ML-based Heterogeneous Container Orchestration Architecture

Mikhail M. Rovnyagin[1], Hrapov Alexander S[2], Guminskaia Anna V[3], Orlov Aleksandr P.[4]
National Research Nuclear University MEPhI (Moscow Engineering Physics Institute),
Moscow, Russian Federation
[1]mmrovnyagin@mephi.ru, [2]arcanius@list.ru, [3]annguminskaya@gmail.com, [4]Alex567.alex@yandex.r

***Abstract* — In recent years, the popularity of containerization technologies has been growing. When they are used, computational tasks are placed in lightweight containers that can be easily moved between different computing nodes. Containerization using Docker is especially popular at the moment. The use of these solutions opens up enormous opportunities for building distributed and cluster computing systems. To maintain the operability of such systems, special tools are used, and one of them is an orchestrator. However, existing orchestrators are focused on not-so-large computing systems in which performance can be maintained by simply moving computational tasks from non-working nodes to working ones. In large systems with many nodes and a huge number of computational tasks, it is also necessary to take into account the uneven consumption of resources by various tasks. This article proposes a system architecture that can solve the problem of container orchestration using machine learning methods and given the uneven consumption of resources by**

Keywords — high performance computing; ML; Docker; monitoring; distribute systems; virtualization; containerization; orchestrator; orchestration

I. INTRODUCTION

Typically, a distributed cluster computing architecture is used to provide scalability to computing systems. This architecture can provide independence for performing computational tasks on various nodes of the cluster with the ability to quickly move processes for best use in accordance with the changing situation. Most often, the ability to seamlessly move tasks between different nodes is achieved utilizing various tools using virtualization [1].

At the same time, the current state of the computing system is hidden from the process wrapped in a virtualized entity. As a result, the use of resources by services looks like the use of hardware with potentially uncertain capabilities. The amount of resources provided to the processes does not depend on the equipment, but on the administrator's limitations. Therefore, the entire cluster can be used by processes as one unified service with always enough resources.

In fact, all user tasks performed in virtual environments are performed on a variety of real computing nodes. And the means of the system should provide constant monitoring of the availability of real hardware resources for virtualized environments so that for each task it really is absolutely not important on which node it is being performed. And since the number of real nodes is always limited, one of the most important tasks is to distribute virtualized entities between real computing nodes in such a way that they do not have a shortage of real resources.

The task of automatically most efficiently distributing container instances among accessible cluster nodes, while ensuring maximum availability of computing resources, is one of the orchestration tasks. Currently, the most promising ways to solve this problem are those that use machine learning methods. Moreover, the implementation of such solutions in real industrial systems can be done quite quickly [2].

One of the frequently described methods for solving this problem is the creation of a "performance model" [3]. It is assumed that with this model it is possible to predict the behavior of the system and the resulting characteristics when some input changes. Another method is to use machine learning to get an estimate of the usefulness of each move action that can be performed.

In any case, the described system will have a core called an orchestrator. It analyzes the current state of the system and processes the available data. However, such an orchestrator will not work immediately, since it is necessary for him to provide a working infrastructure with a special environment in which he can learn. This is what this paper is about.

The solution to this issue is not directly related to the specific virtualization system used, since it is possible to provide the ability to manage virtualization tools through a special API. Moreover, it doesn't essentially matter which of the virtualization methods is used, using the hypervisor, or using containers. If the developed orchestration system works for one of the methods, the developed architecture can, in principle, be adapted to use a different virtualization method. However, in our case, we will use a specific container system – Docker [4].

II. LINKED WORKS

There are several well-known tools for containerization systems that provide existing fixed algorithms for orchestration. For example, the most popular Docker containerization system can work with the Google Kubernetes orchestration system [5]. However, all these tools usually work using policies, for the application of which it is necessary to take into account the specifics of computing operations in containers, which can be quite tedious for the user.

978-1-7281-5761-0/20/$31.00 ©2020 IEEE

As a solution to this problem, it is proposed to use dynamic policies that use the "performance model" during operation [6]. Another well-described solution is container scaling, during which, based on available data, actions are proposed to increase or decrease the number of resources provided to containers [7-8]. But both of these solutions do not take into account the possibility of using a heterogeneous cluster, therefore their application is limited only to homogeneous nodes.

There are also articles devoted to infrastructural architectures for orchestration systems, but they usually do not offer the allocation of special components related to the learning stage, but place more emphasis on monitoring and profiling[8-9].

III. General scheme

The main objective of the presented solution is to ensure the operability of the main component of the system: the orchestrator. All other components of the system are necessary to provide the orchestrator with metrics, on the basis of which the core provides decisions on the need to move existing containers.

The general scheme and its basic elements are presented in Figure 1. As can be seen from the scheme, data on the cluster state and all metrics that describe the processes are sent and stored in the metrics store (blue arrow in figure 1). The orchestrator extracts time series from the store (blue arrow in figure 1) and, based on their analysis, tries to change the current state of the system. Then the corresponding requests for the necessary movements are sent to the containerization system (green arrow in figure 1).

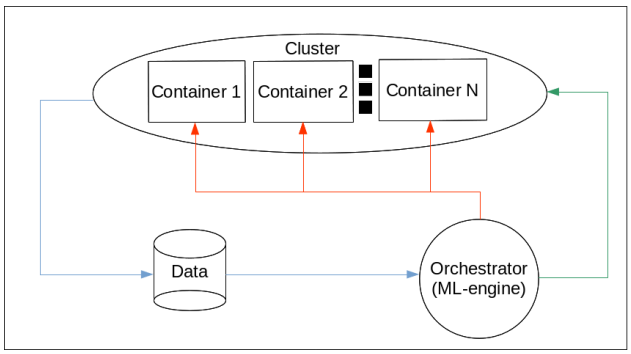

Fig. 1. General scheme of system

In the case under consideration, the orchestrator uses machine learning methods. In fact, it is an ML-engine, the effectiveness of which depends primarily on how the learning process takes place. Therefore, one of the tasks is to provide the ML-engine with a variety of data, based on which decisions can be made. This can be done by placing the system in the most diverse possible situations.

Thus, the general system should actually be able to stay in one of two modes:

- Training mode - in this mode, the ML-engine turns out in various situations, staying in which gives a diverse variety of metric time series. A change of situations should occur sometime after the decision of the machine learning engine in order to save the result of the decision;
- Usage mode - in this mode, the ML-engine works in a variety of situations. After each decision is made by the engine, the containers move in accordance with the recommendations and remain in this state until the next decision of the ML-engine.

It is necessary to provide the ability to generate a given resource consumption on command to enable the system to transition to a suitable state. This can be achieved by organizing a special subsystem for generating load on containers. A schematic representation of such a subsystem also is shown in Figure 1.

Instead of standard containers with user services, during training special containers with the system service stored in them can be used. Its main task is to start consuming the required amount of resources upon request, as a result of which actually increases the container load on the node. But standard containers can also be used, and the correlation of their load depending on requests should be pre-measured.

The orchestrator, which is in training mode, will distribute commands on containers and control the organized load (red arrow in figure 1).

IV. System components

The container orchestration system consists of a large number of various subsystems, each of which independently solves rather diverse tasks. Among these tasks, one can specify the tasks of monitoring and collecting statistics on the current state of the system, the task of visualizing the current state of the system, the task of ensuring a complete environment during the learning process, etc.

In some cases, it is required to develop and implement their own tools. However, to solve many of these tasks can be used ready-made solutions and tools, which greatly simplify the work. Some of these tasks, such as the monitoring task, can be solved using classical tools [10]. However, for systems using container virtualization, specialized tools have appeared that are often used in such situations [11].

If we try to describe all the components that are part of our orchestration system, we can reduce them to the following list:

- Metric collection agents - located directly on the computing nodes and collect metric parameters about the state of each node, as well as collect data on the characteristics of each container. Data collection from containers is performed in such a way as if it was not known about the existence of the host machine and the virtualized space was represented as a real node. In this case, cAdvisor [12] agents are used as agents of this kind;
- Monitoring server - a server that periodically polls existing agents associated with it, as well as collects metrics from them. In addition, this server accepts requests from third-party programs for providing the

collected metric data, and also preprocesses the sent metrics in accordance with the accepted request. In our case, the Prometheus [13] server is used as a monitoring server;

- Database management system - special software that organizes and stores the received metric data. The work of such a system is carried out, first of all, with time series and other statistical data, and, in this regard, suitability for processing and manipulating large volumes of this kind of information is critical. In our case, the built-in system of Prometheus itself is used as a database management system;
- Containerization system - software that performs the tasks of maintaining the operation of virtualized containers. It provides the tasks of creating, pausing, starting and moving containers from node to node based on the received commands. In our case, Docker is used as a similar system, without any additional tools;
- The machine learning engine (ML-engine) is the core of the system we are developing. Provides analysis and processing of metric data supplied to it. Based on the data obtained, a decision is made on the most optimal distribution of containers among computing nodes. The development and implementation of such a machine learning engine is one of the complex objectives;
- Load server - one of the stages of system preparation is the stage of model training based on real data and attempts. To do this, each experimental container must have a server, the purpose of which is to load the container on command and use the amount of resources defined by the algorithm. This server emulates the operation of real user servers running in virtualized containers.
- Data visualization system - software that receives data from a monitoring server, as well as visualization of the received data in a convenient and pleasant for a person representation. This component is not required, since the work of the orchestration system as a whole does not need user intervention. But in our case, this allows us to increase the visibility of the actions taken. In our case, the Grafana [14] service was chosen as a monitoring system.

Among the components described earlier, there are agents whose purpose is to collect data about computing nodes and running containers. And this is enough in order to obtain statistical information. In principle, with their help it is also possible to obtain information about the activity of a container, on the basis of which to make an assumption about the activity of a particular service.

However, if you wish, you can get complete information about the state of the service from inside the container, after it is fully launched. For this, it is necessary that some process running inside the container informs the monitoring server about its operability, and, as a result, about the full operability of the launched service.

Therefore, another component can be added as system components, although it is not required:

- Container activity agents - from this agent, the metrics collection server periodically collects data, mainly information about its activity. This allows you to correlate metric data about the container with indications of its real activity.

V. System Architecture

The proposed orchestration system is based on the use of machine learning algorithms. The processed data is metric data coming from each node of the system and from each container running on the described nodes.

Therefore, one of the important tasks during the development of the system is the development of the architecture of the subsystem that provides the process of continuous collection, transmission, receipt, storage and provision of metric data. But the implementation of this task may in itself lead to the expenditure of significant computing resources. It follows that for its solution it would make sense to allocate a separate computing node.

The general architecture diagram of the described system is shown in Figure 2.

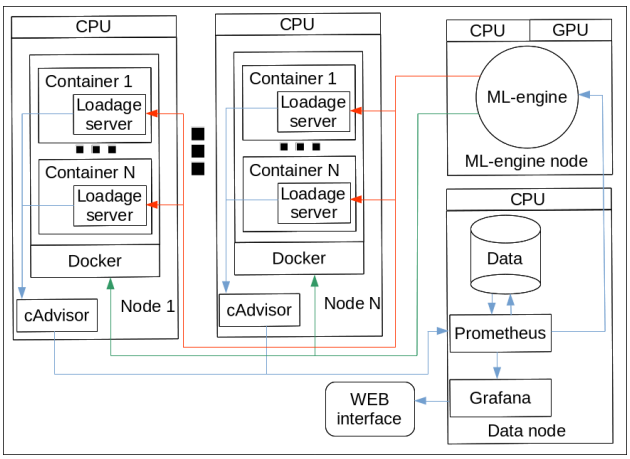

Fig. 2. System architecture

As can be seen from the diagram, there are three types of nodes in the described system:

- Computing nodes - the most numerous type of computing machines in the system. It is their computing power that is distributed between users and used by user services;
- Node with data — the entire infrastructure for collecting and storing metric data is related to the operation of this node;
- Machine learning engine node - the system core is located on this node, making the decision on the need to move containers between nodes.

From a practical point of view, if necessary, the software located on the nodes of the second and third type can be located in one place. This will reduce the equipment required

for work and provide an opportunity to use the vacant machine for the tasks of other computing nodes.

On each computing node used to provide resources to user services, there is a Docker service that supports the operation of virtualized containers. Each node has a cAdvisor agent that periodically polls data from the node and each running container.

The data collected by them is taken by the Prometheus server located on the data node. The metric indicators collected by him, by means of Prometheus itself, are stored in a database connected with it (blue arrows in figure 2).

The Grafana service located on the same node provides a graphical WEB interface for ordinary users to visualize time series. To do this, Grafana makes specialized requests to the Prometheus server, which performs the extraction of the metrics from the database in accordance with the request, as well as some preliminary processing, after which it passes them to Grafana, which displays the requested data in a pleasant form for a person.

The machine learning engine makes requests for information about the metric indicators of the system and the overall current configuration of the system. Based on the obtained data, it makes a prediction of the most optimal distribution of containers among computing nodes, after which it makes requests for moving containers from one node to another in such a way that the new distribution of containers corresponds to the predicted distribution (green arrows in figure 2).

A load server is running inside each container. This server, upon request from the outside (red arrows in figure 2), begins to use computing resources inside the container, thereby emulating the operation of a regular user service. However, its use occurs only at the stage of training the system. Commands for the operation of these servers in the learning process are sent by the machine learning engine, which at the stage of its training tries to simulate various situations, on the basis of which it can learn to give the most useful forecasts.

A significant and important element of the system's operability is the ability of the node on which the machine learning engine is located to have access not only to ordinary processor capacities (designated as CPUs), but also to graphic processors (designated as GPUs). A similar need arises in connection with a significant acceleration of the machine learning process when using graphics processors. Most of the existing machine learning frameworks are optimized for use on video cards due to the high ability to parallelize the computational processes performed in machine learning tasks.

VI. Experiment

To test the operability of the developed system architecture, the system was deployed and tested on a real computing cluster. As such, servers running the CentOS 7 operating system were used. The technical characteristics of the servers are shown in Table 1, and the real scheme of the working system is shown in Figure 3.

TABLE I. TECHNICAL SPECIFICATION OF NODES

Node	CPU	GPU	RAM	Disk
Node-ML	2 cores	NVIDIA Tesla K20X	16 Gb	100 Gb SSD
Node-Data	2 cores	—	4 Gb	100 Gb SSD
Node-1	2 cores	—	8 Gb	100 Gb SSD
Node-2	2 cores	—	8 Gb	100 Gb SSD
Node-3	2 cores	—	8 Gb	100 Gb SSD
Node-4	2 cores	—	8 Gb	100 Gb SSD

The Node-ML node, the only one with access to a video card, is designed to deploy a machine learning engine. The Node-Data node is designed to store metric data collected, and it also hosts monitoring servers. Node-1 nodes Node-2, Node-3, and Node-4 are the compute nodes on which the containers are deployed.

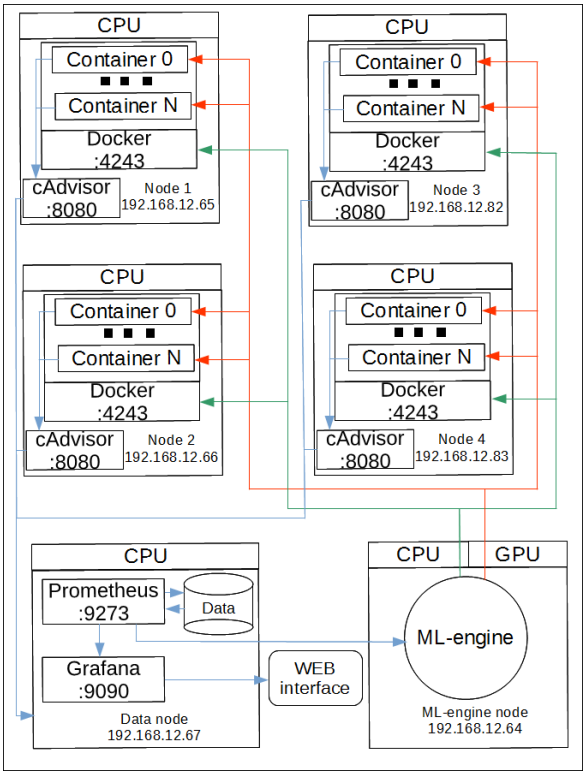

Fig. 3. Scheme of experiment

VII. Conclusion

Thus, this article proposes a new way to organize a container orchestration system. The system is intended for use in conjunction with a machine learning engine. Its use can significantly simplify the training of an ML agent, providing him with real data for work, including the possibility of receiving them directly during training.

References

[1] Andrea Tosatto, Pietro Ruiu and Antonio Attanasio, "Container-based orchestration in cloud: State of the art and challenges," in 2015 NinthInternational Conference on Complex, Intelligent, and Software Intensive Systems(CISIS). IEEE, 2015, pp. 70–75.

[2] Mikhail M. Rovnyagin, Kirill V. Timofeev, Aleksandr A. Elenkin and Vladislav A. Shipugin, "Cloud Computing Architecture for High-volume ML-based Solutions" in EIConRus., Moscow, Russia, 2019, pp.315-318.

[3] Emiliano Casalicchio, "Autonomic Orchestration of Containers: Problem Definition and Research Challenges," in 10th EAI International Conference on Performance Evaluation Methodologies and Tools. EAI, Taormina, Italy, 2016,

[4] Enterprise Container Platform | Docker. [Online].Available: https://www.docker.com/

[5] Production-Grade Container Orchestration - Kubernets. [Online].Available: https://kubernetes.io/

[6] Alejandro Pelaez, Manish Parashar and Andres Quiroz, "Dynamic adaptation of policies using machine learning," in 2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), Cartagena, Colombia, 2016, pp. 501-510.

[7] Fabiana Rossi, Matteo Nardelli and Valeria Cardellini, "Horizontal and Vertical Scaling of Container-based Applications using Reinforcement Learning," in Proc.of IEEE CLOUD '19., Milan, Italy, 2019, pp.329-338.

[8] Akkarit Sangpetch, Orathai Sangpetch, Nut Juangmarisakul and Supakorn Warodo, "Thoth: Automatic Resource Management with Machine Learning for Container-based Cloud Platform," in Proceedings of the 7th International Conference on Cloud Computing and Services Science - Volume 1: CLOSER, Porto, Portugal, 2017, pp. 103–111.

[9] Mikhail M. Rovnyagin, Anna V. Guminskaia, Alexey A. Plyukhin, Aleksandr P. Orlov, Fedor N. Chernilin and Alexander S. Hrapov, "Using the ML-based Architecture for Adaptive Containerized System Creation," in EIConRus., Moscow, Russia, 2018, pp. 358-360.

[10] Mikhail M. Rovnyagin, Viktor V. Odintsev, Dmitrii Y. Fedin, "Cloud Computing Architecture for High-volume Monitoring Processing," in EIConRus., Moscow, Russia, 2018, pp. 361-365.

[11] Emiliano Casalicchio and Vanessa Perciballi, "Measuring docker performance: Whata mess!!!" in Proceedings of the 8th ACM/SPEC on InternationalConference on Performance Engineering Companion, ser. ICPE '17Companion., New York, NY, USA, 2017, pp. 11–16.

[12] google/cadvisor: Analyzer resource usage and perfomance characteristics of running containers. [Online].Available: https://github.com/google/cadvisor

[13] Prometheus - Monitoring system & time series database. [Online].Available: https://prometheus.io/

[14] Grafana: The open observability platform | Grafana Labs. [Online].Available: https://grafana.com/