ISSN 2278-3091
Volume 9, No.4, July – August 2020
International Journal of Advanced Trends in Computer Science and Engineering
Available Online at http://www.warse.org/IJATCSE/static/pdf/file/ijatcse220942020.pdf
https://doi.org/10.30534/ijatcse/2020/220942020

High Performance Computing using Containers in Cloud

Manish Kumar Abhishek[1], D. Rajeswara Rao[2]
[1]Department of CSE, Koneru Lakshmaiah Education Foundation, Vaddeswaram, India, manish.abhishek@gov.in
[2]Department of CSE, Koneru Lakshmaiah Education Foundation, Vaddeswaram, India, rajeshpitam@gmail.com

ABSTRACT

Virtualization is playing as a core component in cloud computing but performance overhead impact in its one of layer is forcing the researchers to think its usage in research areas with cloud for high performance computing. Where containers usage based on operating system virtualization widespread now a days is handling the performance impact via facilitating one of the lightweight layers of virtualization in terms of computing resources such as network bandwidth, throughput, CPU, memory at a large extent. Docker is one of the popular containerized platforms to be used in cloud computing and high-performance computing research areas. In this paper, we are proposing a basic strategy to avoid the performance impact and feasibility using Docker engine along with its evaluation in terms of containers vs virtual machine. Later, we have described the containerized placement now a days and performance specific problems resolution using container images for the development and deployment of HPC applications. The results and related work are focusing on a scalable approach in terms of availability and portability.

Key words: Cloud Computing, Docker, High Performance Computing (HPC), Virtualization, Virtual Machine (VM)

1. INTRODUCTION

The core key component of cloud computing is virtualization. OS virtualization in terms of being light weight in nature signifies containers are best fit in cloud for HPC application deployment within clusters. The demanding feature of development, deployment, upgradation and migration in HPC under Cloud can be eligible as one of the suitable fit to containerization. Clusters are going to be very complex mainly aimed for HPC but their strong and powerful computing is making its higher demand in cloud over several years in research and scientific communities. HPC clusters require multiple CPU cores with higher network bandwidth, low latency and huge computing resources with a well-defined infrastructure. Their effective and efficient use with multiple CPU cores, GPU can extend the frontier of their volume and features having capability to run scalable applications. There will be a need to add the resiliency design consideration in parallel to this. In today's IT world, HPC is considered as a solution to solve the complex research datasets processing and leveraged as a solution to achieve parallelism and performance. Where data is growing day by day, HPC is also getting considered in Cloud Computing. HPC applications are considered in form of multiple parallel jobs execution with defined set of allocated computing resources with different tools and techniques like Message Passing Interface. Cloud computing offers multiple services in terms of infrastructure, platform, and software with multiple models. Virtualization will be considered as a basic underlying layer of infrastructure where Linux OS is suitable for developing the interconnected systems cloud computing to form the HPC clusters in terms of containers with a well-defined set of computing resources in terms of GPU, memory, multiple core processor, latency, higher network bandwidth etc.

We have considered the various expects of performance specific to the execution of HPC applications using containerization with a measurable scale. Due to infrastructure independence, OS virtualization and isolation nature of containers, scalability and performance can be easily improved. In addition to parallel computing, virtualization introduced Cloud computing. To achieve virtualization, various hypervisors such as KVM, Xen can be easily used in Cloud computing but due to performance impact, their adoption has been avoided in HPC clusters. The lack of multi core processor optimization and with higher speed of fabrics, they have not been considered as a suitable option. On other side, Containers which are totally based on OS virtualization gained popularity in HPC community. Containers and Virtual machines both are based on virtualization. In Cloud computing, Infrastructure as a Service or platform as a Service is based on VM provision with the help of hypervisors. On the other hand, containers are also getting considered with a different underlying technique of virtualization which is resulting into a large bucket of features like light weight, scalable in nature and much more. Virtual machines are mainly based on the hardware layer abstraction where containers are going to have the abstraction over the operation system layer. In this paper, we will start first with the comparison of VM and containers via analyzing the platform and then in later sections using Docker engine, will deploy the HPC application over the cluster built on the top of own private cloud infrastructure

using OpenStack. To evaluate the application performance, containers deployment has been considered and with standard tools i.e. Graph500 [1] and LINPACK [2]. Results will represent the comparison of performance with Virtual machines where Docker is significantly providing higher performance.

2. MATERIAL AND METHODS

2.1 Virtualization in High Performance Computing

Analysis around high performance computing usually demands multiple core performance, GPU and high memory to resolve huge data set issues. For the execution of application in HPC cluster, OS is going to differ from host to host and workstation workloads. Via leveraging Virtualization, we can easily have several OS on an individual physical server with the help of knowingly techniques such as hypervisors or the usage of containerization that is totally based on underlying layer of operating system virtualization [3]. It provides encapsulation, scalability, partitioning and isolation which is mainly needed to process the large scale of dataset size application problem.

A. Virtual machine

It is a well-known abstraction of underlying layer of hardware based on totally isolation. With the help of hypervisors, multiple virtual machines can be easily provisioned to form an HPC cluster on the top of individual or group of physical servers with limited performance but with higher security [4]. Each individual virtual machine is considered to have its own execution environment to execute the multiple parallel processes in terms of operating system with well-defined computing resources [5]. Figure 1 shows the whole abstraction with multiple underlying layers with the application.

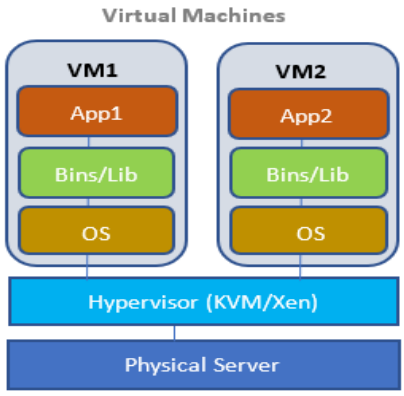

Figure 1: Virtual Machine underlying layers

B. Container

It is also a well-known abstraction of underlying layer of operating system with well-defined namespaces at the level of process isolation [6]. Using Docker platform, we can easily manage the multiple containers spawning which are going to be very light weight in nature and can be instantly provisioned using defined images with native performance. Each container will represent a unique process having its own individual process id (PID), There is no need of separate hypervisors. Complete lifecycle of containers are going to be managed by Docker engine. Even same image can be leveraged for the containers in terms of reusability. Docker is going to be mainly holding the responsibility of developing, porting, integration and execution of application like distributed micro services [7] environment to overcome the challenges of monolith application. Figure 2 shows the diagrammatic representation of underlying layers of a container. All dependencies required to run an application are going to be bundled inside the container only with the help of defined Docker image.

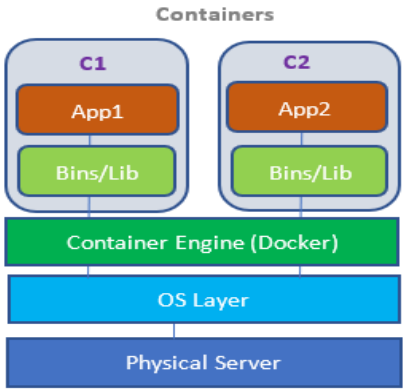

Figure 2: Container underlying layers

C. Docker

It is a well-known emerging technology and trending now days for containers handling to execute multiple distributed applications. It is an open source platform [8] to support the containerization and widely used in one of the Cloud models i.e. Infrastructure as a Service (IaaS). Here, we do not really require the usage of hypervisors. It is going to be very light weight mainly consisting three components i.e. Docker Registry, Containers and Images. Containers can easily share the same operating system kernel to get provisioned instantly with the effective usage of defined resources in terms of processor and memory. Basically, it is using the union file system to have the distributed Docker images for containers with underlying benefit of copy-on-write feature which makes it fit to gain the scalability under cloud infrastructure. Docker file is mainly used to execute the Docker images in form of containers which will get spawned as a container using Docker commands and can be easily managed such as start, stop, bash, upgrade etc. It is a real example of reusability in terms of containers and HPC application usage ease. All the required dependencies are getting bundled inside the container to execute the application and end user can manually or automated defined images to run the process push the same in Docker registry and execute it.

3. PROPOSED MODEL FOR EXECUTING HPC APPLICATION USING DOCKER

As described in above section that virtual machine is having the dependency of complete file system and operating system dependency in individual environment, it will surely increase the performance overhead in terms of having OS and libs during the phase of provisioning of multiple number of virtual machines to execute the HPC application. Docker is based on containerization which is holding the OS level virtualization via sharing the same OS and dependent needed libraries. In addition to that, they will also share the same file as Docker images for containers are built from file systems [9] layered architecture. Each container will be treated as a unique process having PID. Application which will be deployed using the containers sitting on the top of a single physical server will share the same libs/bin or any dependency of it. It will help to overcome the computing resources availability issue and scalability problems that we can face with VMs. Table 1 shows the comparison between VMs and containers computing resources for the spawned computing instances to have HPC Cluster.

Table 1: Comparison between VMs and Containers resources

Computing Instances	Virtual Machines (vCPU, running process, vRAM)	Container (process count)
8	8,8,16	8
16	4,4,8	4
32	2,2,4	2

HPC cluster environment has been built up with a group of six containers with well-defined computing resources to handle even huge datasets from application perspective. For application, we defined the Docker images to make them run as an individual process. Images have been pushed on private harbor registry which will be pulled via ansible scripts to deploy the containers. Spring boot application has been deployed. We have computed the performance results with respect to containers in comparison to bare metal and found them comparable [10]. Figure 3 shows the computed performance using NVIDIA tesla.

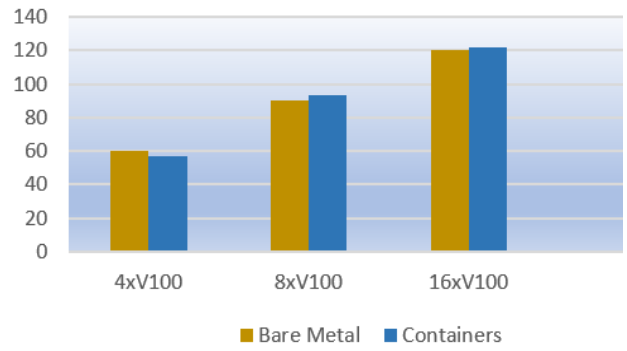

Figure 3: Performance analysis of Containers vs Bare metal

The consumption of computing resources is always a concern, but we can compute it and defined in override.yaml file for a better handling in terms of min and max memory, CPU, I/O threads. We have designed an algorithm using which next container will be determined that need to be spawned with required computing resources. Algorithm 1 shows the pseudocode for the computation of container's computing resources within HPC Cluster to run the application and to overcome the performance overhead. We are going to have a set of physical servers in our private infrastructure and on top of that we need another set of containers to form an HPC cluster. In our cloud environment containers has been spawned instead of VMs. We have computed the resulted container which is going to have maximum computing resources as a remaining quota of both running containers and used physical servers. The next container which is going to be spawned to achieve scalability with respective to running application will be the resulting one which has been considered as an output of our algorithm.

Algorithm 1. Pseudocode for the computation of container that needs to be spawned next with required computing resources.

Input: $C_{nm,}$ PS_m
Output: map C_s
1: $C_s = \varphi$
2: **for each** container ∈ PS_m **do**
3: **for each** m ∈ C_{nm} **do**
4: CS = φ
5: compute free CScpu, C_{nm}
6: compute free CSmem, $_{Cnm}$
7: compute isGPU, t, f
8: compute Affinity, t, f
9: $CS_{total} = C_{nm} + PS_m$
10: **map** CS_{total} with container id
11: **end for**
12: **end for**
13: **return** map C_s**;**

- **C_{nm}**: Set of running containers on top of physical server m having count n
- **PS_m**: Set of physical servers having count m
- **C_s**: Computed Container that is going to be spawned next with required computing resources.
- **CS**: Computing resources

4. EVALUATION

Using our built environment, we have tested the application deployment efficiency followed by reduction of performance overhead in terms of restricting the number of unused micro services having the application with respect to multiple inter dependencies. The key standard chosen for VMs is native performance overhead via benchmarking the different test cases around containers and VMs configurations. The computing resources in terms of CPU, GPU, affinity and memory allocation for VMs and containers needs to be saturated without testing environment.

Our built environment consist a set of16 containers running on the top of HPC cluster with a physical server. Computing resources includes GPU enabled, 18 physical cores and 132 GB RAM with Intel core processor. Gigabit Ethernet has been used for network. VMs have been provisioned via hypervisor KVM/QEMU 4.2.0. Docker version 19 has been used to spawn the containers with Ubuntu as an Operating System. From testing perspective, we have defined our testbeds. At first, we have targeted the medium level of dataset problem to get the best case for containers as well as VMs performance and later we did the benchmarking via increasing the instances count with varying computing resources which actually shows the performance variance results. For HPC application benchmarking, HPL benchmark has been used which is the implementation of LINPACK for the random dense linear system equations [11]. HPL internally uses the double-precision floating point arithmetic and MPI for portable routines. So, we have used HPL benchmark along with its math lib OpenBLAS [12] and OpenMPI [13] for our testing environment mainly targeted for HPC cluster mounted in OS and installed on physical server host. It provides the elasticity of multiple containers deployment even on single compute instance in private cloud infrastructure [14]. Usually metric size raises the issue of HPL that is directly proportional to the performance computation impact. Here, we came with a list for both containers as well as VMs matrix sizes which raises from 65% to 80% of memory in terms of RAM. Figure 4 shows the container efficiency during the phase of application execution within HPC cluster using our built deployment model. Figure 5 shows the memory usage comparison of VMs vs Containers.

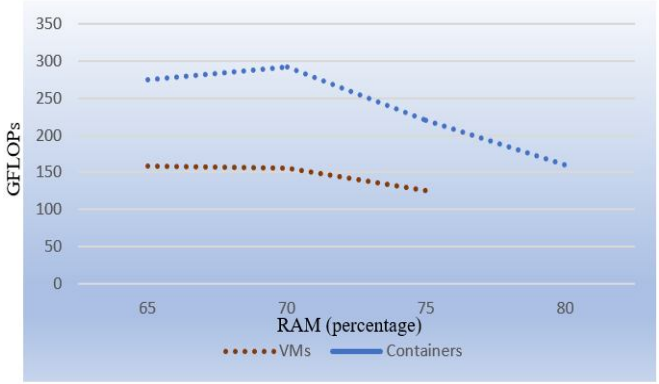

Figure 4: Benchmarking using HPCL for VMs vs. Containers

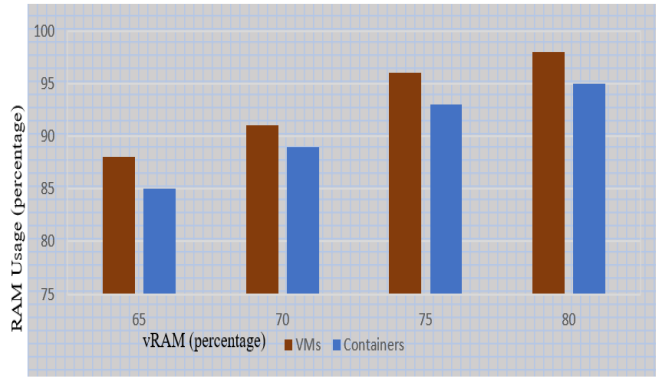

Figure 5: Memory usage comparison for VMs vs. Containers

Using the results, the best case for native performance can be considered with an approximation of 290 GFLOPs which clearly makes containers best fit over VMs. Containers can be considered with a matrix size greater than 70% but less than 75% where VMs can be considered below 70% to 60%. The evaluation of HPC results above 80% was difficult to achieve. On other side memory consumption in terms of RAM get higher for virtual machines in comparison to containers which states that performance is indirectly proportional to matrix size. It means if memory consumption increases, performance impact can be seen where on other side in case of containers it is totally in reverse manner. Figure 6 shows the same performance results in terms of configuration i.e. containers are preferable over VMs. To conclude for HPC cluster, we have reviewed the workload distribution across containers and observed an issue that with the increase of workload distribution along with consideration of multiple process execution within a container. Figure 7 shows that during HPL run if it reaches to maximum requirement of defined computing resources then ability of its computation are getting impacted.

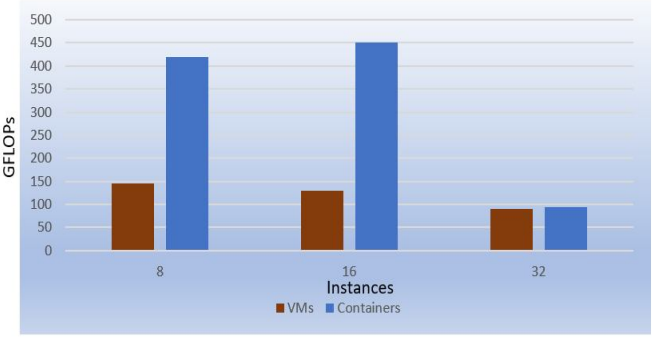

Figure 6: HPCL Benchmarking for Computing Instances

Figure 7: Graph500 based HPCL benchmarking with count increment of VMs vs. containers computing Instances.

We got better result with reduced computation but found that point to be noted and taken care in terms of CPU cycles as a part of Docker resource management. Containers can be stop-start instantly and easily using Docker commands but as count increases, data traceability in form of resource management need to be worried. Here, we are evaluating our proposed model for dynamic allocation resources, usage of free computing resources for non-HPC application, performance, profiling, queue Using Grpah500 based HPCL

benchmarking, we found that as count goes beyond 25 and reaches to 32 or later, due to OS kernel sharing we face the issue in resource control. There is a need to address this issue which we can take as a future work in our testing environment.

5. CONCLUSION

This paper presents the research work to reduce the performance overhead and containers consideration to run an HPC application in cloud environment. Docker is used to spawn the containers on the top of private cloud infrastructure built using OpenStack. Comparison between VMs and containers has been evaluated to get the better performance results. An effective and efficient report has been generated around performance results which significantly showing the containers adaption to run HPC application. Research work was aimed for containers. The adaption of virtualization in terms of Containers via Docker can be leveraged in cloud computing to run HPC applications.

ACKNOWLEDGEMENT

A special vote of thanks to the Koneru Lakshmaiah Education Foundation for helping and facilitating me required infrastructure and my guide as well as staff members who have helped me to complete this research work.

REFERENCES

1. K. Ueno and T. Suzumura, "**Highly scalable graph search for the graph500 benchmark**," in Proceedings of the 21st international symposium on High-Performance Parallel and Distributed Computing. ACM, 2012, pp. 149–160.
2. J. J. Dongarra, P. Luszczek, and A. Petitet, "**The LINPACK benchmark: past, present and future**," Concurrency and Computation: practice and experience, vol. 15, no. 9, pp. 803–820, 2003.
3. R. Morabito, J. Kjallman, and M. Komu, "**Hypervisors vs. lightweight virtualization: a performance comparison**," in Cloud Engineering (IC2E), 2015 IEEE International Conference on. IEEE, 2015, pp. 386–393.
4. P. Patel, V. Tiwari and M. K. Abhishek, "**SDN and NFV integration in openstack cloud to improve network services and security**," 2016 International Conference on Advanced Communication Control and Computing Technologies (ICACCCT), Ramanathapuram, 2016, pp. 655-660, doi: 10.1109/ICACCCT.2016.7831721.
5. G, Nida. (2020). "**A Virtual Machine Introspection in Cloud Computing for Intrusion Detection**." International Journal of Advanced Trends in Computer Science and Engineering. 9. 2662-2666. 10.30534/ijatcse/2020/26932020.
6. S. Soltesz, H. Potzl, M. E. Fiuczynski, A. Bavier, and L. Peter- ¨ son, "**Container-based operating system virtualization: a scalable, high performance alternative to hypervisors**," in ACM SIGOPS Operating Systems Review, vol. 41, no. 3. ACM, 2007, pp. 275–287.
7. Sarita and S. Sebastian, "**Transform Monolith into Microservices using Docker**," 2017 International Conference on Computing, Communication, Control and Automation (ICCUBEA), Pune, 2017, pp. 1-5, doi: 10.1109/ICCUBEA.2017.8463820.
8. (2014) **Docker homepage**. [Online]. Available: https://www.docker.com/
9. C. P. Wright and E. Zadok, "**Kernel korner: unionfs: bringing filesystems together**," Linux Journal, vol. 2004, no. 128, p. 8, 2004.
10. A. M. Joy, "**Performance comparison between linux containers and virtual machines**," in Computer Engineering and Applications (ICACEA), 2015 International Conference on Advances in. IEEE, 2015, pp. 342– 346.
11. A. Petitet, R. C. Whaley, J. Dongarra, and A. Cleary. (2012) **Hpl - a portable implementation of the high-performance linpack benchmark for distributed-memory computers**. [Online]. Available: http://www.netlib.org/benchmark/hpl/
12. Z. Xianyi, W. Qian, and Z. Chothia, "**OpenBLAS**," URL: http://xianyi. github. io/OpenBLAS, 2014.
13. M. Open, "**Open MPI : Open source high performance computing**," 2012.
14. Karimunnisa, Syed & Kompalli, Vijaya. (2019). "**Cloud Computing: Review on Recent Research Progress and Issues"**. International Journal of Advanced Trends in Computer Science and Engineering. 8. 216-223. 10.30534/ijatcse/2019/18822019.
15. J. Leskovec, D. Chakrabarti, J. Kleinberg, and C. Faloutsos, "**Realistic, mathematically tractable graph generation and evolution, using kronecker multiplication**," in Knowledge Discovery in Databases: PKDD 2005. Springer, 2005, pp. 133–145.
16. T. Adufu, J. Choi, and Y. Kim, "**Is container-based technology a winner for high performance scientific applications?**" in Network Operations and Management Symposium (APNOMS), 2015 17th Asia-Pacific. IEEE, 2015, pp. 507–510.
17. J. Turnbull, **The Docker Book: Containerization is the new virtualization**. James Turnbull, 2014.
18. D. Bader, J. Berry, S. Kahan, R. Murphy, J. Riedy, and J. Willcock, "**The graph 500 list: Graph 500 reference implementations**," Graph500. http://www. graph500. org/reference. html (2 Febuary 2012), 2010.
19. D. Ziakas, A. Baum, R. A. Maddox, and R. J. Safranek, "**Intel R quickpath interconnect architectural features supporting scalable system architectures**," in High Performance Interconnects (HOTI), 2010 IEEE 18th Annual Symposium on. IEEE, 2010, pp. 1–6.
20. Chung, Minh & Nguyen, Hung & Nguyen, Manh-Thin & Thoai, Nam. (2016). **Using Docker in High Performance Computing Applications**. 10.1109/CCE.2016.7562612S. Chen, B. Mulgrew, and P. M. Grant. High Performance Computing (HPC) on AWS. [http://aws.amazon.com/hpc-applications.]