Grenze International Journal of Engineering and Technology, Jan Issue

Initial Docker Container Placement Algorithm in Cloud Computing

Dr.Jigna Acharya
Lecturer, K. D. Polytechnic, Patan, Gujarat, India
Email: jignaforever@gmail.com

***Abstract*—Recent developments in Cloud Computing have resulted in a new virtualization technology called Docker Containers, which supports services that are placed on machines that share resources of the operating system. Additionally, it is a lightweight virtualization technology that can be deployed on cloud machines. Thousands of Docker containers can be placed to handle millions of user requests. Using Docker architecture, failures are minimized as well. It is very challenging, however, to manage and place various types of containers on cloud machines. Docker Swarm is a solution that supports this challenge. Through it, containers are placed and managed on clusters of machines. Swarm supports a default strategy of container placement called spread strategy, which distributes containers equally across machines in the cluster. Spread strategy does not consider how to best utilize the resources of the docker machine. As a result, load becomes unequal on machines, like some become overloaded and some under loaded. Our main goal is to distribute containers on the docker machine based on how much of its resources are being used. To determine where to place containers on each machine, we consider how much memory is being used. We propose an algorithm for determining which machine has the freest memory and placing a container on that machine. Experimental results are compared between our proposed algorithm and the default spread algorithm. By placing containers on machines in such a way as to prevent overloading or under loading, we have proposed a container placement algorithm. In addition, this strategy minimizes the number of physical machines in a docker cluster.**

***Index Terms*— Container placement, Docker, Cloud computing, Cluster, Spread Strategy.**

I. INTRODUCTION

In today's world, cloud computing is a popular method of providing virtualized IT resources online. The cloud data center relies heavily on virtualization technology. Virtualization techniques used by cloud data centers are mostly hypervisor based or container based. Hypervisor-based virtualization technology allows running a separate OS on a virtual machine. In contrast, container-based virtualization runs containers directly on the physical machines in the cluster. Such a virtualization technology is lightweight. The overhead of container-based virtualization is low compared with virtual machine virtualization.[1].

One of the most popular technologies for virtualizing containers is Docker. Containers can be virtualized using Docker, an open source framework. Docker containers run independently of the host computer. As opposed to hypervisor based virtualization, users can design, test, and deploy their task more quickly. Utilizing isolated containers, resources can be restricted, and processes can be granted access to a private view of the operating system, including their own process IDs, file system structures, and network interfaces. [2] Docker consumes

Grenze ID: 01.GIJET.10.1.530

little CPU or memory on physical machines. [3] Kubernetess, Docker Swarm, Apache, and Mesos are among the cloud computing frameworks that provide container-based virtualization. Container management and handling in Docker-based clouds is a very challenging task. Docker swarm's main challenge is to schedule or place container applications on available machines. The orchestration system should place applications on a resource as soon as possible as soon as they are submitted for deployment, taking into account the application's specific constraints and maximizing utilization. [4] Docker swarm frameworks deploy containers on machines using spread scheduling algorithms. The disadvantage of spread algorithms is that they distribute containers equally across all machines available. Therefore, the load distribution is unbalanced in this method.

The aim of this research is to distribute container applications in docker swarm according to the resource utilization of every machine in the cluster. In this case, we assessed the memory usage of every machine within the cluster. Using our new initial container placement algorithm, we can find the machine with the maximum amount of memory available to place container applications. We compare our proposed placement algorithm with the existing algorithm.

Following is the organization of the paper. Section 2 presents Related Works. Defining the problem and formulating it follows in Section 3. Section 4 presents an algorithm proposal. Section 5 presents the experimental environment, while results and its discussion will discuss in the section 6. Section 7 describes conclusions and future work.

II. Related Work

Continue text A major component of cloud computing is placement and load balancing, which directly impacts performance and load. Docker-based cloud computing research focuses on container placement and load balancing. This paper describes existing work in two areas: initial placement of containers and load balancing.

We begin with a review of existing container placement research. To minimize energy consumption, Smimite et al. [5] proposed container placement and migration algorithms that take into consideration the QoS requirements of different users. To reduce unnecessary power consumption in the data center, a dynamic approach using the threshold of RAM usage for hosts and virtual machines is recommended. For container placement, Xu et al. [6] propose a stable machine theory algorithm. The algorithm reduces customer response times and increases resource utilization. The new container placement scheme proposed in et al.[7] Is dubbed Dynamic and Resource Aware Placement Scheme (DRAPS). Based on dynamic demands and concurrent services, it distributes tasks to different containers. An ant colony method based multi criteria container placement algorithm was proposed by Lin et al. [8]. In this algorithm, the validity of pheromone updates is verified using a quality evaluation functions of the feasible solutions, and multi-objective heuristics are combined to improve the selection probability. Guo et al. [9] Given Scheduling strategy for containers using neighborhood division in micro services (CSBND).It is important to balance system load and respond quickly to optimize system performance. In their paper, Al-Ahmad et al.[10] propose a new approach for scheduling container services that is based on availability-awareness in order to enhance accessibility of the cloud platform for container-based apps. Containers are scheduled efficiently by selecting VMs and hosts with higher availability values. Using different types of Service Level Agreements (SLA), Cerin et al.[11] Create new scheduling algorithms for containers. The provisioning process involves provisioning a container that will execute a service based on the SLA class of the user, based on a dynamic calculation of how many CPU cores the container should have based on the load on the infrastructure's parallel machines. Li et al.[12] Developed a he Docker platform uses a particle swarm optimization algorithm for container scheduling. This algorithm improved resource utilization and load distribution. Burvall et al.[13] Developed an algorithm for placing multi-objective containers based on an ant colony algorithm. Using replicated redundant containers, it optimizes network usage and virtual machine (VM) costs in simulation and when deployed in Docker Swarm mode in a cloud environment.

Secondly, here is some existing work on load balancing in docker cloud. Li et al.[14] proposed an RPBG strategy for load balancing in cloud clusters. This method combines collaborative resource placement, dynamic resource evaluation, and automatic node load adjustment to accommodate the heterogeneity of each node's memory, CPU, and network I/O and resolve container conflicts based on graph theory. Chiang et al. [15] developed a clustering algorithm based on machine learning and contention awareness. This algorithm ensures proper load balancing in cloud clusters. Mendes et al. [16] proposed a new energy conscious container scheduling algorithm to enhance the CPU and memory utilization of cloud centers. Fan et al.[17] Propose a locality live migration strategy that can significantly improve the utilization of system resources and can improve the balance between system resources. Li et al. [18] Proposed a multi-algorithm collaboration scheduling strategy for docker containers based on different algorithms to improve load balancing. Monsalve et al. [19]

make Dynamic CPU Resource Allocation in Containerized Cloud Environments may help control and mitigate some of the negative performance impacts of oversubscription.

III. Problem Description & Formulation

In the cloud computing center, resource utilization and load balancing are two main issues. The focus of this paper is on load balancing of cloud data centers If Schedule containers in a cloud center on one machine with an adequate amount of CPU and memory, you can achieve compact network conduction and improve application quality in a cluster to some extent. Based on the memory usage of every machine in the cluster, we designed and implemented an initial container placement algorithm. Prior to placing a container on a machine, we calculate the free or available memory. Container will be placed on a machine with maximum free memory in the Docker cloud cluster.

Suppose we have $T = \{t_1, t_2, \dots, t_n\}$applications to make and run containers in the environment of cloud computing. Each machine in the cluster is represented by$S = \{s_1, s_2, \dots, s_m\}$. Reading the application and running the container on the machine takes place at the following time:

$$T \to [0, \infty]$$

$$, \text{Time } (t_j) = \text{installation time of application} t_j$$

$$= \text{Time } (j)$$

We propose a placement algorithm that determines which machine in the cluster has the highest memory usage at a particular moment in order to place containers. The details are as follows,

$$MEM: [0, \infty] \to IR^n$$

$$\text{MEM (Time (i))} = (CU_1, CU_2, \dots, CU_n)_{\text{Time(i)}}$$

By determining the memory usage of all machines,

our proposed placement algorithm will determine which machine in the cluster has the freest memory. Using the max function, it is defined as follows:

$$Min : IR^n \to IR$$

$$Min : (x_1, x_2, \dots, x_n) = x_i$$

$$Min: (CU_1, CU_2, \dots, CU_n) = CU_i$$

Max function will return maximum free memory of machine. The index of the system with the highest memory availability can be represented as follows:

$$I: IR \to S$$

$$I(CU_i) = S_i$$

Here is our strategy for placing the container on the machine after obtaining the index of the machine:

$$System: T \to S$$

$$\text{System } (T) = \{(I) \circ (Min) \circ (CUO) \circ (Time) \circ (t_j)\}$$

$$= \{(I) \circ (Min) \circ (CUO)\} \circ \{Time(j)\}$$

$$= (I) \circ (Min)(CU_1, \dots, CU_n)_{\text{Time(i)}}$$

$$= I(CU_i)$$

$$= s_i$$

$$System(t_j) = s_i$$

IV. Proposed Algorithm

Algorithm: Proposed Placement algorithm for containers

Input: $\{S_i\}$, $\{T_i\}$

Output: $System(t_j) = s_i$

1 For each T_i do

2 For each $S_i$do

3 Calculate MEM(Time(i))

4 End

5 For each CU_i do

6 Calculate $Min(U_i)$

7 End

8 Place container T_i on S_i

9 End

V. EXPERIMENTAL ENVIRONMENT

We use three docker engine machines inside physical machine in cluster. Table-I represents he detail specification of physical machine in this research work. We create docker swarm that represents one manager machine and two worker machine using docker engine machines. Inside docker swarm, we create workload file contains various different docker image application. Also for monitoring load of machines we use monitoring tools. The details of the container workload file and the monitoring tools are given in Table-II.

TABLE I. PHYSICAL COMPUTER SPECIFICATION

Item	Specification
Processor	Intel Core I3,3.4 GHz
CPU	1 core
Memory	5GB
Hard Disk	500 GB
Operating system	Linux Ubuntu 18.04

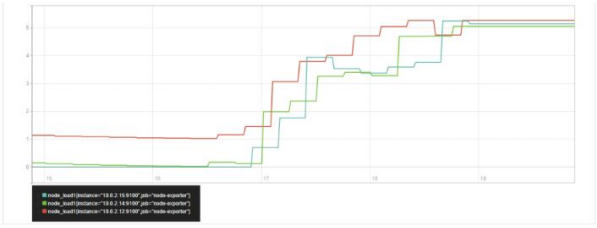
TABLE II. DOCKER SWARM APPLICATION SPECIFICATION

Item	Specification
Workload File	Docker image applications (Ex.-nginx, apche web server, sorting of arrayetc)
Load Monitoring Tools	Prometheus, grafana, cadvisor and node exporter

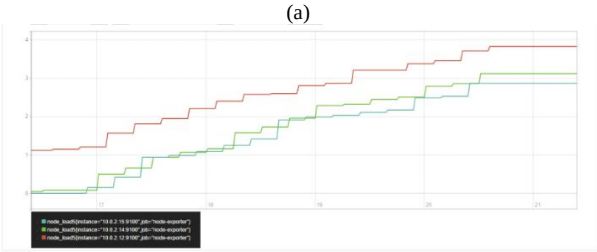
To monitor utilization of memory, we create python script that run on manger machine. This script use Prometheus and node exporter tool for fetching memory utilization of every machine in JSON format. Then it will calculate the available free memory of every machine and find the machine having maximum free memory. The maximum free memory machine is then selected for placing new container in the cluster.

VI. RESULTS AND DISCUSSION

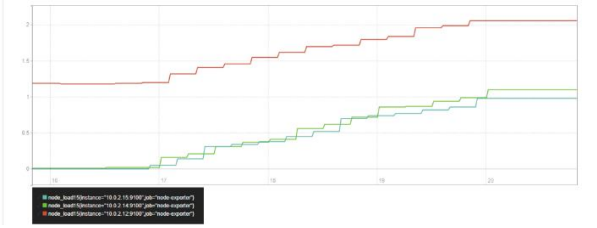
Containers are created by reading images one by one from workload files on the manager machines. Starting and placing initial containers on machines is done using existing spread container placement algorithm as well as our proposed initial container placement algorithm. A proposed initial container placement algorithm aims to place containers on the machine with the lowest memory usage. The following figures compare the results with those obtained by the spread container placement algorithm. Here we take three machines for comparisons and all are indicated with different color and labels in load monitoring tool Prometheus. It gets data with the help of node exporter. Different Prometheus queries are applied for comparing results. Three machines were tested under different conditions using these queries.
In Figure 1, (a), (b), and (c), you can see the load on every machine in a Docker cluster with different number of containers and also with different time intervals using the existing spread container placement algorithm.



(a)



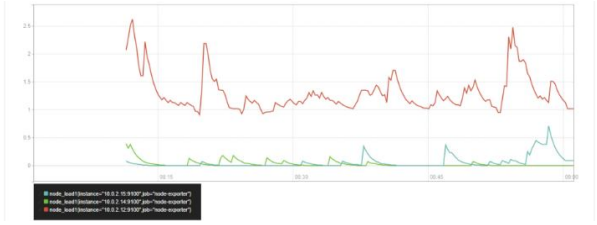
(b)



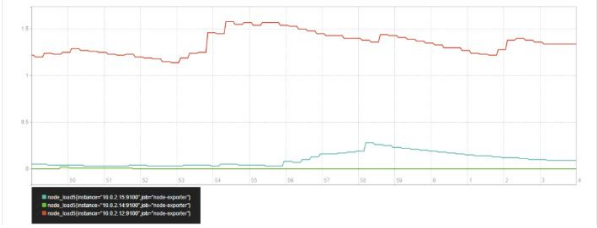
(c)

Figure 1 Initial container placement using spread algorithm (a) Node_load1 usage query (b) Node_load5 usage query (c) Node_load15 usage query

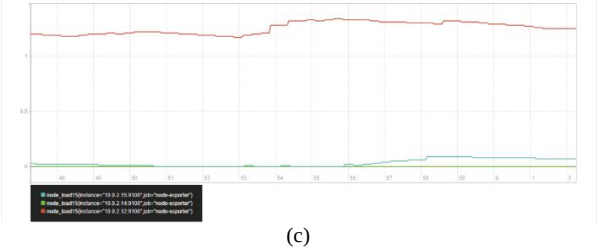
Figure 2 (a), (b), and (c) depict the load of each machine using the proposed initial container placement algorithm with different number of containers at various intervals of time.



(a)



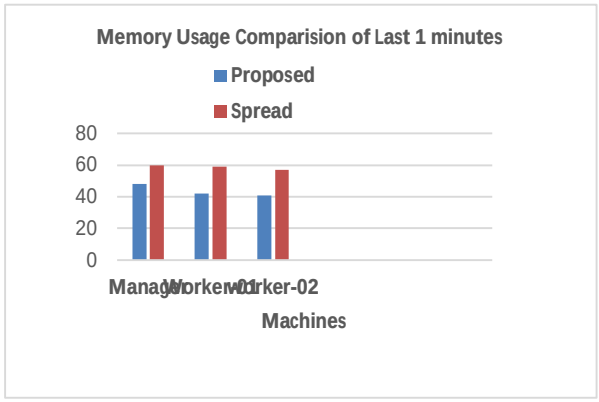
(b)



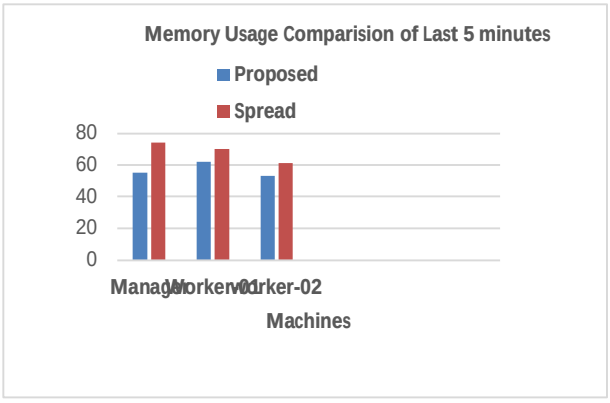
(c)

Figure 2 Initial container placement using proposed algorithm (a) Node_load1 usage query (b) Node_load5 usage query (c) Node_load15 usage query

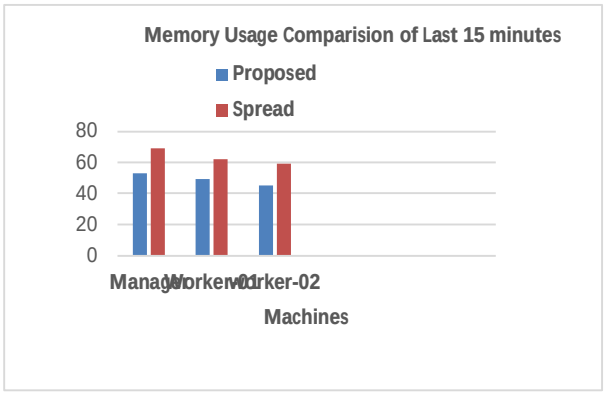
We compare the proposed initial container placement algorithm and the existing spread algorithm. Proposed initial container placement algorithm is an efficient load balancing algorithm, which consider the problem of overload and also distribute the container on machine based on memory utilization of every machine. Figure 3 shows comparison results of proposed initial container placement and existing spread algorithm in terms of Memory usage (%) of every machine at different time interval under different sizes.



(a)



(b)



(c)

Figure 3 Memory usages (%) of every machine

VII. Conclusions and Future Work

After the experiment, we can conclude that existing spread container placement algorithm never consider memory usage of machine while placing container on machines. So, that loads of machines become unequal. Thus, we developed a container placement scheduling algorithm for Docker Swarm.

Prior to placing containers, find out the memory usage of each machine, and then use the one with the lowest memory usage to start and place containers. This algorithm can balance load properly of machine and decrease resource waste. In the future work, we will focus on combined memory and CPU usage of machine before placing containers on machine. In addition, we will attempt to implement other scheduling strategies such as bin pack and random and compare them with our own.

REFERENCES

[1] M. R. M. Bella, M. Data, and W. Yahya, "Web Server Load Balancing Based On Memory Utilization Using Docker Swarm," 3rd Int. Conf. Sustain. Inf. Eng. Technol. SIET 2018 - Proc., pp. 220–223, 2018.

[2] E. N. Preeth, J. P. Mulerickal, B. Paul, and Y. Sastri, "Evaluation of Docker containers based on hardware utilization," 2015 Int. Conf. Control. Commun. Comput. India, ICCC 2015, no. November, pp. 697–700, 2016.

[3] Z. Kozhirbayev and R. O. Sinnott, "A performance comparison of container-based technologies for the Cloud," Futur. Gener. Comput. Syst., vol. 68, pp. 175–182, 2017.

[4] M. A. Rodriguez and R. Buyya, "Containers Orchestration with Cost-Efficient Autoscaling in Cloud Computing Environments," 2018.

[5] O. Smimite and K. Afdel, "Containers Placement and Migration on Cloud System," Int. J. Comput. Appl., vol. 176, no. 35, pp. 9–18, 2020.

[6] X. Xu, H. Yu, and X. Pei, "A novel resource scheduling approach in container based clouds," Proc. - 17th IEEE Int. Conf. Comput. Sci. Eng. CSE 2014, Jointly with 13th IEEE Int. Conf. Ubiquitous Comput. Commun. IUCC 2014, 13th Int. Symp. Pervasive Syst. , pp. 257–264, 2015.

[7] S. In et al., "A Resource-aware Container Management Scheme in a Heterogeneous Cluster BY," no. May, 2021.

[8] M. Lin, J. Xi, W. Bai, and J. Wu, "Ant Colony Algorithm for Multi-Objective Optimization of Container-Based Microservice Scheduling in Cloud," IEEE Access, vol. 7, pp. 83088–83100, 2019.

[9] Y. Guo and W. Yao, "A container scheduling strategy based on neighborhood division in micro service," IEEE/IFIP Netw. Oper. Manag. Symp. Cogn. Manag. a Cyber World, NOMS 2018, pp. 1–6, 2018.

[10] Y. Alahmad, T. Daradkeh, and A. Agarwal, "Availability-Aware Container Scheduler for Application Services in Cloud," 2018 IEEE 37th Int. Perform. Comput. Commun. Conf. IPCCC 2018, pp. 1–6, 2018.

[11] C. Cérin, T. Menouer, W. Saad, and W. Ben Abdallah, "A New Docker Swarm Scheduling Strategy," Proc. - 2017 IEEE 7th Int. Symp. Cloud Serv. Comput. SC2 2017, vol. 2018-Janua, pp. 112–117, 2018.

[12] L. Li, J. Chen, and W. Yan, "A particle swarm optimization-based container scheduling algorithm of docker platform," ACM Int. Conf. Proceeding Ser., pp. 12–17, 2018.

[13] B. Burvall, "Improvement of Container Placement Using Multi-Objective Ant Colony Optimization," 2019.

[14] H. Li, N. Chen, B. Liang, and C. Liu, "RPBG: Intelligent orchestration strategy of heterogeneous docker cluster based on graph theory," Proc. 2019 IEEE 23rd Int. Conf. Comput. Support. Coop. Work Des. CSCWD 2019, pp. 488–493, 2019.

[15] R. C. Chiang, "Contention-aware container placement strategy for docker swarm with machine learning based clustering algorithms," Cluster Comput., vol. 2, 2020.

[16] S. Mendes, J. Simão, and L. Veiga, "Oversubscribing micro-clouds with energy-aware containers scheduling," Proc. ACM Symp. Appl. Comput., vol. Part F1477, pp. 130–137, 2019.

[17] W. Fan, Z. Han, Y. Zhang, and R. Wang, "A Locality Live Migration Strategy Based on Docker Containers," Proc. - 4th IEEE Int. Conf. Big Data Secur. Cloud, BigDataSecurity 2018, 4th IEEE Int. Conf. High Perform. Smart Comput. HPSC 2018 3rd IEEE Int. Conf. Intell. Data Secur., pp. 51–54, 2018.

[18] Q. Li and Y. Fang, "Multi-algorithm collaboration scheduling strategy for docker container," 2017 Int. Conf. Comput. Syst. Electron. Control. ICCSEC 2017, pp. 1367–1371, 2018.

[19] J. Monsalve, A. Landwehr, and M. Taufer, "Dynamic CPU resource allocation in containerized cloud environments," Proc. - IEEE Int. Conf. Clust. Comput. ICCC, vol. 2015-Octob, pp. 535–536, 2015.