Technology for making real-time decisions based on neural network forecasting

A.A. Starovoytov
Faculty of Applied Mathematics and Computer Science
Belarusian State University
Minsk, Belarus
starovoytovaa@bsu.by

V.V. Krasnoproshin
Faculty of Applied Mathematics and Computer Science
Belarusian State University
Minsk, Belarus
krasnoproshin@bsu.by

***Abstract*—This paper considers a current applied problem related to the construction of decision support systems for critical computing services. An original approach based on neural network forecasting is proposed, within which a method of dynamic local approximation using neural network models (DLANN) has been developed. The scheme and architecture of the control system have been designed and implemented. Experiments have been conducted, confirming the effectiveness of the method and the overall approach.**

Keywords—decision-making, information system, proactive management, uncertainty in external load, neural networks, critical IT service.

I. Introduction

Supporting critical IT services and systems (such as banking, telecommunications, and industrial systems) in an operational state with guaranteed computational resource levels is a relevant problem in today's digital society.

Uncertainty in external loads and failures of computing equipment lead to failures in the operation and performance degradation of critical IT systems. As a result, the loss of operational efficiency in processing information and conducting banking and other operations can have serious consequences, including financial losses and major incidents.

Making operational decisions for the management of critical IT services allows for the reduction or prevention of negative consequences. However, the human factor often contributes to a decrease in operational efficiency. Therefore, various automated solutions are actively being developed to enhance efficiency and proactivity in the management of critical systems.

One of the promising approaches to solving the problem is the use of intelligent systems that realize the scheme: data collection - building a predictive model - proactive decision-making - performing control actions.

Several authors have developed various critical IT service management systems using neural network models, contributing to effective decision-making [1], [2], [3], [4]. However, in these systems, only one neural network model is trained for specific types of external loads. It is assumed that these models will successfully predict the values of required parameters for other types of loads associated with uncertainty.

In these works, training datasets are prepared in advance, containing long-time series with a large number of elements. Training on such datasets takes a considerable amount of time and requires high-performance resources to effectively train models within an acceptable timeframe.

In this paper, an approach is proposed based on the idea that a managed IT system can exist in various states (in terms of computational resource capacity), and for each state during its lifetime, a neural network model can be created and trained. This model is capable of predicting the average %CPU utilization of computational modules, based on which a control decision can be made to change the system's state.

II. Model System of Critical IT Service

Due to the difficulty of developing the relevant systems in laboratory conditions (without direct interaction with the actual critical infrastructure), a model system was developed for researching the discussed issue. This model system conceptually corresponds to a critical information service model.

The system consists of a set of computational modules where instances of application software operate, and external requests are load-balanced between them. A stress system is used to generate requests, allowing for the configuration of the load and the retrieval of statistics on processed requests and response times. The model system supports a scaling mechanism (state change). Let's briefly describe the main components of the model system:

A. Computational Modules Block

In the works [1], [2], [3], [4], virtual machines (VMs) were used as computational modules for application software. These VMs were deployed in various public clouds such as Amazon, Google, and others, or in private clouds and data centers. The management of VMs (creation, startup, shutdown, deletion) was achieved using the capabilities of cloud services or data center management systems. While VMs offer good application isolation, they do require more computational resources since each VM needs resources for its operating system. Additionally, a module with functionality similar to a cloud service for managing VMs is necessary. This module handles the configuration of operating systems, installation, configuration, and management of web applications, and adheres to the Infrastructure as Code (IaC) model.

After considering various options (VM, Kubernetes Cluster, Docker Compose), it was decided to use Docker Compose for the computational modules in the model system. This decision allows for the creation of computational modules as Docker containers, their management, and the collection of various utilization metrics. There are ready-made libraries like Docker SDK for Python for working with the Docker API Engine. Load balancing across containers with the web application is accomplished using Docker Compose's built-in service naming features. The configuration of services and resources can be described in Yaml format.

B. Application Web Service Block

To save resources, ensure stability, simplify configuration, and operation, it was decided to implement an application web service as a microservice based on the popular high-

performance minimalistic web framework, Echo.labstack, written in the high-level Go programming language.

C. Load Generation Block

Two software options were considered as load generators: Apache JMeter, written in Java, and Locust, written in Python. Apache JMeter is more feature-rich but complex to configure and resource-intensive. In contrast, Locust has fewer features, is easy to configure, allows for load profile descriptions in the form of Python classes, is less resource-intensive, supports distributed configurations of instances, and allows load profiles to be defined as functions or pre-prepared data. Locust was chosen as the load generator due to its suitability for the model system.

III. Approach Description

Operational decision-making involves several key aspects:

- The adequacy of forecasting the system's key parameters.
- The speed of preparing a forecast.
- The forecasting horizon into the future.

The situation is complicated by the fact that the load behavior in critical systems can change quite rapidly. For example, the load can sharply increase within a short period of time, leading to a lack of computational resources. Additionally, the load profile can vary significantly from day to day.

The currently used approaches for making predictions based on large datasets of historical data may lose forecasting quality and require initial preparation of a training dataset covering a long observation period. In addition, they involve complex neural network models (e.g., LSTM layers with memory and others) that require extended training times on high-performance resources for the given dataset.

A pre-trained model on a large dataset may have insufficient generalization capability, and in some cases, it may not be able to generate an accurate forecast for a new load pattern. Additionally, all critical systems have entirely different load profiles, and a model trained on data from one system may not be suitable for another.

What happened with the system over an extended period of time is not important as the ability to make accurate forecasts based on a small amount of last real-time accumulating data. Furthermore, model training on this data and subsequent predictions should be carried out rapidly, in parallel with the accumulation of new data.

The main idea is that the system's load profile can be divided into small segments during operation. For each segment (during its existence), a neural network model can be constructed to describe the behavior within that segment. This model enables a set of predictions to be made, based on which the system transitions to a new segment. This process then repeats for the next segment, and so on.

A collection of models will accumulate, each of which will have better accuracy within its specific segment compared to a global model trained on data over a longer period.

In [5], a similar approach is proposed in the form of a Local Approximation (LA) method. The main idea behind this method is to divide the domain of a function into several local regions, construct approximating models, and estimate the parameters of these models separately within each region.

If the function is smooth, the regions can be small enough so that the function does not change too abruptly within each of them. This allows for relatively simple models, such as linear ones, to be applied in each region. The key condition for the effective use of LA method is the successful choice of the size of the local region, i.e., the number of neighbors.

This method was used for forecasting economic time series, where similar trends for specific days on stock price charts acted as neighbors. The main challenge lay in selecting suitable neighbors, as the quality of the forecast depends heavily on this choice. The paper qualitatively compares global and local approximation and suggests a similar idea that, compared to the global model, less informative local approximation may be preferable when accuracy is the more important criterion. A global model may not achieve the required accuracy due to the accumulation and averaging of a larger amount of data.

IV. Forecasting Method

In the considered case, the state of the critical system is related to the volume of computational resources required for the system to function normally under a specific load. The state is determined by the number of used computational modules. Data for building the model are taken from the local time segment corresponding to a specific system state. The metric used is the average %CPU utilization across a set of computational modules.

The local time segment requires specific consideration. The lifetime of a particular state depends on the nature of the load. The transition to another state is determined by the level of the measured metric (the transition threshold). Maximum and minimum levels of average %CPU utilization across computational modules are defined. When the utilization crosses the maximum threshold, the system automatically transitions to a new state with more resources, while crossing the minimum threshold leads to a transition to a state with fewer resources.

Local time segments for different states may have varying numbers of data samples. This quantity should include data samples required for model training (training also requires a certain amount of time).

Forecasting within the local time window leads to the following challenge:

Let's consider a critical system that can be in a finite set of states $S = \{S_1, S_2, \dots, S_n\}$, determined by external load on the system and transition levels. In each state i the system generates a discrete signal in the form of a short time series $X_t^{S_i}$ (or a set of time series). The task is to build a predictor based on a neural network using a portion of this signal, denoted as $\tilde{X}_t^{S_i}$, which forecasts the system's transition to a new state S_{i+1}.

To solve this problem was developed a method of dynamic local approximation by neural network models (DLANN). The essence of which is that during the operation of the system for each of its states simple neural network models are built on a part of the local time segment data (e.g., with one hidden and one output layer).

It is assumed that the model trained on a portion of the data will adequately make forecasts for the entire local segment. During training, a quality criterion is saved for each model – the validation error. After training, based on incoming data, forecasts of the average %CPU utilization across a set of computational modules are generated with a specified horizon into the future. These forecasts are compared to the transition levels, and when they are reached, a control decision is generated. This process then repeats.

For simplify, the parameters that determine the size of the time segment for training and the horizon into the future are predetermined and can be related to characteristics of specific system. Automatic selection of these parameters based on signal characteristics is also possible.

To simplify the process, neural networks with the same architecture are used. There is also the possibility of automatically adjusting the architecture (e.g., changing the number of neurons in the hidden layers or adding hidden layers) depending on the complexity of the signal, which can be assessed using some metric (e.g., entropy, variance, etc.)

There are situations where, due to a sudden change in the load's nature, it's impossible to gather a sufficient amount of data to train the model and prepare predictions for making control decisions. In such cases, the control decision is generated based on the current average %CPU utilization across computational modules, which is compared to the transition levels.

By combining the work of these two predictors, we obtain a combined control system with both reactive and proactive approaches.

Based on the method, a control system was implemented, and its operational principles and architecture are described in more detail in the following sections.

V. Model Library Method

In the process described above, various neural network models are created, each associated with a specific state determined by the number of computational modules. Each model memorizes the characteristics of a particular system state.

The set of models can be saved and accumulated as a library of neural models. There can be multiple models for each state. For each model, training quality metrics (validation error) and signal complexity assessment are saved. The library of models can be used for predictions before training a new model in various scenarios:

A. To choose the best model for a state

In this case, when the system transitions to one of the known states, a model is selected from the library based on the one with the lowest training error.

B. To create an ensemble from a set of models that correspond to one state

In this case, an ensemble (composition without training) is created from predictions of existing N models for one state, with weights associated with validation errors:

$$\hat{y}_{predict} = \sum_{i=1}^{N} \varepsilon_i \hat{y}_{model_i} \quad (1)$$

$$\varepsilon_i = \frac{\left(\frac{1}{val\_err_i}\right)}{\sum_{i=1}^{N}\left(\frac{1}{val\_err_i}\right)} \quad (2)$$

C. To create an ensemble from a set of models that correspond to different states)

In this case, an ensemble (composition without training) is created from existing models for different states, with weights associated with the signal's complexity. The formulas are similar to (1) and (2), but instead of the validation error, values related to the signal's complexity on which the neural models were trained are used.

All of these options can be used during the training of a new model as an additional predictor when the model for the current signal is not yet ready. You can compare the forecast results with the accumulating data. If the forecast is adequate, you can allow the use of predictions from this predictor for making control decisions. This has the potential to increase proactivity.

The model library allows to create stacked models that can be trained on data from a new state.

Over time, the model library will continue to grow. There is no need to store many models for the same state. Therefore, it is possible to implement a mechanism for forgetting models, where for a specific state, only a certain number of models with the lowest validation errors are retained.

VI. Operating Principle of the Management System

The management of critical IT system resources is handled by an agent that receives real-time data on the utilization of computational modules and makes decisions about scaling the managed system. The agent uses a combination of reactive and proactive control. For each state of the managed system, a separate dataset is automatically generated, a neural network model is trained based on this dataset, and predictions of resource utilization parameters are made.

The agent compares the current data on the average load across computational modules with the prediction results for a specific state of the system and makes decisions about changing the state (scaling).

If a peak in load appears and the neural network model is not ready yet or there is no prediction for the average utilization of modules, or the prediction does not give such behavior, then the reactive component is activated.

If the forecast of average parameters exceed the threshold values, then a proactive decision is made in advance to change the system's state (proactive component).

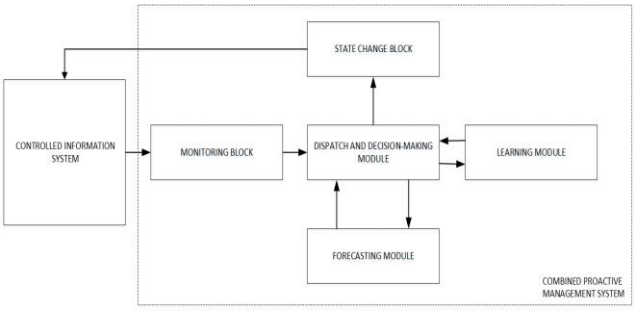

Fig. 1. Structural block diagram of the combined control system

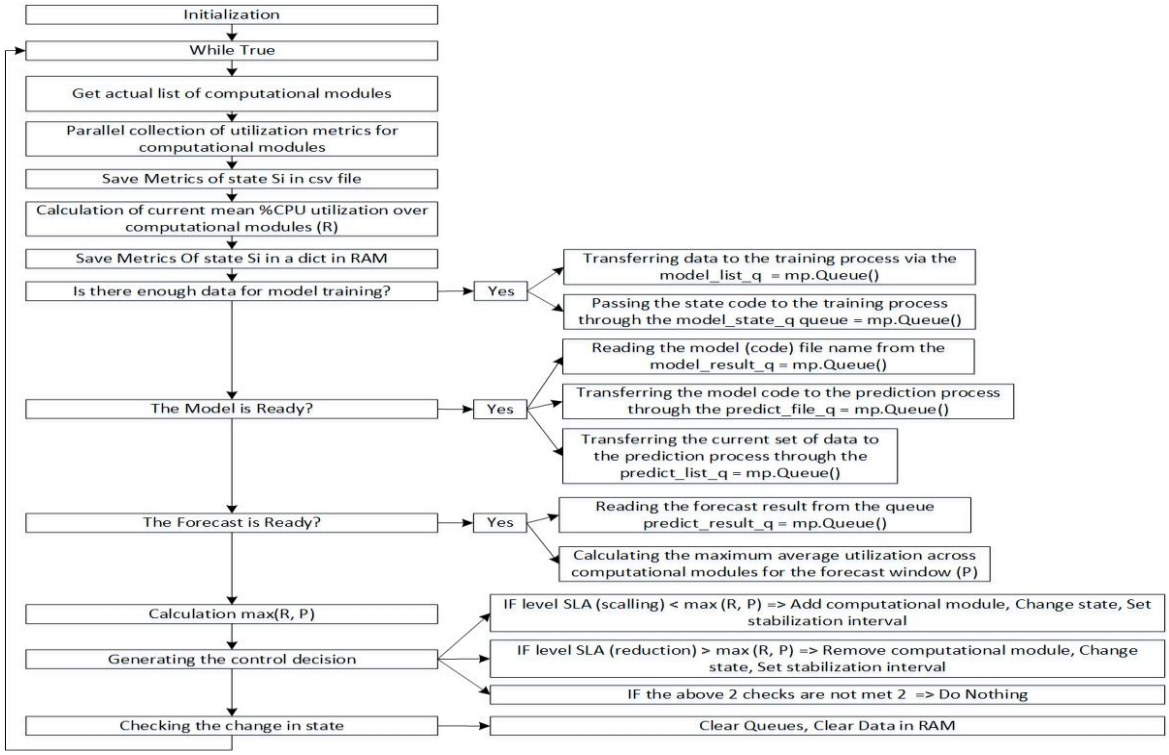

Fig. 2 Control system operation algorithm.

VII. ARCHITECTURE

The system architecture consists of 5 main blocks, which are depicted in Fig 1.

- The monitoring block is responsible for collecting performance metrics of the computational modules of the system and adding new metrics (when the system state changes).
- The state change block is responsible for sending control commands (which change the state of the managed system) and monitoring the correctness of state changes.
- The dispatch and decision-making block.
- The training module - creates a neural network model for a set of historical data for the state of the managed system.
- The forecasting module - provides forecasts for resource utilization parameters with a specific horizon into the future for the current state of the managed system.

The dispatch and decision-making block is central. It, along with the other blocks, is implemented in the high-level Python language. More details about its operation algorithm are presented in Fig 2.

During initialization, resource utilization thresholds (system SLAs) are set, reaching which leads to a change in the state of the managed system. Thresholds for resource addition (A) and removal (D) are set separately. A system stabilization parameter (cool_period) is defined, which determines the number of cycles during which no state-changing actions are performed in the system. A parameter that specifies the number of iterations for accumulating data for one state to create a model (M) is set. A lookahead parameter (Z) is also defined, indicating the number of forecast points into the future.

In the main process, after the initialization stage, there is a loop in which each iteration involves defining the list of computational modules, obtaining current utilization values of these modules, and making decisions about changing the state of the managed system.

Metrics are collected in parallel using the joblib library, with data being collected at a frequency $\sim 2\ s^{-1}$. Historical data about utilization of each computational module for different system states are saved in CSV files. Data on module utilization for the current state is accumulated in memory in a dictionary. Based on the latest received data, the main process calculates the average value over the set of computational modules (R).

If R exceeds the threshold for addition (A), a decision is made to add resources to the managed system. If R is less than the removal threshold (D), a decision is made to remove resources. After the decision is made, a command is sent to the state change block (reactive component).If R exceeds the threshold for addition (A), a decision is made to add resources to the managed system. If R is less than the removal threshold (D), a decision is made to remove resources. After the decision is made, a command is sent to the state change block (reactive component).

In this process, mechanisms of non-blocking interaction with other processes are implemented, which run in parallel to the main process and are responsible for creating and training neural networks and forecasting resource utilization parameters for a specific state with a specified horizon into the future. This mechanism is implemented using the multiprocessing library. Interprocess communication uses the multiprocessing.Queue mechanism, which forms FIFO (first input first output) queues.

Three queues (model_list_q, model_state_q, model_result_q) are used for communication between the main process and the process responsible for creating and training neural network models, and three queues (predict_file_q, predict_list_q, predict_result_q) are used for communication with the prediction process.

VIII. Network Parameters (Параметры Сети)

It's assume that each subsequent element of the series depends on some number of previous elements in the series. Lagged values of the time series are used as independent parameters for autoregression. The number of these parameters determines the time window (tw) for the series. The time window of 30 elements was selected empirically, assuming that the main contribution to the approximating function for the next element in the series comes from a linear combination of the 30 previous elements.

PyTorch is used for creating and training the model.

The preprocessing of the original series with scaling into the [0,1] interval is not performed because it introduces additional errors into the raw data and leads to additional overhead for scaling before and after training. A short time series of 64 data points is used, with a time interval of 2 seconds between data points. The total duration of the series is 128 seconds. The original series is divided into two sets: the training set (70% - 43 data points) and the test set (30% - 21 data points). Considering that the time window is 30, the number of training examples is 14. Training is performed for 100 epochs with a batch size of 1. A two-layer neural network is used with one hidden layer and one output layer. The number of neurons in the hidden layer is 90, which is three times larger than the time window size. There is one neuron in the output layer, serving as an accumulator. A fully connected linear layer (nn.Linear) is used. The activation function is ReLU, and dropout is employed for regularization with a dropout probability of 0.015. The loss function used is nn.MSELoss (mean squared error), and the optimization method is torch.optim.Adam with a learning rate of 0.002.

The training time for the neural network with the specified architecture for the dataset is ~ 2s. The prediction performing speed, taking into account a forecast horizon of 40 samples, is less than 1 second.

IX. Adding Proactivity to System Cycle

After obtaining the model for the current state, the main process transfers information about the model to the prediction process. The model file name is placed in the predict_file_q queue, and the current utilization data is placed in the predict_list_q queue. The prediction process checks for data in these queues at intervals corresponding to the data collection frequency. Once the data is read from the queues, the prediction is executed, and the forecasted data is sent to the model_result_q queue.

After receiving the forecast results for the current state, the main process calculates the maximum average utilization (P) across the computational modules for the prediction window (Z). Then, max(R, P) is computed, which represents the maximum between the current utilization value and the maximum forecasted value. This value, along with the state code, the number of computational modules, and the stabilization limit, is used to make decisions regarding system scaling. If max(R, P) exceeds the addition threshold (A), a decision to add resources to the managed system is made. If max(R, P) is less than the removal threshold (D), a decision to remove resources is made. After making the decision, a command is sent to the state change block. This introduces a proactive component into the decision-making process.

X. Example of System Operation under Workload

As an example, the system's performance is illustrated under a gradually increasing load, reaching up to 600 users performing various requests over approximately 0.5 hours.

As a result of running the system, it changed its state six times, with proactive changes occurring 4 times and reactive changes occurring 2 times.

On Fig. 3-8, examples of utilization forecasts for different states are presented (green color represents the input model data, blue color represents the forecast).

XI. Results

The article considered the problem related to making operational decisions in managing the computational resources of a critical IT service in conditions of uncertainty in external loads.

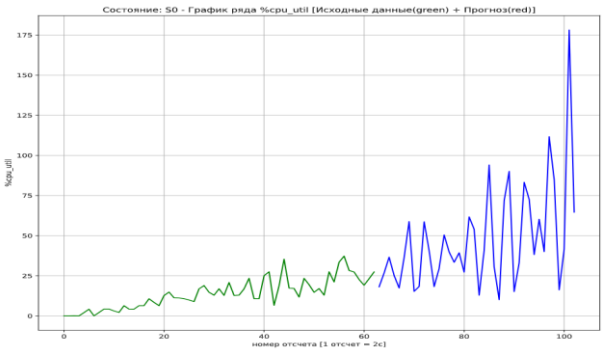

Fig. 3. Forecast 1 for state S0

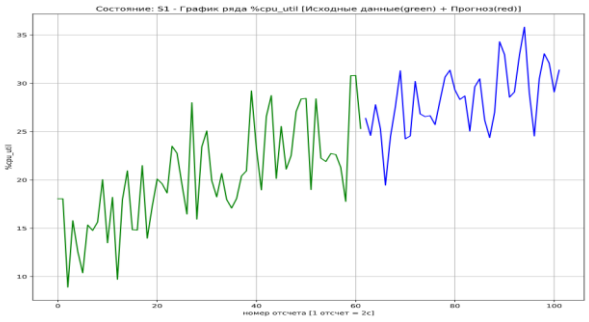

Fig. 4 Forecast 1 for state S1

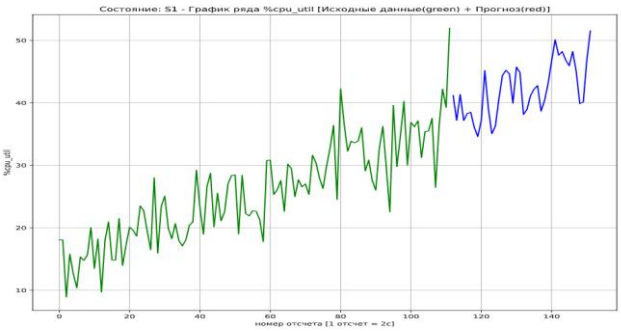

Fig. 5. Forecast 51 for state S1

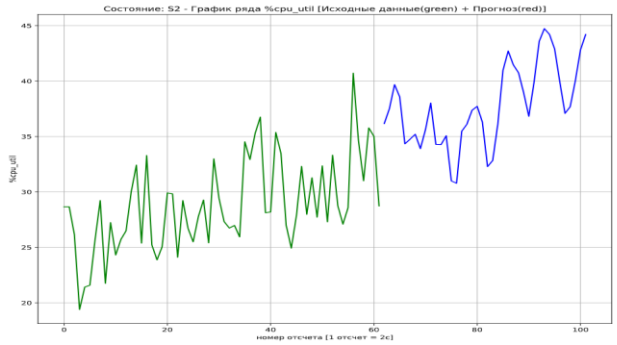

Fig. 6. Forecast 1 for state S2

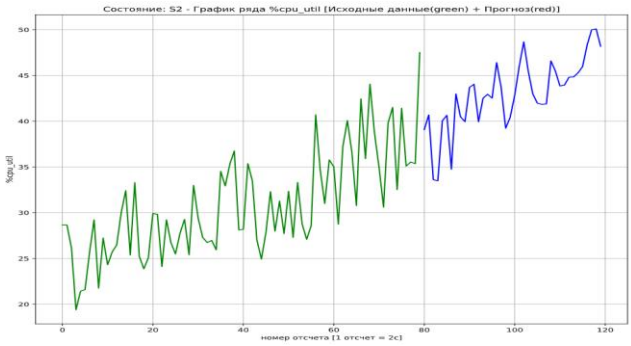

Fig. 7. Forecast 19 for state S2

As a result of the conducted research, a model system for a critical service has been developed. An original approach to solving the forecasting problem for decision-making has been proposed, which enhances the adaptive properties and stability of the managed system to external loads. Within this approach, a new method of dynamic local approximation using neural network models (DLANN) has been developed.

The structure and architecture of the combined control system have been described. A technology for making real-time management decisions has been developed, described and implemented in practice.

The experiments have been conducted that confirm the proposed technology works.

The structure and architecture of the combined control system have been described. A technology for making operational management decisions has been developed and described, which has been implemented in practice.This approach allows for the creation of a library of neural models capable of making predictions for system states. Furthermore, using this technology in the future opens up the possibility of implementing adaptation mechanisms in the system, conceptually similar to those found in the natural world. These mechanisms involve adapting to changing circumstances during an organism's life cycle to better perform their functions, one of which is survival. In this context, mutation mechanisms with natural selection are employed (where organisms that make better predictions of the current environment survive) as well as crossover mechanisms, resulting in offspring with combinations of parental characteristics.

In the context of the discussed process, each neural network model can be compared to a set of DNA encoding the characteristics of a specific state corresponding to external influences. Mutation with natural selection can be seen as the survival of those models that make better predictions of changes in resource utilization parameters, i.e., they lead to the best adaptation to a specific external influence. Crossover can be likened to the mechanism of stacking the best models. This analogy draws parallels between the evolutionary process in biology and the development of adaptive neural network models.

The iterations of building models, it becomes possible to create a model that can provide predictions for a generalized representation of the states of the managed system.

REFERENCES

[1] M. Straesser, J. Grohmann, J. von Kistowski, S. Eismann, A. Bauer, S. Kounev. Why Is It Not Solved Yet?: Challenges for Production-Ready Autoscaling // In ICPE '22: ACM/SPEC International Conference on Performance Engineering, Bejing, China, April 9 - 13, 2022, P. 105–115, ACM, 2022.

[2] N. Khan, D. A. Elizondo, L. Deka, M. A. M.–Cabello. Fuzzy Logic applied to Sys-tem Monitors // IEEE Access, Vol. 9, P. 56523-56538, 2021.

[3] B. M. Nguyen, G. Nguyen, Giang. A Proactive Cloud Scaling Model Based on Fuzzy Time Series and SLA Awareness // Procedia Computer Science (International Conference on Computational Science ICCS 2017), 108, P.365–374, 2017.

[4] V. Persico, D. Grimaldi, A. Pescape, A. Salvi, S. Santini. A Fuzzy Approach Based on Heterogeneous Metrics for Scaling Out Public Clouds // IEEE Transactions on Parallel and Distributed Systems, Vol. 28, No. 8, P. 2117–2130, 2017.

[5] Loskutov A.Yu. & Zhuravlev D.I. Application of a local approximation technique for forecasting economic indicators. Voprosy' analiza i upravleniya riskom, 2003, (1), pp. 21-31. (In Russ.).