Presentation of the PaaS System State for Planning Containers Deployment Based on ML-Algorithms

Mikhail M. Rovnyagin[1], Alexander S. Hrapov[2]
National Research Nuclear University MEPhI (Moscow Engineering Physics Institute),
Moscow, Russian Federation
[1]m.rovnyagin.2015@ieee.org, [2]arcanius@list.ru

***Abstract*—In modern world one of the most important technologies is virtualization. And one of the most promising types of virtualization is OS-level virtualization, also known as containerization. Its use greatly simplifies the task of deploying stable computing system services that are performed on suitable hardware depending on the current situation. Various additional tools are used to automating the process of managing the location of the containers. However, most existing container management tools provide only the simplest behaviors. One of the more complex tasks that cannot be solved by such tools can be represented as follows: there are several virtualized entities (containers) that can be executed on cluster nodes. Each entity contains a task that consumes a certain amount of computing resources. It is necessary to distribute entities among nodes in such a way that each of them has enough resources. This paper proposes a more complex methodology that solves the proposed problem of service management using machine learning methods.**

Keywords—ML, reinforcement learning, distribute systems, virtualization, Docker, containerization, orchestration

I. Introduction

In the modern world, cloud technologies are becoming increasingly popular, and more and more companies are using them to solve their business problems. Cluster computing infrastructure and task virtualization are often used to build cloud systems.

When using virtualization, the user of cloud services may not pay attention to the fact that his service is performed on one computing node along with the services of other users. He will always receive as many resources as requested.

This can be achieved using any type of virtualization. But recently, the OS-level virtualization has become extremely popular. This method is also called containerization for brevity, and virtualized entities generated by it are called containers. One of its advantages is greater performance compared to virtualization using virtual machines[1-2].

However, the cloud administrator must ensure that all running virtualized entities are executed correctly and each of them has enough requested amount of resources. There are various tools to facilitate this task, for example for the Docker containerization case, Docker Swarm and Google Kubernetes. However, existing tools provide only the execution of containers on working nodes, and do not analyze the current consumption of resources. As a result, a situation may arise when several entities are executed on a working node and consume a lot of resources, thereby interfering with each other's work, while the other node has no load at all.

A container location management technique using machine learning methods is proposed to solve this problem. In this case, reinforced learning is used, in which one of the key aspects is the reward function. A properly selected reward function allows you to achieve the desired system behavior without explicitly setting the work algorithm.

II. Related works

There are a lot of papers related to the task of container orchestration. Some of these papers describe the specifics of container orchestration[3], some focus on general architecture[4], and some look at specific orchestration techniques to solve specific problems[5].

Despite the fact that there are not so many articles describing the use of ML-algorithms to solve this problem, reinforced learning is widely used to solve various problems that are not similar to each other. In particular, in situations where random events play a role, models based on Q-learning are used. These algorithms are best known as convenient tools for developing AI for video games[6-7]. However, they are also used to solve more practical technical problems, for example, in the construction of cooling systems for data centers[8].

The development of agents using Q-learning is well described in various technical literature[9]. Therefore, the main tasks when using Q-learning is to determine useful features and create the principle of rewarding for this particular task.

III. General scheme

In the general case, a system using reinforcement learning has a common scheme[10]. There is some external environment about which the machine learning engine collects data. Based on the received data, It sends a control action to the environment. After each action, a new state of the environment is evaluated, and the engine receives some

978-1-7281-2572-5/20/$31.00 ©2020 IEEE

reward. The scheme in the case of the proposed methodology is depicted in Figure 1.

In our case, the external environment is computer nodes. At each such node, several virtualized containers can be deployed. All information about nodes with containers running on them is collected in the metrics store.

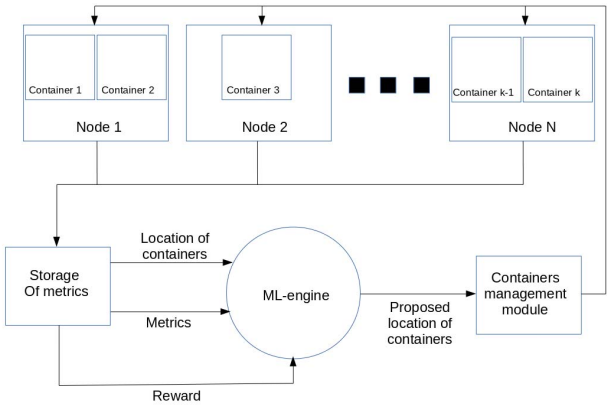

Fig. 1. General scheme of the system

There are two types of metrics that will be extracted and used in the future. These metrics are input parameters that describe the state of the external environment. The first is the current location of containers by nodes at a given point in time. Since each container can be located on any of the computing nodes, it is necessary to transmit accurate information about the location and activity of each container.

The second type is metric data that describes the internal load and the use of various computing resources. Metric data is not extracted all, but only for the last some time. The access of the engine not only to the data describing the current state of the environment, but also the previous state allows increasing the accuracy of the forecast.

The output is the forecast of the usefulness of every from possible variants of container location. The machine learning engine selects the container location variant, which is most profitable at the current time. After that, the container management module moves containers according with the requirements.

IV. Current location

Firstly the machine learning engine should have knowledge of the current containers location by nodes. To do this, it is necessary to transfer the current location of the containers on each of the virtual nodes.

However, the input data of the neural network should not have discrete values that have sharp transitions between states and where different numbers are associated with fundamentally different situations. Instead, various distributions from zero to one are best suited.

Therefore, one of the simplest ways to present containers location without using discrete values is to form a matrix. The row number is the virtual node number in this matrix, and a column number is container number located on these nodes. Each matrix element can take values 1 or 0, depending on whether the container is located on this node or not.

Thus, if the cell is a one at the intersection of a row and a column, then the container with the row number is located on the virtual node with the column number. And if the cell is zero at the intersection then it means that the container with the row number is not located on the node with the column number.

As a result, in order to find the real location of each container, you need to find the row corresponding to the container you are looking for and find the column number on which the unit is in this row. And vice versa, to find all the containers located on the host, you need to find the column corresponding to the required virtual node, and remember the numbers of all the lines in which there is one.

It should be noted that each container can be located on only one node, and cannot be located on several nodes at once. As a result of this, only one unit will always be located in each row.

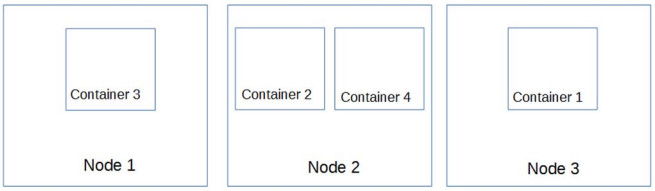

Fig. 2. Containers location example

Figure 2 shows an example of the arrangement of containers, and figure 3 shows an example of a matrix describing this arrangement. The dimension of the matrix is k × n, where k is the number of containers, n is the number of nodes.

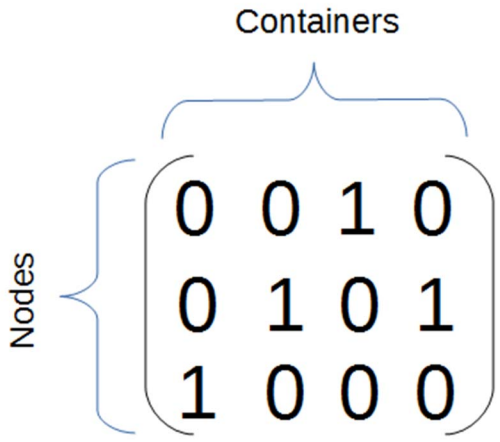

Fig. 3. Container location in matrix form

V. Current configuration

Numerous metric indicators obtained from various sources, as a rule, are a set of numbers corresponding to measurements in various metric units. They are not discrete values, and can conditionally be considered as distributions of various values.

However, it is recommended to scale input values for improving convergence and for eliminating unwanted effects [11]. Therefore, the best solution is to normalize these values relative to the maximum achieved value. This leads to the fact that all metric values will take the form of floating-point numbers and range from zero to one.

A time series with a fixed number of measurements will be extracted as metrics. Moreover, all these measurements will be the last taken values for the defined period of time. Thus, the last recorded value will correspond to the current point in time. Such a solution will make it possible to analyze those states of containers and nodes that preceded the current state.

Extracted from containers and computational nodes metrics may seem as having different types of data. However, the main thing is that all the obtained values are metrics in the form of a time series. It does not make sense to separate them by type, since the machine learning engine will independently decide the role of each obtained value, without regard to its real physical origin or meaning.

It is easy to form a time series set of many metric parameters and Figure 4 shows such values representation in a matrix form for the current case.

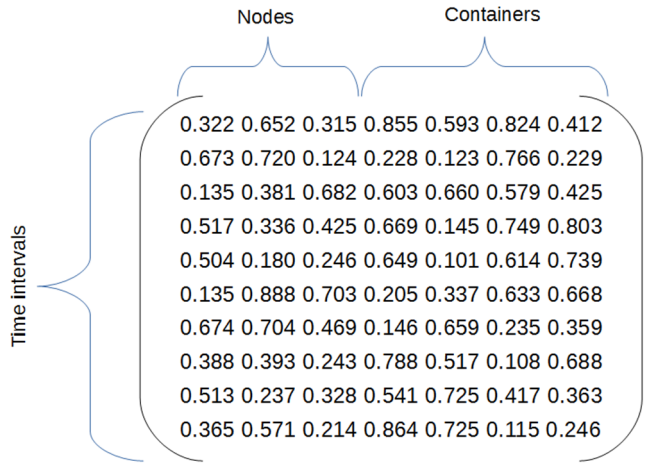

Fig. 4. Metrical time series in matrix form

Each individual row contains different values at some fixed point in time. Each individual column contains values obtained from the same source, node or container, but at different points in time. An increasing of the number of nodes and containers leads to an increase in the number of columns. An increasing of the analyzed time interval leads to an increase in the number of rows.

VI. Forecast

As an output, the machine learning engine should provide a prediction of the usefulness of each action that can be performed. At the first sight, the best solution is to associate each action with every possible movement of the container.

But here a problem arises - it is necessary to evaluate the result of each action. And building a matrix with a prediction for each possible operation is quite hard.

Therefore, another, simpler solution is used. Each output action can be associated with the entire target configuration. In other words, the machine learning mechanism at the output does not give a prediction of the usefulness of movements, but the usefulness of a specific arrangement of containers. And the container management module, knowing the target distribution of containers, will figure out how to move containers to achieve the desired state.

It remains only to associate each existed configuration of containers in nodes with a specific action that has a specific number. One option is to simply sort all existing states and try to number them in a completely arbitrary way, and then save the mutual mapping of configurations to numbers. But another option is not to use mappings, but instead to algorithmically encode the configuration number. Thus, it will be possible to determine the location of containers using only one number, and vice versa, it can be possible to convert the matrix of configuration into a number.

You can do this as follows. Since it is assumed that each container can only be located on one node at a time, it follows that for the container, you can unambiguously indicate the number of the node on which it is located. Figure 5 shows how to do this for one of the matrix options.

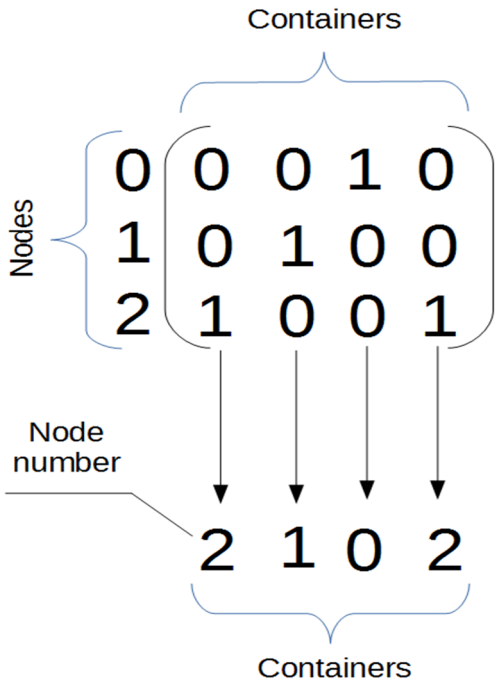

Fig. 5. Converting the arrangement of containers in matrix form to the numbers of the target nodes

As an option, the action number can be decomposed into digits in an arbitrary number system. Since each container exists only in a single copy, then the container number can be represented as the number of the digit. Thus, the container number is the number of the digit, and the node number is the weight of the digit.

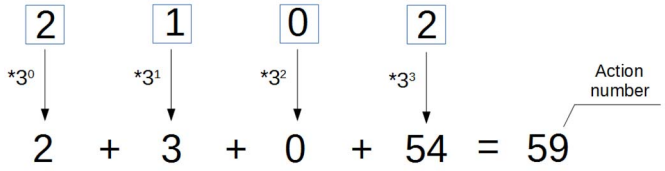

Fig. 6. Convert matrix form to action number

A result is a number in an arbitrary positional notation, where the radix of the number system is equal to the number of nodes, and the number of digits is equal to the number of containers. Consequently, the presentation of the desired arrangement of containers is reduced to one single number, which is the action number. Thus, the action number is calculated using the following formula:

$$n^0 \times i_0 + n^1 \times i_1 + \cdots + n^{k-1} \times i_{k-1} = N_{act} \quad (1)$$

In this formula, n is the number of nodes in the system, k is the number of containers in the system, i0 ... i_k-1 are the numbers of nodes on which the container with the specified number is located. Figure 6 shows an example of using the formula for the configuration in Figure 5.

The formula works in the opposite direction. To do this, you need to take a number and split it into digits using a number system where a base is a number of containers. Then you need to turn each number into a vector in which all elements are zeros, except for one, whose number is equal to a given number. This item is assigned a value one. After that, the resulting vectors will be concatenated into the matrix of containers location.

VII. Current configuration

The choice of action is made by the ML-engine based on the award received after each action. In the described problem, the best reward is the profit received as a result of the transition from the previous state to the new one. Thus, the reward received for completing each action can be calculated by the formula:

$$R = S(t+1) - S(t) \quad (2)$$

In this formula, R is the reward received, after the transition to the new state, S(t) is the cost of the current state of the system, S(t+1) is the cost of the new state of the system after performing the N_{act} action.

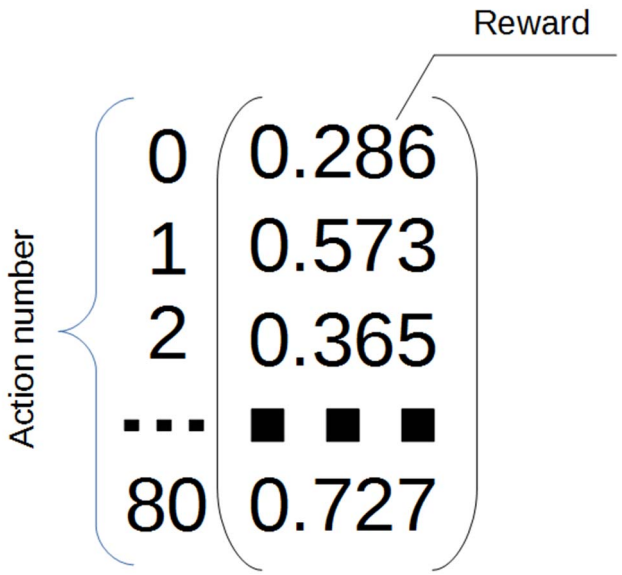

Fig. 7. Reward matrix

Thus, as an output, the engine presents a matrix with rewards that will be received for each action. An example of such a matrix is presented in Figure 7. One of the dimensions of the matrix is one, and the second dimension is calculated by the following formula:

$$N = N_k \quad (3)$$

To select the best action, the ML-engine selects the row with the highest award in the matrix, and then returns its number as the number of the performed action.

The most valuable condition is the condition of the containers on the nodes, in which the load on all nodes is distributed most evenly. It is worth considering that all metrics are normalized and ranged from 0 to 1.

Due to the need to maximize the value of a state with the most uniformly distributed metrics, it may be useful to use the Euclidean distance for each significant metric and each node. But it must be taken into account that all the arguments accepted are less than one.

The formula for describing the state cost for one metric will look like this:

$$S = \sqrt{(1 - M_1)^2 + \cdots + (1 - M_N)^2} \quad (4)$$

In this formula S is the value of the current state, M1 ... MN are the metric indicators of each of the nodes in the range [0; 1], N is the number of nodes in the system.

As can be seen from formula (4), cost maximization occurs in the case of minimizing the concentration of large metric values on a node. This leads to an even distribution of containers.

TABLE I
VALUE CALCULATION FOR THE CASE WITH THREE CONTAINERS AND TWO NODES

State	Node 0	Node 1	Value
0	Container 0, Container 1, Container 2	—	0.93
1	Container 1, Container 2	Container 0	0.97
2	Container 0, Container 2	Container 1	1.00
3	Container 2	Container 0 Container 1	1.01
4	Container 0, Container 1	Container 2	1.01
5	Container 1	Container 0, Container 2	1.00
6	Container 0	Container 1, Container 2	0.97
7	—	Container 0, Container 1, Container 2	0.93

The theoretical case of applying the Euclidean distance with different loads can be considered. For example, there are two computing nodes on which three containers are deployed, each of which has different values of occupied resources. For

example, the zero, the first and the second containers are loaded at 0.1, 0.2 and 0.3 respectively. Table 1 presents the cost of every state in this case.

As can be seen from the Table 1, by increasing the uniformity of the load distribution, the cost of the state also grows. Moreover, the state where the nodes have an equal load, 0.3 units each, has the greatest value.

VIII. Experiment

During the system operability check, testing was performed and the level of distributed containers was measured. The level of distribution was calculated in accordance with formula (4). The random access memory used randomly by each container was selected as the loaded value. The experiments were carried out on a cluster of four nodes.

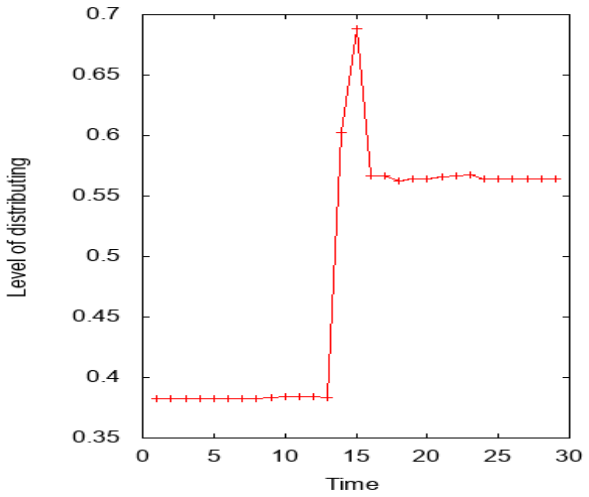

Fig. 8. Container distribution level during a container moving operation

The results of one of these measurements are shown in Figure 8. The abscissa axis counts the time during which the containers are redistributed, the ordinate level of the container distribution, where the levels of RAM utilization on each computing node are used as metric values. The main objective of the system is the transition to a state in which the distribution level of containers is greatest for a long time.

Initially, all container nodes are located on one node, as a result of which the distribution level is low. During the movement of containers, there is a sharp increase in the level of distribution, which is expressed as a peak on the graph, after which the load is stabilized, and the level of distribution is at a higher level than before the redistribution of containers.

IX. Future works

This paper provides a general description of the concept and evidence of its applicability. However, the proposed objective function is quite simple and solves the problem when working with only one metric.

As further development, it would be possible to use more complex objective functions, including those that solve the problem for many metrics and taking into account the nature of the resources consumed.

X. Conclusion

This article proposes a new technique for managing the location of containers on cluster nodes using reinforcement learning algorithms. The container location management system using the described technique is really capable of placing containers depending on the level of load on the tasks performed in them. The methodology used was tested for the case of OS-level virtualization using Docker, but in theory it can be used for other types of virtualization.

References

[1] P. Padala, X. Zhu, Z. Wang, S. Singhal and K. Shin, "Performance Evaluation of Virtualization Technologies for Server Consolidation," HP Lab. Palo Alto, California, USA, Rep. HPL-2007-59, Apr. 2007. [Online]. Available: http://www.hpl.hp.com/techreports/2007/HPL-2007-59R1.pdf

[2] A. Chierici and R. Veraldi, "A quantitative comparison between xen and kvm," in J. Phys.: Conf. Ser. 219 042005, Prague, Czech Republic, 2009. [Online]. Available: https://iopscience.iop.org/article/10.1088/1742-6596/219/4/042005/pdf

[3] A. Tosatto, P. Ruiu and A. Attanasio, "Container-Based Orchestration in Cloud: State of the Art and Challenges," in CISIS., Blumenau, Brazil, 2015, pp. 70-75.

[4] M. Rovnyagin, A. Guminskaia, A. Plyukhin, A. Orlov, F. Chernilin and A. Hrapov, "Using the ML-based Architecture for Adaptive Containerized System Creation," in EIConRus., Moscow, Russia, 2018, pp. 358-360.

[5] S. Hoque, M. Brito, A. Willner, O. Keil and T. Magedanz, "Towards Container Orchestration in Fog Computing Infrastructures," in COMPSAC., Turin, Italy, 2017, pp. 294-299.

[6] L. Galway, D. Charles, and M. Black, "Machine learning in digital games: a survey," Artificial Intelligence Review, vol. 29, no. 2, pp. 123-161. Apr. 2008, DOI. 10.1007/s10462-009-9112-y.

[7] N. Justesen, P. Bontrager, J. Togelius, S. Risi, "Deep Learning for Video Game Playing," IEEE Transactions on Games, 2019.

[8] Y. Li, Y. Wen, K. Guan and D. Tao, "Transforming Cooling Optimization for Green DataCenter via Deep Reinforcement Learning," Artificial Intelligence, 2017.

[9] S. Ravichandiran, "Temporal Difference Learning" in Hands-On Reinforcement Learning with Python, Birmingham, UK: Packt Publishing Ltd., 2018, ch. 5, pp. 189-214.

[10] L. Kaelbling, M. Littman and A. Moore, "Reinforcement Learning: A Survey," Journal of Artificial Intelligence Research, vol. 4, pp. 237-285. May 1996, DOI. 10.1613/jair.301.

[11] Y. LeCun, L. Bottou, G. Orr and K. Müller, "Efficient BackProp," Neural Networks: Tricks of the Trade, vol. 7700 LECTURE NO, pp. 9-48, 2012, DOI. 10.1007/978-3-642-35289-8-3.