Comparison of Container Virtualization Tools for Utilization of Idle Supercomputer Resources*

Julia Dubenskaya[1[0000-0002-2437-4600]], Stanislav Polyakov[1[0000-0002-8429-8478]], Minh-Duc Nguyen[1[0000-0002-5003-3623]], and Elena Fedotova[1[0000-0001-6256-5873]]

[1] Skobeltsyn Institute of Nuclear Physics, Lomonosov Moscow State University, Moscow, Russia
jdubenskaya@gmail.com

Abstract. In this paper, we present the results of comparing container virtualization tools to solve the problem of using idle resources of a supercomputer. On average, as much as 10% of computational resources of a supercomputer may be underloaded due to various reasons. Our basic idea is to maintain an additional queue of low-priority non-parallel jobs that will run on idle resources until a regular job from the main queue of the supercomputer arrives. Upon arrival of the regular job, the low-priority jobs temporarily interrupt their execution and wait for the appearance of new idle nodes to be resumed there. This approach can be implemented by running low-priority jobs in containers and using the container migration mechanism to freeze these jobs and then run them from the point they were frozen at. Thus, the selection of a specific container virtualization tool that is best suited to our goal is an important task. Preliminary analysis allowed us to choose Docker and LXC software products. In this work, we make a detailed comparison of these tools and show why Docker is preferable for solving the above problem.

Keywords: Container virtualization, Supercomputer scheduling, Jobs execution, Docker, LXC.

1 Introduction

Supercomputer time is expensive and, ideally, it should perform calculations using all its resources on a non-stop basis. In fact, some of the resources of a supercomputer are often idle. The reasons are different, sometimes related to the fact that the main job queue is empty, and sometimes not: there may be restrictions on the policy of using the supercomputer (when there is a limit on the number of jobs from one user), or there may not be enough jobs, or some computational jobs are too demanding on resources and it is not possible to run them all in parallel, or the users give inaccurate estimates of execution time used for planning the schedule in advance, etc. As a result, for many supercomputers, the average load is approximately 90% and can be as

* The work was supported by the Russian Foundation for Basic Research grant 18-37-00502

low as 70% [1, 2]. So, our intention is to increase the load on a supercomputer forcing the use of its idle resources.

Our main idea is to create an additional queue of low-priority computational jobs that are not resource-intensive and not urgent. With such a queue we'll be able to load idle resources of a supercomputer with these jobs until regular jobs from the main queue appear. Upon arrival of a regular job, the low-priority jobs will interrupt their execution and wait for the appearance of new idle nodes to be resumed there.

There are two key requirements for low-priority jobs - they must be small and not urgent. Being small, they will perfectly fill the idle resources. Being not urgent, they can be interrupted at any time. A good example of this kind of jobs is the Monte Carlo simulation, which is required in many scientific experiments. These jobs are available in effectively infinite numbers and can be executed sequentially and independently, thus they can utilize any available idle resources. An example of this approach is the use of idle resources of the Titan-2 supercomputer by the ATLAS project [3].

The suggested approach can be implemented by running low-priority jobs in containers and using the container checkpoint/restore mechanism to freeze these jobs and then run them from the point they were frozen at. It should be emphasized that after restoring the process does not start from the beginning, but from the moment it was interrupted. Thus, it is possible to complete any low-priority job in several stages.

In this work, we compare different computer virtualization tools with support for checkpoints that can be used to run low-priority jobs on a supercomputer.

2 Technologies

The key product used in our work is Checkpoint/Restore In Userspace (CRIU) [4] - a software tool for the Linux operating system intended to safely and quickly interrupt and later resume a running process. CRIU provides the means to create a so called stateful checkpoint that is a collection of files containing all the information for restoring the process execution sometimes in the future. The process can be restored later, and restoring is possible on the same computational node or on another one. In our work, we intend to apply CRIU to a container with a running computational job.

The main goal of this work is to compare different container virtualization tools with CRIU support and to select one that is most suitable for our task. Currently, CRIU support is built into OpenVZ [5], Docker [6], and LXC [7] container virtualization systems. All of these systems use the same special tools of the Linux operating system to manage resource isolation - control groups and namespaces. Control groups (cgroups) is a Linux kernel mechanism that limits and isolates computing resources (processor, network, memory, I/O) of a collection of processes. Namespaces are a feature of the Linux kernel that partitions kernel resources (process IDs, hostnames, user IDs, file names) such that one set of processes sees one set of resources while another set of processes sees a different set of resources.

3 Candidate virtualization tools

Initially, we considered three container virtualization tools - OpenVZ, Docker, and LXC. However, a detailed analysis showed that the functionality implemented in OpenVZ is not quite the one that is required to solve the problems considered in our project. Namely, in OpenVZ, only the so-called "live migration" is implemented by standard means, which is the transfer of a working container from one computer to another over the network (that is, a checkpoint is created and immediately restored in another place). For the purposes of our project, it is important that you can create a checkpoint for a working container and store it for some time in the storage. We did not find how to implement this functionality with standard OpenVZ tools, so we opted for Docker and LXC projects, whose standard tools explicitly allow you to take a break between creating a checkpoint for a working container and restoring it.

When comparing the capabilities of Docker and LXC, it was noted that Docker containers start and stop somewhat faster than LXC containers, the launch of which is more like starting a classic virtual machine. At the same time, the LXC project has the best support for the ZFS file system, which significantly speeds up the process of writing checkpoints to disk, as well as restoring containers from checkpoints. The ZFS file system is primarily known for its ability to provide high speed access to data, this high speed is achieved due to the features of its implementation. LXC is specially optimized to work with ZFS, and without ZFS, LHC cannot compete with Docker, which is much more lightweight. Therefore, it was decided to test the capabilities of these virtualization tools in conjunction with the file system. Thus, we tested LXC together with ZFS, and Docker together with a regular file system (Ext4 in our case).

Hereinafter, for brevity, only the name of the container virtualization tool will be used, but the joint operation with the file system is implied.

4 The testbed

We compared virtualization tools on a testbed. We did not use a supercomputer, but the most ordinary computer; nevertheless it allowed us to draw certain conclusions about the capabilities and performance of the tools.

Computer specifications are as follows.

A processor with two cores, the characteristics of each are:

product: Intel(R) Xeon(R) CPU E5620 @ 2.40 GHz
capacity: 2401 MHz
width: 64 bits

Memory:

size: 15 GB

Also we prepared a special test computational job. This job was launched, checkpointed and restored using containers of each type (respectively, Docker and LXC) 20 times to collect statistics. The job was specially implemented as non-optimally as possible, which made it almost endless. Thus, sudden container stops due to the com-

pletion of the job do not affect the test results. According to the top command, the job consumed 1.3 to 1.6 GB of memory, while the processor was 100% loaded.

5 Test results

When comparing the capabilities of Docker and LXC, we started from the comparison of the container launch. The comparison results are presented in **Ошибка! Неверная ссылка закладки.**.

Table 1. Comparison of container create/start times in Docker and LXC.

	Docker	LXC	
	create/start a container	create a container	start a container
Time, sec	1.5 ± 0.2	14 ± 2	1.5 ± 0.2

A feature of Docker is that you can immediately start a container without creating it separately. The launch of LXC container is more complicated: you should create a container and then it can be started (at this particular moment or later). So, the launch takes longer and includes an additional step. However, an acceptable workaround is to create a number of LXC containers in advance.

The next step was to compare the times to checkpoint/restore. These results are presented in **Ошибка! Неверная ссылка закладки.**.

Table 2. Comparison of container checkpoint/restore times in Docker and LXC.

	Docker		LXC	
	checkpoint	restore	checkpoint	restore
Time, sec	70 ± 3	62 ± 3	13 ± 2	5.5 ± 0.5

As we can see, creating an LXC checkpoint is almost 5 times faster, and restoring from the checkpoint is more than 10 times faster (comparing to Docker).

Surprisingly, a more thorough study of LXC revealed a recurring problem: the same container was correctly restored from a checkpoint only once. Attempts to checkpoint a container that was previously restored from a checkpoint resulted in an error with the loss of the container state. Most likely, this behavior is the result of the features of the ZFS file system, and not the disadvantage of the LXC tool itself. However, since our project assumes a multiple checkpoint and restore of the same container as the main scenario, the above feature prevents us from using LXC with ZFS for the needs of our project.

Thus, we opted for the Docker project, which stably and correctly checkpoints/restores any container multiple times, while maintaining the current state of the processes running there.

6 Conclusion

We implemented a prototype system to increase the effective load of supercomputer resources using Docker containers [8]. Our simulation experiments demonstrate that under some assumptions the system increases the effective utilization of computational resources. So, testing of the prototype proved the viability of the proposed approach, as well as the reliability and stability of the checkpoint/restore mechanism in Docker containers.

Also, we believe that the results obtained can be of practical use to other researchers when choosing a container virtualization tool for their needs.

References

1. Antonov, A.S. et al.: Examination of supercomputer system jobs flow dynamic characteristics. Computational methods and programming, 14(4), 104-108 (2013) (in Russian).
2. Leonenkov, S., Zhumatiy, S.: Supercomputer Efficiency: Complex Approach Inspired by Lomonosov-2 History Evaluation. In: Russian Supercomputing Days: Proceedings of the International Conference (September 24-25, 2018, Moscow, Russia), pp. 518-528. Moscow State University (2018).
3. Barreiro Megino, F. et al. [ATLAS Collaboration]: Integration of Titan supercomputer at OLCF with ATLAS Production System. Journal of Physics: Conference Series, 898(9) p. 092002 (2017).
4. The CRIU project, https://www.criu.org/, last accessed 2020/06/30.
5. The OpenVZ project, https://openvz.org/, last accessed 2020/06/30.
6. The Docker project, https://www.docker.com/, last accessed 2020/06/30.
7. The LXC project, https://linuxcontainers.org/, last accessed 2020/06/30
8. Polyakov, S.P., Dubenskaya, Yu.Yu.: Improving the load of supercomputers based on job migration using container virtualization. In: Selected Papers of the 8th International Conference "Distributed Computing and Grid-technologies in Science and Education", CEUR Workshop Proceedings, vol. 2267, pp. 243-247. M. Jeusfeld c/o Redaktion Sun SITE, Informatik V, RWTH Aachen (2018).