Cloud Computing Architecture for High-volume ML-based Solutions

Mikhail M. Rovnyagin[1], Kirill Timofeev V. [2], Aleksandr A. Elenkin[3], Vladislav A. Shipugin[4]
National Research Nuclear University MEPhI (Moscow Engineering Physics Institute), Moscow, Russian Federation
[1]m.rovnyagin.2015@ieee.org, [2]timofeev.log@yandex.ru, [3]alekselenkin@gmail.com, [4]vladshipugin@gmail.com

***Abstract*—A large number of modern projects use machine learning technology to perform a variety of business calculations. There are two main ways to integrate machine-learning models into the logic of industrial applications. The first way is to rewrite models from the data analysis language (for example R or Python) to the industrial development language (for example Java, Go or Scala). The second way is to equip models with a web-interface and integrate it into the calculation. In this article, we explore the second method. A deployment architecture for machine learning in the clouds is proposed. The possibilities of the proposed scheme for scaling are described. Examples of practical use of the proposed architecture for organizing data storage with compression are also given.**

Keywords— High performance computing; ML; Docker; micro-services; distribute systems; virtualization; containerization

I. INTRODUCTION

In modern applications, machine learning models are increasingly used for a variety of purposes. For example, financial models, image recognition models, time series prediction models and many others. Previously, models were rarely embedded directly into computational processes (mainly batch execution was used with uploading model results to a file and then importing them into a computational application), today the situation has changed. Increasingly, the construction of online forecasts is required, where the model is accessed directly during the execution of user queries. Such an organization of systems requires not only the model itself and the data it processes, but also the launch and execution environment, the authorization infrastructure, and the execution of queries.

II. RELATED WORKS

Currently, a large number of works [1,2,3] are devoted to the subjecting of containerization of software modules. We can talk about both the isolated launch of software components within the framework of the virtualization technology of network functions [4,5], and the containerization of IoT programs [6]. Containerization of ML models is no exception [7,8]. The most significant practical work in this area is the creation by Microsoft Corporation of a server for launching R / python-models [9]. Also, a great contribution to the development of startup virtualization techniques was made by the companies Facebook, Google and Uber [10]. Comparison of approaches to virtualization (hypervisor vs. containers) are presented in articles [7,11].

III. ARCHITECTURE DESIGN

Based on the above mentioned studies, this article proposes an architecture for launching ML models based on containerization technologies. Each machine learning model is packaged in a separate container. In addition to the python/R code of the model itself, files with model coefficients, dependencies, configuration files, etc. are placed in the container.

Certain requirements are imposed on the model code, for example, the main function *predict* (entry point) must have a single argument string and a single return value string. The model developer must also provide the *init* initialization function, which hosts the one-time code for loading libraries and dependencies of the ML model. The model is supplied as a zip file with all the listed elements.

The model version is formed as a version of the code + version of the coefficients (version of the training set). The architecture involves wrapping the *predict* and *init* functions provided by the model developer into a REST service and then packing it into a container using the Docker [12] technology. Any model of clustering of Docker containers, for example, Docker Swarm, Kubernetes, OpenShift [13], can be used to deploy the model.

To perform experiments with the proposed architecture, the ML-RUN software framework was developed (Figure 1).

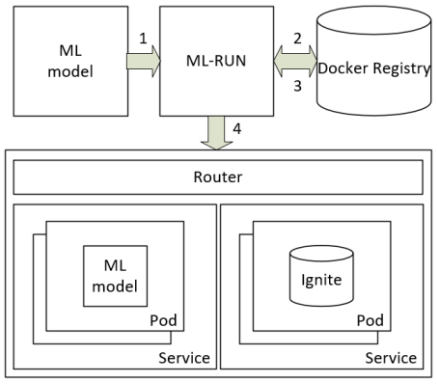

Fig. 1. ML-models execution system architecture

978-1-7281-0339-6/19/$31.00 ©2019 IEEE

Model life cycle in ML-RUN:

1. Download the zip file with the model in the software framework;
2. Formation of module wrappers and Dockerfile;
3. Building the Docker image and publishing it in the Docker registry
4. Deploy the model in OpenShift.

The created Docker container is published in the OpenShift cluster. In addition to the ML model Pods, containers with Apache Ignite, Apache Spark [14,15] and any other technologies for organizing parallel computations in memory can also be placed in the cluster. At the same time, these resources will be accessible from the code of the ML models deployed in containers.

The proposed authorization scheme is based on the Bearer token mechanism (Figure 2). According to this approach, the user addresses (1) the software framework using his login / password (or through the use of LDAP technology, etc.). In response, the user receives a token, which is valid for a limited time. The user accompanies all his requests to the model with the received token (3). The ML-model access authorization module checks the presence of the user-provided token in the cache and its lifetime. If there is no entry in the cache, the authorization module requests information about the token in ML-RUN. Next, the module updates the information in the cache and skips (or does not skip) the user request to the predict method.

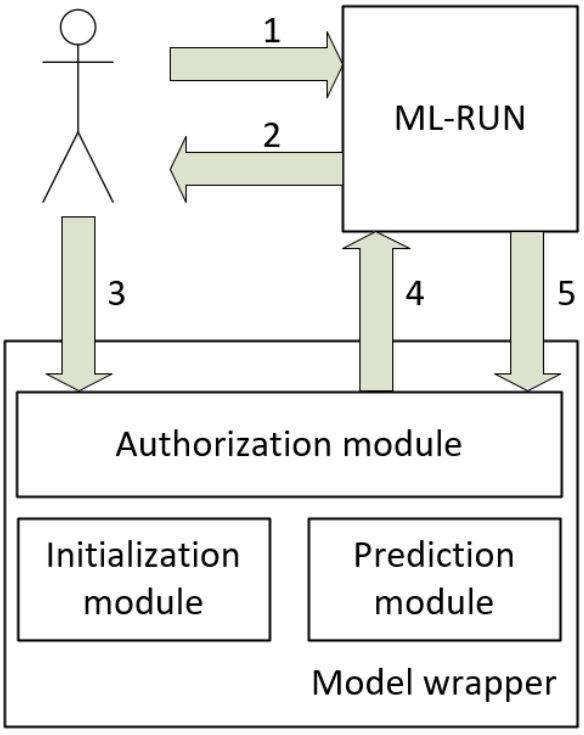

Fig. 2. Token authorization schema

IV. Design modelling as a queue system

The architecture described above can be easily represented as an open queuing network. To do this, we introduce the following notation:

- I- intensity of the incoming flow of requests into the system;
- I_w- the intensity of the flow of requests in the module-wrapper ML-model;
- λ_i- the intensity of the flow of requests in the subsystem *i;*
- p_{ij}- probability of transferring a request from subsystem *i* to subsystem *j;*
- p_{i0}- probability of transferring a request from subsystem *i* to the external environment;
- S_c- authorization token verification subsystem;
- S_t- authorization token obtaining subsystem;
- S_p- query processing subsystem in the ML-model;
- S_a- authorization subsystem;
- S_r- routing subsystem;
- K- the number of instances of the module wrapper ML-model;
- M- number of replicas of the authorization subsystem;
- N- number of routing subsystem replicas.

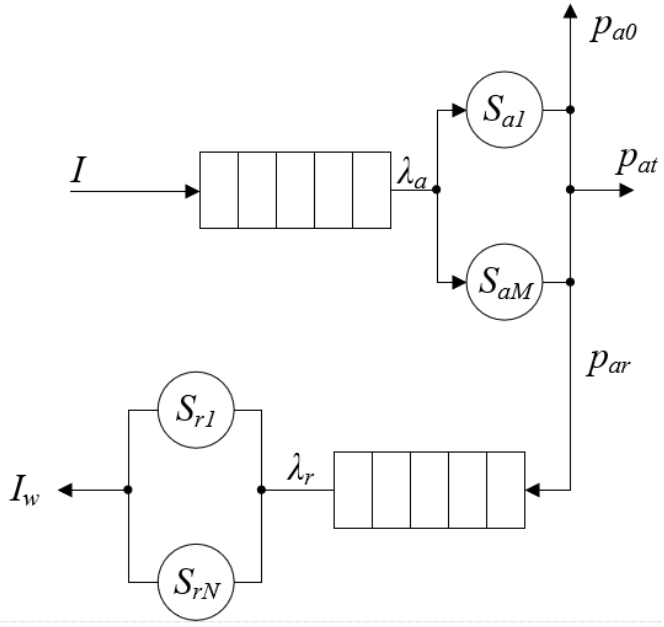

Fig. 3. ML-RUN as a queue system

The corresponding queuing network designations are shown in Figures 3 and 4. In the framework of this work, the placement of the ML-model of the same type with the number of replicas K is simulated.

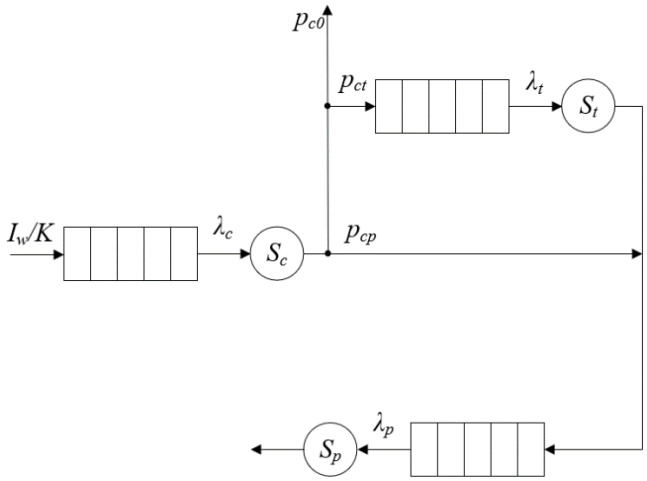

Fig. 4. Model wrapper as a queue system

According to queuing theory and based on the fact that the flow of requests to ML-models is Poisson, we can describe the systems presented above by relations (1) and (2):

$$\begin{cases} \lambda_r = p_{ar}\mathrm{I} = (1 - p_{a0} - p_{at})\mathrm{I} \\ \lambda'_a = \frac{\lambda_a}{M} \\ \lambda'_r = \frac{\lambda_r}{M} \\ \mathrm{I} = I_t + I_i \\ I_w = \lambda_r \end{cases}, \quad (1)$$

$$\begin{cases} \lambda_p = p_{cp}\lambda_c + \lambda_t \\ I_t = \lambda_t = p_{ct}\lambda_c \\ \lambda_c = I_w \\ p_{co} = 1 - p_{ct} - p_{cp} \end{cases}. \quad (2)$$

V. Experiments

In the course of the experiments, the main measured characteristic was the intensity of the flow of applications. This is a random variable, so it can be characterized by such parameters as the mathematical expectation (the estimate is the arithmetic average of the sample) and the variance (the estimate is the sample variance). According to the theory of the organization of the experiment, for an experimental study of a certain value, it is necessary to perform a series of experiments of a certain length with the established indicators of accuracy and confidence level. The arithmetic average of the sample is determined by the formula:

$$\bar{x}_V = \frac{1}{n}\sum_{i=1}^{n} x_i \quad (3)$$

where n is the number of elements in the sample (the number of experiments), x_i i-th sample value (the value of the measured parameter in the i-th experiment)

The sample variance is a biased estimate of the general variance, therefore, in conducting the experiment, the standard deviation in the sample is used, which is expressed in terms of the variance using the formula:

$$s_V = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x}_V)^2} \quad (4)$$

For a large number of experiments (n> 100), the Laplace function is used to estimate the confidence probability:

$$P\{\bar{x}_V - \delta < x < \bar{x}_V + \delta\} \approx 2\Phi(t) = p \quad (5)$$

where $\Phi(t)$-Laplace function, δ-accuracy, $\{\bar{x}_V - \delta < x < \bar{x}_V + \delta\}$- confidence interval, p- confidence probability.

For normal distribution, the accuracy is expressed by the formula:

$$\delta = \frac{ts_V}{\sqrt{n}} \quad (6)$$

Relations (1) and (2) are composed under the assumption that the flows are Poisson. For large n, the Poisson distribution can be approximated by a Gaussian distribution with $\sigma \approx \sqrt{\lambda}$. Thus, the number of experiments is determined by the inequality:

$$n \geq \left(\frac{ts_V}{\delta}\right)^2 \quad (7)$$

Thus, the sequence of actions during a series of experiments (taking into account the estimated number of measurements) can be set as follows:

1) A preliminary series of measurements is carried out (n ≥ 100), the average value of the sample is determined $\bar{x}_V$ and standard deviation s_V;
2) Confidence probability p is set, the value of the parameter t is determined in accordance with formula (5);
3) The accuracy δ is set;
4) The number of experiments is determined by the formula (7).

According to the technique written above, a series of experiments were conducted. The average values for the input request flow with an intensity of 10,000 requests per second, a confidence level of 0.95, and an accuracy of 0.0001 are shown in Figures 5 and 6.

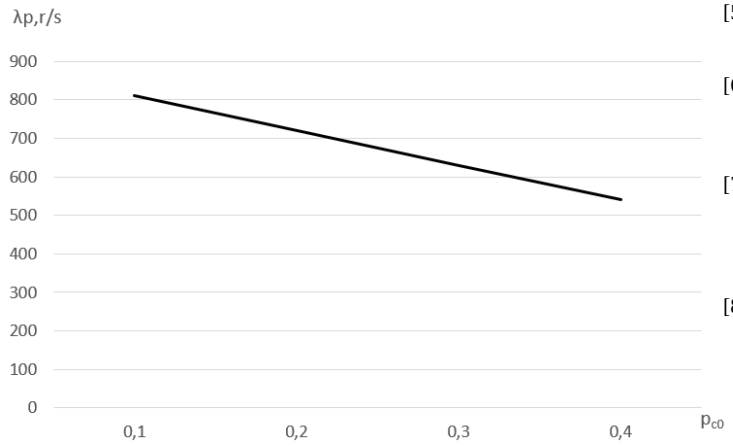

Fig. 5. Intensity of requests to ML-model on bad token probability

The intensity of requests directly to the ML-model depends on the probability of obsolescence (or non-validity) of the authorization token at the verification stage linearly.

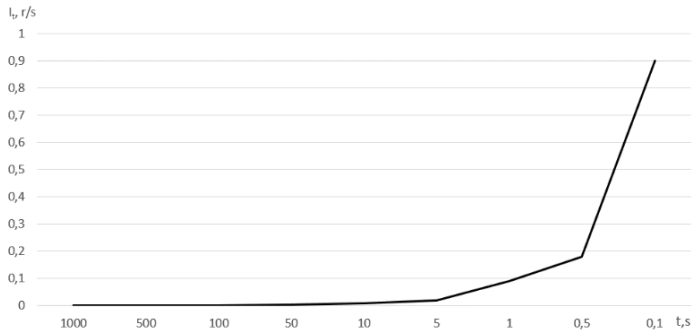

Fig. 6. Intensity of token requests on token TTL

Thus, this article presents the architecture for an isolated launch of ML-models in a distributed execution environment. The advantages of the proposed approach are: high fault tolerance and scalability (provided by a software framework for distributed launch of containers), support for model versioning, and availability of an authorization scheme for access to models. It should be noted that according to theoretical and experimental estimates, the presence of an authorization scheme based on Bearer token does not lead to significant performance losses.

References

[1] S. Kumar Pentyala, "Emergency communication system with Docker containers, OSM and Rsync," *2017 International Conference On Smart Technologies For Smart Nation (SmartTechCon)*, Bangalore, 2017, pp. 1064-1069.

[2] O. Sallou and C. Monjeaud, "GO-Docker: A Batch Scheduling System with Docker Containers," 2015 IEEE International Conference on Cluster Computing, Chicago, IL, 2015, pp. 514-515.

[3] Qiang Liu; Wei Zheng ; Ming Zhang ; Yuxing Wang ; Kexun Yu "Docker-Based Automatic Deployment for Nuclear Fusion Experimental Data Archive Cluster", IEEE Transactions on Plasma Science, 2018 , vol. 46, pp. 1281 - 1284

[4] Rovnyagin, M. M., Sergey S. Varykhanov, Yury V. Maslov, Iuliia S. Riakhovskaia and Oleg V. Myltsyn. "NFV chaining technology in hybrid computing clouds." 2018 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus) (2018): 109-113.

[5] Cziva, Richard and Dimitrios P. Pezaros. "Container Network Functions: Bringing NFV to the Network Edge." IEEE Communications Magazine 55 (2017): 24-31.

[6] Morabito, Roberto, Ivan Farris, Antonio Iera and Tarik Taleb. "Evaluating Performance of Containerized IoT Services for Clustered Devices at the Network Edge." IEEE Internet of Things Journal 4 (2017): 1019-1030.

[7] Rovnyagin, M.M., Guminskaia, A.V., Plyukhin, A.A., Orlov, A., Chernilin, F.N., & Hrapov, A.S. (2018). Using the ML-based architecture for adaptive containerized system creation. 2018 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus), 358-360.

[8] Al-Rakhami, Mabrook, Mohammed A. Alsahli, Mohammad Mehedi Hassan, Atif Alamri, Antonio Guerrieri and Giancarlo Fortino. "Cost Efficient Edge Intelligence Framework Using Docker Containers." 2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing, 4th Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress(DASC/PiCom/DataCom/CyberSciTech) (2018): 800-807.

[9] Microsoft Machine Learning Server: [Online] / https://docs.microsoft.com/en-us/machine-learning-server/what-is-machine-learning-server

[10] Introducing MLflow: an Open Source Machine Learning Platform: [Online] / https://databricks.com/blog/2018/06/05/introducing-mlflow-an-open-source-machine-learning-platform.html

[11] Walter Blair, Aspen Olmsted, Paul Anderson, " Docker vs. KVM: Apache spark application performance and ease of use" 2017 12th International Conference for Internet Technology and Secured Transactions (ICITST), 2017, pp. 199-201.

[12] Docker: Enterprise Container Platform: [Online] / https://www.docker.com

[13] OpenShift: Container Application Platform by Red Hat, Built on Docker and Kubernetes: [Online] / https://www.openshift.com

[14] Meng, Xiangrui, Joseph K. Bradley, Burak Yavuz, Evan R. Sparks, Shivaram Venkataraman, Davies Liu, Jeremy Freeman, D. B. Tsai, Manish Amde, Sean Owen, Doris Xin, Reynold Xin, Michael J. Franklin, Reza Bosagh Zadeh, Matei Zaharia and Ameet S. Talwalkar. "MLlib: Machine Learning in Apache Spark." Journal of Machine Learning Research 17 (2016): 34:1-34:7.

[15] Shanahan, James G. and Laing Dai. "Large Scale Distributed Data Science using Apache Spark." KDD (2015).