APPLICATIONS OF ON-DEMAND VIRTUAL CLUSTERS TO HIGH PERFORMANCE COMPUTING[1]

I.G. Gankevich, S.G. Balyan, S.A. Abrahamyan, V.V. Korkhov

Saint Petersburg State University
igankevich@cc.spbu.ru, serob.balyan@gmail.com,
suro7@live.com, vkorkhov@csa.ru

Virtual machines are usually associated with an ability to create them on demand by calling web services, then these machines are used to deliver resident services to their clients; however, providing clients with an ability to run an arbitrary programme on the newly created machines is beyond their power. Such kind of usage is useful in a high performance computing environment where most of the resources are consumed by batch programmes and not by daemons or services. In this case a cluster of virtual machines is created on demand to run a distributed or parallel programme and to save its output to a network attached storage. Upon completion this cluster is destroyed and resources are released. With certain modifications this approach can be extended to interactively deliver computational resources to the user thus providing virtual desktop as a service. Experiments show that the process of creating virtual clusters on demand can be made efficient in both cases.

Introduction

Virtualisation is employed for many components of computing system such as processor, memory, storage subsystem and network interface, however, virtualising all of them at once is not required in high performance computing where efficiency of the resulting computing system is of utmost importance. First, most of the virtualisation comes at a cost of slight performance decrease, which grows with the size of the application. Second, virtualisation of some components (i.e. network interfaces, processors) complexifies system's architecture without simplifying problem solution. This led to scarce use of virtualisation technologies in high performance computing; contrary to this we feel that there is at least one way of using it to simplify architecture of the system and decrease its maintenance effort.

This way consists of storing applications which run on cluster and their dependencies inside lightweight containers. Upon submitting a task containers are mounted on each host given by a resource manager and parallel application is executed inside each of them. The benefit of this approach is that a separate operating system and optimised libraries can be installed for each application without the need to alter host computer configuration. In addition to this, it is easy to maintain different versions of applications (due to licensing or compatibility problems) in different containers and to update them in contrast to common shared-folder-for-all-applications configuration where there is a lot of manual work. Finally, since container virtualisation does not imply processor virtualisation overheads this approach can be made efficient in terms of application performance.

The rest of the paper discusses related work (Section 1), describes system's architecture with containers (Section 2) and evaluates its efficiency on basis of real-world application and some synthetic tests (Section 3).

1 Related work

Research works on the subject of virtual clusters can be divided into two broad groups: works dealing with provisioning and deploying virtual clusters in high performance environment or GRID

[1]The research was carried out using computational resources of Resource Center Computational Center of Saint Petersburg State University (T-EDGE96 HPC-0011828-001) and partially supported by Russian Foundation for Basic Research (project No. 13-07-00747) and Saint Petersburg State University (project No. 9.38.674.2013).

and works dealing with overheads of virtualisation. Works from the first group typically assume that virtualisation overheads are low and acceptable in high performance computing and works from the second group in general assume that virtualisation has some benefits for high performance computing, however, the authors are not aware of the work that touches both subjects in aggregate.

In [1] authors evaluate overheads of the system for on-line virtual cluster provisioning (based on QEMU/KVM) and different resource mapping strategies used in this system and show that the main source of deploying overhead is network transfer of virtual machine images. To reduce it they use different caching techniques to reuse already transferred images as well as multicast file transfer to increase network throughput. Simultaneous use of caching and multicasting is concluded to be an efficient way to reduce overhead of virtual machine provisioning.

In [2] authors evaluate general overheads of Xen para-virtualisation compared to fully virtualised and physical machines using HPCC benchmarking suite. They conclude that an acceptable level of overheads can be achieved only with para-virtualisation due to its efficient inter domain communication (bypassing dom0 kernel) and absence of high L2 cache miss rate when running MPI programs which is common to fully virtualised guest machines.

In contrast to these two works the main principles of our approach can be summarised as follows. Do not use full or para-virtualisation of the whole machine but use virtualisation of selected components so that overheads occur only when they are unavoidable (i.e. do not virtualise processor). Do not transfer opaque file system images but mount standard file systems over the network so that only minimal transfer overhead can occur. Finally, amend standard task schedulers to work with virtual clusters so that no programming is needed to distribute the load efficiently. These principles are paramount to make virtualisation lightweight and fast.

2 System's configuration

The system comprises many standard components which are common in high performance computing, these are distributed parallel file system which stores home directories with experiment's input and output data, cluster resource scheduler which allocates resources for jobs and client programmes to pre- and post-process data; the non-standard component is network-attached storage exporting container's root files systems as directories. Linux Container technology (LXC) is used to provide containerisation, GlusterFS is used to provide parallel file system and TORQUE to provide task scheduling. The most recent CentOS Linux 7 is chosen to provide stable version of LXC (>1.0) and version of kernel which supports all containers' features. Due to limited number of nodes each of them is chosen to be both compute and storage node and every file in parallel file system is stored on exactly two nodes. Detailed hardware characteristics and software version numbers are listed in Table 1.

.

Component	Details	Component	Details
CPU model	Intel Xeon E5440	Operating system	CentOS Linux 7 (Core)
CPU clock rate (GHz)	2.83	Kernel version	3.10
No. of cores per CPU	4	LXC version	1.0.5
No. of CPUs per node	2	GlusterFS version	3.5.1
RAM size (GB)	4	TORQUE version	5.0.0
Disk model	ST3250310NS	OpenMPI version	1.6.4
Disk speed (rpm)	7200	IMB version	4.0
No. of nodes	12	OpenFOAM version	2.3.0
Interconnect speed (Gbps)	1		

Table 1. Hardware and software components of the system.

Creating virtual cluster in such environment requires the following steps. First, a client submits a task requesting particular number of cores. Then according to distribution of these cores among compute nodes a container is started on each node from the list with SSH daemon as the only program running inside it. Here there are two options: either start containers with network virtualisation (using “macvlan” or “br” LXC network type) and generate sufficient number of IP addresses for the cluster or use host network name space (“none” LXC network type) and generate only the port number to run ssh daemon on. The next step is to copy (possibly amended) node file from host into the first container and launch submitted script inside it. When the script finishes its work SSH daemon in every container is killed and all containers are destroyed.

For this algorithm to work as intended client's home directory should be bind-mounted inside the container before launching the script. Additionally since some MPI programmes require “scratch” directories on each node to work properly, container's root file system should be mounted in copy-on-write mode, so that all changes in files and all the new files are written to host's temporary directory and all unchanged data is read from read-only network-mounted file system; this can be accomplished via Union or similar file system and that way application containers are left untouched by tasks running on the cluster.

To summarise, only standard Linux tools are used to build the system: there are no opaque virtual machines images, no sophisticated full virtualisation appliances and no heavy-weight cloud computing stacks in this configuration.

3 Evaluation

To test the resulting configuration OpenMPI and Intel MPI Benchmarks (IMB) were used to measure network throughput and OpenFOAM was used to measure overall performance on a real-world application.

The first experiment was to create virtual cluster, launch an empty (/bin/true) MPI programme in it and compare execution time to ordinary physical cluster. To set this experiment up in the container the same operating system and version of OpenMPI as in the host machine was installed. No network virtualisation was used, each run was repeated several times and the average was displayed on the graph (Figure 1). The results show that a constant overhead of 1.5 second is added to every LXC run after the 8th core: one second is attributed to the absence of cache inside container with SSH configuration files, key files and libraries in it and other half of the second is attributed to the creation of containers as shown in Figure 2. The jump after the 8th core marks bounds of a single machine which means using network for communication rather than shared memory. The creation of containers is fully parallel task and takes approximately the same time to complete for different number of nodes. Overhead of destroying containers was found to be negligible and was combined with “mpirun” time. So, usage of Linux containers adds some constant overhead to the launching of parallel task depending on system's configuration which is split between creation of containers and filling the file cache.

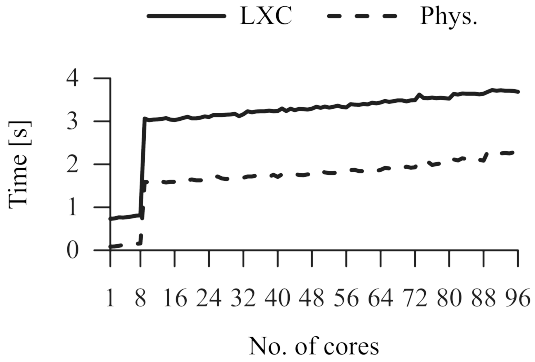

Figure 1. Comparison of LXC and physical cluster performance running empty MPI programme.

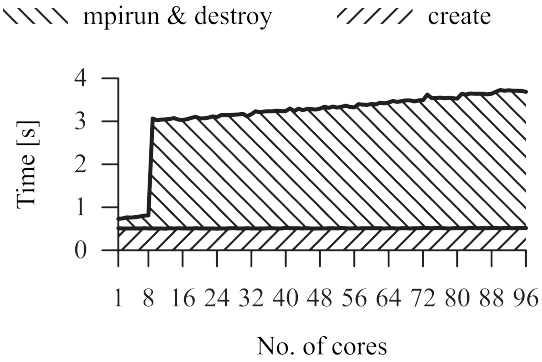

Figure 2. Breakdown of LXC empty MPI programme run.

The second experiment was to measure performance of different LXC network types using IMB suite and it was found that the choice of network virtualisation greatly affects performance. As in the previous test container was set up with the same operating system and the same IMB executables as the host machine. Network throughput was measure with “exchange” benchmark and displayed on the graph (Figure 3). From the graph it is evident that until 2^{14} bytes message size the performance is approximately the same for all network types, however, after this mark there is a dip in performance of virtual ethernet. It is difficult to judge where this overhead comes from: some studies report that under high load performance of bridged networking (veth is always connected to the bridge) is decreased [3], but IMB does not have high load on the system. Additionally, the experiment showed that as expected throughput decreases with the number of cores due to synchronisation overheads (Figure 4).

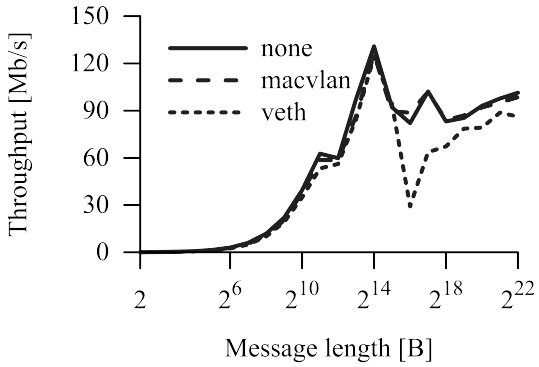

Figure 3. Average throughput of "exchange" MPI benchmark.

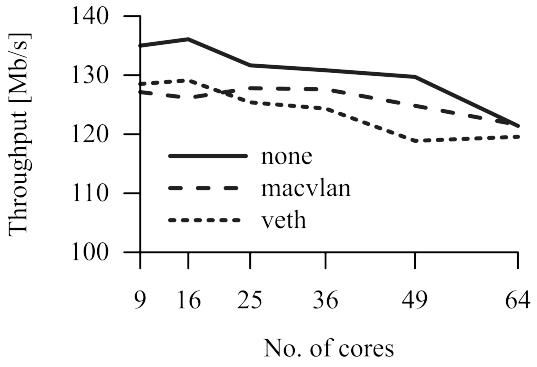

Figure 4. Throughput for 16Kb messages.

The third and the last experiment dealt with real-world application performance and for this role the OpenFOAM was chosen as the complex parallel task involving large amount of network communication, disk I/O and high CPU load. The dam break RAS case was run with different number of cores (total number of cores is the square of number of cores per node) and different LXC network types and the average of multiple runs was displayed on the graph (Figure 5). Measurements for 4 and 9 cores were discarded because there is a considerable variation of execution time for these numbers on physical machines. From the graph it can be seen that low performance of virtual ethernet decreased final performance of OpenFOAM by approximately 5-10% whereas “macvlan” and “none” performance is close to the performance of physical cluster (Figure 6). So, the choice of network type is the main factor affecting performance of parallel applications running on virtual clusters and its overhead can be eliminated by using “macvlan” network type or by not using network virtualisation at all.

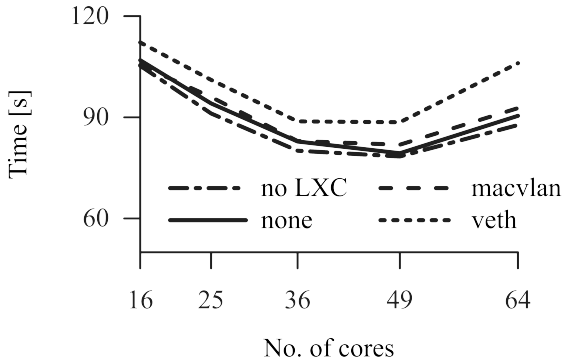

Figure 5. Average performance of OpenFOAM with different LXC network types.

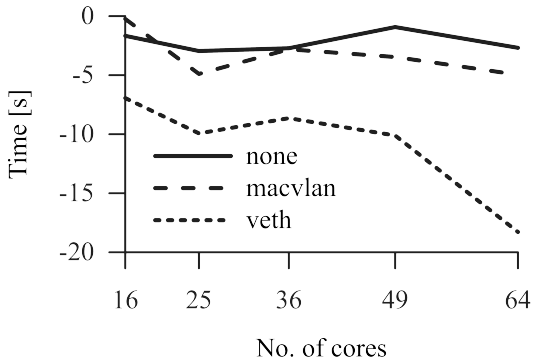

Figure 6. Difference of OpenFOAM performance on physical and virtual clusters. Negative numbers show slowdown of virtual cluster.

To summarise, there are two main types of overheads when using virtual cluster: creation overhead which is constant and small compared to average time of a typical parallel job and network overhead which can be eliminated by not using network virtualisation at all.

Conclusions and future work

Presented approach for creating virtual clusters from Linux containers was found to be efficient and its performance comparable to ordinary physical cluster: not only usage of containers does not incur processor virtualisation overheads but also network virtualisation overheads can be totally removed if host's network name space is used and network bandwidth saved by automatically transferring only those files that are needed through network-mounted file system rather than the whole images. From the point of view of system's administrator storing each HPC application in its own container makes version and dependencies control easy manageable and their configuration does not interfere with the configuration of host machines and other containers.

Acknowledgements

Authors would like to thank Andrey Zarochentsev for his advice on using UnionFS to make root file system writeable which was the turning point to eliminate existing over-sophisticated solution.

References

[1] Chen Y., Wo T., Li J. An efficient resource management system for on-line virtual cluster provision // Cloud Computing, 2009. CLOUD'09. IEEE International Conference on. – IEEE, 2009. – C. 72-79.

[2] Ye K. et al. Analyzing and modeling the performance in Xen-based virtual cluster environment // High Performance Computing and Communications (HPCC), 2010 12th IEEE International Conference on. – IEEE, 2010. – C. 273-280.

[3] James T. Y. Performance evaluation of Linux Bridge // Telecommunications System Management Conference. – 2004.