Scalable Launch Working Group Outbrief

Software Tools Workshop
August 14, 2024

Participants

- John Mellor-Crummey, Rice University
- Jonathon Anderson, Rice University
- Ben Griego, Sandia National Laboratory
- Matt Legendre, Lawrence Livermore National Laboratory
- Todd Gamblin, Lawrence Livermore National Laboratory

The Problem

- Applications for modern supercomputers are all dynamically-linked
    - Libraries supplied by GPU vendors are all dynamically-linked
- Launching dynamically-linked applications on supercomputers is problematic
    - Each of thousands of MPI ranks will try to look up each of their shared libraries
    - Today, shared libraries built with Spack often have many RPATH entries
    - To find a library, a process will look for the library in each directory in an RPATH
        - This turns into a thundering herd of processes pounding on the file system

Possible Space of Solutions

- Spack prebinding
- Shrinkwrap
  - Uses full paths in DT_NEEDED, which prevents the search storm that arises with long RPATH
- Spindle
  - Dynamically caches and multicasts dependence to processes that need them
- Simpler volume mounts

Shrinkwrap Experience

- Didn’t work in practice on Frontier
  - ROCm and MPI libraries loaded in an order with an HSA-related error message
  - Spindle also ran into a similar issue on El Cap (in preliminary testing)
  - May be some improvements in Spack’s link_type:bind – needs further testing

Spack Approaches

- You can use config:shared_linking:bind: to pre-resolve all DT_NEEDEDs in your programs
  - Spack keeps RPATHs in case they are needed for dlopens
  - ld.so should load all libraries directly with one open() instead of searching lots of RPATHs
  - https://spack.readthedocs.io/en/latest/config_yaml.html#shared-linking-bind
- Shrinkwrap does this a bit differently:
  - Pre-resolves absolute paths for DT_NEEDEDs of lib/executable and all transitive dependencies
  - Puts DT_NEEDEDs *on the root executable/library* for each pre-resolved absolute path
- Spack is only doing this for DT_NEEDEDs on each library
  - Doesn’t (yet) give you control over lib races in transitive dependencies
  - Intended use case for Shrinkwrap was incompatible openmp libraries
  - e.g., be sure to load Cray OpenMP instead of GOMP for programs that use both b/c Cray has both sets of bindings

Simpler Volume Mounts

- Much of the overhead of loading is shared filesystem metadata
- Don’t really need a shared filesystem for executable binaries
- LLNL uses iSCSI block volume mounts for TOSS image on compute nodes
  - OS image gets mounted read-only over iSCSI
  - *Seems* to be relatively fast
  - Boots happen less frequently than job launches
- Most clouds come with some service like this
  - EBS, Google Block, etc.
  - Default volume for instances is some block device – you pick the volume when you deploy the instance
- LLNL developing user-level block service that is similar
  - Can make a file in Lustre and mount it to compute nodes as a loopback device
    - Performance is comparable to local NVME (slighlty higher latency)
  - Block devices are not safe for regular users (can mount in bad places, override kernel)
  - Need to do this with some controlled setuid program
  - Hobbes at LC has developed SLURM and Flux plugins for ephemeral volumes
    - Makes a Lustre volume
    - Mounts it with user permissions in known location when job starts
    - Destroys it at job end
  - Testing this out at LC right now, likely to be used for CI
  - TODO:
    - persistent volumes with names that live beyond one job
    - Sets of volumes for, e.g., file-per-process checkpoints or large read-only data sets
- CSCS squashFS mounted environments
  - https://cug.org/proceedings/cug2023_proceedings/includes/files/pap143s2-file1.pdf

Action Items

- We need to identify what problems arise with library loading that causes errors when using Spack prebinding
- Investigate auditor composition
  - Spindle auditor + HPCToolkit auditor
  - Can spindle
- Jonathon needs LLNL credentials for joint work with Matt and Todd on El Cap TDS
- Productize Spindle
  - Revisit building Spindle with Spack
    - Can the right auto-detection be done in the context of a Spack build?
    - Can’t detect
      - file systems on the back end
      - Slurm policies
    - Need resource manager plugins

Story time

- HPCToolkit ran into a Glibc bug that affects TLS
  - Running OpenMP + ROCProfiler, TLS just gets wiped between things happening
  - Triggered on a dlopen() of an already-loaded library
  - Potentially could affect the Spack prebinding approach