Nikita Kuznetsov, MSc-1, ACS

Nikita.Kuznetsov@skoltech.ru

Essay (Ex. 3)

High-performance computing is nowadays widely used in different scientific and industrial applications. Starting from modeling complex systems like galaxies or molecules and concluding with training GPT-3 model for text processing. All these tasks use the cluster to perform needed calculations. But it all won't be possible without complicated and carefully organized systems as supercomputers. Since they are pretty powerful machines, a lot of people want to use them for one's purposes, and to provide fair facilities for everyone; it is needed to use specific tools and rules for operating the cluster,

The key point that plays a valid role in computer science and, on the other hand, makes many programmers suffer is memory. And while using high-performance clusters, anyone definitely would face such a problem. To simplify the procedures, there is a concept as SLUM, which states for Simple Linux Utility for Resource Management. This software help with scheduling tasks for Linux and Unix-like kernels. The essential functions are allocating exclusive or non-exclusive access to computer nodes to users for some time; providing a framework for MPI programs on allocated nodes; and managing a queue of pending jobs.

One of the ways to use SLURM is to set up job arrays for the task being performed on a cluster. Job array itself is a group of identical jobs with the same time, memory, and CPU allocations. It offers a mechanism for submitting these similar jobs much faster because it saves SLURM a lot of work which will reuse previous computations for scheduling. It is important to note that the job array is supported for batch jobs. Therefore, it can be used only with *sbatch* command but not with *salloc*. Besides the fact that it is possible to form a batch of tasks, it is also possible to provide an order in this batch, called job dependencies. It tells SLURM not to start a job until another job has begun or successfully or unsuccessfully completed.

But practice is usually not as easy as it may look on paper. For example, most high-performance processors have a cache buffer between slow memory and high-speed registers. Accessing a memory location causes a slice of actual memory, and after that memory location is requested to be copied into the cache. However, simultaneous updates of individual elements in the same cache line coming from different processors may disable the whole cache lines, even though these updates are logically independent of each other. Other processors accessing a different element in the same line see that the line is invalid. Because of it, they are forced to fetch a more recent copy of the line from memory or elsewhere, even though the element accessed has not been modified. As a result, there will be an increase in interconnect traffic and overhead. Also, while the cache-line update is in progress, access to the elements in the line is inhibited.

This situation is called false sharing. If this occurs frequently, the performance and scalability of the application will dramatically decrease. False sharing degrades performance in such cases: shared data is modified by multiple processors; multiple processors update data within the same cache line, and updates occur frequently. But it is important to note that such a problem won't appear if shared data is in read-only mode. In general, false sharing can be reduced by using private data as much as possible. But mainly, techniques for solving false sharing problems depend on the particular application. For example, sometimes change in the data allocation procedure can help.

Another way to operate programs on the high-performance cluster is to use virtual machines or containers. Speaking about the virtual machine, they can be separated into two types by hypervisors. The hypervisor itself is a kind of emulator that creates and runs virtual machines presenting the guest operating system with a virtual operating platform and manages the execution of the guest operating system.

Getting back to types of virtual machines. Type 1 can be described technically as an operating system because it is the only program that runs in privileged mode. Such hypervisor supports multiple copies of the actual hardware, similar to a standard operating system's processes. Type 2 is different. It is a program that relies on some operating system to allocate and schedule resources, like a normal process. Both types of hypervisors must execute the machine's instruction set safely. In both cases, the operating system running on top of the hypervisor is called the guest operating system. For a type 2 hypervisor, the operating system running on the hardware is called the host operating system. The most famous type 2 hypervisor on the market was VMware Workstation.

In the beginning, the containers were mentioned. And one of the popular ways to work with containers is to use Docker. Docker is an application to run an isolated environment. It originally used LinuX Containers (LXC) but later switched to runC, which runs in the same operating system as its host. This allows it to share a lot of the host operating system resources. Also, Docker uses a layered filesystem and manages networking. There is a particular advantage of this approach: the app runs in the same environment; simplifies work with others projects; no overhead that provided by a virtual machine. From the previous paragraph, it is possible to conclude that Docker is a type 2 virtual machine. But to name its virtual machine, it should satisfy specific requirements:

1. Safety. The hypervisor should have full control of the virtualized sources.
2. Fidelity. The behavior of a program on the virtual machine should be identical to that of the same program running on bare hardware.
3. Efficiency. Much of the code in the virtual machine should run without intervention by the hypervisor.

From these points, it is easy to see that Docker does not fulfill safety conditions as it must do virtual machines. Virtual machines are stand-alone with their kernel and security features. Therefore, applications needing more privileges and security run on virtual machines. But providing root access to applications and running them with administrative premises is not recommended in the case of Docker containers because containers share the host kernel. Since container technology has access to the kernel subsystems, it results in a single infected application being capable of hacking the entire host system. Due to these security reasons, Zhores runs only Singularity containers. Singularity obeys security restrictions with immutable images and does not provide user and process namespace isolation.

Container and especially Docker is widely used in the world of programming. Since my work is mainly connected with machine learning and data science, then Docker files can be helpful to run the trained model on different devices. It may be beneficial because such a model may be sensitive for library versions, and libraries may be implemented for specific OS (like JAX for Linux). Therefore, transferring and delivering results for other researchers passing Docker containers to run seems to be a good idea.