Distributed Computing Infrastructure Based on Dynamic Container Clusters

Vladimir Korkhov(✉), Sergey Kobyshev, Artem Krosheninnikov, Alexander Degtyarev, and Alexander Bogdanov

St. Petersburg State University,
7/9 Universitetskaya Nab., St. Petersburg 199034, Russia
v.korkhov@spbu.ru

Abstract. Modern scientific and business applications often require fast provisioning of an infrastructure tailored to particular application needs. In turn, actual physical infrastructure contains resources that might be underutilized by applications if allocated in dedicated mode (e.g., a process does not utilize provided CPU or network connection fully). Traditional virtualization technologies can solve the problem partially, however, overheads on bootstrapping a virtual infrastructure for each application and sharing physical resources might be significant. In this paper we propose and evaluate an approach to create and configure dedicated computing environment tailored to the needs of particular applications, which is based on light-weight virtualization also known as containers. We investigate available capabilities to model and create dynamic container-based virtual infrastructures sharing a common set of physical resources, and evaluate their performance on a set of test applications with different requirements.

Keywords: Virtualization · Containers · Virtual cluster

1 Introduction

Constant development of computer hardware and software together with the development of computing methods and algorithms stimulates new ways of bringing together software and hardware, matching application requirements and resources, thus mapping programs to computing infrastructures. Virtualization technologies started a new era of tailoring computing environment to the needs of users and applications. However, flexibility of full- and para-virtualization approaches is hold back by some limitations causing extra overheads, resource consumption and lack of dynamics. Container-based virtualization, a new generation of virtualization techniques, can give better answers to create a flexible and dynamic distributed computing infrastructure with small overhead.

Containers as a way to create a dedicated environment for running applications have been around for years. However, the boost of new interest to

O. Gervasi et al. (Eds.): ICCSA 2016, Part II, LNCS 9787, pp. 263–275, 2016.
DOI: 10.1007/978-3-319-42108-7_20

them started when new technologies and tools to orchestrate their operations appeared. One of the most commonly used tool to manage container infrastructures is Docker [8].

Traditional hypervisor-based virtualization is still widely used to deploy and run applications on a wide range of platforms, however, it suffers from a number of restrictions:

- Significant overheads while running fully-virtualized guest operating systems, in particular, overheads to boot up virtual machine instances
- Lack of flexibility to allocate resources to particular processes
- Downfalls of application performance due to virtualization overheads, hypervisor mediation etc.

While hypervisor-based virtualization provides flexibility in building variety of environments, e.g. allowing to simulate and run completely different architectures and operating systems on top of each other, container-based or operating system-level virtualization is restricted by using the same core components within host and containers, in particular operating system kernel. Nevertheless, the variety of supported platforms is often not required, but low overheads and dynamics are needed.

Container-based virtualization, also referred as operating-system level or light-weight virtualization, follows a different paradigm compared to hypervisor virtualization. Containers are based on the host operating systems itself rather than on hypervisor. Containers do not virtualize hardware, which would require virtualized operating system images on each guest OS. Instead, containers virtualize OS by sharing the host OS kernel and other resources between original host environment and environments run in containers. Thus, containers provide an isolated and controlled environment that provides everything an application might need for execution without extra overheads caused by virtualizing hardware.

In this paper, we evaluate the capabilities obtained while using the OS-level virtualization technology to build a computational environment with configurable computation (CPU, memory) and network (latency, bandwidth) characteristics. Such configuration enables flexible partitioning of available physical resources between a number of concurrent applications utilizing a single physical infrastructure. Depending on application requirements and priorities of execution each application can get a customized virtual environment with as much resources as it needs or is allowed to use.

Our main interest is to use container-based computing infrastructures for parallel high-performance computing applications: parallel programs that consist of a number of processes running on computing nodes and communicating during the execution. Using containers as computing nodes can help us to control and share available computing and networking resources between concurrent parallel applications. Thus, applications with complementary requirements (e.g. fast CPU-slow network + slow CPU-fast network) can co-exist on a single physical node or a VM without affecting each other much.

The approach that we propose is complementary to the traditional queue-based batch processing used in HPC systems. Applications would not have to wait in the queue until worker nodes become fully free and available. Instead, the scheduler can control fraction of resources allocated for each application thus enabling immediate execution for applications with requirements fitting still available fraction of resources. In addition, flexible quality of service (QoS) and service-level agreement (SLA) policies can be built on top of such infrastructure: applications might be ready to get smaller amount of resources right away rather than wait in line to acquire more resources.

The paper is structured as follows: Sect. 2 gives an overview of related work in the area of container management software. Section 3 takes a closer look at comparison or containers and virtual machines for building distributed computing infrastructures. Section 4 presents and approach to simulate and predict actual application requirements that can be used for configuring container-based DCI created for particular application. Section 5 presents an experimental evaluation of building container-based computing environment for a number of test applications. Section 6 discusses the results and Sect. 7 concludes the paper.

2 Related Work

Containers are an easy way to generate large amounts of compute units, and robust monitoring, management, and orchestration are needed to cope with container crowds, where containers can be mislocated or left running forgotten.

There are a number of available tools and technologies that provide means to manage containers, maintain their lifecycle, orchestrate and monitor their execution.

One of the most popular tools for managing containers is Docker [8]. Docker introduced the concept of the container image. The Docker container image is a straightforward way to package an application and all its dependencies so that it can be executed on any modern Linux servers supporting Docker. Such portability is very important for distributed infrastructures that can be based on various platforms and versions of operating systems. In addition, Docker has tools for container deployment and orchestration, including Docker Machine, Docker Compose, and Docker Swarm. Docker Machine provides means to easily deploy Docker Engines local computer, on cloud providers, and in data centers. Docker Swarm is a native clustering solution for Docker containers. It pools together several Docker Engines into a single virtual host. Docker Compose is a way of defining and running multi-container distributed applications with Docker.

Kubernetes (originally by Google, now is a part of the Cloud Native Computing Foundation) is an open-source platform for automating deployment, scaling, and operations of application containers across clusters of hosts, providing container-centric infrastructure [11]. Kubernetes defines a set of building blocks ("primitives") which collectively provide mechanisms for deploying, maintaining, and scaling applications. These primitives are designed to be loosely coupled and

extensible so that the infrastructure can meet a wide variety of different workloads. The extensibility is provided in large part by the Kubernetes API, which is used by internal components as well as extensions and containers running on Kubernetes.

Apache Mesos can be used to deploy and manage application containers in large-scale clustered environments. It abstracts CPU, memory, storage, and other compute resources away from machines (physical or virtual), enabling fault-tolerant and elastic distributed systems to easily be built and run effectively [9]. At a high level Mesos is a cluster management platform that combines servers into a shared pool from which applications or frameworks like Hadoop, Jenkins, Cassandra, ElasticSearch, and others can draw. Mesos allows developers to conceptualize their applications as jobs and tasks. In combination with a job system like Marathon, it takes care of scheduling and running jobs and tasks. Marathon is a Mesos framework for long-running services such as web applications, long computations and so on [10].

CoreOS is a Linux distribution designed to make large multiple-machine deployments secure, consistent, and reliable. Instead of installing packages via yum or apt, CoreOS uses Linux containers to manage services at a higher level of abstraction. A single service's code and all dependencies are packaged within a container that can be run on one or many CoreOS machines [13]. It uses "fleet" for cluster management and "etcd" for service discovery and keeping configuration up to date across the cluster.

3 Deploying and Running Applications in DCIs: Containers vs Virtual Machines

Container virtualization allows to virtualize physical servers at the level of operating system kernel. OS virtualization layer provides insulation and security of resources between different containers. Virtualization layer makes each container similar to a physical server. Each container maintains therein an application and workload. The main advantages of container virtualization are the following:

- Containers are maintained on the level of physical servers. Lack of virtualized hardware, the use of real equipment and direct access to drivers allows to achieve high performance.
- Each container can be scaled to the resources of a physical server.
- Virtualization on the OS level allows to achieve the highest density among the other available virtualization solutions. You can create and launch hundreds of containers on a single physical server.
- Containers use a single OS, making their support and update very simple. Applications may also be deployed in a separate environment.

In addition to the light-weight virtualization benefits, containers provide flexible and convenient ways to package and distribute software. The older ways to package, deploy and distribute applications were installing them directly on

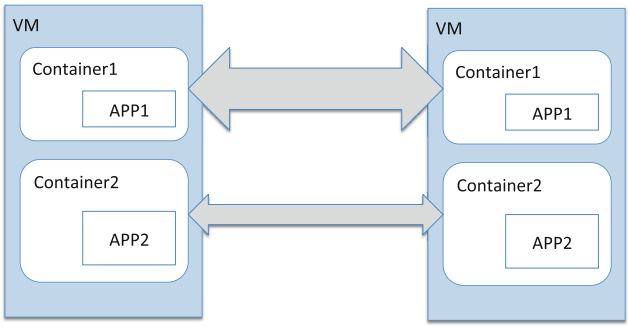

Fig. 1. Concurrent applications on container cluster

every machine with the help of operating system package managers, or creating a separate virtual machine for a particular software deployment that could be rather heavy-weight and non-portable.

Containers are based on operating-system-level virtualization rather than hardware virtualization, nevertheless they are still well isolated from each other and from the host. Containters have their own filesystems, they cannot see and influence each others processes, and their computational and network resource usage can be bounded. The latter brings us the possibility to create fully defined and controlled container-based clusters, configured to the needs of particular users and applications (Fig. 1).

Containers are not coupled to particular underlying infrastructure of filesystem; they are easy to build and portable across different types of operating systems in data centers or in clouds.

Containers are light-weight and fast; normally there is one-to-one relation between an application and a container image which enables composing a computing environment in a loosely coupled manner, built from individual blocks that can be easily created at build/release time rather than deployment time.

An important feature of containers for our research is the application-centric management that raises the level of abstraction from running an OS on virtual hardware to running an application on an OS using logical resources configured according to application requirements.

4 Simulating Container-Based Distributed Computing Infrastructure

Conducting offline simulations is often needed to preliminary assess the correctness and adequacy of the applications that interact with the network, in particular to evaluate performance of network-related algorithms. Moreover, the ability to perform reproducible experiments is necessary in the development of software. This is especially important when large-scale network applications are concerned since real network environment is dynamic, and application behavior

needs to be checked in advance in variety of conditions. The simulation tools like ns [7] and OMNet++ [14] allow to perform simulations and evaluate the efficiency and scalability of algorithms or protocols without running them in real networks.

Experimentation with real programs, however, is mainly focused on the measurement of makespan, CPU usage, memory usage, network performance, etc. One of the complex examples is an application running on multiple nodes over a network. Using the Internet in this case is not encouraged because there will be no exact repetition of the experiment conditions, and many parameters can not be controlled. In addition, network applications may interact with different hosts, but a change in the network topology and its parameters on this real testbed is time consuming and can be error prone. On the other hand, the topology of choice in accordance with the specific tasks may speed up the task and show a great performance. However, the transition to a different type of tasks would require changing the entire network topology.

Here we consider an approach to create a virtual testbed, having virtual links between nodes, that allows us to create a variety of network topologies. The created nodes may have different restrictions, e.g. the limitation on the memory or the data transfer speed between nodes. Furthermore, it is possible to simulate the conditions of poor communication between two or more nodes, to limit the bandwidth to add delay when transmitting packets, to change the error ratio, i.e. the ratio of the number of incorrectly received bits (1, instead of 0 and vice versa) to the total number of bits transmitted in transmission data between nodes.

When we talk about the emulation of the network we usually refer to the ability to control artificially created environment for effective interaction with real computer networks and real-time traffic. We should note that already in 1996 David Tennenhouse et al. proposed an idea of software-centric networks, called Active Networks, but then it was not widely acknowledged [2]. Recently similar projects such as OpenVSwitch [16], Mininet [1] and others appeared; together with the container virtualization technologies they can perform various tasks, like simulating many nodes with connections of controlled quality, simulated latency, packet loss etc., even within a single node.

While reviewing the existing software, we paid attention to the following points:

1. Virtualization of nodes. The ability to simulate multiple independent sites that are running on a single physical machine can be achieved through a variety of virtualization techniques, e.g., QEMU, VirtualBox, VMware, that deliver maximum isolation between nodes, but achieve this due to the intensive use of memory, hard drive, and processor capacity. Paravirtualization reduces the load on the I/O subsystem. Various container virtualization technologies, along with the namespaces, creates the illusion of running many nodes with different network characteristics between each of them on the same machine with a totally negligible overhead.

2. Network emulation. In addition to simulating a connection, the ability to model deterioration of network quality is needed, for example to emulate packet loss, latency, BER (the number of bits inverted with respect to the original) and so on.
3. In addition, a graphical interface is preferable, at least in the form of a web-client, because it allows us to not keep in mind the whole network topology, it is easy to change network settings and to visualize the results.

We have checked a number of tools that can be used for the purposes of simulating distributed computing infrastructure:

- User-mode Linux (UML) [15]: virtualization solution that allows you to run processes isolated from the rest of the operating system in the user space. In addition, it is possible to run network services. Important difference of UML from other solutions is that it allows you to run Linux kernel version different from the version on the host. Currently, this technology is integrated into the Linux kernel, but the development is slow, in addition, the performance is lower than other available solutions.
- Manage Large Networks (MLN) [12]: means of virtual machine management, working with Xen, VMware and User-mode Linux. In addition, the MLN was able to work with Amazon EC instances, but the development of the software was frozen in 2009.
- Marionnet [18]: software for network virtualization. It allows users to define, configure and run complex simulated networks. It is also possible to combine real and virtual networks. Unfortunately, Marionnet is also based on the User-mode Linux, so we cannot consider this technology new and evolving. Moreover, for the management of network properties it uses VDE (Virtual Distributed Ethernet), which does not support bandwidth limitation mechanisms, delays and other things.
- Integrated Multiprotocol Network Emulator / Simulator - IMUNES [3]: framework on the basis of today's popular products like Docker and Open VSwitch, which allows you to create nodes, connections between them, and fine-tune the network properties. It is possible to emulate the delay in data transmission, damage to transport packets, limit the bandwidth, etc. This product also has a graphical user interface that simplifies the creation of topology and access to running nodes.

5 Building and Evaluating Container-Based Distributed Computing Environment

We have implemented a prototype of container-based distributed computing infrastructure on the cloud resources provided by Microsoft Azure. To build the infrastructure we used a set of 8 virtual machines residing in different regions (5 machines in East US; 3 machines in North Europe) with the following characteristics (see Fig. 2):

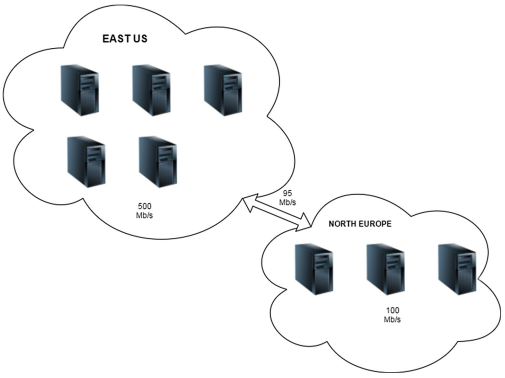

Fig. 2. Resources of experimental testbed in MS Azure

– Instance type: A1
– Cores: 1
– Memory: 1.75 GB

Current prototype implementation does not use any specific container management tools and relies only on Docker and a custom python-based toolkit developed to configure and execute containers for parallel applications (with OpenMPI deployed and configured) and control resource usage with help of a separately maintained database (see Fig. 3).

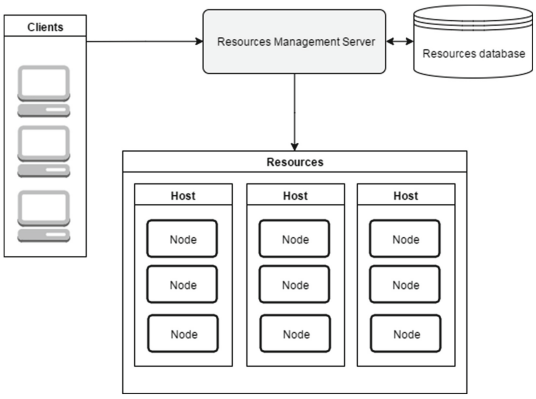

Fig. 3. Schematic view of the system

Figure 4 shows sequence diagrams illustrating the functionality of the system to manage resources and created container clusters.

We used several programs from NAS Parallel Benchmarks (NPB) suite as the applications with various requirements to the underlying infrastructure. NPB is

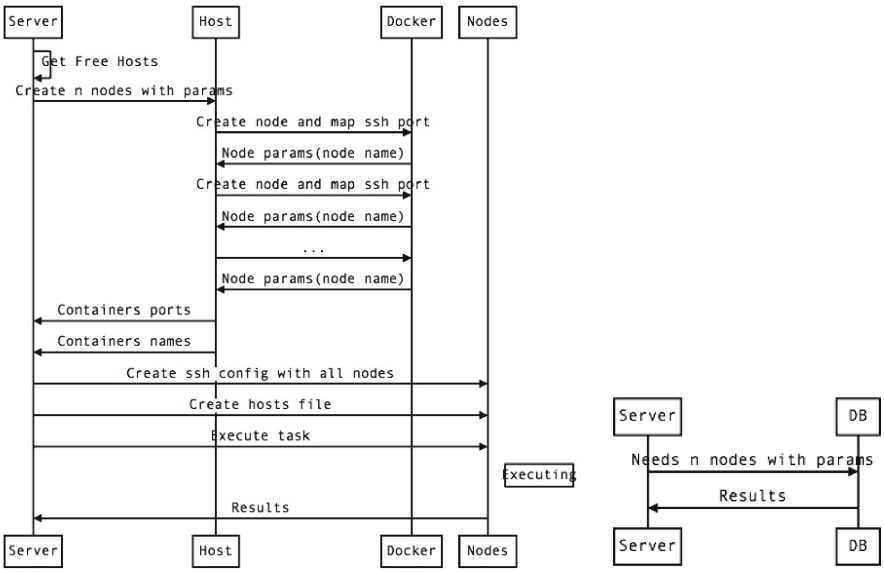

Fig. 4. Sequence diagrams of system's functionality (left: server-resources communication; right: server-database communication)

a small set of programs designed to help evaluate the performance of parallel computers and clusters. The benchmarks are derived from computational fluid dynamics (CFD) applications and consist of five kernels and three pseudo-applications. Moreover, the benchmark suite contains benchmarks for unstructured adaptive mesh, parallel I/O, multi-zone applications, and computational grids [17]. In our experiments we used MG, FT, and CG kernels:

- MG - Multi-Grid on a sequence of meshes, long- and short-distance communication, memory intensive
- FT - discrete 3D fast Fourier Transform, all-to-all communication
- CG - Conjugate Gradient, irregular memory access and communication

The aims of the experiments were the following:

- Investigate performance of parallel applications on the prototype of distributed container-based infrastructure
- Check how performance of applications with different requirements on cpu/memory/network varies depending on infrastructure configuration
- Evaluate possibilities of concurrent execution of parallel applications, minimizing their influence on each other

The first set of experiments was performed to evaluate the resource saturation point for an application: the point when adding more memory (or available cpu, or network bandwith) would not increase application performance anymore. Sample results are presented in Fig. 5. We can see that after some point the

performance of applications does not increase with increasing the amount of allocated resources per application. Namely, the left plot demonstrates that for the application (FT class S) the performance stops increasing after increasing available bandwidth between the nodes more than 900 Kbit/s; the right plot shows that the amount of available memory is crucial for the application to start (FT class A, the application does not start with less than 70 MB of memory available) but does not influence the performance when amount of memory is increased.

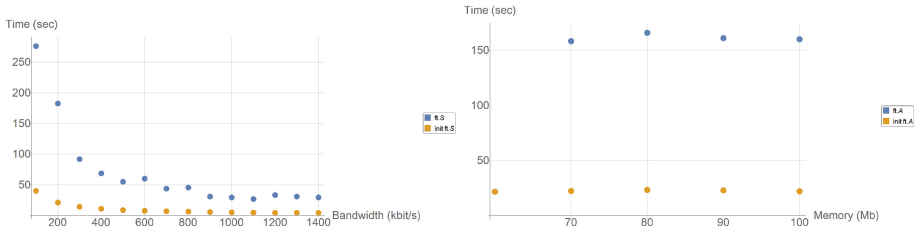

Fig. 5. Experimental results: saturation of resource requirements for FT kernel. (Color figure online)

The results presented in Fig. 6 illustrate details of application performance for a particular configuration of computing infrastructure.

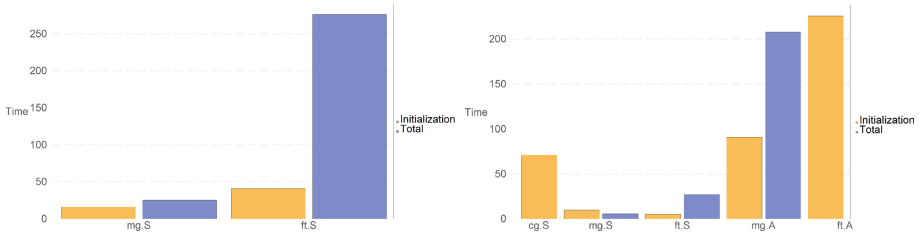

Fig. 6. Experimental results. Left: mem 256 MB, net 100 Kb/s; Right: mem 512 MB, net 1024 Kb/s. (Color figure online)

Next, we have evaluated sequential and concurrent execution of two different application kernels, MG and FT, to ensure that concurrent execution of both applications will not affect their performance in case container clusters are configured to meet the individual requirement of the applications. Figure 7 illustrates observed differences in initialization and benchmark time of MG and FT kernels in a particular virtual hardware configuration (512 MB memory, 120 Kbit/s network) for sequential and concurrent execution. Here sequential execution means allocation of the whole set of resources to each of the applications and executing

them one by one; concurrent execution means execution of both kernels simultaneously in separate containers with given limitations. Figure 8 shows experimental comparison of shared and concurrent execution, where shared execution means running both MG and FT kernels in a single container simultaneously. We can observe that in this case kernels can compete for shared resources which results in overall performance degradation.

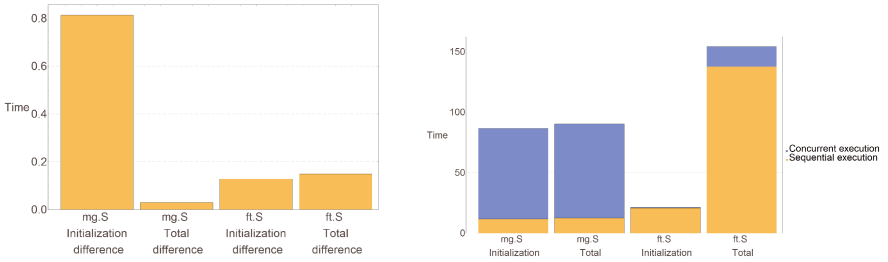

Fig. 7. Experimental evaluation of separate and concurrent execution: difference in init and benchmark time; mem 512 MB, net 120 Kb/s. (Color figure online)

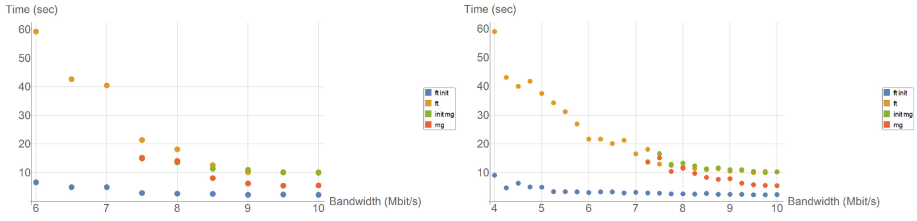

Fig. 8. Experimental comparison of shared (left) and concurrent (right) execution: mem 100 MB, lan 120 Kb/s. (Color figure online)

In order to confirm results gained on Amazon cluster, a bunch of tests were made on a local machine with IMUNES software. A star topology was generated with 8 nodes, all nodes were created from the same Dockerfile so they were completely identical. As IMUNES uses Docker and OpenVSwitch, network capacity was easy to configure. FT class A was used with different memory limitations but it wasn't possible to make test fail due to memory constraint as it was on a cluster. However the amount of memory available indeed doesn't influence the time when we allocate more than 80 mb per node. Overall, we cannot completely rely on results gained in a simulated testbed but we can at least understand how applications can scale (see Fig. 9).

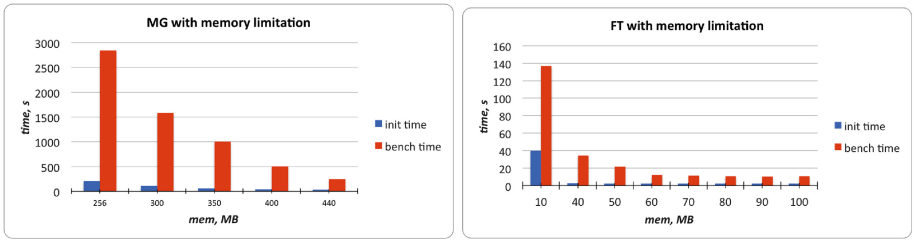

Fig. 9. Experimental results of simulation with IMUNES framework (Color figure online)

6 Discussion

The presented work continues the developments presented in [6]; this approach can be used as an enabling part of the virtual supercomputer concept [4,5] to ensure proper and efficient distribution of resources between several applications. Knowing the application demands in advance we can create appropriate infrastructure configuration giving just as much resources as needed to each particular instance of a virtual supercomputer running a particular application. Here we use containers as an enabling part of the computing infrastructure. In such a way, free resources can be controlled and granted to concurrent applications without negative effect on other executions.

7 Conclusions and Future Work

In this paper we proposed and evaluated usage of concurrently running container clusters that are created based on application requirements and have minimal effect on each other by resource allocation control. We demonstrated a proof-of-concept prototype running on Microsoft Azure cloud resources and showed experimental evaluation of its performance on a set of NAS benchmarks based on real application kernels.

Our future work will be to look more closely into container cluster management and orchestration software (e.g. Docker Swarm, Kubernetes, or Mesos) to delegate the functionality of maintaining the cluster to these tools so that we could concentrate on mechanisms of application requirements evaluation and concurrent execution of applications on distributed container resources.

Acknowledgments. The research was supported by Russian Foundation for Basic Research (projects N 16-07-01111, 16-07-00886, 16-07-01113) and St. Petersburg State University (project N 0.37.155.2014).

References

1. Lantz, B., Heller, B., McKeown, N.: A network in a laptop: rapid prototyping for software-defined networks. In: Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks, Hotnets-IX, NY, USA, 2010, pp. 19: 1–19: 6
2. Tennenhouse, D.L., Wetherall, D.J.: Towards an active network architecture. Comput. Commun. Rev. **26**(2), 5–18 (1996)
3. Salopek, D., Vasic, V., Zec, M., Mikuc, M., Vasarevic, M., Koncar, V.: A network testbed for commercial telecommunications product testing. In: Proceedings of the Softcom 2014 22th International Conference on Sotware, Telecommunications and Computer Networks, Split, September 2014
4. Gankevich, I., Gaiduchok, V., Gushchanskiy, D., Tipikin, Y., Korkhov, V., Degtyarev, A., Bogdanov, A., Zolotarev, V.: Virtual private supercomputer: Design and evaluation. CSIT 2013–9th International Conference on Computer Science and Information Technologies (CSIT), Revised Selected Papers. pp. 1–6 (2013). doi:10.1109/CSITechnol.2013.6710358
5. Gankevich, I., Korkhov, V., Balyan, S., Gaiduchok, V., Gushchanskiy, D., Tipikin, Y., Degtyarev, A., Bogdanov, A.: Constructing virtual private supercomputer using virtualization and cloud technologies. In: Murgante, B., Misra, S., Rocha, A.M.A.C., Torre, C., Rocha, J.G., Falcão, M.I., Taniar, D., Apduhan, B.O., Gervasi, O. (eds.) ICCSA 2014, Part VI. LNCS, vol. 8584, pp. 341–354. Springer, Heidelberg (2014)
6. Korkhov, V., Kobyshev, S., Krosheninnikov, A.: Flexible configuration of application-centric virtualized computing infrastructure. In: Gervasi, O., Murgante, B., Misra, S., Gavrilova, M.L., Rocha, A.M.A.C., Torre, C., Taniar, D., Apduhan, B.O. (eds.) ICCSA 2015. LNCS, vol. 9158, pp. 342–353. Springer, Heidelberg (2015)
7. NS-3 Project Homepage. http://www.nsnam.org
8. The Docker platform. https://www.docker.com
9. Apache Mesos. http://mesos.apache.org/
10. Mesosphere Marathon. https://mesosphere.github.io/marathon/
11. Kubernetes. http://kubernetes.io/
12. Manage Large Networks. http://mln.sourceforge.net/
13. CoreOS. https://coreos.com/
14. OMNeT++. https://omnetpp.org/
15. Dike, J.: User Mode Linux, p. 352. Prentice Hall, Englewood Cliffs (2006)
16. Open vSwitch. http://openvswitch.org/
17. NAS Parallel Benchmarks. http://www.nas.nasa.gov/publications/npb.html
18. Marionnet: a virtual network laboratory. http://www.marionnet.org