Benchmark informed software upgrades at Quest-NUiT

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

Alper Kinaci
NUiT RCS, 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 perfromed to guide software upgrades. We performed extensive evaluation of all mpi libraries 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 and Alper Kinaci. 2020. Benchmark informed software upgrades at Quest-NUiT. In *Proceedings of PEARC '20*. ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Quest is a heterogenous HPC cluster[4] at Northwestern University consisting of Intel Haswell/Broadwell/Skylake nodes with varying interconects which uses the slurm[23] as resource manager and job scheduler (should we mention fairhsare ?). The cluster operates with very high uptimes and only shuts down once every academic year for maintainence (for approximately two weeks). While this high uptime is great for research throughput, it compresses critical maintaince tasks into those two weeks and makes the operators prioritize in place uprgrades 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. The software stack was kept stable even through maintaince cycles that involved major and minor OS upgrades.

This has led to a bloated software stack with inconstent naming schemes for modules and executables that 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 performant software for our users while reducing the maintainence 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 slurm update is on the agenda for the next downtime.

In this article we present the results of our benchmarking studies that inform our plans for deprecating modules and prelimary tests on our beta cluster for a strategy to deploy optimized builds for each architecture that are dynamically loaded at runtime based upon the nodelist for the job.

2 MPI LIBRARIES

Over time, multiple versions of libraries proving functionality specified by the message passing interface standard [15, 16] were installed. Some of these are quite old (before major OS version upgrades) and the naming scheme for the corresponding modulefiles is inconsistent. To test these libraries for funcaionaly and performance, we compiled and executed two point-to-point benchmarks, bandwith and latency from the OSU micro-benchmarks suite [6]. Since none of these were installed with slurm support, we wrote a bash script to run the tests. The results are presented in the figure 1.

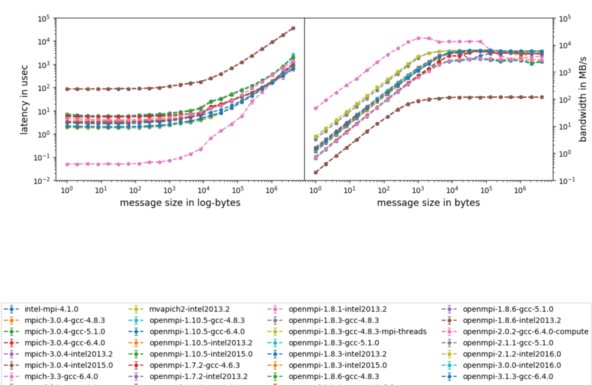

Figure 1: Benchmark of currently available mpi builds

In total, 42% of the available mpi libraries were faulty with 28% being nonfunctional (failure to compile or run) and the rest being nonperformant.

2.1 Improvements

Spack[17] 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 [11, 22]. 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 1.

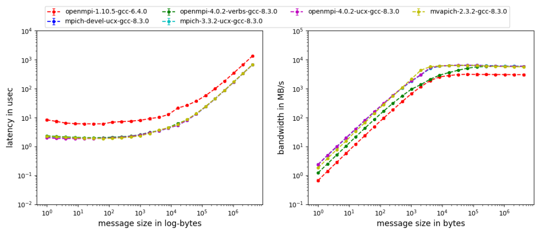

Figure 2: Benchmark of new mpi builds

As indicated in a recent OpenMPI deployment guide [21], 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 [19, 20] in slurm to use the PMIx process management standard [7, 14] (implemented via the OpenPMIx library [5]) which improves the job startup time. We also note that the slurm plugin for PMIx allows it use UCX for communication.

2.2 Deploying UCX and PMIx

While UCX was installed on the GPFS filesystem for convinience of testing, we plan to install it on the node-local filesystem (specifically at */usr/local*) of each compute node since it is a runtime dependency. 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 crash during the job start which reuqires ucx to be installed with 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 infibinand dirvers, 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 libaires.

Thus, we plan to use spack [17] 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 maintaing a complex software stack that includes a multitude of combitaions 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, in light of the current plans to purchase new "Cascade Lake" processors, such a deployment stragegy severly degrades productivity of the cluster. Thankfully, due to recent developments in the spack package manager, we are able to build multiple versions of each library, each optmized for a differenct architecture for optimal productivity of the cluster.

Table 1: Optimized builds on Haswell nodes

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

3.1 Benchmarks

We benchmarked optimzed builds for two of our most commonly used applications, LAMMPS[18] 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 [12, 13] 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.

3.2 Deployment Strategy

On Quest, users are not reuqired to choose a partition and the jobs are assigned to nodes dynamically based on availability. Thus, we 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 confiure a task prolog script for slurm that automatically sets the modulepath based on *SLURM_NODELIST*. To test this, we provisioned a virtual slurm cluster on a laptop by by four docker containers tied together via docker-compose using an existing repository [10] for the same developed by SciDAS (Scientific Data Analysis At Scale).

```
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

Thus, we have effectively used benchmarking tests to inform our software upgrade strategy for the NUiT-Quest high performance compute cluster. This work will inform our plans to deprecate and eventually remove non-functional and non-performant software and executing the future node-architecute opmized sotware stragey would enable a sifnificant enhancement in the productivity of the cluster.

ACKNOWLEDGMENTS

To Alex Mamach from NUiT for significant help with ucx/pmix testing and execution of sysadmin duites for the beta cluster, 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. The Unified Communication X Library. Retrieved April 19, 2020 from https://www.openucx.org/

[12] 2020. *Unified European Applications Benchmark Suite*. Retrieved April 19, 2020 from https://prolinktrials.co.uk/prace/training-support/technical-documentation/benchmark-suites/

[13] 2020. *Unified European Applications Benchmark Suite*. Retrieved April 19, 2020 from https://repository.prace-ri.eu/git/UEABS/ueabs

[14] 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

[15] 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/

[16] 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/

[17] 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

[18] 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

[19] 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

[20] 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).

[21] 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).

[22] 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.

[23] 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.