Martin *et al. Hum. Cent. Comput. Inf. Sci.* (2018) 8:1
https://doi.org/10.1186/s13673-017-0124-3

Human-centric Computing and Information Sciences

RESEARCH Open Access

Exploring the support for high performance applications in the container runtime environment

John Paul Martin[1*], A. Kandasamy[1] and K. Chandrasekaran[2]

*Correspondence:
johnpm12@gmail.com
[1] Department of Mathematical and Computational Sciences, National Institute of Technology Karnataka, Surathkal, Karnataka, India
Full list of author information is available at the end of the article

Abstract

Cloud computing is the driving power behind the current technological era. Virtualization is rightly referred to as the backbone of cloud computing. Impacts of virtualization employed in high performance computing (HPC) has been much reviewed by researchers. The overhead in the virtualization layer was one of the reasons which hindered its application in the HPC environment. Recent developments in virtualization, especially the OS container based virtualization provides a solution that employs a lightweight virtualization layer and promises lesser overhead. Containers are advantageous over virtual machines in terms of performance overhead which is a major concern in the case of both data intensive applications and compute intensive applications. Currently, several industries have adopted container technologies such as Docker. While Docker is widely used, it has certain pitfalls such as security issues. The recently introduced CoreOS Rkt container technology overcomes these shortcomings of Docker. There has not been much research on how the Rkt environment is suited for high performance applications. The differences in the stack of the Rkt containers suggest better support for high performance applications. High performance applications consist of CPU-intensive and data-intensive applications. The High Performance Linpack Library and the Graph500 are the commonly used computation intensive and data-intensive benchmark applications respectively. In this work, we explore the feasibility of this inter-operable Rkt container in high performance applications by running the HPL and Graph500 applications and compare its performance with the commonly used container technologies such as LXC and Docker containers.

Keywords: Cloud computing, Containers, High performance computing, Core OS Rkt, Docker, LXC, Linpack, Graph 500

Introduction

Cloud computing is being used for innumerable applications these days. The end-users vary from naive clients to expertised technicians. Cloud is a pool of resources shared among number of users [1]. Presently, in the world of cloud computing, it is the era of XaaS (Anything-as-a-Service) which means that the providers offer a wide variety of services [2, 3]. One of the most recent services provided through the cloud is high performance computing (HPC) environments for the complex applications. Virtualization is the technology which enables users to share a single entity among a group of users.

Based on the position of the virtualization layer, virtualization can be of different types like full virtualization, paravirtualization and OS level virtualization.

Traditional HPC clusters are composed of many separate dedicated servers called nodes and may be shared among different organizations. The requirements of each user or organization will be different, which demands the creation of customized environments without affecting others. This is not an easy task in traditional HPC systems. As a solution for this, virtualization was adopted for HPC. Virtualization materializes the task by creating separate customized virtual environments of the system based on the requirements of each user.

The overhead associated with virtualization hindered its usage in HPC environments. There exists different kinds of virtualization techniques [4]. One of the popular techniques involves a Hypervisor (Virtual Machine Manager). Here virtualization services are mainly provided through virtual machines (VM), but this creates an additional overhead due to the running of a fully installed OS. The guest OS in VMs creates calls to the hypervisor rather than direct communication with the hardware, which causes some reduction in application performance. This overhead and limitation results in insufficient adaptability to the HPC environment. Virtualization technique based on the OS level offers a model called Container Virtualization as a solution to all these overheads, which gives near native performance [5]. Container virtualization allows to deploy and run applications without creating separate VMs for each user. Multiple isolated containers are run on a single host with sharing a single kernel. The Linux features such as namespaces, chroot and cgroups provides secure execution of containers in the same kernel. When compared to traditional virtualization, since containers do not use separate OS instances, it requires less CPU, memory and storage, thus the same host can incorporate more number of virtualized containers. The time required to create and deploy containers is very less compare to the virtual machine manager based systems.

The Hypervisor based virtualization employs a full guest OS in each virtual machine along with the necessary binaries and libraries for the applications. Containers will hold only the necessary binaries required by any application to run [6]. Containers possess packaged, ready to deploy applications or parts of applications, and if necessary middleware and business logic to run those applications [7]. Figures 1 and 2 shows the two different virtualization architectures [8].

There exists different container management technologies for managing the entire life cycle of containers. This includes creating, deleting and performing modifications on images and tools associated with it. Linux LXC, Docker and the latest release Rkt are the major managing technologies. All the technologies work on the same base principle that the application space should be isolated within the operating system.

The Rkt container runtime was recently introduced to overcome the limitations of the existing container runtimes. Being a more recent technology, the Rkt container is expected to provide better support for HPC applications. Most of the current generation applications demand high performance capabilities. It is essentially required to enhance the support available for such applications. Researchers have been on the quest for technologies that will enable them to achieve performance which most resembles the bare metal scenarios. In alignment with this research interest, we have identified the following objectives:

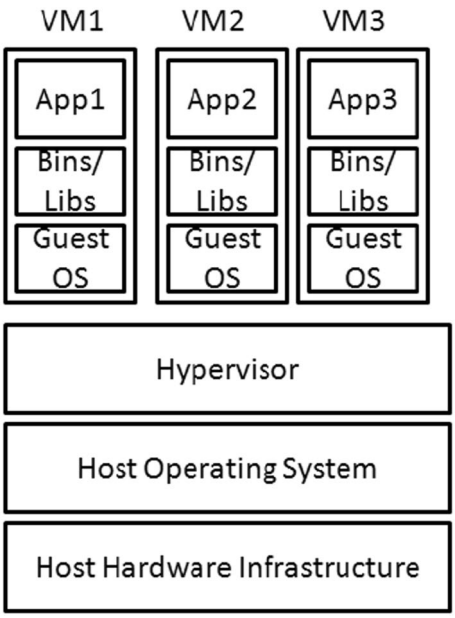

Fig. 1 Hypervisor-based virtualization

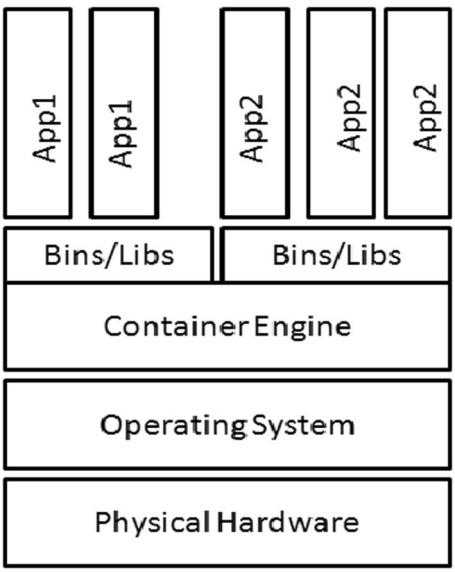

Fig. 2 Container virtualization

- Investigate the existing works which aim at comparing and contrasting the various container technologies in a performance-oriented perspective.
- Explore the features which differentiate the Rkt container runtime from the other runtimes.
- Implement archetypes to assess the variation in support provided by the Rkt container runtime for applications with high performance requirements.

- Analyze how the different features impact the performance of computationally challenging tasks.
- Analyze the impact of the features specific to the Rkt container runtime on the data intensive tasks.

Our primary aim is thus to explore the feasibility of the recent Rkt container [9] for HPC environments and compare its performance with LXC and Docker containers. We are mainly analyzing the performance results of computing intensive and data intensive practical applications in all the scenarios. To the best of the authors' knowledge, this is the first work attempting to analyze the support provided by the Rkt container runtime and contrast it with its predecessor, Docker. The results of the work presented in this paper will be of equal interest to researchers attempting to enhance the features of Rkt and the group of researchers looking for the adoption of Rkt containers in high performance environments.

The rest of this paper is structured as follows: "Background and related work" section contains the history of containers and similar works in this area. Detailed description and specification of experiment setup and various benchmarking tools is elaborated in "Experimental setup and benchmarking tools" section. "Results and discussions" section contains the obtained results and analysis done based on it. Finally, the conclusion and future scope of research is described in "Conclusion" section.

Background and related work

High performance computing is an activity which requires more than a normal computer's ability to execute it. Such activities are generally executed using parallel programming efficiently. High performance computing centers are now beginning to use container based cloud environments for solving their complex problems [10]. Adopting containers in HPC is not an easy task-it involves a lot of challenges [11].

Container technology has been there for more than a decade and it allows the operating system to be virtualized and share the same instance of the OS. Containerization was initiated by UNIX operating system in 1979 with their system called chroot. Then, in 2000 Free BSD jail container technology evolved, which was similar to the chroot, but incorporated features for isolating file system, users and networking. Linux VServer was another jail mechanism and an initial implementation of virtual private servers. OpenVZ containers emerged in 2005 which uses patched linux kernel. Each of the OpenVZ container possess isolated file systems. The first complete implementation of linux container manager is LXC and it evolved in 2007 [12]. Later, people begin to think of containers as processes with extra isolation, and thus helps in reducing the overhead associated with virtual machines. Heroku PaaS provider initiated this concept of containers to deploy applications. In 2013, Docker came up with an entire ecosystem for managing containers. Rocket containers came into existence for solving the drawbacks of Docker and to provide more stringent security measures. In 2016, Microsoft Windows introduced a container technology called Windows Containers for supporting windows server systems. The standardised code in the containers can be easily plugged-in and run in the operating systems. This eases the portability of the applications.

The technologies available in the linux such as namespace, cgroup are used by the container management systems for managing the various operations in containers. Major existing technologies for container management are Linux LXC, Docker and Rkt. The common objectives of these technologies are achieved in a different manner.

- *LXC* LXC is an interface for the linux kernel containment features and allows the users to run multiple isolated images on a single host. The isolation among LXC containers are provided through kernel namespaces. LXC containers uses PID namespace, IPC namespace and file system namespace for virtualizing and isolating PIDs, IPCs and mount points respectively. Network namespace is used to connect the virtual interface in a namespace to the physical interface and supports route based and bridge based configurations. Resource management is done through cgroup. Some other key responsibilities of cgroup are process control, limiting the usage of CPU and isolating containers and processes. LXC has the unprivileged option to create User space containers. It may seem advantageous in some aspects but in other aspects it may create some security issues. Container portability allows the same image to run in different distributions and hardware configurations without much changes. In this regard, LXC provides only partial portability because it will work across only Ubuntu distributions [13]. LXC allows multiple applications in a container.
- *Docker* Docker is one of the leading container life cycle management tools. Docker allows to run and manage applications side-by-side in isolated containers. Similar to LXC, Docker also make use of the features of the Linux kernel such as cgroups and kernel namespaces [14].

    - namespace: Docker uses namespace for creating isolation among containers. Following are the types of namespace used by Docker.
        pid: pid namespace ensures that process in one container does not affect process in a different container.
        uts: used for kernel version isolation.
    - mnt namespace: provides view of own file system and mount point.
        ipc namespace: provides isolation for interprocess communication.
        net : network isolation is provided through this namespace.
    - Union file system: union file system allows to stack different layers and present it as a single file system. The writeable layer exists only at the top.
    - Control group: resource management or efficient sharing of hardware among containers is allowed.

    The lightweight nature of the Docker container allows several containers to run in a single server or virtual machine simultaneously [15]. The major limitation of Docker is its security issues, that is, an attacker can easily get superuser privileges. Lack of interoperability is the another limitation of the Docker technology [16].

- *Rkt* Core OS Rkt is a more secure, interoperable, and open source alternative to Docker. It allows to run multiple isolated images sharing a common kernel space. Rkt

provides more security compare to the Docker Containers in various aspects. For example, while downloading an image docker does not ensure any kind of security but rkt does a cross checking of the signature of the publisher of the image [9]. Rkt has different stages.

- Stage 0
  Interacts with the user.
  Fetches the image and verifies it.
  Handles image store operations.
  Image rendering.
- Stage1
  Pod isolation from others.
  Relevant networking established.
  Initialise file systems.
- Stage 2
  Execution of the user application.

Coreos, host, fly and kvm are the different modes of execution supported by Rkt in Stage0. Coreos and host uses Linux namespaces for isolation such as pid, network and so on. Fly mode is the lighter security mode-this does not have any isolation for network, CPU and memory. SELinux is also not enabled in this mode. KVM mode is the most secure mode, in which the Rkt container will behave like a lightweight virtual machine itself. Sharing of kernels is not permitted in this mode. Several researchers explored the area of virtualization from the past decades onwards, and majority of the works were about virtual machines. Morabito et al. [17] made a comparison between hypervisor based virtualization and lightweight virtualization. They considered KVM as an example of hypervisor-based virtualization and Docker, LXC as the representatives of Lightweight virtualization. Xavier et al. [18] made a similar comparison between virtual machine and different containers. They considered Xen as the VM technology and LXC, Vserver and OpenVZ are the containers. They found that among those containers LXC is the suitable one for HPC environment. Chung et al. [19] made a detailed evaluation about the suitability of Docker in HPC environments and found that Docker is more suitable for data intensive applications. Kozhirbayev et al. [20] made a comparison among the container technologies to find out which performs better in Cloud environments. They made a comparison between Docker and Flockport with reference to the native performance and they found that only I/O intensive operations suffer the impact of a higher overhead for containers. Docker and Flockport doesnt suffer the overheads in terms of memory and processor. They claim that containers reduces the difference between the Infrastructure as service and baremetal systems by providing near native performance. Varma et al. [21] performed an analysis of network overheads when Docker containers are used in Big Data environments. The Hadoop benchmarks were executed in different experimental setups, by varying the number of containers against the number of virtual machines. The networking and latency aspects were considered. The throughput of the network was observed to be inversely proportional to the number of nodes in a virtual machine. The Docker containers were found to offer fair sup-

port for big data applications. Chung et al. [22] evaluated the performance of virtual machines and Docker HPC environments where the infrastructure is connected by Infiniband and found that the overall performance of containers is better than the virtual machines. C. de Alfonso et al. explored the practical feasibility of running scientific workloads with high throughput requirements on containers [16]. Clusters of machines running applications in containers were used to provide the requirements of the scientific applications. A middleware receives the user requests and spawns the appropriate number of virtual nodes. The CLUES manager is employed to provide the required elasticity. Distributed computing paradigms such as the Cloud, is the basis of the IT era. Docker containers offer an efficient option to run applications in the Cloud [23]. The Docker containers may be executed on single host or on multiple hosts. When large number of Docker containers are run on multiple hosts, the management of the system becomes tedious, which can be tackled by developing infrastructure solutions enabling the administrators to automate the tasks of management of the system. Several open-source software solutions have been developed for the Docker ecosystem. There is relatively less work focused in the Rkt container in HPC environment, so, we decided to explore the feasibility and performance impacts of this most secure container in HPC environment and to check whether Rkt can meet the challenges of an HPC environment.

Experimental setup and benchmarking tools

Our experiments were performed on an HP EliteDesk 800G1 Tower system with two Intel 4th Generation Core i7 Processors for a total of 16 cores. We used Ubuntu 16.04.2 LTS (Xenial Xerus) 64 bit with Linux kernel 4.4.0-62-generic, Docker container 1.3.0, LXC 2.0.7, and Rkt container 1.24.0. For consistency, we used the same Ubuntu base image on these different platforms.

CPU throttling was disabled in all the experiments because of the usage of BLAS (Basic Linear Algebra Subprogram) specification in the ATLAS library. The ATLAS library requires disabling of CPU Throttling for its proper execution.

High performance computing applications can be either Computation Intensive or Data Intensive. We evaluated the performance of Rkt container in both scenarios. HPL benchmarking tool (computing intensive application) and Graph500 (data intensive applications) are used for analysing the suitability of respective container platform and for measuring the performance in high performance computing applications. For analysis, the Linpack 2.1 CPU-intensive benchmark and Graph 500 2.1.4 Data-intensive benchmarks were executed and results were collected.

Computing intensive applications

Linpack is the benchmarking tool which we used for analysing computing intensive applications. It has two implementations. One is optimized only for Intel machines and the other one is supported by all machines. HPL is the portable implementation of Linpack, is written in C and requires an MPI implementation, BLAS interface implementation ATLAS library. HPL tests the performance of a system by generating and solving a system of equations using LU decompositions [24].

HPL generates and solves dense system of linear equations with LU factorization and partial pivoting. It mainly involves multiplication of a scalar value with vector and adding the results in to a vector, all these operations are carried out with values with double precision floating point. HPL measures rate of floating point execution for solving these linear equations and gives us the performance results. Mathlibrary, HPL package and Message Passing Interface are required for running this in a distributed environment. The two major steps involved in the problem are:

- Lower upper factorization of a random matrix.
- Lower upper factorization used to solve the linear system consisting of the random matrix and a scalar.

The performance results from HPL are measured in Gflops (Billions of floating point operations per second):

- $Gflops = ops/(cpu * 100000000)$

Following are the test cases considered while analysing the performance of various container platform.

- Varying the problem size (N).
- Varying block size (NB).
- Varying the rfact and panel fact.

The input given for the various tests can be summarized as shown in Table 1.

With these input parameter configurations around 120 tests were performed in all the different container runtimes and the averaged experiment results on several runs were used for the purpose of analyzing the performance.

Data intensive applications

We measured the performance impacts on data intensive applications in the container environment using Graph500 benchmarking tool [25]. Graphs are a core part of analytic

Table 1 Values of the input parameters

Parameter	# of values	Values
Problems size (N)	5	10000 11000 12000 13000 15000
Block size (NBs)	4	100 92 104 98
Process grids (P × Q)	1	P = 4, Q = 8
Threshold	1	16
Panel fact	3	Right left crout
Recursive stopping criterium (NBMin)	1	4
Recursive panel fact (RFACTs)	2	Right left
Panels in recursion (NDIVs)	1	2
Swapping threshold	1	64

workloads. Graph500 is a data intensive super computing application which uses a large scale graph to evaluate the performance. The execution of Graph500 involves two steps:

- Data generation
  - generates edge list.
  - construction of graph from the generated edge list.

- BFS (Breadth First Search) on the constructed graph
  - randomly selects 64 unique search keys whose degree is greater than or equivalent to one.
  - parent array of each key is computed and check whether it is a valid BFS search tree.

There exists different problem classes based on the size of the problem. They are toy (17 GB) called as level 10, mini (140 GB)-level 11, small (1 TB)-level 12, medium (17 TB)-level 13, large (140 TB)-level 14, huge (1.1 PB)-level 15.

The input parameters to be given to Graph500 is SCALE and Edgefactor and this determines the size of the graph. Number of vertices of the graph is calculated based on the input parameter SCALE. Let N be the number of vertices in the graph.

$$N = 2^{SCALE}$$

Number of edges M in the graph can calculated as

$$M = N * Edge\,factor$$

Results and discussions

We explore the performance of CoreOS Rkt container in HPC environment and compare it with existing popular container runtimes such as Docker and Linux LXC. HPC environment mainly comprises of applications which are data intensive or computation intensive in nature. We considered both of these important scenarios for our experimental analysis. The tests were carried out in single node environment and later in a cluster environment. We have tuned the HPL benchmark and tests were carried out.

Exploring computing intensive applications

The behavior of a processor varies with the increase in the problem size (N) of the HPL benchmark, based on that the behavior can fall into one of three different zones:

- *Rising zone* The low problem size wont invoke memory and processor to their maximum performance.
- *Flat zone* Problem invokes only processor to its maximum performance.
- *Decaying zone* Problem size is too large and cache memory is not large enough to keep all the necessary data, because processor is running in top speed.

The performance results of HPL in the CoreOs Rkt environment is in given in Fig. 3a. The GFlops on the Y axis gives the rate of execution of floating point operations on a scale of billion. We have examined different problem sizes and varying block sizes. The results show that Rkt container performs well in computing intensive applications. We

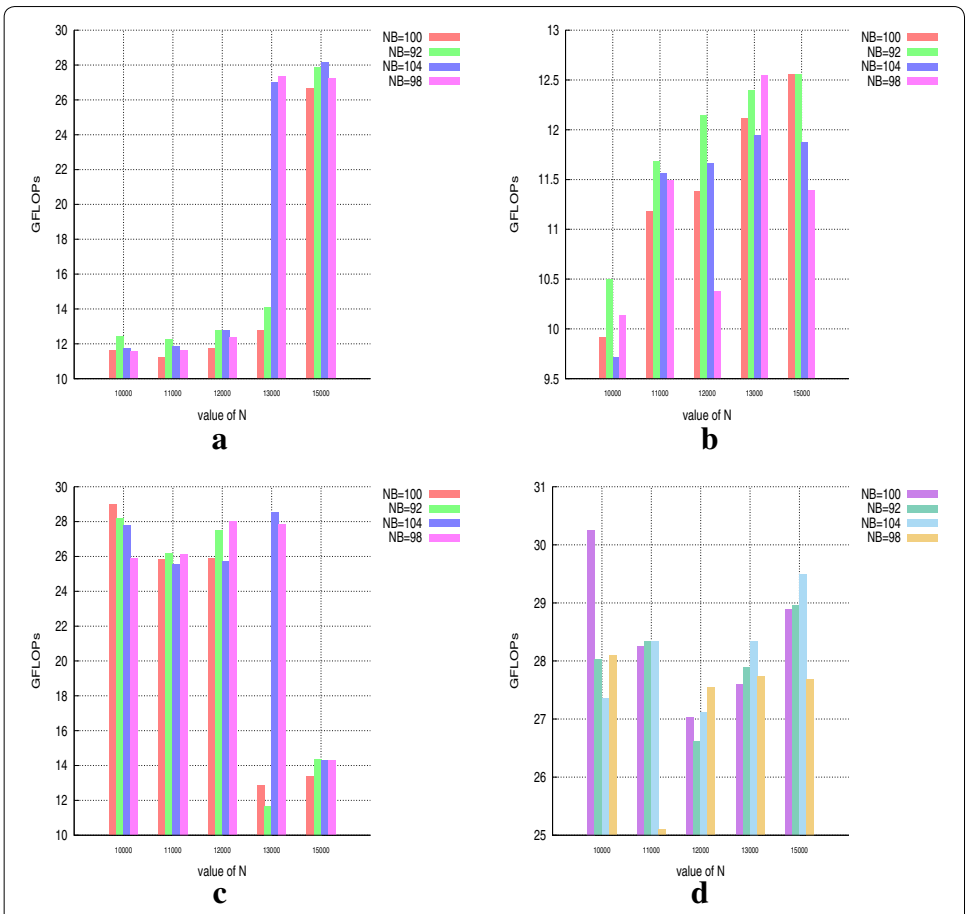

Fig. 3 HPL on containers **a** Rkt, **b** LXC container, **c** Docker container, **d** native

can see a significant increase in execution rate with the increase in problem size (N values). Rkt runs in its rising zone of HPL. Figure 3b shows the results of HPL in Linux LXC, Fig. 3c shows the results obtained when HPL is run on Docker containers and Fig. 3d is the baseline performance obtained when running HPL on the native system. The rate of execution in billions is increasing almost linearly with increase in problem size. Comparing with Rkt for larger problem sizes it does not perform well. The results of Docker in this environment gives better performance for smaller problem sizes and there is a drastic degradation in the performance with an increase in the problem size. When we compare these results with the results of a native system Rkt container gives near native performance for larger problem sizes whereas for smaller problem sizes Docker outperforms Rkt and gives near native results, as can be observed in Fig. 4.

The Rkt container runtime runs for a longer time in the flat zone implying that the load experienced by the containers are less challenging. Even when the Docker container subsides to the decaying zone, the Rkt container continues in the flat zone. This is supporting the intuition that Rkt container engines provide better HPC support. This can be attributed to the absence of a daemon process in the Rkt runtime stack. The Rkt containers are started based on command clients. This also resolves compatibility issues with init systems (which are existing in Docker Runtime). The absence of such a daemon

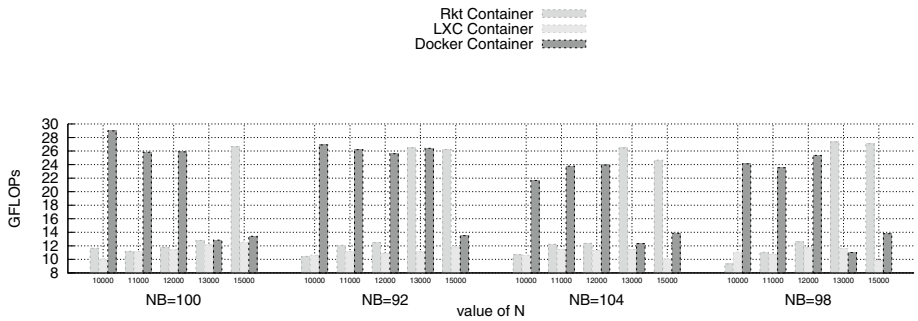

Fig. 4 Comparison among containers-single node setup

increases the startup time slightly, but greatly lowers the overhead experienced by the containers. This is the major reason why the Rkt containers fare better in the high performance scenarios.

Many of the high performance applications are executed in cluster environments, so for better understanding the performance in such an environment, cluster of containers was created and performance was measured. The CoreOS Rkt cluster performance is compared with other platforms. The results show that the clustered environment shows almost same characteristics as that of the single node environment. The obtained results are illustrated in Fig. 5a–c and a better understanding may be obtained from Fig. 6. The Rkt container gives better performance results for larger problem sizes. In the case of Docker containers, by default they use a time sharing algorithm such that each container gets the CPU only for 100 ms and after that the CPU switches to the next container. This creates an overhead as it is required to save the process state information before switching to the next process. The computing intensive applications which takes long time for execution will be affected by this overhead and will cause a degradation in its performance.

Exploring data intensive applications

Graph500 represents a data intensive application benchmark. This benchmark attempts to obtain the BFS of a large graph and the performance is evaluated. Data traceability is the criteria which is used for evaluation, it is the ability to keep track of data and accessing on a system. We examined the performance of Graph500 with varying Edge factors. We took edge factors varying from 16 to 36. Results are measured in TEPS which is the Traversed Edges Per Second. We explored the simple graph problem size in the Graph500 benchmark. The results given in Fig. 7a–d show that LXC, Docker and Rkt containers give near native performance because they are not emulating an entire system like the host and they are virtualized only at the level of operating system. Containers are sharing the same kernel of the host system where it is running and this enables them to access data in a very fast manner. Compared to virtual machines, containers fare better in data intensive applications and Rkt container gives similar performance results along with other containers such as Docker and LXC.

From the results shown in Fig. 8, it can be deduced that the Rkt container performs well in data intensive applications, but, the performance is lower when compared to the

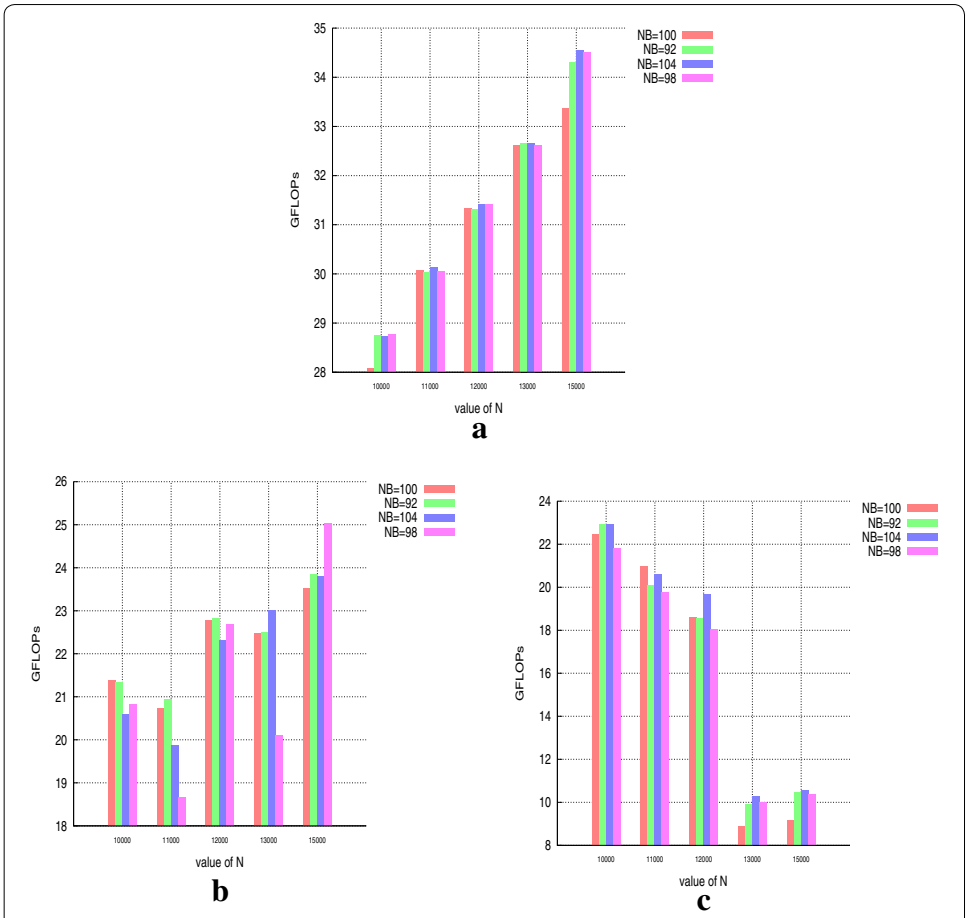

Fig. 5 Performance analysis on container clusters **a** Rkt, **b** LXC, **c** Docker

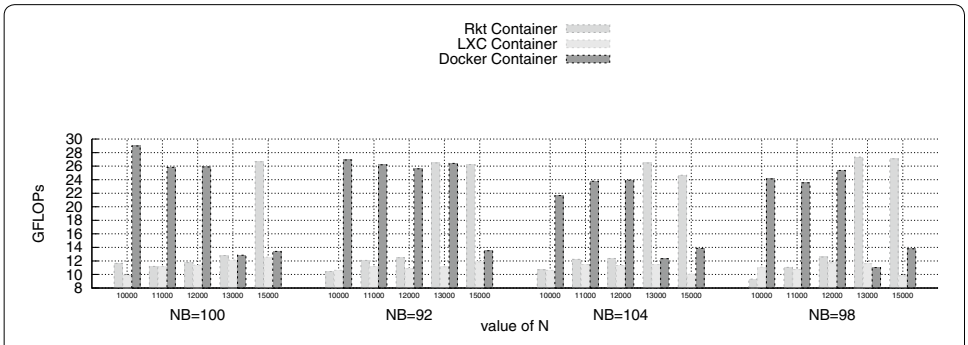

Fig. 6 Comparison among container clusters

other container environments. The LXC container has the performance closest to the native performance in data-intensive applications.

The results shows that Rkt container performs well in both computation intensive and data intensive high performance application environments. However, the Rkt containers are better suited for computation-intensive applications. The improved support for computation-intensive applications coupled with advanced security features connotes

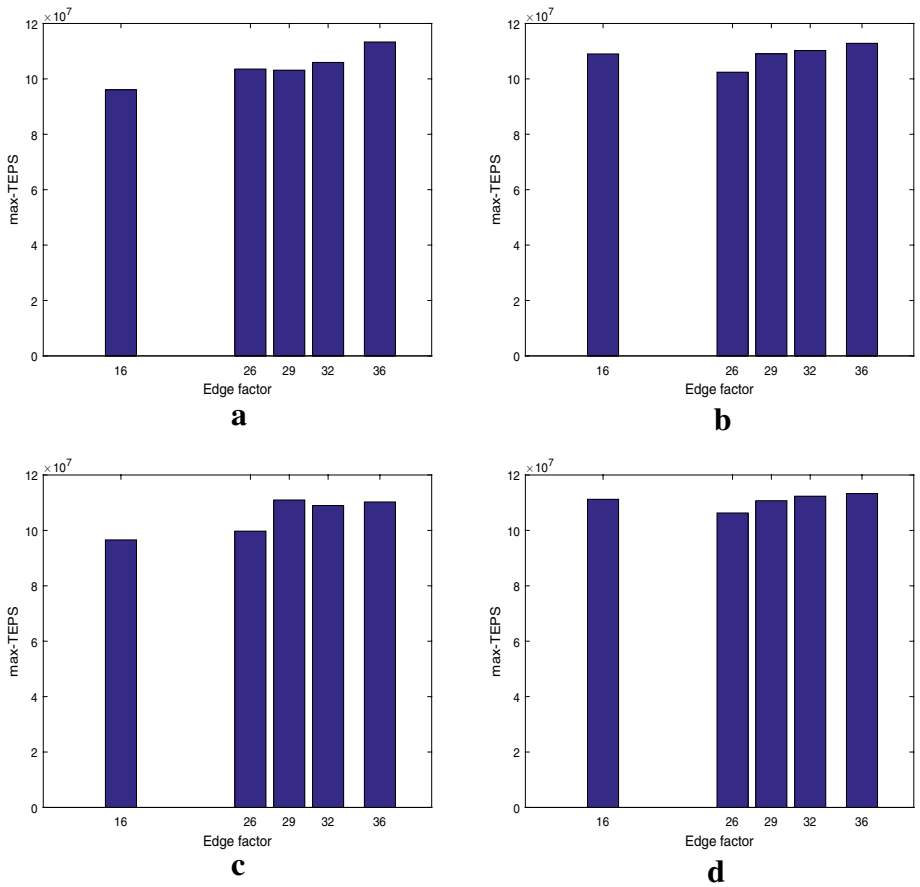

Fig. 7 Data intensive performance analysis on containers **a** Rkt, **b** LXC, **c** Docker, **d** Native

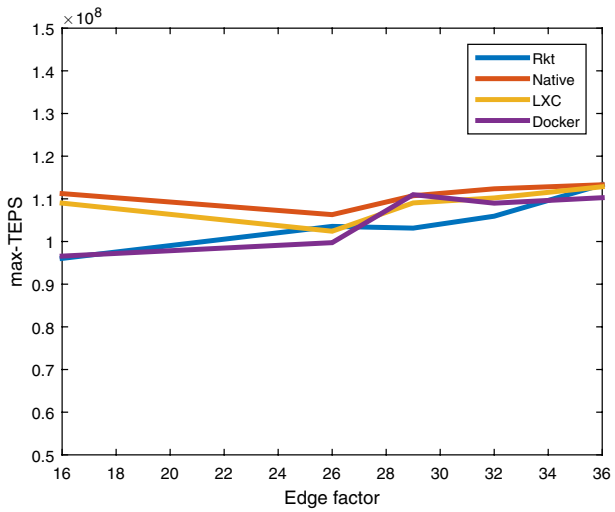

Fig. 8 Data intensive computing performance comparison among containers

that the Rkt containers offer a viable option for HPC. There are more challenges to be resolved to deliver the Rkt environment as the best option for HPC applications. Being a successor of Docker, the Rkt containers are better in some aspects, on the other hand, being an emerging technology still in its evolving stage, there remains more scope for improvement.

Conclusion

Container technology is becoming a widespread platform used in Cloud computing. High performance computing centers are now using containers for their complex applications because of the flexibility and gain in productivity. Adopting containers is not an easy task. In this work, we explore the feasibility of CoreOS Rkt container in the HPC world. We performed evaluations in two scenarios related to HPC. One is the data intensive application environment and the other is the computation intensive environment. We explored the widely used containers such as Docker and LXC in the same scenarios and provided a comparison of the results.

The results of the experiments show that the CoreOS Rkt container gives near native performance in computational intensive and data intensive high performance application environment. The results are promising and indicate that Rkt is well suited to run HPC applications as well, especially for computation-intensive workloads. Future research can experiment various HPC applications, measure the communication performance and tune the algorithms accordingly. The enhanced security features, interoperability and better performance will lead Rkt to a prominent position in HPC environments in the near future.

Authors' contributions
JPM conducted the experiments, analysed the results and drafted the manuscript. AK and KC provided valuable suggestions on improving the standards of the manuscript. All authors read and approved the final manuscript.

Author details
[1] Department of Mathematical and Computational Sciences, National Institute of Technology Karnataka, Surathkal, Karnataka, India. [2] Department of Computer Science and Engineering, National Institute of Technology Karnataka, Surathkal, Karnataka, India.

Acknowledgements
The authors thank the reviewers for their suggestions which helped in improving the quality of the paper.

Competing interests
The authors declare that they have no competing interests.

Availability of data and materials
Not applicable.

Consent for publication
Not applicable.

Ethics approval and consent to participate
Not applicable.

Funding
Not applicable.

Publisher's Note

Received: 13 July 2017 Accepted: 20 December 2017
Published online: 08 January 2018

References

1. Moon Y, Yu H, Gil J-M, Lim J (2017) A slave ants based ant colony optimization algorithm for task scheduling in cloud computing environments. Hum-centric Comput Inf Sci 7(1):28
2. Zhu W, Lee C (2016) A security protection framework for cloud computing. J Inf Process Syst 12(3):538–547
3. Kar J, Mishra MR (2016) Mitigate threats and security metrics in cloud computing. J Inf Process Syst 12(2):226–233
4. Huh J-H, Seo K (2016) Design and test bed experiments of server operation system using virtualization technology. Hum-centric Comput Inf Sci 6(1):1
5. Yu H-E, Huang W (2015) Building a virtual hpc cluster with auto scaling by the docker. arXiv preprint arXiv:1509.08231
6. Louati T, Abbes H, Cérin C, Jemni M (2018) Lxcloud-cr: towards linux containers distributed hash table based checkpoint-restart. J Parallel Distrib Comput 111:187–205
7. Ciuffoletti A (2015) Automated deployment of a microservice-based monitoring infrastructure. Procedia Comput Sci 68:163–172
8. Babu A, Hareesh M, Martin JP, Cherian S, Sastri Y (2014) System performance evaluation of para virtualization, container virtualization, and full virtualization using xen, openvz, and xenserver. In: 2014 fourth international conference on advances in computing and communications (ICACC). IEEE, New York, pp 247–250
9. CoreOS. https://coreos.com/rkt. Accessed 23 June 2017
10. Julian S, Shuey M, Cook S (2016) Containers in research: initial experiences with lightweight infrastructure. In: Proceedings of the XSEDE16 conference on diversity, big data, and science at scale. ACM, New York, p 25
11. Jacobsen DM, Canon RS (2015) Contain this, unleashing docker for hpc. In: Proceedings of the Cray User Group
12. Felter W, Ferreira A, Rajamony R, Rubio J (2015) An updated performance comparison of virtual machines and linux containers. In: 2015 IEEE international symposium on performance analysis of systems and software (ISPASS). IEEE, New York, pp 171–172
13. Linux LXC. https://linuxcontainers.org/lxc/. Accessed 23 June 2017
14. Docker Hub. https://hub.docker.com/. Accessed 23 June 2017
15. Kniep C (2014) Containerization of high performance compute workloads using docker. doc.qnib.org
16. de Alfonso C, Calatrava A, Moltó G (2017) Container-based virtual elastic clusters. J Syst Softw 127:1–11
17. Morabito R, Kjällman J, Komu M (2015) Hypervisors vs. lightweight virtualization: a performance comparison. In: 2015 IEEE international conference on cloud engineering (IC2E). IEEE, New York, pp 386–393
18. Xavier MG, Neves MV, Rossi FD, Ferreto TC, Lange T, De Rose CA (2013) Performance evaluation of container-based virtualization for high performance computing environments. In: 2013 21st euromicro international conference on parallel, distributed and network-based processing (PDP). IEEE, New York, pp 233–240
19. Chung MT, Quang-Hung N, Nguyen M-T, Thoai N (2016) Using docker in high performance computing applications. In: 2016 IEEE sixth international conference on communications and electronics (ICCE). IEEE, New York, pp 52–57
20. Kozhirbayev Z, Sinnott RO (2017) A performance comparison of container-based technologies for the cloud. Fut Gener Comput Syst 68:175–182
21. Varma PCV, Kumari VV, Raju SV et al (2016) Analysis of a network io bottleneck in big data environments based on docker containers. Big Data Res 3:24–28
22. Chung MT, Le A, Quang-Hung N, Nguyen D-D, Thoai N (2016) Provision of docker and infiniband in high performance computing. In: 2016 international conference on advanced computing and applications (ACOMP). IEEE, New York, pp 127–134
23. Peinl R, Holzschuher F, Pfitzer F (2016) Docker cluster management for the cloud-survey results and own solution. J Grid Comput 14(2):265–282
24. HPL—a portable implementation of the high-performance Linpack benchmark for distributed-memory computers. http://www.netlib.org/benchmark/hpl/. Accessed 4 July 2017
25. Top 10 (2016) http://www.graph500.org/. Accessed 4 July 2017