Algorithm of ML-based Re-scheduler for Container Orchestration System

Mikhail M. Rovnyagin[1], Hrapov Alexander S[2], Dmitriev Sviatoslav O[3], Kozlov Valentin K[4]
National Research Nuclear University MEPhI (Moscow Engineering Physics Institute),
Moscow, Russian Federation
[1]mmrovnyagin@mephi.ru, [2]arcanius@list.ru, [3]sodmitriev96@gmail.com, [4]kozlov.valent@gmail.com

***Abstract* — Due to the gradual increase the number of companies that use cloud technologies, there is an increase the number of the enterprises deploying and using an internal private cloud. Due to this trend, there is a growth of interest in various technologies that ensure the efficiency of the cloud infrastructure. One of such technologies is the orchestration technology, and the core of which is a scheduler - a special component that allows efficiently distribute virtualized entities with running tasks across computational nodes. However, schedulers usually only plan the locations schemes of tasks that was not started yet; often they do not plan to make changes to the arrangement of already running entities. To create the plan of changing the state of already running tasks deschedulers and reschedulers are additionally used. This article proposes a solution using a Reinforcement Learning based rescheduler and an algorithm of its preparation.**

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

I. Introduction

In the last decade, there has been an unceasing growth in the popularity of cloud computing in comparison with classical methods of deploying a computing infrastructure [1]. These changes are especially relevant for commercial companies that do not directly specialize in working in IT, but the costs of IT infrastructures for which can be serious. Therefore, the use of cloud computing allows them to significantly reduce the overhead costs of maintaining their own IT infrastructure, reduce possible costs for scaling, and also use resources in the manner of utilities, paying only for the consumed capacity according to the tariff.

However, it is not only the large cloud service providers are actively interested in building clouds for leasing computing resources. Many companies create their own private clouds both for use in deploying commercial products under development for external use, and for the internal needs of their own departments. As a result, they receive such advantages as fast system scaling, ensuring a high level of availability of company services and the ability to continuously deploy software. In such cases, the owners of private clouds can either completely not use the services of external providers, or build a hybrid solution, allocating not very important functions to outsourcing.

In such a situation, when companies use their own equipment, the issue of the severe limitation of the available hardware capacities becomes relevant. Therefore, for the continuous and reliable availability of the services provided, it is required to provide a sufficient amount of resources requested by the software. At the same time, the cloud technologies used are based on technologies such as clustering of computational nodes and virtualization, which can be performed using virtual machines or using containerization. Therefore, under these conditions, the problem of distributing computing resources between services can be formulated as a problem of the correct distribution of the existing virtualized entities between different nodes of the cluster. Creating a good distribution scheme between nodes allows each service, wrapped in a container or virtual machine, to receive the requested resources in full, and also have some additional reserve in case of load increase.

To automate the solution of this task, special complexes for load orchestration are used - orchestrators. They allow you to represent the existing set of computational nodes as a cluster, and to perform operations for deployment, coordination, balancing and management of virtualized entities in the described cluster, automating many routine operations. In some cases, they may be required to take more intelligent actions, such as processing various logs and notifications[2].

One of the important components of an orchestrator is the scheduler. It performs the task of estimating the current state of the entire cluster and calculating priorities that determine which of the available nodes is the most suitable for executing entities preparing to launch on it. In fact, it designates on which hardware which container (or group of containers) will run on. Subsequently, the scheduler does not analyze the state of already launched objects, it is only interested in the available load, which helps to evaluate the most suitable candidates for launching subsequent objects on them.

In the resulting situation, the scheduler does not estimate the situation over time, as a result, the need for resources from already distributed virtualized entities can change dramatically. Indeed, after its initial placement, the load on some services can significantly increase and continue to grow, and on some decrease. Because of this, a layout that was satisfactory at first may become ineffective due to the dynamic nature of the system requirements for services. It is possible that as a result of this, the process of timely provision of access to hardware resources at an object with a load that has begun to grow may be disrupted.

An illustration of this situation can be seen in Figure 1. The initial distribution pattern was fairly balanced. However, the resource requirements of the containers at the 1st node increased, and at the second, they decreased. As a result, the current scheme has ceased to fit the needs of the system.

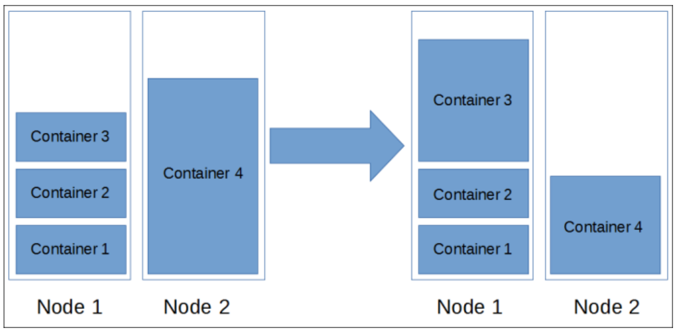

Fig. 1. An example of changing the state of the environment

One solution to this problem is to create a re-scheduler that will allow the orchestrator to change the existing distribution scheme in accordance with the current state of the cluster. This article proposes the use of a rescheduler, whose work is based on the use of machine learning algorithms, and more specifically on reinforcement learning. Unlike existing similar tools, the proposed rescheduler processes information about the current situation in the form of time series, which can allow estimate the state of the cluster not only based on the presentation of the current moment, but also based on the description of some previous state. But at the same time, the success of the rescheduler's work depends on how successfully his initial preparation was made.

This article proposes a rescheduler that allows the orchestrator that uses it to solve the described issue, and also presents an algorithm for its preparation. In the following sections, the Docker containerization system is chosen as the virtualization mechanism due to its long-growing popularity[3]. In theory, the described rescheduler can be similarly configured to work with other containerization systems, and to work with virtual machines.

The rest of the sections in this article are organized as follows. The second section mentions related work on this topic. The third section provides an overview of the system in general. The fourth section describes the preparation algorithm. The fifth section contains information regarding the experimental verification of the algorithm. The sixth section is the conclusion of the work.

II. Linked works

Existing popular orchestration systems prioritize container resiliency rather than placement efficiency. Therefore, the problem of resheduling becomes relevant for them only in a situation of failure of one of the nodes and the need to restart the disconnected containers on the operating equipment[4]. However, in the case of popular systems, there are additional components that can perform descheduling: destruction of non-optimal containers in order to relocate them using the standard scheduler [5]. However, the problem with this solution is the use of rough static rules that do not always take into account the state of the system.

There is another approach in which this problem is considered as a packing problem in a linear space [6-7]. In such cases, it is important to become not only monitoring the nodes, but also for each of the containers, which allows the assessment of the container size. However, such an approach does not allow taking into account the dynamic processes occurring inside the container, although it shows itself well in the case of a predetermined specific task performed in a container with previously known coefficients[8].

Another approach suggests the use of dynamic policies that actually use some kind of predictive model and allow scaling existing tasks according to the prediction [9-10]. However, in this case, only homomorphic nodes with the same fixed characteristics can be used.

III. General scheme

To implement the described solution, it is necessary to prepare a basic infrastructure framework, which will allow not only to prepare the necessary environment for work, but also to take on the solution of issues related to routine operations that do not affect the operation of the developed mechanism. Thus, the presented rescheduler is only a component of a complete system that performs the functions of basic orchestration and maintaining the operation of containers in the cluster. The architecture can be based on any of the existing described implementations [11], which, despite significant differences, have some similar elements.

A general view of the architecture of such a system is shown in Figure 2 as follows. There is some environment in which the execution and operation of containers takes place. In this case, such an environment is a cluster, with a set of software deployed on its nodes to provide containerization of virtualized entities. To abstract from working with specific nodes and containers, there is a separate module, the manager, which accepts all commands to start, suspend, restore and move containers in the cluster, allowing another components not to access the nodes directly.

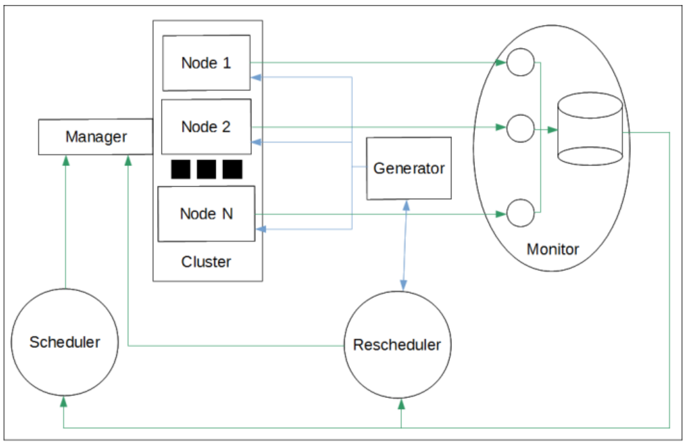

Fig. 2. General scheme of system

At the same time, the manager receives commands from the corresponding component that composes the container launch plan and associates each of them with the corresponding working node - the scheduler. It makes a decision on further actions when a new previously unidentified task appears.

The planner extracts data for its assessment of the situation from the analyzer, which produces a complete collection of information about the environment and performs the primary processing of the information received, as well as providing access to historical information. With its help, you can, in theory, receive notifications about significant changes in the environment.

Acting according to logic similar to the logic of scheduler, the rescheduler also receives data about the current state of the system from the analyzer, but its purpose is not to schedule for tasks in the queue, but to reschedule existing containers in accordance with the principle of maintaining a balance under changing conditions. Therefore, for its assessment, it is not the state of the nodes is more useful, but the state inside the containers and the dynamics of events occurring to them. Using a mechanism for identifying patterns and predicting the consequences of subsequent changes, the rescheduler estimates the degree of usefulness of making any changes, as well as predicting the new most profitable layout of running containers.

The Rescheduler may have to interact not only with the manager, which directly makes changes to the environment, but also with the scheduler. Scheduler may not be able to independently correlate changes in the environment and layout, and try to work in accordance with the previously developed plan. Therefore, to avoid this, it may be necessary to send immediate notifications of necessary changes.

The mechanism of the proposed rescheduler is based on the use of reinforcement learning, so its core will be a machine learning agent. This agent will not be able to start acting immediately, and needs a preliminary training process, during which he will interact with the same environment as during the use phase. Therefore, the machine learning engine can operate in two main modes: learning mode and prediction mode. It is in the prediction mode that the main practical use of the system takes place.

The training mode is needed to train the agent in various situations similar to practical ones. Therefore, in this mode, the load on containers will not be due to receiving real user requests for processing, but due to receiving training requests from a special load generator. Its use allows using a special algorithm for a short time to recreate around the agent a large number of various dissimilar situations, similar to practical ones, but at the same time having a low probability of occurrence in each individual case.

IV. Algorythm

The algorithm of the learning process is a set of various experiments in which the machine learning engine is set up in various situations in which it must find the best behavior strategy by trial and error. In this case, not only the variety of situations encountered is important, but also the opportunity to experiment for a long time in one situation, because this allows the opportunity to get any positive result in it.

Thus, the best way to organize a sequence of experiments that implements the learning process is to represent it in the form of an iterative algorithm. In our case, it can be represented as two loops in each other:

- loop of episodes;
- loop of attempts.

The existence of a loop of episodes is associated with the need to create a variety of experiments, at the beginning of each a new situation occurs with a new distribution of the load over tasks, which differs from the previous situation. At the beginning of this loop, random generation of new levels of load on containers is performed, after which the required commands are sent to the corresponding load servers located in each running container.

The loop of attempts exists because of the need to provide the machine learning engine with the ability to experiment with a situation for some specified time. This loop is located inside the loop of episodes, and there are no actions that change the load on the containers. Instead, it is a direct attempt by the machine learning engine, through predictions, to make changes in the distribution of containers across nodes.

```
1: procedure Training(agent, current_layout)
2:        for i ← 0 to episodes do
3:            for c ∈ containers do
4:                GENERATE_LOADAGE_LEVEL(box_num)
5:            end for
6:            for j ← 0 to attempts do
7:                state ← GET_CURRENT_STATE()
8:                new_layout ← PREDICTION(agent, state)
9:                REARRANGE(new_layout)
10:               new_state ← GET_CURRENT_STATE()
11:               reward ← ESTIMATE(new_state) - ESTIMATE(old_state)
12:               PUSH (agent, state, current_layout)
13:               PUSH(agent, new_state, new_layout, reward)
13:               current_layout ← new_layout
14:           end for
15:           LEARN_STACK(agent)
16:       end for
17: end procedure
```

Fig. 3. Rescheduler learning algorithm

Thus, the general algorithm of the machine learning engine during the training process is described in Figure 3. It shows the two training cycles described earlier, as well as the process for generating a new load on containers. The algorithm for work must receive information about the current layout and the ability to interact with the machine learning agent and each of the containers. In addition, the number of iterations for the episode loop and the attempt loop must be predetermined. During each iteration of the described cycle, direct learning occurs, which consists of the following several actions.

During each iteration of the sequence of episodes, before starting the execution of the attempts loop, further load values are generated for each container. The values for the load level are generated as follows. For each loaded container in the system, for each individual loaded resource, there is, with a certain degree of probability, either the generation of a conditional shutdown of the consumption of this resource, or the generation of a random value that corresponds to the value of the occupied resource. This scheme allows generating not only a situation with the load on containers, but also a situation with the absence of a significant use of any resource.

During the loop of attempts, first of all, the available metric data about the state of the load on the computational nodes and containers are retrieved from the monitoring server. Next, information is retrieved about the current distribution of running containers among computational nodes. In fact, after completing these two steps, all the necessary inputs for the trained model were obtained.

At the next step, the process of transferring the previously received input data and obtaining from the agent predictions about the optimality of various variants of the system distribution at the current moment takes place, and on the basis of these data the agent returns the most optimal placement. It should be taken into account that to ensure the diversity of the attempts made by the model, with a certain degree of probability, not a real prediction, but a completely arbitrary distribution can return. This avoids following only one randomly chosen, and possibly not the most optimal strategy at the beginning of experiments. Obviously, with an increase in the number of experiments performed, it is necessary to decrease this probability, due to the increase in the experience accumulated by the model.

After that, based on the obtained prediction of the most optimal placement, an appropriate movement of containers is performed in order to bring the current distribution of containers in accordance with the selected state. In theory, it is not necessary to try to achieve the most optimal state, since the need to make a large number of container movements makes a rather significant part in the efficiency of work. In some cases, the reward received as a result of the move is so small that performing an extra move operation is absolutely redundant.

After the displacements have been made, the current metric indicators of the system are extracted again, while the current distribution of the moved containers among the nodes of the system must correspond to the one proposed by the agent earlier.

Based on the data obtained, the process of estimating the usefulness of the current state and the estimating of the previous state takes place. The difference in these estimates is the reward of the model from the transition to this state. The reward can also have a negative value if the machine learning engine, by its actions, has worsened the state of the entire system. The main goal of the engine is to achieve the highest possible reward.

After calculating the reward, all sorts of information is stored on the stack before and after the movement of containers, as well as the reward received for making this movement. This stack is a kind of repository of experience accumulated by the machine learning engine during attempts during the current load of containers. Before moving on to the next load of containers, the experience gained in the stack is applied, as a result of which the model is normalized, and at the next iteration of the episode cycle it shows the best results.

V. Experiment

In order to check the applicability of the described algorithm suitable architecture | was deployed on a real computing cluster. Servers with OS CentOS 7 were used as nodes of the computing system, Docker was chosen as the containerization system. One of the existing described architectures was taken as an infrastructure basis for the orchestrator[12]. Table 1 describes the server specifications.

TABLE I. Technical specification of nodes

Node	CPU	RAM
Node-1	2 cores	16 Gb
Node-2	2 cores	4 Gb
Node-3	2 cores	8 Gb
Node-4	2 cores	8 Gb
Node-5	2 cores	8 Gb

The existing nodes have a different amount of RAM, which made it possible to start and test the system in an environment consisting of heterogeneous nodes. Infrastructure components, auxiliary services and the rescheduler were located on a separate node that was not used for container deployment.

VI. Conclusion

This article proposes a new algorithm of preparation for reschedulers that use reinforcement machine learning. After using this algorithm, the rescheduler can work in a variety of very different situations. In addition, the training process of the rescheduler core - ML-agent is accelerated and simplified.

References

[1] K. Bilal, S. U. R. Malik, S. U. Khan, A. Y. Zomaya, "Trends and challenges in cloud data centers," IEEE Cloud Computing, vol. 1, no. 1, May, pp. 10–20, 2014.

[2] A. Orlov, M. Rovnyagin, A. Aminova, A. Guminskaia, F. Chernilin, A. Hrapov, "Using the machine learning methods for resource management of high availability broadcasting containerized system," Procedia Computer Science, vol. 169, pp. 773-779, 2020.

[3] D. Merkel, "Docker: lightweight linux containers for consistent development and deployment," Linux Journal, vol. 2014, no. 239, Mar., pp. 76-91, 2014.

[4] M. A. Rodriguez, R. Buyya, "Container-based cluster orchestration systems: A taxonomy and future directions," Software: Practice and Experience, vol. 49, no. 5, May, pp. 698-719, 2018.

[5] OM. Ungureanu, C. Vlădeanu, R. Kooij, "Kubernetes cluster optimization using hybridshared-state scheduling framework," In Proceedings of the 3rd International Conference on Future Networks and Distributed Systems (ICFNDS), 2019, pp. 1-12.

[6] D. Zhang, B. Yan, Z. Feng, C. Zhang, Y. Wang, "Container oriented job scheduling using linear programming model," In Proceedings of 3rd International Conference on Information Management (ICIM), 2017, pp. 174-180.

[7] A. Havet, V. Schiavoni, P. Felber, M. Colmant, R. Rouvoy, C. Fetzer, "GENPACK: A Generational Scheduler for Cloud Data Centers," In Proceedings of 5th IEEE International Conference on CloudEngineering (IC2E), 2017, pp. 95-104.

[8] X. Ding, Y. Liu, D. Qian, "JellyFish: Online Performance Tuning with Adaptive Configuration and Elastic Container in Hadoop Yarn," In Proceedings of IEEE 21st International Conference on Parallel and Distributed Systems (ICPADS), 2015, pp. 831-836.

[9] A. Pelaez, M. Parashar, A. Quiroz, "Dynamic adaptation of policies using machine learning," In Proceedings of 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), 2016, pp. 501-510.

[10] A. Quiroz, M. Parashar, N. Gnanasambandam, N. Sharma, “Autonomic policy adaptation using decentralized online clustering,” In Proceedings of the 7th International Conference on Autonomic Computing (ICAC), 2010, pp. 151–160.

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

[12] M. M. Rovnyagin, A. S. Hrapov, A. V. Guminskaia, A. P. Orlov, "ML-based Heterogeneous Container Orchestration Architecture," In Proceedings of 2020 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus), 2020, pp. 477-481.