Intelligent Docker Container Orchestration for Low Scheduling Latency and Fast Migration in Paas

Mikhail. M. Rovnyagin
National Research Nuclear University MEPhI
(Moscow Engineering Physics Institute)
Moscow, Russia
m.rovnyagin.2015@ieee.org

Dmitry M. Sinelnikov
National Research Nuclear University MEPhI
(Moscow Engineering Physics Institute)
Moscow, Russia
dsinelnikov96@gmail.com

Sergey S. Varykhanov
National Research Nuclear University MEPhI
(Moscow Engineering Physics Institute)
Moscow, Russia
masmx64@gmail.com

Timur R. Magazov
National Research Nuclear University MEPhI
(Moscow Engineering Physics Institute)
Moscow, Russia
timur71398@gmail.com

Amir A. Kiamov
National Research Nuclear University MEPhI
(Moscow Engineering Physics Institute)
Moscow, Russia
amirkiy01@gmail.com

Tatyana A. Shirokikh
National Research Nuclear University MEPhI
(Moscow Engineering Physics Institute)
Moscow, Russia
tanya.plys@bk.ru

Abstract — Most modern cloud Platform as a Service (PaaS) systems are built on the basis of application containerization technology. The containerization schedulers currently in use do not have intelligent algorithms for placing containers across nodes. They work on the principle of placing in the "first suitable slot". This article discusses two issues related to intelligent decision-making about moving containers between servers: the lethality of obtaining metrics for monitoring the current state of the platform and taking into account the influence of container deployment parameters on the speed of transformations. The article proposes a way to determine the necessary and sufficient number of monitoring services for latency management. The paper also considers the influence of microservices launch parameters, the size of the container image on the time of transferring the container between nodes, and proposes a model for assessing the complexity of transferring containers between nodes.

Keywords — distributed computing, ML scheduling, Docker, containerization, orchestration, PaaS, Cloud, monitoring, migration

I. Introduction

The transition to cloud computing dating back to 1994 (the creation of PersonaLink Services by AT&T) [1] marked a turning point in the industry - small companies were able to compete with large ones. Now you can rent data center capacity to deploy your applications without worrying about the hardware [2]. This made it possible for developers to focus on developing business logic.

Cloud computing is characterized by the following elements [3]:

- On-demand self-service — issuance of access to computing resources occurs automatically.
- Broad network access.
- Resource pooling — the user does not have access to the internal structure of the cloud.
- Rapid elasticity — flexibility and high speed of changing the configuration of the cloud, for the user, the allocated resources may seem unlimited.
- Controlled Infrastructure — the cloud system automatically adjusts at different levels based on its own measurements (Measured service).

One of the most common types of cloud computing services – PaaS (platform as a service) – is the provision of a platform to the user with support for some programming languages, libraries and tools. In PaaS systems, the user has no control over the computing infrastructure.

To provide the PaaS service, virtualization technologies [4] are used, often these are virtual machines with a hypervisor. Along with provisioning virtual machines, containerization is also used. Containerization is a technology that requires much less resources to use [4]. The de facto industry standard for containerization is Docker technology [5].

Like any large technological system, cloud computing faces many problems at different levels – from building a topology and hierarchy of a computer network to ensuring efficient virtualization at the hardware level [6].

One of the problems that PaaS faces is load balancing between processing servers. Within the framework of this problem, a couple of common tasks can be distinguished:

1. Collection and processing of metrics for timely updating of server status;
2. Efficient transfer of containers from one node to another to redistribute the load.

These issues are discussed in our article.

II. Related Works

Current approaches to load balancing involve the construction of processing algorithms and architectures. The algorithmic part uses heuristic approaches. For example, in [7] the authors focus on improving the genetic task scheduling algorithm in cloud computing by adding new strategies there. In another paper [8], the authors use a different genetic algorithm and also whole swarm optimization to solve the problem of load balancing in the data center. Work [9] uses Reinforcement Learning (RL) techniques, in particular DQN building and training, to improve resource allocation. In our practice, the RL algorithm was also proposed – training a Q-learning agent to solve this problem [10].

Regarding the development of the processing architecture, the following works can be distinguished at the moment: [11] focuses on proposing overall architecture

979-8-3503-9759-8/22/$31.00 ©2023 IEEE

concept, as we concentrate on adding machine learning techniques into cloud balancing architecture in [12].

III. EXPERIMENT

In this paper, we simulate the distribution of containers over cluster nodes, as well as the collection and aggregation of metrics from these nodes. Accordingly, the first part of the article is devoted to the study of the collection and aggregation of metrics, during which the dependence of the number of nodes-processors (agents) of metrics on the number of nodes and containers under monitoring at the target value of processing latency is studied. The second part of the article is devoted to the issues of estimating the time of container migration between nodes depending on their characteristics.

IV. LOW LATENCY METRICS GATHERING

PaaS involves infrastructure management on the side of the service provider, so cloud monitoring is an important task. Cloud monitoring faces the following challenges [13]:

- Selection of health states, which denote the alive node;
- Creation of a homogeneous (Unified) monitoring environment for the possibility of scaling control and management;
- Development of an optimal strategy for behavior in case of overload (failover).

The last of these problems, in turn, relies on getting up-to-date metrics with the minimum possible delay.

To solve this issue, we propose to evaluate the capabilities of monitoring agents (Prometheus) depending on the number of nodes they control. For example, starting from a certain number of controlled nodes, the agent cannot cope with the aggregation of metrics from them, in which case its statistics begin to lag by some amount Δt (Fig. 1).

In this case, the solution to the problem will be to increase the aggregation delay to Δt is to add monitoring agent nodes, redistributing new nodes to them. From here we can also obtain an algorithmic estimate of this quantity.

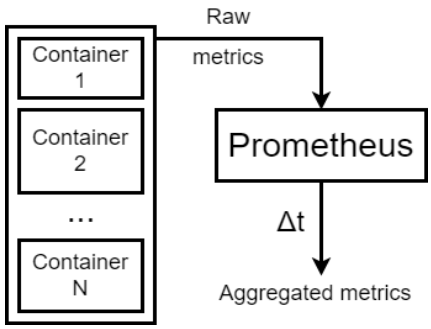

Fig. 1. – Low latency metrics aggregation problem

To simulate the problem, the following prototype was built (Fig. 2):

- Master node: Windows10, Intel Core i7 CPU 1.30GHz, Cores 8, RAM 16GB, SSD Samsung 970;
- Controlled nodes (Server 1 - 5): Debian 11, Intel Core i7 CPU 2.40GHz, Cores 4, RAM 4GB, SSD Samsung 860.

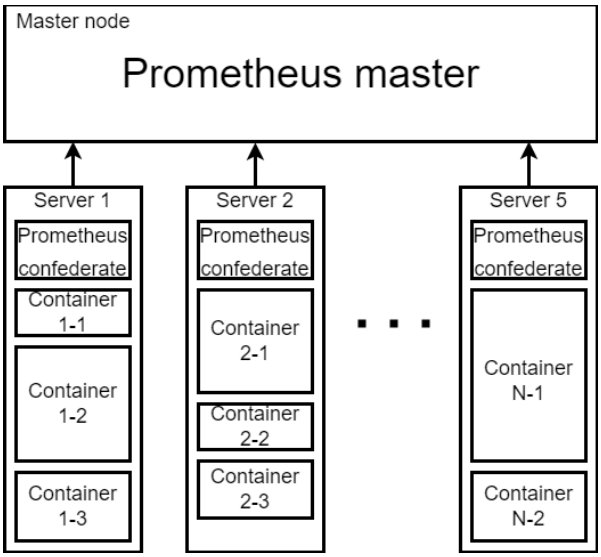

Fig. 2. – Metrics aggregation experimental model architecture

For the experiment, an environment based on Docker containers and Prometheus monitoring agent nodes was chosen due to their unification and low requirements for computing resources.

To reduce the scale of the experiment, the following aggregation function was chosen:

```
rate(node_memory_SwapCached_bytes[10m])                /
rate(node_memory_SwapFree_bytes[10m])C_destination)
```

and the criteria Δt was set to 10% of scrape time.

The data obtained during the experiment are presented in the table 1.

TABLE I. Experiment results

Num of Prometheus instances	Num of containers	Delta (%)
1	5	2.6
1	10	3.2
1	15	4.6
1	20	5.6
1	30	7.2
1	40	8.8
1	60	13.0
2	60	7.2
2	80	9.0
2	100	11.3
3	100	8.0
3	120	9.4
3	160	10.7
4	160	8.9

Based on these data, a graph was built (Fig. 3):

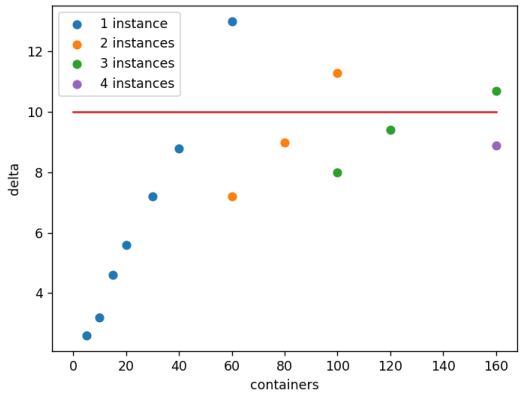

Fig. 3. – Results graph

From the data obtained, we can draw the following conclusion about the dependence of the number of monitoring nodes on the number of controlled nodes:

$$N_{Monitoring} \square O(N_{Monitored})$$

The analytical value formula can be used to estimate the cost of scaling the cloud monitoring infrastructure.

V. CONTAINER MIGRATION COST ESTIMATION

Cloud services that provide their resources to consumers to solve problems face the problem of migration of containers in a cluster when balancing the load. The problem lies in the need to ensure a minimum downtime for the user service.

Container migration is the movement of: processor, memory, network, and storage states. The move process involves stopping the service on the source node and starting the service on the target node when the migration is complete [14].

There are the following techniques for moving containers:

- Cold – complete stop, move, start;
- Pre-copy migration: In pre-copy phase, memory pages are iteratively copied from source to destination host when the virtual system is running at the source. At the first set, all the memory pages are copied while the subsequent iterations only the modified or dirty pages are copied. The modified pages are monitored by the hypervisor using a dirty bitmap. If the number of dirty pages reaches the limit, the pre-copy phase is terminated. In stop-and-copy phase, the Container is suspended at the source and the remaining dirty pages are transferred to the destination. Later, the Container is resumed on the target host [15];
- Post-copy migration: In the post-copy technique, the stop-and-copy phase is followed by pull phase for migration. The Container is suspended at the source system, and a minimal state of the Container is transferred to the destination. Later, Container is resumed on the target host where it fetches memory pages that are then transferred to the destination host [16]. A page fault occurs when the Container tries to access the memory pages which are not yet migrated. Maximum migration time and minimum downtime are achieved by this technique [17];
- Hybrid migration: It is referred to as a particular case of post-copy migration where the post-copy algorithm is preceded by a pre-copy stage. The most often used memory pages are transferred before the Container execution is resumed at the destination host. The performance degradation caused due to page faults in post-copy migration can be reduced by using this technique;
- Live migration: moves a running container from one server to another. Containers continue to run during the live migration process, which results in shorter container Downtime and longer Migration Time [18].

All existing migration techniques are evaluated by two parameters – migration time (Total migration) and downtime (Downtime) [19].

$$TotalMigrationTime = \underbrace{Initialization + Reservation}_{Pre_{migrationOverhead}} + \sum_{i} Pre_{copy_i} + StopAndCopy + \underbrace{Commitment + Activation}_{Post\_migrationOverhead}$$

$$TotalDowntime = StopAndCopy + \underbrace{Commitment + Activation}_{Post\_migrationOverhead}$$

System load balancing provides longer lifetime of cluster components and allows for greater fault tolerance []. The need to rebalance resource consumption can arise when new containers are added to the PaaS or when they are removed from the PaaS.

When optimizing the choice of virtual machines for which the process of moving from the source node to the target will be carried out, a number of the following characteristics should be taken into account: the size of the virtual machine, network bandwidth, the quantitative characteristic of the node resources consumed by the container for some last finite period of time, or the "activity" of the container, the average time to stop and start the container.

All of the characteristics described above can be used to estimate the cost of container migration.

To form a model that estimates the cost of migration, a system prototype has been developed that uses the following components:

- A database that stores information about containers and migration statistics;
- Two worker nodes with Docker;
- The control node, which includes a module for saving all the current characteristics of the container and the network at the time of the start of migration and the end of migration;
- A component that adds new containers to the system or removes them from there from time to time.

The control node balances the system containers based on information about the consumption of node resources by each container.

The following containers were used to collect statistics: busybox, nginx, nextcloud.

The architecture for collecting statistics is as follows:

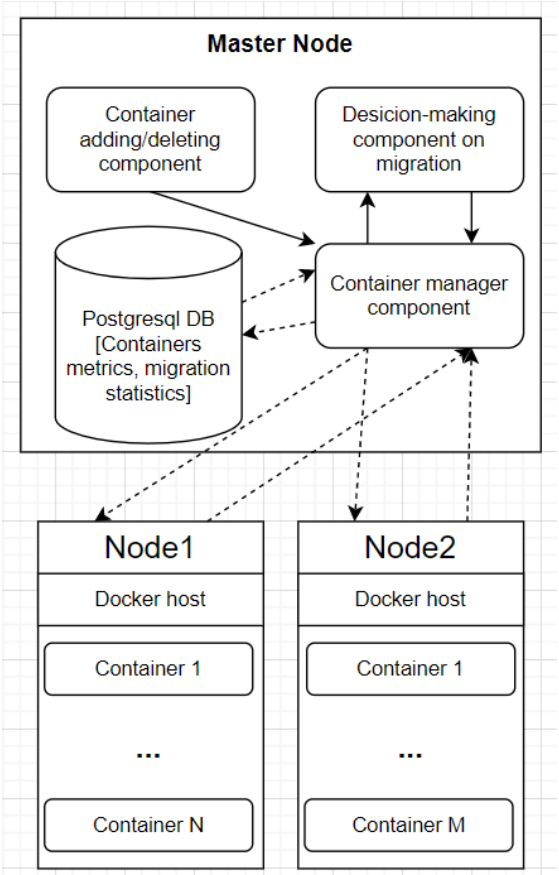

Fig. 4. – Experiment architecture

Node characteristics:

- MasterNode: Windows10, Intel Core i7 CPU 3.20GHz, Cores 16, RAM 16GB, SSD Samsung 860;
- Node1, Node2: Ubuntu20.05, Intel Core i7 CPU 2.40GHz, Cores 4, RAM 4GB, SSD Samsung 860.

The following set of parameters is formed based on aggregated metrics:

- network throughput in a given period of time;
- peak performance of one node core;
- host storage system read and write speed.

Estimating the migration time requires estimating the time to start a container, the time to save the state of the container, and the time to completely stop the container. Therefore, when adding a new image to the cloud, from which it is required to form a container, these parameters are evaluated on a standard node with known parameters. The container starts and stops N times, then the average value of the container parameters is calculated.

A model for estimating the time of cold migration of a container from a SourceNode to a DestinationNode from the container parameters and network characteristics at a given time has been developed:

```
TotalMigrationTime >= StopTime * (r_source / r0) + SaveTime * (r_source / r0) + LoadImageTime * (r_destination / r0) + StartTime * (r_destination / r0) + ContainerSize / min(C_network, C_source, C_destination)
```

, where:

- r – peak performance per core,
- r0 – single core performance of a standard node,
- ContainerSize – selected container size,
- StopTime, SaveTime, StartTime – measured time to stop and start a container on a standard node,
- C_network – network bandwidth between SourceNode and DestinationNode nodes,
- C_source – the speed of reading information from the host storage,
- C_destination – node storage write speed.

The obtained values of network bandwidth and write and read speeds of node storages:

TABLE II. Throughput values for Nodes

	Node1	**Node2**
C_network, MB per sec	10.9	10.9
C_source, MB per sec	550	550
C_destination, MB per sec	520	520

To confirm the adequacy of the model, containers with the following characteristics were used:

TABLE III. Container characteristics

	busybox	**nginx**	**nextcloud**
Size, Mb	1.24	142	994
Start time, s	0.557	0.61	0.493
Stop time, s	10.182	0.38	2.221
Save time, s	0.29	3.1	18.34
Load image time, s	0.25	0.85	72.1
migration time, s	13.25	19.71	188.84
migration estimation, s	11.39	17.97	184.34

The estimated time obtained is a lower estimate and accurately predicts the migration time without taking into account errors.

The developed evaluation model can be used to form container weights, based on which the load balancing component in PaaS will decide which containers should be migrated.

VI. Conclusion

This work is devoted to the construction of intelligent methods for processing metrics of cloud infrastructure nodes. The proposed options for estimating the required number of monitoring agents and a more accurate estimate of the time for transferring containers will allow more efficient planning of operations related to balancing containers across PaaS cluster nodes.

VII. References

[1] https://www.wired.com/2014/05/tech-time-warp-cloud-is-born/ (accessed through WaybackMachine in 2022/X).

[2] Shaw, Subhadra & Singh, A (2014). A survey on cloud computing. 1-6. 10.1109/ICGCCEE.2014.6921423

[3] Peter Mell, Timothy Grance. The NIST Definition of Cloud Computing (Draft). NIST. 2011

[4] R. Dua, A. R. Raja and D. Kakadia, "Virtualization vs Containerization to Support PaaS," *2014 IEEE International Conference on Cloud Engineering*, 2014, pp. 610-614

[5] https://www.docker.com [Online] (accessed in 30.10.2022)

[6] Fatemi Moghaddam, Faraz & Ahmadi, Mohammad & Eslami, Mohammad & Sarvari, Samira & Golkar, Ali. (2015). Cloud Computing Challenges and Opportunities: A Survey. IEEE journal.

[7] Verma, Amandeep & Kumar, Pardeep. (2012). Independent Task Scheduling in Cloud Computing by Improved Genetic Algorithm. International Journal of Advanced Research in Computer Science and Software Engineering. 2. 111-114.

[8] Musa, Nehemiah & Gital, Abdulsalam & Zambuk, Fatima & Muhammad Usman, Ali & Almutairi, Mubarak & Chiroma, Haruna. (2020). An Enhanced Hybrid Genetic Algorithm And Particle Swarm Optimization Based on Small Position Values for Tasks Scheduling in Cloud. 1-5.

[9] Chraibi, Amine & Ben Alla, Said & Ezzati, Abdellah. (2021). Makespan Optimisation in Cloudlet Scheduling with Improved DQN Algorithm in Cloud Computing. Scientific Programming. 2021.

[10] M. M. Rovnyagin and A. S. Hrapov, "Presentation of the PaaS System State for Planning Containers Deployment Based on ML-Algorithms," 2020 Moscow Workshop on Electronic and Networking Technologies (MWENT), 2020, pp. 1-5

[11] M. Belkhouraf, A. Kartit, H. Ouahmane, H. K. Idrissi, Z. Kartit and M. El Marraki, "A secured load balancing architecture for cloud computing based on multiple clusters," 2015 International Conference on Cloud Technologies and Applications (CloudTech), 2015, pp. 1-6

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

[13] Pourmajidi, William & Steinbacher, John & Erwin, Tony & Miranskyy, Andriy. (2018). On Challenges of Cloud Monitoring.

[14] https://www.politesi.polimi.it/bitstream/10589/114581/3/2015_Dicembre_Pace.pdf [Online] (accessed in 30.10.2022)

[15] M. Rosenblum and T. Garfinkel, “Virtual machine monitors: Current technology and future trends,” Computer, vol. 38, no. 5, pp. 39–47, 2005.

[16] S. C. Pasumarthy, “Live Migration of Virtual Machines in the Cloud: An Investigation by Measurements,” 2015.

[17] M. Bunyakitanon and M. Peng, “Performance Measurement of Live Migration Algorithms,” 2014.

[18] F. Romero and T. J. Hacker, “Live migration of parallel applications with openvz,” in Advanced Information Networking and Applications (WAINA), 2011 IEEE Workshops of International Conference on, 2011, pp. 526–531

[19] Akoush, S., Sohan, R., Rice, A., Moore, A., Hopper, A.: Predicting the performance of virtual machine migration. In: Proceedings of 2010 IEEE International Symposium on Modeling, Analysis Simulation of Computer and Telecommunication Systems (MASCOTS 2010), pp. 37–46. IEEE (2010)