Orchestrating Docker Containers in the HPC Environment

Joshua Higgins, Violeta Holmes, Colin Venters

The University of Huddersfield,
Queensgate, Huddersfield, UK
{joshua.higgins, v.holmes, c.venters}@hud.ac.uk

Abstract. Linux container technology has more than proved itself useful in cloud computing as a lightweight alternative to virtualisation, whilst still offering good enough resource isolation. Docker is emerging as a popular runtime for managing Linux containers, providing both management tools and a simple file format. Research into the performance of containers compared to traditional Virtual Machines and bare metal shows that containers can achieve near native speeds in processing, memory and network throughput. A technology born in the cloud, it is making inroads into scientific computing both as a format for sharing experimental applications and as a paradigm for cloud based execution.However, it has unexplored uses in traditional cluster and grid computing. It provides a run time environment in which there is an opportunity for typical cluster and parallel applications to execute at native speeds, whilst being bundled with their own specific (or legacy) library versions and support software. This offers a solution to the Achilles heel of cluster and grid computing that requires the user to hold intimate knowledge of the local software infrastructure. Using Docker brings us a step closer to more effective job and resource management within the cluster by providing both a common definition format and a repeatable execution environment. In this paper we present the results of our work in deploying Docker containers in the cluster environment and an evaluation of its suitability as a runtime for high performance parallel execution. Our findings suggest that containers can be used to tailor the run time environment for an MPI application without compromising performance, and would provide better Quality of Service for users of scientific computing.

Keywords: Linux containers, Docker, cluster, grids, run time environment

1 Introduction

Cloud computing has been driven by the Virtual Machine (VM). They are widely deployed to achieve performance and resource isolation for workloads by constraining the amount of virtual memory and processor cores available to a guest system. This allows resource sharing on a massive scale; VMs can be provisioned with any software environment and kept separate from other guests on the same physical server[18].

Linux container technology is classed as an operating system virtualisation method. It allows the creation of separate userspace instances in which the same kernel is shared.This provides functionality similar to a VM but with a lighter footprint. The Docker project provides a management tool and its own library for communicating with containment features in the OS kernel[2].

Resource isolation takes a back seat in HPC systems which generally execute a user's job within the same OS environment that runs directly on the hardware to gain the best performance. This poses a problem for application portability, especially in grid systems where a remote resource may lack the libraries or support software required by a job. This undermines efforts by middleware vendors to unify resources and provide a common format for accessing heterogeneous systems[8]. In this respect some features of the cloud are desirable in cluster computing.

Docker containers offer an opportunity to create cloud-like flexibility in the cluster without incurring the performance limitations of a VM. This paper investigates the performance gains that can be achieved using Docker containers for executing parallel applications when compared to the KVM hypervisor[5] in a typical scientific cluster computing environment. We also propose a method of executing an MPI job encapsulated in a Docker container through the cluster resource manager.

In the sections of this paper, a short review of Docker containers versus Virtual Machines will be conducted. The current work will be then discussed. Section 4 describes the proposed Docker in the HPC cluster solution. The results from the deployment of this implementation will be evaluated and future work identified.

2 Docker vs KVM

2.1 Architecture

KVM is a popular hypervisor for Linux that introduces virtualisation support into the Linux kernel. The hypervisor provides an illusion to the guest OS that it is managing it's own hardware resources [15]. A request from the guest must be translated into a request to the underlying physical hardware; a process in modern hypervisors that is highly optimised and transparent to the guest. This allows the hypervisor to host a VM without modifications to the OS that has been chosen.

Docker containers do not require a layer of translation. On Linux, they are implemented using 'cgroups'; a feature in the Linux kernel that allows the resources (such as CPU, memory and network) consumed by a process to be constrained[14]. The processes can then be isolated from each other using kernel 'namespaces'[13]. This fundamental difference requires that the guest system processes are executed by the host kernel, restricting containers on a Linux host to only other Linux flavours. However, it means that an executable within the container system essentially runs with no additional overhead compared to an

executable in the host OS. A container is not required to perform any system initialisation - its process tree could contain just the program being run and any other programs or services that it depends on.

2.2 Performance

A benchmark by IBM Research demonstrates that the performance of a Docker container equals or exceeds KVM performance in CPU, memory and network benchmarks [12]. The results show that both methods do not introduce a measurable overhead for CPU and memory performance. However the Linpack performance inside KVM is shown to be very poor; the hypervisor abstracts the hardware and processor topology which does not allow tuning and optimisation to take place.

They also suggest that the preferential scaling topology for Docker containers is by processor socket. By not allowing a container to span cores distributed over multiple processor sockets, it avoids expensive inter-processor bus communication. This is in line with the philosophy already ingrained in cluster computing applications in which a process per core is executed on compute nodes. However, the effect may not be appreciable in distributed memory applications where the bandwidth of the network interconnect may be many orders of magnitude slower.

2.3 Storage

A Virtual Machine is traditionally accompanied by a disk image which holds the OS and run time applications. To create or modify this disk image would require the user to hold the knowledge of installing the OS, or systems management experience. The resulting image file may span several gigabytes. This places a large burden on storage and may be inconvenient to transfer between systems.

The Docker project introduces a concept of a 'Dockerfile' which allows the userspace of a container to be described as a list of directives in a text file that construct the real image[3]. Each directive produces a layer of the final system image, which are combined at run time using a copy-on-write method to appear to the container as a single unified image. This allows multiple containers to share common image layers, potentially reducing the amount of data required to transfer between systems. The Dockerfile itself is significantly easier to customise and can be easily version controlled or shared.

3 Current Work

The flexibility of the cloud allows one to create multiple clusters where each individual virtual cluster can have it's own customised software environment. This draws parallels with the development of the now defunct OSCAR-V middleware[17] and Dynamic Virtual Clustering[11] concepts, which are able to provision a virtual cluster based on the requirements of a job at the time of submission. These systems still incur the overhead of running a VM as the job execution environment and inherit the associated performance limitations.

The Agave API is a gateway platform that provides services typically found in grid systems tailored for scientific users. It has a strong focus on web standards and boasts support for running applications packaged in Docker containers in a cloud[10]. However, Agave orchestrates a single container per job, which limits the scope for running parallel applications[9].

HTCondor is a middleware for high-throughput computing that has support for running jobs in parallel on dedicated machines, using 'universes' to distinguish between different execution paradigms. HTCondor itself already supports 'cgroups' to constrain the resources available to a job on Linux hosts and a universe is under development to provide support for the Docker container format[16]. HTCondor has powerful resource discovery features but is the usefulness of a container in not needing to know?

4 Docker in the HPC Cluster

To implement existing container and VM solutions in HPC systems requires modifications to the software stack of the local HPC resources. The HPC systems would already have resource management and job scheduling in place. The methodology should follow the concept of containers, that is to abstract the application from the software stack of the resource. Modifying core components of this stack to support containers introduces a portability problem. Any standard resource manager provides all the information required to orchestrate a Docker container within a resource.

A resource manager such as Torque uses a script that is the core of the job execution and is responsible for configuring the environment and passing the required information to a process starter, such as 'mpirun'. We use both the MPICH[6] and OpenMPI[7] launchers regularly in our systems, which support SSH to enable remote process execution.

4.1 Choosing a container model

Since we concluded that running a container has no appreciable overhead over running a normal executable, we propose two different container models for parallel execution. A single container could be started to mimic the worker node, as shown in Figure 1, which would hold all the processes assigned to that node.

Secondly, a container per process could also be orchestrated as shown in Figure 2. Whilst it is unlikely that processes within the same job would require different run time environments, this presents an interesting opportunity for resource usage accounting and can offer real time adjustment of the resource constraints per process through 'cgroups'. It can also enforce a process to be mapped to a specific processor core if this functionality is missing from the process launcher.

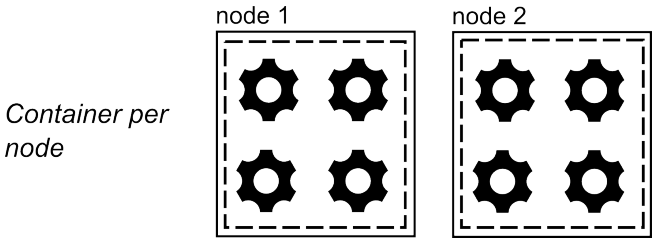

Fig. 1. A container per node that holds all respective processes

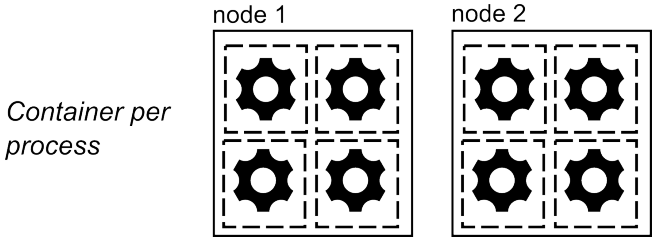

Fig. 2. A container per process is orchestrated on each node

4.2 Container model implementation

The resource manager will not be aware of any containers, so the script that is central to the job execution will be used to prepare the nodes and start the containers, before the process launcher starts the parallel job. An overview of this process is described in Figure 3. We cannot assume that the user is able to SSH to the worker nodes to run the container preparation commands. In this case the bootstrap script can also be invoked through the process launcher first and then the process launcher is called a second time to invoke the parallel job. The overview of the script process supports both container models.

When the process launcher is called, the environment has been modified in two ways: the container will randomise it's SSH port and expose it on the same interface as the host. The SSH configuration is temporarily changed so that the process launcher will use this port instead of the default, thereby launching the process within the container and not on the node. However, if the process launcher forks the process on the rank 0 node instead of using SSH, the process will run outside the container. To avoid this condition, the script will write a unique hostname alias for each node into the SSH configuration that maps to the container on that node. These are substituted into the list of nodes provided by the resource manager before being passed to the process launcher.

5 Deployment of container model on HPC cluster

Eridani is the general purpose, 136 core cluster within the QueensGate Grid at the University of Huddersfield. Like the other clusters within this campus grid, it

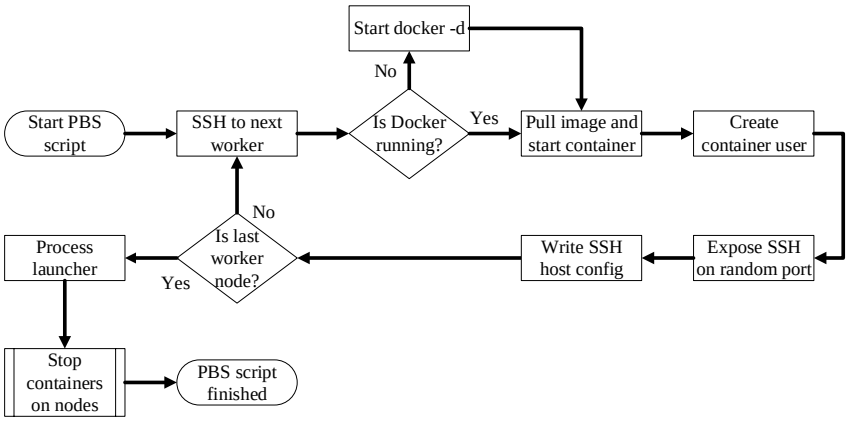

Fig. 3. Overview of script process

uses the Torque resource manager running on CentOS 6.5 and accepts jobs in a PBS script format. The Intel-optimised parallel LINPACK version 11.2.1.009[4] benchmark was used to measure the performance of 2 nodes with 4 cores each.

In order to validate the claim in [12] that for the non-optimised use case there is no appreciable difference in the CPU execution performance between a Docker container and a VM, the same benchmark was run using the reference BLAS version 3.5.0 library without architecture specific optimisations[1].

The containers were orchestrated in both container per-core and container per-node models, using the implementation described in the previous section. The VMs were created to consume all available cores on the host and appropriate memory to fit the LINPACK problem size.

5.1 Results

Figure 4 shows the experimental results using both the Intel-optimised parallel LINPACK and generic BLAS library comparing native, Docker container models and KVM performance. The results have been obtained from 10 runs of each configuration. The peak performance observed per configuration is shown.

5.2 Evaluation

The LINPACK experimental results echo those obtained by previous research[12], showing that the performance of Docker containers has no appreciable difference compared to running natively, whilst the VM achieves approximately half the peak performance of the container. However, this work differs significantly in that the parallel LINPACK uses a distributed memory model, not shared memory, utilising a network interconnect for message passing to achieve data sharing between processes.

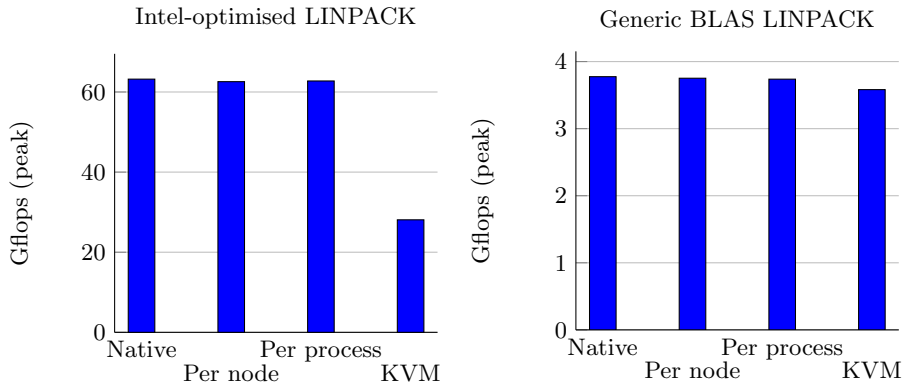

Fig. 4. LINPACK benchmarking results

Without optimisation for the processor architecture, the performance of a VM and container are mostly comparable. However, the overall peak performance is considerably lower for this application. This suggests that the Docker container is therefore a more appropriate execution method for high performance parallel applications where we are likely to employ these types of optimisation.

There is no difference in performance between the two container orchestration models proposed. This is expected given that a process within the container is essentially a process running in the host kernel with 'cgroup' limitations.

6 Summary

One of the requirements of grid computing is to run a job transparently to the user on any resource they desire without requiring knowledge of the local software configuration. Based on our research and experimental results conducted, it is evident that Docker containers can facilitate this by abstracting the software environment of the local HPC resource without compromising performance. This improves Quality of Service for our users by

- Allowing parallel jobs to run on traditional PBS clusters with arbitrary run time environments
- Reducing the entry level of customising the run time environment to that of the average user
- Running jobs on resources within the grid that was previously not possible due to software configuration

7 Future Work

The container per process model offers many advantages by allowing us to apply 'cgroup' constraints to each process in a HPC job. This would allow resource

management to be improved based on job requirements as more fine grained control can be achieved for network and disk I/O usage in addition to CPU time[14]. It also provides scope for optimising the power consumption of a job as limits can be changed in real time without restarting the process.

In our future work we will perform benchmarking to appreciate the impact of Docker's NAT networking on message passing. We will also investigate orchestration of Docker containers that contain parallel applications in the cloud environment as opposed to traditional cluster computing.

Acknowledgments. The experimental results for this work could not have been obtained without the resources and support provided by the QueensGate Grid (QGG) at The University of Huddersfield.

References

1. Blas (basic linear algebra subprograms). `http://www.netlib.org/blas/`.
2. Docker. `https://www.docker.com/`.
3. Dockerfile reference - docker documentation. `https://docs.docker.com/reference/builder/`. Version 1.4.
4. Intel math kernel library linpack download. `https://software.intel.com/en-us/articles/intel-math-kernel-library-linpack-download`.
5. Kernel based virtual machine. `http://www.linux-kvm.org/`.
6. Mpich high performance portable mpi. `http://www.mpich.org/`.
7. Open mpi: Open source high performance computing. `http://www.open-mpi.org/`.
8. Charlie Catlett. Standards for grid computing: Global grid forum. *Journal of Grid Computing*, 1(1):3–7, 2003.
9. Rion Dooley. Agave docker quickstart. `https://bitbucket.org/deardooley/agave-docker-support/`, 09 2014.
10. Rion Dooley, Matthew Vaughn, Dan Stanzione, Steve Terry, and Edwin Skidmore. Software-as-a-service: The iplant foundation api. In *5th IEEE Workshop on Many-Task Computing on Grids and Supercomputers*, November 2012.
11. W. Emeneker and D. Stanzione. Dynamic virtual clustering. In *Cluster Computing, 2007 IEEE International Conference on*, pages 84–90, Sept 2007.
12. Wes Felter, Alexandre Ferreira, Ram Rajamony, and Juan Rubio. An updated performance comparison of virtual machines and linux containers. *technology*, 28:32, 2014.
13. Michael Kerrisk. Namespaces in operation, part 1: namespaces overview. `http://lwn.net/Articles/531114/`, 1 2014. Accessed 7 Feb 2015.
14. Paul Menage. Cgroups. `https://www.kernel.org/doc/Documentation/cgroups/cgroups.txt`. Accessed 7 Feb 2015.
15. Jim Smith and Ravi Nair. *Virtual Machines: Versatile Platforms For Systems And Processes*. Morgan Kaufmann, 2005.
16. Todd Tannenbaum. Htcondor and hep partnership and activities. Presented at the HEPiX Fall 2014 Workshop, University of Nebraska, Lincoln, 13-17 October 2014, 2014.
17. Geoffroy Vallee, Thomas Naughton, and Stephen L Scott. System management software for virtual environments. In *Proceedings of the 4th international conference on Computing frontiers*, pages 153–160. ACM, 2007.
18. Aaron Weiss. Computing in the clouds. *netWorker*, 11(4):16–25, December 2007.