Performance overhead of CO frameworks for multi-tenant database deployments

Indranil Ghosh

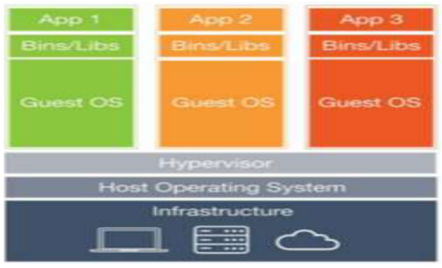
Summer term 2020

1 Introduction

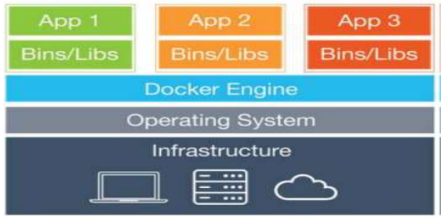
The paper promises to look into the production assessment of Docker engine in the current accentuation and industry acquisition of Container Orchestration(CO) framework. Like Docker Swarm and Kubernetes for the motive of spontaneous placement of cloud based features. Specifically when to manage a CPU bound CPU-bound Cassandra workload in Open Stack.

2 Comparison between Docker Engine and Virtual IP networking

Virtualisation mean abstracting the operating system, storage or network and give a feeling of real hardware or software. The concept of virtualization has been used in cloud computing. Virtualisation resources will be taking key role in solving the problem by the utilization of cloud computing technique. There are mainly two types of virtualisation. One is Full virtualisation and another one is Para virtualisation. In Full virtualisation the hardware simulation will allow the guest os to be run in isolation. Hypervisor is a virtual platform which will reside between Host operating system and guest operating system. Where as in Para virtualisation will not be done for VMs. The installation of Hypervisor will be done on physical server and the guest OS will be installed into the environment. The main difference here is that the guest are aware of their virtualisation but in Full virtualisation the guest will not have this in their knowledge. On the other hand Docker is a container based technology. Container is a set of isolated process that is run by image files which provides the necessary support to run the process. The docker tool will manage the container and this will be used to provide OS level virtualisation. Containers will use only the resources which will be required. The main advantages that we can get by Docker over Virtualisation are speed, portability and scalability[2].



Virtual Machine architecture



Docker Container architecture

[2]

3 Architecture for running multi node Cassandra Cluster with Swarm Cluster and Kubernetes

3.1 Apache Cassandra

The advancement of social media made us to realize the importance of the database system that is capable of handling unstructured data unlike relational database management system. Apache Cassandra has found to be a good option for this. Cassandra is against the single point failure as it is using P2P model. Although Casandra does not support the full relational data model but for improving the storage performance it uses data partition and data replication technologies[3].

3.2 Docker

Docker is an open source project that is allow the developer to wrap up their code and deploy to a sand box environment called container[3]. Containerization is a process of deploying the application by the help of container. The main advantage of containerization is containers are light weight, loosely coupled, secure, scalable, portable and flexible. Docker image provide all the supports needed to run the container. Each container is allowed to access it's own private file system provided by docker image. Docker commands are provided by the Docker client to the user to communicate with Docker demon.

3.3 Docker Swarm

Docker swarm is a container orchestration tool inbuilt in Docker Engine for distributed system which involves Docker Node, Docker Services.There are manly two node types.

- Docker Manager
- Docker Worker

3.4 Kubernetes

Kubernetes or k8s (kubernetes) is an open-source container orchestration system for automating deployment, scaling, and managing of applications. It was formerly developed by Google and now it is maintained by the Cloud Native Computing Foundation.

3.5 Problem Statement

Two approaches of deploying Cassandra cluster have been practiced so far. Either each Cassandra instance can be deployed on each server or server can be virtualized and multiple VMs can be initiated to deploy Cassandra clusters. But the performance overhead is much larger than the native physical sever. The study has pointed out this issue and Docker Swarm has been introduced to deploy Cassandra clusters[3].

3.6 Overview of compared deployments

There has been a comparison brought up by 4 deployments of Cassandra cluster with three nodes[1].

- A native deployment where Cassandra v2.0.17 is directly installed in an Ubuntu VM
- A Docker engine deployment where database containers are started from the official cassandra:2.0 Docker image in similarly sized Ubuntu VMs
- A Docker Swarm deployment where database containers are inter-connected by means of a stable Service IP address
- A Kubernetes deployment where database containers discover each other via a DNS name

Mesos-based CO frameworks have been excluded at the time of running the experiment as in any Mesos-based CO framework the virtual networking solution of Mesos has not been used.

3.7 Result

At the first level after inspecting the memory usage it has been found that the Docker image with Cassandra 2.0 consumes less memory in comparison to natively installed deployment as Docker image has been configured for less memory consumption. Whereas Kubernetes consumes more memory. It can be happening because of facilitating more features

than Docker swarm. The second level of comparison has been done based on how quickly they can reach 100 percentage CPU utilization and by average response latency. With respect to response latency Docker performed in much better way. In both the case of Docker Swarm and Kubernetes the network usage appeared high. We have also found that although Kubernetes consumes more CPU time at soft interrupts than Docker Swarm, it still has proven it's betterment in terms of response latency. This result has been found b inspecting the I/O rate at which Cassandra is able to commit it's write request into it's commit log. In comparison with Docker engine deployment the disk I/O rate of Docker Swarm has found much lower. Where as the disk I/O rate of Kubernetes has not deceased like Docker Swarm. So we can conclude that in host mode the Docker engine deployment of Cassandra behaves in par with the native virtual machine deployment[1].

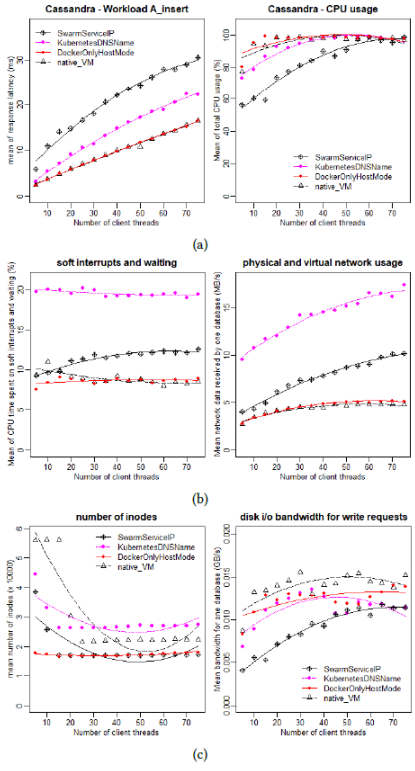


Figure 1: Performance overhead of Kubernetes and Docker Swarm in comparison to VM+Docker and VM-based deployments[1]

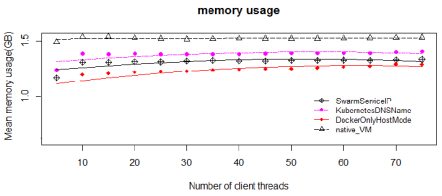


Figure 2: Memory usage[1]

3.8 Proposed System

In the current situation each Cassandra instance would be host by each server system and based on P2P connection establishment the clusters can be established. But in this kind of structure we can find unnecessary connection between every system. In this design one instance of Cassandra has been deployed to each system, so if the server specification is high then there can be a possibility of high amount of wastage[3].In the production environment and number of processors per system will not be utilized. the server will be of high configuration and every processors will not be utilized. This issue can be overcome by Docker. By Docker every container can get their own IP address and sandbox environment. According to this proposed model multiple instances of the container are deployed on the same physical host server machine[3].There are few drawbacks of this proposed model. If physical disk fails then a problem of high availability will occur.This issue an be overcome by storing the data of different containers into different physical storage of the same machine[3]. Because of the cheaper price of storage device than server machine the total cost can be reduced.

3.9 Experiment

To test the suitability of the proposed model experiment has been performed on this model Linux systems with the following specifications in the below mentioned table.

Create, Read,Update and Delete operations have been run in Cassandra system to take the response time of these operations. A client program was written by Java to test the operations in the Cassandra database in a specific sequence. 2-Node and 3-Node Cassandra clusters were used to run the test on Server machines and also on Docker[3]. The

System Requirement	Specification
Linux system	16 GB RAM, Intel (R) Core (TM) i5-3570
processor	@ 3.40 GHz, Quad Core
Server	64 bit Oracle Linux Server
Operating system	6.7 operating system Server

Table 1: System specification for the suitability of the proposed model experiment. [3]

Operations	Gain Percentage for 2-Node Docker Cassandra Cluster	Gain Percentage for 3-Node Docker Cassandra Cluster
Create	49.56%	60.67%
Read	52.40%	63.28%
Update	48.30%	60.06%
Delete	48.94%	63.73%

[3]

below table result indicates how performance has been improved by introduction of Docker for Cassandra Cluster deployment.We can observe an improvement of almost 50 percent for CRUD operations on 2-Node Cluster and more than 60 percent for 3-Node Cluster by the implementation of Docker[3].

4 Conclusion

In this paper the performance overhead has been evaluated of container orchestration frameworks to run and manage database clusters in an Open Stack private cloud by a comparison with native and Docker engine deployments in OpenStack VMs. In host mode networking the performance overhead of Docker engine has been found relatively small. Though substantial overhead will come into picture for extra CPU consumption for virtual network bridges in service networking solutions of CO frameworks. Finally it has been concluded that Co framework should further advance the isolation of container networking approaches that depends on host mode networking and should refine the performance of volume plugins for local persistent storage.

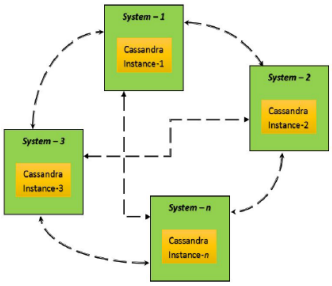


Fig. 1 Existing Cluster Configuration

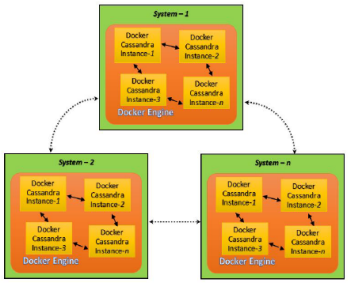


Figure 3: Architecture of Existing Cluster Configuration and Proposed Cluster Configuration[3]

References

[1] E. Truyen, D. Van Landuyt, B. Lagaisse, and W. Joosen. “Performance overhead of container orchestration frameworks for management of multi-tenant database deployments”. In: *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*. 2019, pp. 156–159.

[2] B. B. Rad, H. J. Bhatti, and M. Ahmadi. “An introduction to docker and analysis of its performance”. In: *International Journal of Computer Science and Network Security (IJCSNS)* 17.3 (2017), p. 228.

[3] A. Jaison, N. Kavitha, and P. Janardhanan. “Docker for optimization of cassandra NoSQL deployments on node limited clusters”. In: *2016 International Conference on Emerging Technological Trends (ICETT)*. IEEE. 2016, pp. 1–6.