Approach of Program's Concurrency Evaluation in PaaS Cloud Infrastructure

Elza F. Mingazhitdinova[1], Mikhail M. Rovnyagin[2], Dmitry M. Sinelnikov [3],Viktor V. Odintsev[4],Sergey S. Varykhanov[5]
National Research Nuclear University “MEPhI”
Kashirskoye highway 31, 115409,
Moscow, Russian Federation
[1]mingazhitdinova@gmail.com,
[2]m.rovnyagin.2015@ieee.org,[3]dsinelnikov96@gmail.com,[4]zakhams@gmail.com,[5]masmx64@gmail.com

In the most cases programs have got different level of concurrency. According to Amdahl's Law, changing the amount of resources may not give a gain in computational efficiency. The article’s goal which was determined by the authors is to find out the balance between increasing the amount of resources and improving the efficiency that can be obtained for computation in the K8s cluster. In an effort of resolving the task authors have decided to explore the correlation between increasing the number of podes and time of Spark program processing (data-intensive and compute-intensive computations) in the local minikube for following deployment in K8s.

Keywords—Kubernetes, K8s, PaaS, container orchestration tool

I. Introduction

Kubernetes has become one of the most popular container’s orchestrator tool and the most widespread PaaS technology in the last decade. A huge amount of companies around the world use K8s nowadays. Increased popularity of Kubernetes is caused by functions that are provided by this technology: it allows to deploy applications quickly, fast scale and manage containerized applications. To compare, in the past only physical servers were available for deploying applications and it could took a weeks to make a release of new function and a couple of weeks to find out the problems which could be caused by application’s separability and resources sharing: memory, CPU resources and network [2].

Revolution in the infrastructure approach and moving from physical servers deployment to containerization technology cloud deploying has gave start to Kubernetes ecosystem which tried to solve problems related with long and difficult deployment: now it can be processed within a seconds to deploy new version of an application and the resiliency of system was raised [3-4].

As it was mentioned before, Kubernetes is used by a lot of companies as a part of their production infrastructure. To use K8s ecosystem efficiently there should be correct managing of the resources which are allocated for cluster. There is an issue related with Kubernetes how to find out the balance between efficiency and how to predict the resources that should be used.

Due to Amdahl's law increasing the number of processors has got the limit in concurrency [1]. Therefore, increasing the number of processors and memory resources in the K8s cluster do not ensure the growth of application parallelism.

The aim of this article is finding out the correlation between the growth of resources that are allocated for cluster and computation time of Spark program which emulate two types of processing (data-intensive and compute-intensive).

Spark will be used for the article experiment because it is unified analytics engine for large-scale data processing where can be calculated all types of computing.

Also Apache Spark has become one of the key big data distributed processing frameworks in the world [6]. Spark can be deployed in a variety of ways, provides native bindings for the Java, Scala, Python, and R programming languages, and supports SQL, streaming data, machine learning, and graph processing. Additionally, Spark could be deployed to Kubernetes and at the nodes of K8s cluster could be run containerized Apache Spark applications. In this case each Spark app will be fully isolated from the others and packages its own version of Spark and dependencies within a Docker image.

Our Spark compute-intensive calculation will be counting the factorial of BigInteger which should be parallelized on different pods as it possible and data-intensive will be grouping by the array by the array’s keys.

II. Theory

Amdahl's law or it called Amdahl’s argument is a formula which gives the theoretical speed-up in latency of the execution of a task at fixed workload that can be expected of a system whose resources are improved [1]. This law could help to determine a certain balance between parallel and sequential program (as the number of available processors increases) and optimize the available speed of execution of a particular sequential program by itself. For example, it could help to find the trade-off for such examples: Have four processors executing a given program for 40% of its total execution, or use only 2 processors to execute the same program, but at the same time twice as long? Consequently, Amdahl’s law allows to estimate the maximum limit of any potential improvement in speed resulting from parallel computing.

Concurrency value of computational task could be calculated with mathematical formula of Amdahl’s law. In this formula we have got the p – numbers of processors, α – the proportion level of the total computation volume which could be obtained by consecutive calculation, $1-\alpha$ – the proportion level which could be obtained by parallel calculation, speed-up that could be received on a computing system with p processors is S_p:

$$S_p = \frac{1}{\alpha + \frac{1-\alpha}{p}}$$

In our experiment Amdahl's law is related to compute intensive calculation in Kubernetes cluster which will check the concurrency of factorial counting.

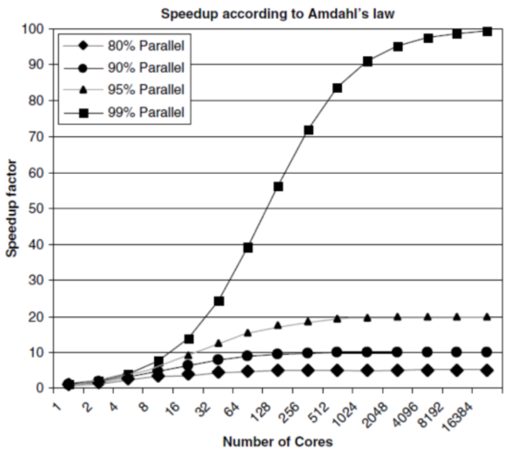

Fig. 1. *Amdahl's Law*

Kubernetes cluster architecture (Fig.2) could be divided to two groups: control layer and computational layer [5]. The last one consist of nodes – working machines and they could be physical or virtual. Each node consists of his working component and main abstraction which is called Pod and is managed by the control plane. Pods are basic deployment objects that represent a single instance of a running process in the cluster, with container pods running on each node in the cluster. In the control layer, Kubernetes components that manage the cluster, and data about the state and configuration of the cluster. These components control that containers are running with sufficient numbers and with the required resources. The control layer also provides an API (kube-apiserver) for working with the orchestration system, a task scheduler (kube-scheduler), a container manager (kube-controller-manager), and a configuration database (etcd).The computational layer contains the pods and the services which are required to run them. These services include the node agent (kubelet), which monitors the work under the node, interacts with the control layer and executes commands coming from this layer, as well as the proxying service (kube-proxy), which handles network interactions inside and outside the cluster.

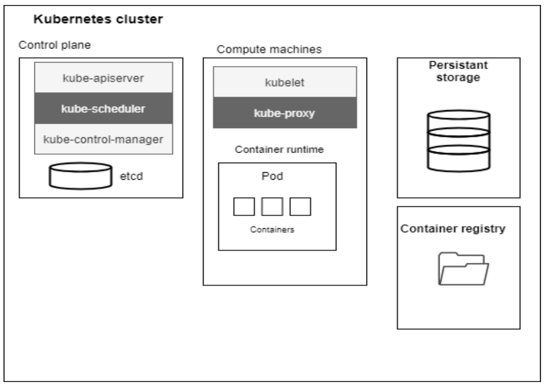

Fig.2 Kubernetes cluster architecture In addition to managing the containers that run applications, Kubernetes can also manage application data, allowing persistent storage to be accessed. At the same time, the interaction of the storage is built in a way that the services, when accessing it, do not need to know the underlying storage infrastructure, which simplifies the work with the storage. Also persistent storage relates to the cluster doesn't refer to the compute node therefore the lifetime of the cluster's data could be longer than the lifetime of the data on the nodes. The container orchestration stores images of containers that can be deployed in a cluster, images could be either a local registry or an outside one.

In addition to managing the containers that run applications, Kubernetes can also manage application data, allowing persistent storage to be accessed. At the same time, the interaction of the storage is built in a way that the services, when accessing it, do not need to know the underlying storage infrastructure, which simplifies the work with the storage. Also persistent storage relates to the cluster doesn't refer to the compute node therefore the lifetime of the cluster's data could be longer than the lifetime of the data on the nodes. The container orchestration stores images of containers that can be deployed in a cluster, images could be either a local registry or an outside one.

As our experiment program will be computed in Spark, it is necessary to observe Spark architecture details. System consists of a driver program, a cluster management node and computation nodes. The driver program is the main part of the Apache Spark cluster where the application context is created. The driver interacts with the cluster management node to distribute tasks between compute nodes. Cluster management node control the cluster. Apache Spark supports a variety of managers, including YARN, Mesos, and the standalone cluster manager [7]. This node allows distributed computing and coordinates the interaction of computing nodes with each other.

The computing node is the machine where the actual computation works. These nodes inform the cluster management node of the available resources on the node. At each computation node, a daemon starts, which monitors the state of this node.

The main abstraction in Apache Spark technology is Resilient Distributed Dataset or RDD [7]. It is a collection of items distributed across the cluster nodes which could work in parallel. Distribution of collection elements among the cluster nodes is made by the framework Spark. Also, using RDD provides automatic data recovery in case of node failure, which is very important part of building resilient systems. It is possible to create an RDD, as not only from an existing dataset in the program also from a dataset in an external storage system, for example, HDFS or any other data source assuming Hadoop InputFormat.

III. Experiment

Experiment will be processed in the part of Kubernetes infrastructure - Minikube. It is a special tool which is running locally on the one node at virtual machine. Architecture of Minikube which is running on local virtual machine is on Fig.3. Pods are running on Minikube, their amount is set in program's configuration. Partitions are distributed evenly on each pod. Pods are connected between themselves by virtual network connection. Network latency in this case is latency of the virtual network stack because all podes are located on the one virtual machine.

Experiment computation consist of two different types of calculation:

- compute-intensive calculation
- data-intensive calculation

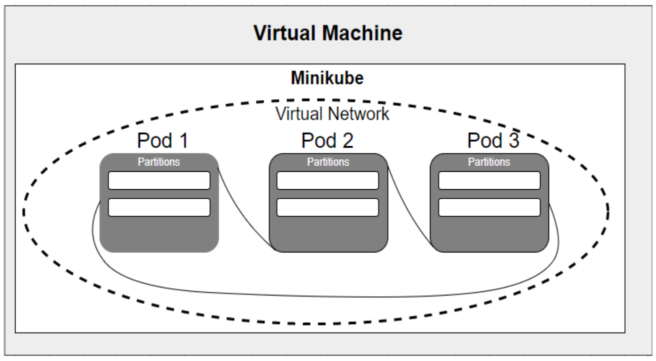

Fig.3. Architecture of Minikube running on virtual machine

First type of calculation is factorial computation of each element of BigInteger array. This type of calculation uses map transformation in Spark. Map function applies to each element of Resilient Distributed Dataset and it returns as the result new RDD. The length of BigInteger array for experiment will be 30,000.

Second type of calculation is data-intensive. To evaluate it we use Spark transformation groupBy to group BigInteger array by key. The length of this array for experiment will be the same as compute-intensive – 10,000,000.

To evaluate the level of concurrency of two types of compute-intensive calculation we measure the execution time of factorial computation depending on number of cores and amount of podes which are used for parallelizing. The number of partitions equal the number of podes.

On the Fig.4. there is the result of compute-intensive calculation. In this experiment was used number of cores (0.25,0.5,1) and number of podes (2,4,8,16). We can see that increasing number of podes can give a great boost in parallelizing, e.g. with core equal to 0,25 increasing number of pods from 2 to 16 has gave speed-up in 4,5 times.

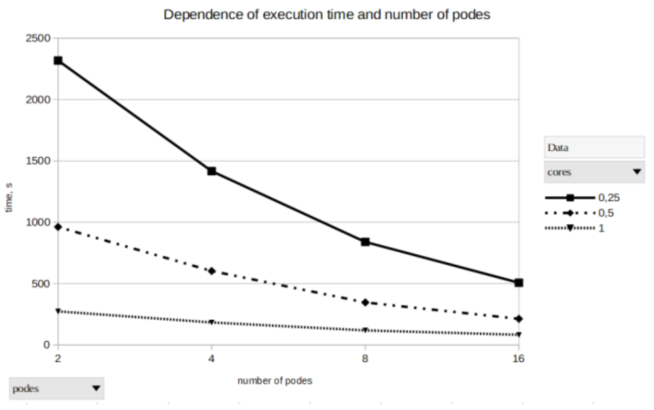

Fig.4. Dependence of execution time and number of pods for compute-intensive

On the Fig.5 there is the result of data-intensive calculation. In this experiment was used the same number of cores and number of podes as for compute-intensive calculation. The result of concurrency is very much different comparing with the previous experiment. In this case we can see that increasing number of podes do not give any speed-up. On the core equal to 1 there is no boost at all with increasing number of podes.

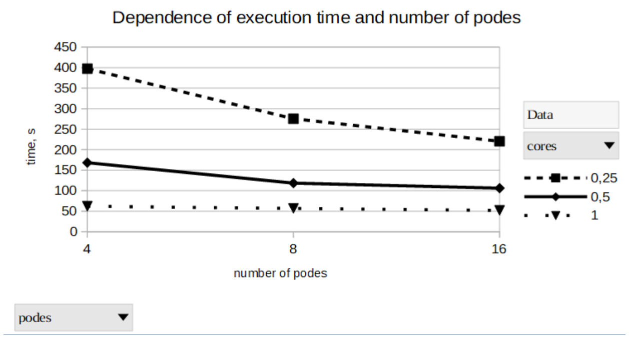

Fig.5. Dependence of execution time and number of pods for compute-intensive

IV. CONCLUSION

The aim of this work was finding out the correlation between concurrency level and amount of resources which could be used in computation. We have made the experiment where different types of calculations were done. As the result, we see that compute-intensive calculation is an excellent task for parallelizing. There is the correlation between increasing the number of resources, in our case it were number of podes and number of cores and concurrency level. Increasing number of resources has gave decreasing execution time of compute-intensive task. Data-intensive task's result differs and there was no boost after multiplication of podes and cores amount.

To sum up, it is clear that parallelizing level is strictly depends on type of program which should be computed. It could be useless to increase resources if type of program is inappropriate for parallelization.

REFERENCES

[1] William J. Song, Saibal Mukhopadhya,Sudhakar Yalamanchili. "Amdahl's law for lifetime reliability scaling in heterogeneous multicore processors 2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)", 2016, pp. 75-86,DOI:10.1109/HPCA.2016.7446097

[2] E. Casalicchio and V. Perciballi, "Auto-Scaling of Containers: The Impact of Relative and Absolute Metrics," 2017 IEEE 2nd International Workshops on Foundations and Applications of Self* Systems (FAS*W), Tucson, AZ, 2017, pp. 207-214, DOI: 10.1109/FAS-W.2017.149

[3] 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,DOI: 10.1109/EIConRus.2018.8317106

[4] 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, DOI:10.1109/EIConRus49466.2020.9039033

[5] Tengfei Hu, Yannian Wang, "A Kubernetes Autoscaler Based on Pod Replicas Prediction," 2021 Asia-Pacific Conference on Communications Technology and Computer Science (ACCTCS), 2021, pp. 358-360, DOI:10.1109/ACCTCS52002.2021.00053

[6] Kamran Ghane, "Big Data Pipeline with ML-Based and Crowd Sourced Dynamically Created and Maintained Columnar Data Warehouse for Structured and Unstructured Big Data",Information and Computer Technologies (ICICT) 2020 3rd International Conference on,2020, pp.60-67,DOI:10.1109/ICICT50521.2020.00018

[7] Preeti Gupta, Arun Sharma, Rajni Jindal, "An Approach for Optimizing the