Containers

Pete Beckman (*PI*), Kamil Iskra, Florence Monna, Swann Perarnau, Valentin Reis

Argo Project

The goal of Argo is to augment and optimize existing OS/R components for use in production HPC systems, providing portable, open source, integrated software that improves the performance and scalability of and that offers increased functionality to exascale applications and runtime systems.

For an overview of Argo, please refer to `https://www.argo-osr.org/`.

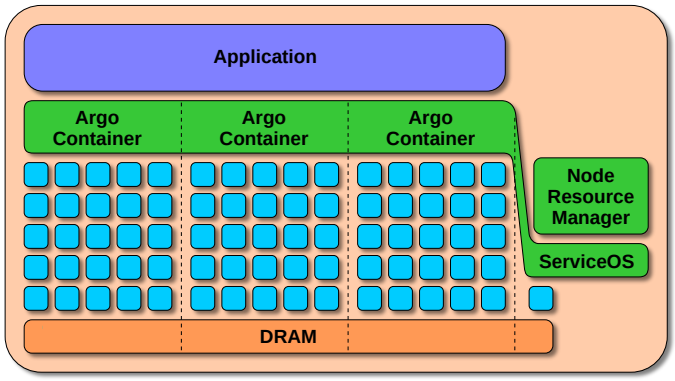
Argo Containers

Argo uses containers for *resource management*. Physical resources on the compute nodes are divided into separate partitions. Containers can be used to *separate individual components* of parallel workloads, in addition to cordoning off system services; this physical separation ensures an *improved performance isolation* between components.

Our containers can currently manage the following:

- CPU cores (hardware threads)
- Memory (including physical memory at sub-NUMA granularity with a patched Linux kernel)
- Kernel task scheduling class

The physical resources are partitioned primarily by using the `cgroups` mechanism of the Linux kernel. Work is under way to extend the management to I/O bandwidth as well as to the partitioning of last-level CPU cache using Intel's Cache Allocation Technology.



Argo containers divide node CPU (blue squares) and memory (orange bar) resources between three application components, with an additional partition left aside for system services.

Containers are managed by the *Node Resource Manager* (NRM), which maps them to the node's hardware topology, arbitrating resources between applications and runtime services. NRM also provides integration with the power management. Resources can be dynamically reconfigured at run time; interfaces are provided for use from applications and from global services.

Argo containers are meant to be transparent to applications; in particular, they have been tested to work with MPI codes. They do not impede communication between application components, since they do not utilize namespaces; they are, however, compatible with (and complementary to) other namespaces-based container runtimes such as Docker, Singularity, or Shifter.

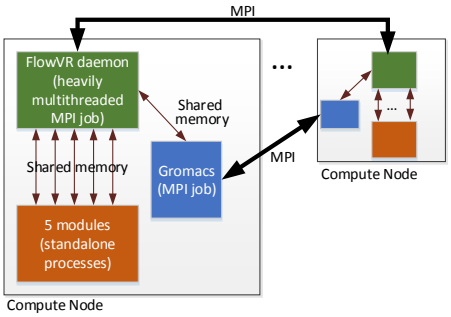
Impact

Partitioning node resources can be particularly important for complex workloads consisting of multiple heterogeneous components, for example, coupled codes, simulations running in parallel with in situ an-

alytics, and workflows. If a compute node is shared among multiple workload components (a likely scenario considering the reduced cost of data transfers), these components—if allowed to freely share node resources—could interfere with one another, resulting in suboptimal performance.

Testcases

The figure on the right depicts a complex application workflow we used during an early development of Argo containers. The simulation component is *Gromacs*, a molecular dynamics simulation package from the biology community. It is a parallel, multinode MPI application. The visualization component performs isosurface extraction. It is a pipeline of five sequential processes on each node. The coupling is managed by *FlowVR*, in situ middleware designed for building asynchronous workflows.

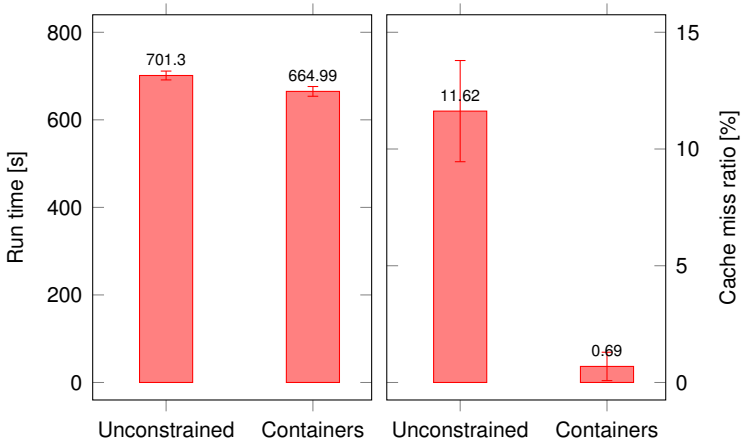


Process interactions in a workflow consisting of a simulation (blue) and an in situ visualization (orange).

Correct placement of application processes on the node is critical to obtaining optimal performance, since the Gromacs processes are highly sensitive to perturbations. With Argo containers, we were able to obtain the best performance by putting Gromacs and the visualization components in two separate containers and by allowing FlowVR threads to run unconstrained. This performance was comparable to that achieved by using manual placement by a human expert.

We are collaborating with PETSc developers on improving the performance of their *libEnsemble* workload, which uses co-scheduling of small (but still parallel) application runs to increase throughput of an ensemble simulation. Argo containers are used to provide resource partitioning and performance isolation to the co-scheduling infrastructure.

The closer figure on the right shows the result of a simple experiment where Argo containers are used to split a two-socket Intel Xeon Gold 6126 node into two containers, one per socket, to co-schedule libEnsemble MPI jobs. We can see that using the containers improves the execution time. We attribute the improvement primarily to the reduced cache miss ratio, as shown in the figure farther to the right.



Status

Our software is open source and can be found at `https://xgitlab.cels.anl.gov/argo` (spread across the `containers`, `nrm`, and `cuttr` repositories). It is under active development, and we are looking for additional collaborators with more workloads. If you think that your project could benefit from running under Argo containers, we encourage you to get in touch with us.

Contact

For more information, please contact Swann Perarnau `swann@anl.gov` or Kamil Iskra `iskra@mcs.anl.gov`.