Benchmark informed software upgrades at Quest, Northwestern's HPC cluster

Sajid Ali
Applied Physics, Northwestern University
Evanston, Illinois
sajidsyed2021@u.northwestern.edu

Alex Mamach
NUIT, Northwestern University
Evanston, Illinois
alex.mamach@northwestern.edu

Alper Kinaci
NUIT, Northwestern University
Evanston, Illinois
akinaci@northwestern.edu

ABSTRACT

We present the work performed at Quest, a high performance computing cluster at Northwestern University regarding benchmarking of software performed to guide software upgrades. We performed extensive evaluation of all MPI modules present on the system for functionality and performance in addition to testing a strategy to deploy architecture optimized software that can be loaded dynamically at runtime.

CCS CONCEPTS

• **Software and its engineering → Software configuration management and version control systems**; *Software maintenance tools.*

KEYWORDS

software management, software builds, software automation

ACM Reference Format:
Sajid Ali, Alex Mamach, and Alper Kinaci. 2020. Benchmark informed software upgrades at Quest, Northwestern's HPC cluster. In *Proceedings of PEARC '20.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Quest is a heterogeneous HPC cluster[4] at Northwestern University consisting of Intel Haswell/Broadwell/Skylake/Cascade Lake nodes with varying interconnects which recently transitioned to SLURM[24] as the resource manager and the job scheduler. The cluster operates with very high uptimes and generally shuts down for a week once every academic year for maintenance. While this high uptime is great for research throughput, it compresses critical maintenance tasks into that week and makes the operators prioritize in place upgrades over major redesigns. While such an operations scheme works in the short run, managing a large set of software stacks that were installed at various points in time becomes challenging since the software stack was kept stable even through downtime cycles that involved major and minor OS upgrades.

This has led to a bloated software stack and sometimes inconsistencies in naming schemes for modules and executables. This is challenging to continuously benchmark for functionality and performance. Thus, we are motivated to develop a strategy to maintain our software stacks that will enable us to provide functional and efficient software for our users while reducing the maintenance and support workload for the operators and software specialists. In addition to the above, we also face an immediate need to make MPI launchers compatible with srun as a SLRUM update is on the agenda for next downtime (as part of the MPI upgrade project).

In this article, we present our ongoing project to modernize MPI installations and the results of benchmarking studies that inform our plans for deprecating modules. We also discuss the preliminary tests on our beta cluster for a strategy to deploy optimized builds for each architecture that are dynamically loaded at runtime based upon the node list for the job.

2 MPI LIBRARIES

Over the 10-year life of Quest cluster, multiple versions of libraries proving functionality specified by the message passing interface standard [16, 17] were installed. Some of these are quite old (before major OS version upgrades) and the naming scheme for the corresponding module files is inconsistent. Thus, we developed the following strategy for an upgrade project with the following milestones (in chronological order) :

- Deployment of a beta cluster with SLURM 19 (later updated to SLURM 20)
- Benchmark existing MPI libraries on the beta cluster
- Benchmark new MPI libraries on beta cluster
- Deployment of updated middleware (UCX, PMIx) on the production cluster
- Communicate module deprecation and upgrade strategy to users
- User tests using recompiled or new MPI applications on beta cluste
- MPI modules transition along with SLURM update on production cluster

During the submission of this report, we are working on a user communication plan. Consequently we will primarily focus on tasks prior this milestone.

2.1 Benchmarking

To test these libraries for functionality and performance, we compiled and executed two point-to-point benchmarks, bandwidth and latency from the OSU micro-benchmarks suite [6]. The MPI libraries

PEARC '20, July 26-30, 2020, Portland, OR

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

were installed without SLURM support and had incostintent naming schemes necessitating the use of bash scripts to execute the tests. The results are presented in the figure 1.

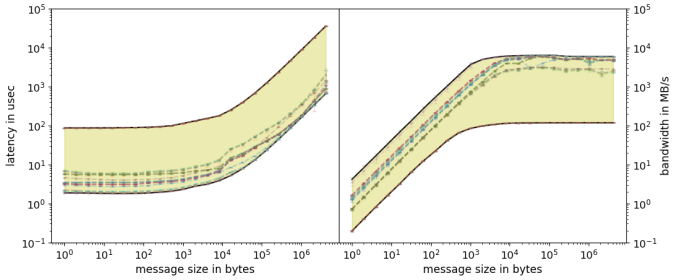

Figure 1: Band of performance metrics measured for the currently available MPI builds indicating a large spread in performance. Some builds fail to use the InfiniBand fabric while others had sub-optimal configurations. The list of libraries tested is listed in section A.1

In total, 42% of the available MPI libraries were faulty with 28% being nonfunctional (failure to compile or run) and 14% being non-performant.

2.2 Improvements

Spack[18] was used to build new versions of MPI libraries with SLURM support. This allows us to automate a large set of parameterized builds and eases the testing. Alongside testing the new installations for srun launcher support, we also tested the relative performance of the UCX transport layer [12, 23]. Unified Communication X (abbreviated as UCX) is a portable, high performance middleware that sits between programming models (like MPI, PGAS, charm++, etc) and network device drivers. The results of benchmarks with new installations are presented in the figure 2.

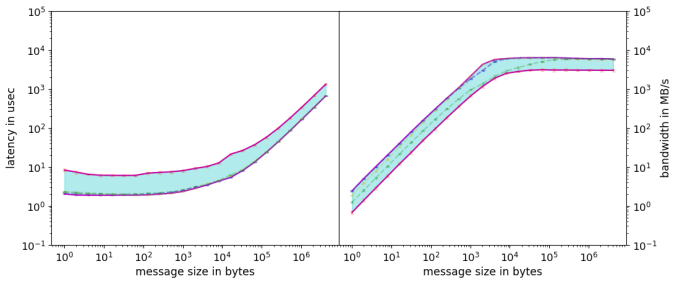

Figure 2: Band of performance metrics measured for the new MPI builds with optimal configuration and updated middleware indicating a reduced spread in performance with newer middleware providing some gains. The list of libraries tested is listed in section A.2

As indicated in a recent OpenMPI deployment guide [22], newer versions of the libraries configured with UCX transport layer perform better. We have thus decided to use the UCX transport layer for all MPI libraries. In addition to this, we also plan to enable the PMIx plugin [20, 21] in SLURM to use the PMIx process management standard [7, 15] (implemented via the OpenPMIx library [5]) which improves the job startup time. We also note that the SLURM plugin for PMIx allows it optionally use UCX for communication via a SLURM plugin for UCX.

2.3 Deploying UCX and PMIx

While UCX was installed on the GPFS filesystem for convenience of testing, we plan to install it on the node-local filesystem (specifically at */usr/local*) of each compute node given its nature as a widely used runtime dependency for MPI libraries. We already had an older version of UCX, 1.4.0 available (as part of the Mellanox driver installations) and used this alongside an installation of OpenPMIx [5], version 2.2.3 for testing. Unfortunately, due to a bug in the SLURM plugin [9], this configuration led to the job start phase crashing. The bug fix involves either installing UCX with the rdma-core [8](which provide the user space components for the IB drivers) dependency or update the drivers to a newer version.

Since there are no plans on updating the InfiniBand drivers, we first attempted at manually creating binaries (in the *.rpm* format using *rpm – builder* tool) for UCX and PMIx that overcome the aforementioned bug. We had no success with this approach as we were unable to properly patch the libraries. Thus, we used Spack [18] to install the libraries to a common prefix (by using a filesystem view in an environment) and create an rpm binary from this for easy deployment on the compute nodes.

3 NODE ARCH DEPENDENT SOFTWARE

Given the challenges in maintaining a complex software stack that includes a multitude of combinations between applications, versions, compilers and dependencies, achieving optimal software performance (as available via generating optimized binaries for each architecture) was not prioritized. While this was not a major concern in the past, increase in the CPU-architecture heterogeneity of the cluster in the recent years, such a deployment strategy severely degrades productivity. Thankfully, due to recent developments in the Spack package manager, we are able to build multiple versions of each library, each optimized for a different architecture for optimal productivity of the cluster.

3.1 Benchmarks

We benchmarked optimized builds for two of our most commonly used applications, LAMMPS[19] and GROMACS [2, 3] against the currently available installations on the oldest processor generation, "Haswell". We chose the Lennard-Jones liquid benchmark for LAMMPS available as part of the official benchmark suite [1] and a benchmark from the Unified European Applications Benchmark Suite [13, 14] for GROMACS. The results of these tests are presented in the Table 1 which shows that there are substantial benefits to deploying node architecture specific software even on our oldest nodes, with speedups of 32.3% 12.4% observed for LAMMPS and GROMACS applications.

3.2 Deployment Strategy

On Quest, users are not required to choose a partition and the jobs are assigned to nodes dynamically based on availability. Thus, we

Table 1: Comparision of performance between currently available and architecture optimized builds of commonly used applications on Haswell nodes

Software	Current	Optimized
LAMMPS	765.8 timesteps/day	1013.4 timesteps/day
GROMACS	1.78 ns/day	2.00 ns/day

are faced with the following deployment challenge : how do we deploy optimized builds for each processor family but not require our users to choose the exact build of the application for their job ?

To answer the above, we have chosen a simple strategy where we configure a task prolog script for SLURM that automatically sets the modulepath based on *SLURM_NODELIST* environment variable. We tested this on a virtual SLURM cluster with two nodes on a laptop provisioned by four docker containers tied together via docker-compose using an existing repository [10] developed by SciDAS (Scientific Data Analysis At Scale). This slurm cluster has two compute nodes named *worker*01 and *worker*02. We present the sample task prolog script below [1] :

```
short_list=${SLURM_JOB_NODELIST##worker}
if [ $short_list == "01" ]
then
echo "export MODULEPATH=/home/path1"
fi
if [ $short_list == "02" ]
then
echo "export MODULEPATH=/home/path2"
fi
```

4 CONCLUSION

We have effectively used benchmarking tests that gives us valuable insight regarding the health of the software on the Northwestern-Quest high performance compute cluster. This work will inform our plans to deprecate and eventually remove non-functional and non-performant software. Moreover, executing the node-optimized software strategy will significantly improve the productivity of the cluster.

ACKNOWLEDGMENTS

To various mailing lists, slack channels and forums including but not limited to mpich-discuss, slurm-info, spack-users.

REFERENCES

[1] [n.d.]. *LAMMPS Benchmarks.* Retrieved April 14, 2020 from https://lammps.sandia.gov/bench.html
[2] 1995. GROMACS: A message-passing parallel molecular dynamics implementation. *Computer Physics Communications* 91, 1 (1995), 43 – 56. https://doi.org/10.1016/0010-4655(95)00042-E
[3] 2015. GROMACS: High performance molecular simulations through multi-level parallelism from laptops to supercomputers. *SoftwareX* 1-2 (2015), 19 – 25. https://doi.org/10.1016/j.softx.2015.06.001
[4] 2019. *NUiT Quest.* Retrieved April 19, 2020 from https://www.it.northwestern.edu/research/user-services/quest/specs.html
[5] 2020. *OpenPMIx repository.* Retrieved April 19, 2020 from https://github.com/openpmix/openpmix
[6] 2020. *OSU Micro Benchmarks website.* Retrieved April 19, 2020 from http://mvapich.cse.ohio-state.edu/benchmarks/
[7] 2020. *PMIx webpage.* Retrieved April 19, 2020 from https://pmix.org/
[8] 2020. *rdma-core repository.* Retrieved April 19, 2020 from https://github.com/linux-rdma/rdma-core
[9] 2020. *Slurm Bug Report.* Retrieved April 19, 2020 from https://bugs.schedmd.com/show_bug.cgi?id=7646
[10] 2020. *Slurm in Docker - Exploring Slurm using CentOS 7 based Docker images.* Retrieved April 19, 2020 from https://github.com/SciDAS/slurm-in-docker
[11] 2020. *Slurm TaskProlog.* Retrieved April 29, 2020 from https://slurm.schedmd.com/faq.html#task_prolog
[12] 2020. The Unified Communication X Library. Retrieved April 19, 2020 from https://www.openucx.org/
[13] 2020. *Unified European Applications Benchmark Suite.* Retrieved April 19, 2020 from https://prolinktrials.co.uk/prace/training-support/technical-documentation/benchmark-suites/
[14] 2020. *Unified European Applications Benchmark Suite.* Retrieved April 19, 2020 from https://repository.prace-ri.eu/git/UEABS/ueabs
[15] Ralph H. Castain, Joshua Hursey, Aurelien Bouteiller, and David Solt. 2018. PMIx: Process management for exascale environments. *Parallel Comput.* 79 (2018), 9 – 29. https://doi.org/10.1016/j.parco.2018.08.002
[16] Message Passing Interface Forum. 2009. *MPI: A Message Passing Interface Standard, Version 2.2.* High-Performance Computing Center Stuttgart, University of Stuttgart. https://fs.hlrs.de/projects/par/mpi//mpi22/
[17] Message Passing Interface Forum. 2015. *MPI: A Message-passing Interface Standard, Version 3.1.* High-Performance Computing Center Stuttgart, University of Stuttgart. https://fs.hlrs.de/projects/par/mpi//mpi31/
[18] T. Gamblin, M. LeGendre, M. R. Collette, G. L. Lee, A. Moody, B. R. de Supinski, and S. Futral. 2015. The Spack package manager: bringing order to HPC software chaos. In *SC15: International Conference for High-Performance Computing, Networking, Storage and Analysis.* IEEE Computer Society, Los Alamitos, CA, USA, 1–12. https://doi.org/10.1145/2807591.2807623
[19] Steve Plimpton. 1995. Fast Parallel Algorithms for Short-Range Molecular Dynamics. *J. Comput. Phys.* 117, 1 (March 1995), 1–19. https://doi.org/10.1006/jcph.1995.1039
[20] Artem Y. Polyakov, Boris I. Karasev, Joshua Hursey, Joshua Ladd, Mikhail Brinskii, and Elena Shipunova. 2019. A Performance Analysis and Optimization of PMIx-Based HPC Software Stacks. In *Proceedings of the 26th European MPI Users' Group Meeting (EuroMPI '19).* Association for Computing Machinery, New York, NY, USA, Article Article 9, 10 pages. https://doi.org/10.1145/3343211.3343220
[21] Artem Y Polyakov, Joshua S Ladd, and Boris I Karasev. 2017. Towards Exascale: Leveraging InfiniBand to accelerate the performance and scalability of Slurm jobstart. (2017).
[22] Howard Porter Jr. Pritchard, Thomas Naughton, and George Bosilca. 2020. Getting It Right with Open MPI: Best Practices for Deployment and Tuning of Open MPI. (2 2020).
[23] Pavel Shamis, Manjunath Gorentla Venkata, M Graham Lopez, Matthew B Baker, Oscar Hernandez, Yossi Itigin, Mike Dubman, Gilad Shainer, Richard L Graham, Liran Liss, et al. 2015. UCX: an open source framework for HPC network APIs and beyond. In *2015 IEEE 23rd Annual Symposium on High-Performance Interconnects.* IEEE, 40–43.
[24] Andy B. Yoo, Morris A. Jette, and Mark Grondona. 2003. SLURM: Simple Linux Utility for Resource Management. In *Job Scheduling Strategies for Parallel Processing*, Dror Feitelson, Larry Rudolph, and Uwe Schwiegelshohn (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 44–60.

A BUILD DETAILS

A.1 Currently installed MPI libraries

All the libraries available to the users were first built at a staging directory for final installation at destination directory followed by modulefile creation. These include :

- intel-mpi-{4.0.1,4.0.3,5.1.3.258}
- mpich-{3.0.4-gcc-4.6.3, 3.0.4-gcc-4.8.3, 3.0.4-gcc-5.1.0, 3.0.4-gcc-6.4.0, 3.0.4-intel2013.2, 3.0.4-intel2015.0,3.3-gcc-6.4.0}
- mvapich2-{gcc-4.8.3, gcc-4.8.3-cuda8, intel2013.2}
- openmpi-1.6.3-{gcc-4.6.3, gcc-4.8.3, intel2011.3, intel2013.2}
- openmpi-1.6.5-{gcc-4.6.3, gcc-4.8.3, intel2013.2}
- openmpi-1.7.2-{gcc-4.6.3, intel2013.2}
- openmpi-1.8.1-{gcc-4.6.3, intel2013.2}

[1]The TaskProlog script is executed before the job starts and has the ability to set environment variables for a job.[11]

- openmpi-1.8.3-{gcc-4.8.3, gcc-4.8.3-mpi-threads, gcc-5.1.0, intel2013.2, intel2015.0}
- openmpi-1.8.6-{gcc-4.8.3, gcc-4.8.3-debug, gcc-5.1.0, intel2013.2}
- openmpi-1.10.5-{gcc-4.8.3, gcc-6.4.0, intel2013.2, intel2015.0, intel2016.0}
- openmpi-{2.0.2-gcc-6.4.0-compute, 2.1.1-gcc-5.1.0, 2.1.2-intel2016.0}
- openmpi-{3.0.0-intel2016.0, 3.1.3-gcc-6.4.0}

The general naming scheme is $mpi-library-name/version/compiler-version/extra-flags$. In addition to our high uptimes, we rarely remove modules out of concerns over disruption to research. Following this strategy while upgrading the OS to a new major versions is the most common cause of non functional modules. These were compiled with older versions of system libraries but don't link with newer ones. The major cause of non performant builds, on the other hand, is errors in compile configuration coupled with lack of benchmarking.

A.2 New builds of MPI libraries

As mentioned in the text, spack[18] was used to automate the builds and which reduces time spent building and allows us to shift out focus to testing. The newly built libraries include:

- openmpi-1.10.5-gcc-6.4.0 {fabrics=verbs}
- openmpi-4.0.2-gcc-8.3.0 {fabrics=verbs, fabrics=ucx}
- mpich-3.3.2-gcc-8.3.0 {device=ch4, netmod=ucx}
- mpich-devl-gcc-8.3.0 {device=ch4, netmod=ucx}
- mvapich-2.3.2-gcc-8.3.0 {fabrics=mrail}

The newer library versions built with ucx in general perform better. The configurations are listed as per package conventions as represented by the corresponding variants in the spack package recipes.