`float`	Delay in messaging between GT and DC

Table 4. A complete description of the SANgo execution options.

transmission t	latency t	read processing t	write processing t	sequential read rate t
3.96e-05	0.01784	0.01229	0.02331	0.00065

Table 5. An example of hard drive parameter values for 8 Kb block size, where "t" is time in seconds.

The SANgo alone has limited precision in terms of simulation quality. The storage system model describes only high-level components and provides a very simplified description of their failure mechanics. However, the strong side of our approach is the adjustable parameters of these components. When the parameters are controlled by an RL-trained agent, the simulation quality can be improved. At the same time, the physical model within the SANgo framework ensures the results are interpretable and physically consistent.

While the software itself is not used in any commercial setting, the product of its application, the diagnostic and prediction tool for the SAN malfunction, is bundled with an undisclosed commercial storage system. The tool is developed and trained using real data and synthetic data obtained from the digital twin of the SAN.

CONCLUSION

A flexible framework SANgo for the creation of an event-driven storage simulator was presented. Within this framework, the storage system models have simplified design, but researchers are allowed to adjust the components' effective parameters during runtime to improve the quality of the simulation. This approach complies with a hybrid simulation method, where a reinforcement learning algorithm adjusts a physical model.

The SANgo framework allows the user to emulate the I/O load on the storage system, as well as

289 failures of its components. It was initially designed to work in tandem with the DeepController program,
290 implementing the RL algorithm. The simulator can also be used as a benchmark for comparison of
291 different learning algorithms, due to support of the OpenAI/gym interface.
292   The purpose of such a hybrid simulator is to serve as a storage system digital twin and provide large
293 amounts of synthetic data. This data can be further used for storage system optimization, diagnostic, and
294 failure prediction.

CONFLICT OF INTEREST

296 No conflict of interest exists: We wish to confirm that there are no known conflicts of interest associated
297 with this publication, and there has been no significant financial support for this work that could have
298 influenced its outcome.

ACKNOWLEDGMENTS

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

REFERENCES

305 Aoyagi, M. (2005). Temperature characteristics of stress-induced migration based on atom migration.
306   *Journal of Vacuum Science & Technology B: Microelectronics and Nanometer Structures Processing,*
307   *Measurement, and Phenomena*, 23(6):2384–2389.
308 Berg, H. M. and Paulson, W. M. (1980). Chip corrosion in plastic packages. *Microelectronics Reliability*,
309   20(3):247 – 263.
310 Blanks, H. S. (1990). Arrhenius and the temperature dependence of non-constant failure rate. *Quality and*
311   *Reliability Engineering International*, 6(4):259–265.
312 Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016).
313   Openai gym. *CoRR*, abs/1606.01540.
314 Cope, J., Liu, N., Lang, S., Carns, P., Carothers, C., and Ross, R. H. (2011). Codes: Enabling co-design
315   of multi-layer exascale storage architectures. IEEE Transactions on Parallel and Distributed Systems.
316 Crockford, D. (2018). ETF RFC 8259 The JavaScript Object Notation (JSON) Data Interchange Format.
317   `https://www.json.org/`.
318 Dhariwal, P., Hesse, C., Klimov, O., Nichol, A., Plappert, M., Radford, A., Schulman, J., Sidor, S., Wu,
319   Y., and Zhokhov, P. (2017). Openai baselines. `https://github.com/openai/baselines`.
320 d'Heurle, F. M. (1971). Electromigration and failure in electronics: An introduction. *Proceedings of the*
321   *IEEE*, 59(10):1409–1418.
322 Dutta, T. and Barnard, A. R. (2017). Performance of hard disk drives in high noise environments. *Noise*
323   *Control Engineering Journal*, 65(5):386–395.
324 Fishman, G. S. (1978). Principles of discrete event simulation.
325 Hamidi, A. and Coquery, G. (1997). Effects of current density and chip temperature distribution on
326   lifetime of high power igbt modules in traction working conditions. *Microelectronics Reliability*,
327   37(10):1755 – 1758. Reliability of Electron Devices, Failure Physics and Analysis.
328 Hill, A., Raffin, A., Ernestus, M., Gleave, A., Kanervisto, A., Traore, R., Dhariwal, P., Hesse, C., Klimov,
329   O., Nichol, A., Plappert, M., Radford, A., Schulman, J., Sidor, S., and Wu, Y. (2018). Stable baselines.
330   `https://github.com/hill-a/stable-baselines`.
331 Hushchyn, M., Sapronov, A., and Ustyuzhanin, A. (2019). Machine learning algorithms for automatic
332   anomalies detection in data storage systems operation. *Advances in Systems Science and Applications*,
333   19(2):23–32.
334 Karpov, M., Arzymatov, K., Belavin, V., Sapronov, A., Ustyuzhanin, A., and Nevolin, A. (2018). Hybrid
335   approach to design of storage attached network simulation systems. *International Journal of Civil*
336   *Engineering and Technology*, 9(11):220–226.
337 Koch, W., Mancuso, R., West, R., and Bestavros, A. (2019). Reinforcement learning for uav attitude
338   control. *ACM Transactions on Cyber-Physical Systems*, 3(2):22.

Kolesnikov, S. (2018). Reproducible and fast dl & rl. `https://github.com/catalyst-team/catalyst`.

Lakshminarayanan, V. and Sriraam, N. (2014). The effect of temperature on the reliability of electronic components. In *2014 IEEE International Conference on Electronics, Computing and Communication Technologies (CONECCT)*, pages 1–6.

Lillicrap, Hunt, Pritzel, Heess, Erez, Tassa, Silver, and Wierstra (2015). Continuous control with deep reinforcement learning. *CoRR*, abs/1509.02971.

Matsunaga, T. and Uegai, Y. (2006). Thermal fatigue life evaluation of aluminum wire bonds. In *2006 1st Electronic Systemintegration Technology Conference*, volume 2, pages 726–731.

Mitchel, T. (1996). International disk drive equipment and materials association japan. *Proceeding of Diskcon '96 International Disk Forum*, pages 1–10.

Molero, X., Silla, F., Santonja, V., and Duato, J. (2000a). Modeling and simulation of storage area networks. In *Proceedings of the 8th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, MASCOTS '00, pages 307–, Washington, DC, USA. IEEE Computer Society.

Molero, X., Silla, F., Santonja, V., and Duato, J. (2000b). Performance analysis of storage area networks using high-speed LAN interconnects. In *Proceedings IEEE International Conference on Networks 2000 (ICON 2000). Networking Trends and Challenges in the New Millennium*, pages 474–478.

Molero, X., Silla, F., Santonja, V., and 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)*, SS '01, pages 133–, Washington, DC, USA. IEEE Computer Society.

Mubarak, M., Carothers, C. D., Ross, R. B., and Carns, P. (2017). Enabling Parallel Simulation of Large-Scale HPC Network Systems. *IEEE Transactions on Parallel and Distributed Systems*, 28(1):87–100.

Muknahallipatna, S., Miles, J., Hamann, J., and Johnson, H. (2010). Large fabric storage area networks: Fabric simulator development and preliminary performance analysis. *International Journal of Computers and Applications*, 32(2):167–180.

Perles, A., Molero, X., Marti, A., Santonja, V., and 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*, pages 227–234.

Rabiei, A. and Evans, A. (2000). Failure mechanisms associated with the thermally grown oxide in plasma-sprayed thermal barrier coatings. *Acta Materialia*, 48(15):3963 – 3976.

Riley, G. F. and Henderson, T. R. (2010). *The ns-3 Network Simulator*, pages 15–34. Springer Berlin Heidelberg, Berlin, Heidelberg.

Roesch, W. J. and Jittinorasett, S. (2004). Cycling copper flip chip interconnects. *Microelectronics Reliability*, 44(7):1047 – 1054.

Sankar, S., Shaw, M., and Vaid, K. (2011). Impact of temperature on hard disk drive reliability in large datacenters. In *2011 IEEE/IFIP 41st International Conference on Dependable Systems Networks (DSN)*, pages 530–537.

Sapronov, A., Belavin, V., Arzymatov, K., Karpov, M., Nevolin, A., and Ustyuzhanin, A. (2018). Tuning hybrid distributed storage system digital twins by reinforcement learning. *Advances in Systems Science and Applications*, 18(4):1–12.

Strom, B., Lee, S., Tyndall, G., and Khurshudov, A. (2007). Hard disk drive reliability modeling and failure prediction. *Magnetics, IEEE Transactions on*, 43:3676 – 3684.

Varga, A. and 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*, Simutools '08, pages 60:1–60:10, ICST, Brussels, Belgium, Belgium. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

Wang, C.-Y., Zhou, F., Zhu, Y.-L., Chong, C. T., Hou, B., and Xi, W.-Y. (2003). Simulation of fibre channel storage area network using sansim. In *The 11th IEEE International Conference on Networks, 2003. ICON2003.*, pages 349–354.

Yeh, E. C. C., Choi, W. J., Tu, K. N., Elenius, P., and Balkan, H. (2002). Current-crowding-induced electromigration failure in flip chip solder joints. *Applied Physics Letters*, 80(4):580–582.

Zhou, Y. and Hashida, T. (2002). Thermal fatigue failure induced by delamination in thermal barrier coating. *International Journal of Fatigue*, 24(2):407 – 417.