Research on Docker Cluster Scheduling Based on Self-define Kubernetes Scheduler

Research on Docker Cluster Scheduling Based on Self-define Kubernetes Scheduler

Gengsheng Zheng[1], Yao Fu [1*], Tingting Wu[1]

[1]School of Computer Science and Engineering, Wuhan Institute of Technology, Wuhan, Hubei, 430205, China

[2] Hubei Key Laboratory of Intelligent Robot, Wuhan Institute of Technology, Wuhan, Hubei, 430073, China

[*]Corresponding author's e-mail: 21807010039@stu.wit.edu.cn

Abstract. To address the issue that native scheduler in kubernetes cannot realize load balancing and cannot give full play to the overall performance of the cluster, we put forward a scheduling strategy based on Docker cluster of self-define kubernetes scheduler. The kubernetes scheduler can improve the native kubernetes scheduling strategy using optimized predicate algorithm model and priority algorithm model. Moreover, it can also be used to conduct contrast experiment between the native scheduling strategy of kubernetes and that of self-define kubernetes scheduler. It is in VMware Workstation environment where Centos system is installed, and where Docker container service and kubernetes container orchestration engine are deployed to establish Docker container cluster. Experimental results show that cluster scheduling strategy of the self-define kubernetes scheduler is better than the native one, because it can save more system resources to improve the fairness of the cluster scheduling and the scheduling efficiency.

1. Introduction

In current times where cloud computing is increasingly applied, its key technology virtualization develops rapidly. Docker container technology attracts more and more researchers due to its portability and high performance.

Applications running on different physical servers cannot expand due to insufficient resource utilization, and it is costly to organize and maintain many physical servers. Therefore, due to their lower start up and termination overhead, containers are rapidly replacing Virtual Machines (VMs) in many cloud deployments, as the computation instance of choice [1]. Docker containers are more lightweight than virtual machines. Docker realizes more efficient virtualization, faster delivery and deployment, more efficient utilization of system resources, consistent running environment and easier expansion, maintenance, and migration. Micro-services and containers are becoming the de-facto way of building software and deploying workloads in an agile manner. Therefore, it is inevitable for Docker container to replace virtual machines in application deployment.

Clustering of Docker containers is the hot field of Docker research and application. Currently, the most applied Docker cluster scheduling strategies mainly include Docker Swarm issued by the Docker Company in 2013 and the Kubernetes by Google in 2014. Docker Swarm is a local cluster of the Dockerized distributed application, which optimizes resource utilization rate and tolerance failure service of the host on the basis of functions provided by Machine. However, given in the current reality,

build-in scheduling algorithm of Docker Swarm does not work well in case of inhomogeneous resource distribution[2].

Container management is the process of controlling a group of hosts, including adding or removing hosts from a cluster, obtaining the current status information of hosts or containers, and starting or managing processes[3]. Kubernetes provides a flexible framework for docker to run distributed systems. K8s is a container scheduling management system designed for production environment. It is designed to run native cloud applications everywhere. It has native support for automatic scaling, load balancing, high availability, rolling upgrade, service discovery and other functions[4].

The default scheduling method of current Kubernetes cluster is a static resource scheduling mechanism, which determines the priority of the scheduling according to request volume of applications to resources and does not consider the actual resources usage in a node. Therefore, the obtained scheduling priority cannot accurately reflect the actual resources usage available in that node.

The resource model used in the native Kubernetes scheduling algorithm only includes CPU and internal RAM, without considerations in the performance of the node. Besides, in the priority, for containers not set with lower limit to CUP or internal RAM, Kubernetes employs the same default value, regardless of the node performance[5].

1.1. Experimental environment and scheduling strategy

Experimental environment:

- Virtual machine installation environment: VMware Workstation Pro.
- Virtual machine node: Centos-7-x86-minimal-2003.
- kubernetes version: 1.15.1.
- Docker version: 19.03.12.

1.2. Self-define scheduling strategy

The load in cluster environment is dynamic, and corresponding resource demand is subject to dynamic change, posing a challenge to high-efficient utilization of cluster resources, thus the strategy of statically pre-allocating peak resources will bring huge waste of resources[6]. We generally need to, in Scheduler scheduling, consider three problems:

- Fairness: Each node can be allocated appropriate resources.
- Resource utilization: Cluster resources can be maximized.
- Scheduling efficiency: The scheduler can quickly complete scheduling work for large batches of Pods.

Based on three points above, schematic diagram of pods scheduling process for self-define Kubernetes scheduler is as shown in follows:

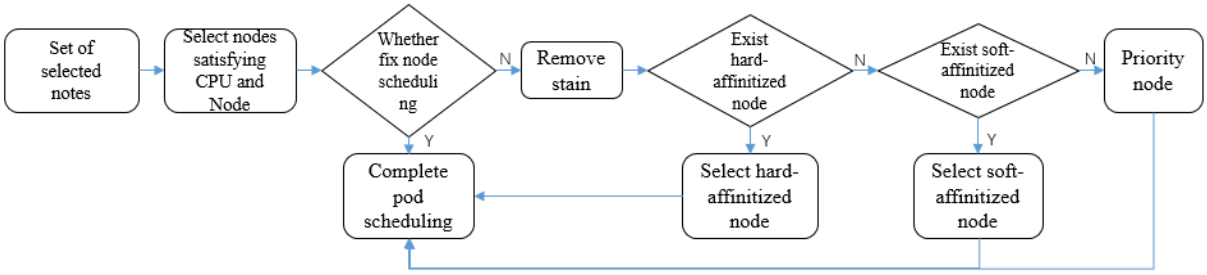

Figure 1.Schematic Diagram of Pods Scheduling Process

1.3. Scheduling algorithm model

Kubernetes is the Google-led container orchestration engine, whose resource scheduling algorithm is classified into predicate and priority[7]. In selecting nodes for non-fixed node scheduling, it is necessary to filter out the nodes with full CPU and RAM and unoccupied port. Its scheduling steps are: firstly, select nodes according their CUP and RAM use status, filter out these with CPU and RAM not meeting requirements; secondly, detect the node port occupancy status, and eliminate these with occupied ports.

This method can both select nodes meeting requirements and avoid that some node load is too high. Among them, the CPU and RAM filter conditions set different thresholds, and the cluster scheduling effect will be very different. CPU, RAM, and port are evaluated using equation (1) as follows:

$$T\left(Node_{(a)},\ C_{(a)},\ \mathrm{R}_{(a)},\ P_{(a)}\right)=\begin{cases} & \mathrm{Cpu}_{(a)}=true \\ Node_{(a)} & \&\&\ Ram_{(a)}=true \\ & \&\&\ Port_{(a)}=true \\ Null & else \end{cases} \tag{1}$$

Where: C, R, and P mean the respective Boolean values indicating whether CPU, RAM, and port meet requirements.

Where the predicate algorithm scheduling firstly screens out fixed node scheduling pod, and gets key value of fixed node scheduling in yaml file in such a way to select or pick out the fixed node scheduling Pod form to-be-scheduled Pod array. If it judges that there exists fixed node scheduling Node, directly select the designated node, otherwise the Pod scheduling will re-select from selected Node set in the scheduling rule of non-fixed node. Fixed-node scheduling is screened out using equation (2) as follows:

$$F\left(Ns, fixed\right)=\begin{cases} Node_{(fixed)} & fixed\neq null \\ Ns & fixed=null \end{cases} \tag{2}$$

Where: fixed represent for name of the node designated for Pod fixed node scheduling, and Ns for to-be-scheduled Node set.

After Node cluster filters out nodes meeting requirements, it is required to remove stain nodes of Pod scheduling, get key value for stain scheduling in yaml file to use it as a mark to remove the stain nodes of Pod scheduling. In case there exists a stain scheduling node, it shall be removed from the cluster. The stain scheduling node is removed using equation (3) as follows:

$$S\left(Ns,\ \ Node_{(a)},\ \ stain\right)=\begin{cases} Ns-Node_{(a)} & stain=true \\ Ns & stain=false \end{cases} \tag{3}$$

Where: stan represents for the Boolean value of Pod stain scheduling, and Ns for to-be-scheduling Node set.

After removing Pod scheduling stain node, Node cluster will provide affinity detection to residual nodes to assure whether these nodes are designated affinitized scheduling nodes. In case there exist designated hard-affinitized nodes and there is no designated hard-affinitized nodes in residual nodes, then the Pod cannot be scheduled. In case there exist soft-affinitized nodes and there are designated nodes in residual nodes, then this Pod can directly operate in designated nodes. In case there exist designated soft-affinitized nodes but no designated node in residual nodes, then this Pod shall be subject to priority scheduling strategy. Affinitized node scheduling is expressed using equation (4) as follows:

$$A\left(\mathrm{Ns},\ \ Ha,\ \ Sa\right)=\begin{cases} Node_{(Ha)} & Ha\neq null \\ Node_{(Sa)} & Ha=null\ \&\&Sa\neq null \\ \mathrm{Preferred}\left(Ns\right) & Ha=null\ \&\&Sa=null \end{cases} \tag{4}$$

Where Ha represents for the name of hard-affinitized node, and Sa for that of the soft one.

Traverse the remaining node set to calculate the score of each node. The Pod is executed by the node with the highest score. The node score is determined by the remaining CPU, RAM, and the ratio of the two. The higher the percentages of remaining CPU and RAM, the higher the score, and the closer the percentages of remaining CPU and RAM, the higher the score of node. This shows that the design is helpful to maximize the utilization rates of node, CUP, and RAM. Node score of priority scheduling algorithm is expressed using equation (5) as follows:

$$Score_{\left(Node_{(a)}\right)}=\left(2-C_{(a)}-S_{(a)}\right)\Big/\left|\frac{1-C_{(a)}}{1-S_{(a)}}\right| \tag{5}$$

Where C and S respectively show CUP utilization rate and RAM occupancy rate.

2. Experiment and analysis

2.1. Experiment procedure

Prepare mirror file for future use, including Centos-7Linux system mirror and kubeadm mirror. Use VMware Workstation to install six Centos-7 virtual machines, of which one acts as Master node and others act as work nodes. Among them, work nodes 1 and 2 represent for high-resource node, 3 and 4 for low-resource node, and 5 for extreme-low-resource node. Virtual machine nodes configuration table is as shown in follows:

Table 1.Virtual machine nodes configuration table.

Name of Node	Quantity of processor (EA)	Quantity of core contained in the processor (EA)	RAM (GB)
MasterNode	1	2	2
WorkNode1	2	2	4
WorkNode2	2	2	4
WorkNode3	2	1	2
WorkNode4	2	1	2
WorkNode5	1	1	2

Complete system initialization for each virtual machine.

Kubeadm is used to install Kubernetes and deploy Docker container service and kubernetes cluster. Prior to the delpoyment of Kubernetes cluster, it is required to prepare basic installing environment of each node. Nodes in Kubernetes cluster mainly play three roles: Etcd server, Kubernetes API server, and application node (Workers)[8].

Build private mirror warehouse. In inner work application environment or cloud application environment, single node or cloud host will generally deploy Docker CE through YUM source. Docker CE may be subject to manual addressing of dependent relationship to directly install rpm for deployment. Docker CE is provided with inner warehouse mechanism. There are many command formats and parameters for option in building a warehouse.

2.2. Results

After scheduling experiment for native kubernetes scheduler and the self-define one through adding different quantities of Pods, CPU utilization rate and RAM occupancy rate of each work node are as shown in follows:

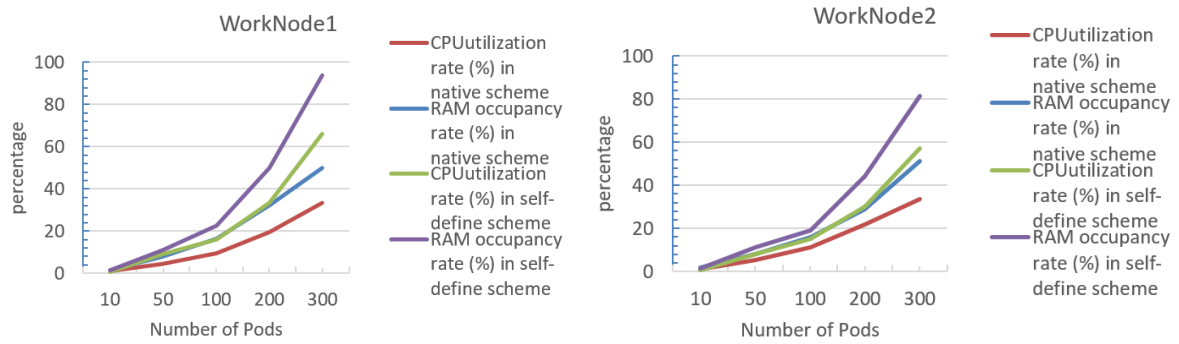

Figure 2. CPU utilization rate and RAM occupancy rate of work node1&node2

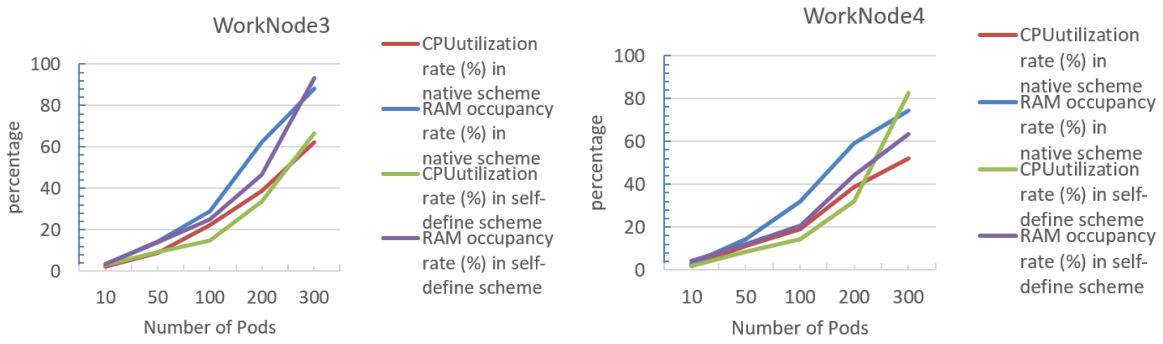

Figure 3. CPU utilization rate and RAM occupancy rate of work node3&node4

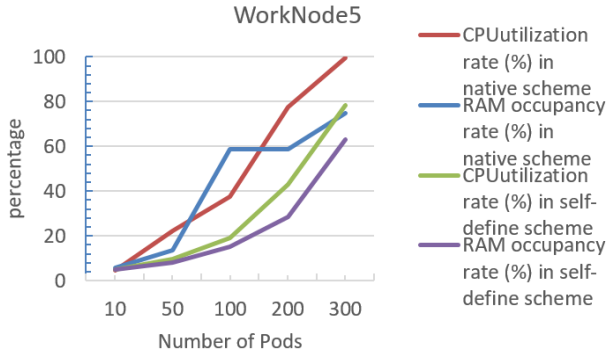

Figure 4. CPU utilization rate and RAM occupancy rate of work node5

Figs. 2, 3, and 4 are respectively the line charts of CPU utilization rate and RAM occupancy rate of work nodes 1, 2, 3, 4, and 5 in different quantities of pods.

3. Conclusion

Conclusions after specifically analyzing results are as shown in follows:

Table 2. Experimental Conclusion

Resource utilization in native scheduling strategy			Resource utilization in self-define scheduling strategy		
High-resource node	Low-resource node	Extreme-low-resource node	High-resource node	Low-resource node	Extreme-low-resource node
Low utilization rate Node resources are not maximized.	High then low utilization rate Node resources are not maximized.	Overhigh utilization rate Node is easy to paralysis.	High utilization rate Node resources are maximized.	High then low utilization rate Node resources are maximized.	Low utilization rate Node is not easy to paralysis.

(1) Compared to native kubernetes scheduling strategy, the self-define one can give better scheduling fairness, and work nodes with high-resource configuration will have higher CPU and RAM utilization rates.

(2) Self-define kubernetes scheduling strategy can make cluster resources better maximized than the native one.

(3) Self-define kubernetes scheduling strategy can make the work node with low-resource allocation slower, reach the resource utilization peak at a lower speed, and make the work node with extreme-low-resource not fall into paralysis, improving the efficiency of cluster scheduling.

4. Discussion

This paper, through using kubernetes container orchestration engine to schedule Docker container cluster, introduces a self-define kubernetes scheduling scheme. In this paper, we use the predicate algorithm and priority algorithm for optimizing kubernetes scheduler in such a way to improve native kubernetes scheduling strategy. It is proved that, through experiment, the improved kubernetes scheduling strategy, compared with the native one, not only improve the fairness of kubernetes cluster scheduling and cluster scheduling efficenty, but also enhance cluster resource utilization rate. Docker container technology is expected to be further studied in the filed of ELK log system application.

References

[1] Víctor Medel,Rafael Tolosana-Calasanz,José Ángel Bañares,et al. (2018)Characterising resource management performance in Kubernetes. Computers and Electrical Engineering., 68.
[2] Li, D.G.(2019)Research on Scheduling Algorithms Based on Docker.Xi'an Technological University.
[3] WANG,J.X, GUO, L. (2018)Enterprise Container Cloud PaaS Solution Based on Kuerbnetes and Docker. Journal of Shanghai Ship and Shipping Research Institute.,41(03):51-57.
[4] Tian, Y.F,Wang, Zh. (2018)PAAS architecture based on k8s and investigation and analysis of typical products in the industry.Scientific and Technological Innovation.,(06):97-98.
[5] Tan, L,Tao, H.C(2019) An Improved Kubernetes Priority Algorithm based on Load Balancing.Journal of Chengdu University of Information Technology., 34(03):228-231.
[6] Zhang, B.T,Rui, J.W,Zhou, P,et al.(2017) Research on Cluster Scheduling Based on CoreOS Oriented Load Integration.Computer Systems & Applications., 26(11):67-75.
[7] Chang, X.Zh,Jiao, W.B.(2020) Improvement and Implementation of Kubernetes Resource Scheduling Algorithm.Computer Systems & Applications., 29(07):256-259.
[8] Sheng, Y.B,Zhou, Q.L,You, W.Q, et al.(2018) Exploration and Practice of Construction for a Kubernetes High Availability Cluster.Computer Knowledge and Technology.,14(26):40-43.