Using the ML-based Architecture for Adaptive Containerized System Creation

Mikhail M. Rovnyagin[1], Anna V. Guminskaia[2], Alexey A. Plyukhin[3], Aleksandr P. Orlov[4], Fedor N. Chernilin[5], Alexander S. Hrapov[6]
National Research Nuclear University MEPhI (Moscow Engineering Physics Institute), Moscow, Russian Federation
[1]m.rovnyagin.2015@ieee.org, [2]annguminskaya@gmail.com, [3]a.pluhin@gmail.com, [4]Alex567.alex@yandex.ru, [5]fedor.chernilin@gmail.com, [6]arcanius@list.ru

***Abstract*—Most of today's applications are built on a micro-service architecture, where one large application is divided into several differently functional parts. Each part is packaged in an isolated container with its own file system, process space and IP address. Contemporary container management tools implement only the simplest strategies for placing containers in cluster systems. In this paper, we propose a method for constructing adaptive containerized infrastructures using machine learning methods. The paper also describes the methodology for designing distributed systems with containers of different types: data-intensive and compute-intensive.**

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

I. Introduction

More and more companies are starting to use cloud technologies. In the clouds are deployed multimodule systems, the configuration of which, in general, requires a lot of time. Also, developers may encounter problems tracking dependencies, scaling an application, and updating individual components that do not directly affect the application itself. Docker-containerization and service-oriented design are trying to solve many of these problems. Applications can be broken down into managed functional components, individually packed together with all their dependencies, and then easily deployed on a non-standard architecture. This also simplifies the scaling and updating of components.

Container-based systems most often work in a cluster. The standard algorithms of orchestration Docker Swarm, Google Kubernetes do not take into account the specifics of computational operations in containers, which can lead to uneven distribution of load on the nodes.

Thus, there is a need for algorithms that work on top of the standard implementation and load balancing between services depending on their performance profiles.

II. Related works

The authors of [1] propose to use an additional Management Server, whose goal is to monitor the system based on the statistics received and dynamically create additional copies of the service.

The book [2] outlines the basic concepts of building distributed applications, such as CORBA, OMA, ORB. These concepts are the standard for building complex distributed object-oriented applications.

A popular solution for load balancing with containerization is to create a task queue for a cluster and a manager that sends these tasks to containers, and creates new containers if necessary.

C. Kaewkasi in his article proposed the following approach for building distributed applications - Swark-Kit Ant Colony Optimization [3], the distinguishing feature is the use of the pipeline scheme and filters to find the appropriate node in the cluster on which the task will be executed.

The authors of [4] use a similar concept of cluster construction, its main components are a web server, a scheduler, an observer and a performer. By adding a new layer of abstraction over the Docker Swarm, the system can react independently to possible changes in the incoming load.

III. Architecture Description

Our approach is focused on the distribution of services on the nodes of the network on the basis of machine learning algorithms [5] in order to reduce latency and increase the throughput of the system as a whole. The adaptive nature of the proposed approach is a necessity due to the dynamic behavior of containers during operation. The architecture of our solution consists of the following components:

- Manager for launching containers based on Docker Swarm;
- Manager for distribution of containers based on ML;
- Node monitoring manager based on metrics collection and kpi nodes and containers.

IV. Comparison of Virtualization Approaches

There are two most commonly used approaches to service virtualization, which, moreover, have implementations in the form of free software:

- virtualization based on the deployment of virtual machines using a hypervisor as a resource manager component;

- container-based virtualization. The hypervisor is absent in this case, and the host OS itself manages the resources.

The proposed infrastructure scheme is abstracted from a specific software method for virtualizing network functions.

A. *Virtualization using hypervisors*

The most obvious representation can be obtained by the example of the virtualization scheme used in KVM. The base Linux kernel (HWN, HardWare Node) interacts with the hardware (processor, memory, I / O). It also, with the help of a special module, provides the ability to work processes in which a full-fledged operating system functions. It's worth noting that these are not exactly normal processes, since the operation of virtual machines is provided through a kernel module that translates system calls from the client operating system to HWN (Figure 1).

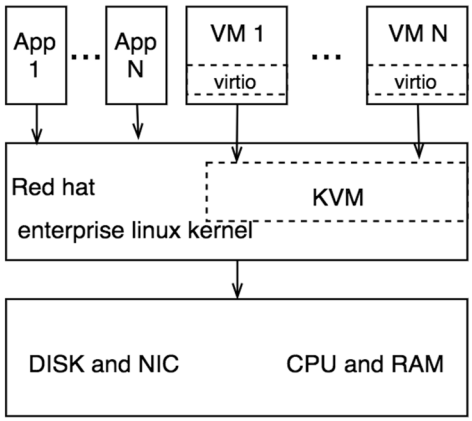

Fig. 1. - Hypervisor-based virtualization scheme

Disadvantages of this type of isolation are the additional delays in the operation of the disk subsystem, the network subsystem, memory and processor due to the use of an additional hardware and software layer of abstraction between the real hardware and the virtual environment. Because of this, the difference in performance when using hypervisors relative to a typical OS is 5-15% [6-7], depending on the type of software used and the system configuration. There are several implementations of this approach: KVM (including OVirt), Xen.

B. *Virtualization using containers*

It uses one core and no virtual machines. At a low level, the Linux kernel HWN interacts with hardware and performs all hardware requests from virtual environments. For security reasons, virtual environments are isolated from each other using the kernel-built mechanisms in Linux.

A modified chroot analog is used. It allows you to create isolated hierarchies within a single file system. Secondly, the mechanism PID namespaces, which inside each chroot environment creates a completely independent from HWN system of process identifiers, where we can have our own process with PID = 1, even if the init server is already running on the server itself. In general, we can create hundreds or even thousands of individual containers that are not connected to each other in any way.

Memory, CPU and hard disk load are limited by the cgroups mechanism, which is also used for the same purposes in KVM technology.

Figure 2 shows the containerization scheme for services.

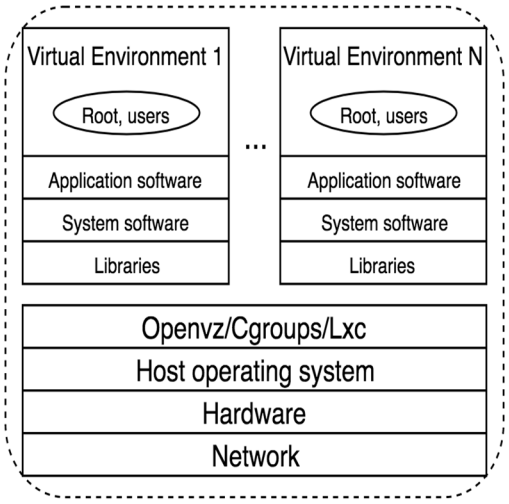

Fig. 2. - Container-based virtualization scheme

Estimates of the overhead loss of containerization are given in various works, and they are in the region of 0.1-1% [8-9], depending on the type of load and the testing methodology. Examples of implementing a container type virtualization include: Docker, OpenVZ, LXC, and so on.

V. Approach Description

This approach involves the allocation of services involved in the observation and maintenance of computing nodes. These services are designed to perform two types of tasks: monitoring the current state of the nodes and generating control actions for redistributing containers at different nodes depending on the current load.

An additional challenge is to visualize various current system characteristics. For more convenient monitoring of the current state of the computing infrastructure.

To obtain the current metrics from the nodes, Telegraf is used - a universal agent for collecting, processing and aggregating all necessary metrics, such as network, memory, processor load. To better understand the state of the Telegraf system, the agent is located not only on each node, but also in each container.

The collected metrics on each node are collected by the Prometheus monitoring system, which stores all the collected metrics in its own local NoSQL database, in which each metric is linked to the time point at which it was measured, and the agent from which it was received.

Using the Prometheus REST interface, the pre-aggregated metrics are read and written to the main database (Apache Cassandra) for further analysis and processing.

ML-engine performs analysis of metrics and generates control actions for the placement manager, which makes

appropriate notes to its database and carries out direct transfer of containers.

The architecture of the solution is shown in Figure 3.

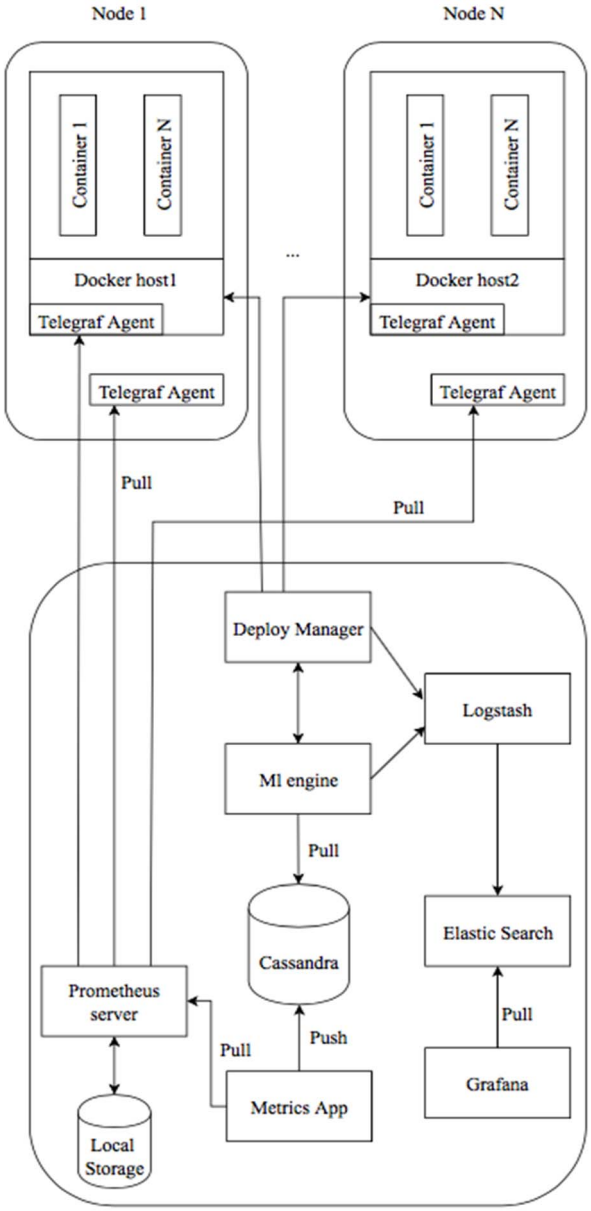

Fig. 3. - Solution architecture

As a protocol for direct management of containers, you can use both normal ssh and higher-level systems such as Docker Swarm or Kubernetes.

VI. FUTURE WORKS

At the moment, the stage of proof of the concept is passed: all components of the proposed architecture work properly. From the triggering of the trigger threshold in the monitoring subsystem to the generation of the control action (container transfer) takes an average of 3-5 seconds depending on the configuration.

Immediately, the transfer of the stateless container takes another 25 seconds, after which the container again responds to healthcheck requests. The resulted results were removed at rigidly set transfer scenario. Currently, studies are being conducted using algorithms based on DNN-analysis of time series (using the Tensorflow library).

VII. CONCLUSIONS

Thus, in this article, a new way of orchestrating container management processes is proposed, based on the analysis of the node state by the scheduler implemented using machine learning algorithms. Using the following scheduler allows you to increase utilization and reduce access times to containers.

REFERENCES

[1] O. Sallou and C. Monjeaud., Go-Docker, A batch scheduling system with Docker containers. IEEE International Conference of Cluster Computing, pp. 514-515, 2015

[2] A. Puder, K. Romer, F. Pihloher Distributed System Application. A Middleware Approach, USA, Morgan Kaufmann Publishers is an imprint of Elsevier. 2006.

[3] C. Kaewkasi and K. Chuenmuneewong, Improvement of Container Scheduling for Docker using Ant Colony Optimization pp. 254 – 259

[4] Y. Li and Y. Xia, Auto-Scaling Web Application in Hybrid Cloud Based on Docker, 5th International Conference on Computer Science and Network Technology (ICCSNT), pp. 75-79, 2016

[5] D.A. Shulga, A.A. Kapustin, A.A. Kozlov, A.A. Kozyrev, M.M. Rovnyagin, "The Scheduling Based on Machine Learning for Heterogeneous CPU / GPU Systems", 2016 IEEE NW Russia Young Researchers in Electrical and Electronic Engineering Conf

[6] Performance Evaluation of Virtualization Technologies for Server Consolidation: [Online] /HP Laboratories. Available: http://www.hpl.hp.com/techreports/2007/HPL-2007-59R1.pdf

[7] A quantitative comparison between xen and kvm: [Online] /Journal of Physics: Conference Series. Available: http://iopscience.iop.org/article/10.1088/1742-6596/219/4/042005/pdf

[8] Performance Overhead Comparison between Hypervisor and Container based Virtualization: [Online] /Archive. Available: https://arxiv.org/pdf/1708.01388.pdf

[9] A comparison of performance between KVM and Docker instances in OpenStack: [Online] /HEPiX Fall 2015 Workshop at BNL. Available: https://indico.cern.ch/event/384358/contributions/909221/attachments/1170419/1689415/151014_hepix_wataru_takase.pdf