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, 2 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Quest is a heterogenous HPC cluster[1] at Northwestern University consisting of Intel Haswell/Broadwell/Skylake nodes with varying interconects which uses the slurm[4] 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 burdern for the operators and software specialists. In addition to the above, we also face an immediate need to make mpi providers 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 builds of libraries proving the message passing interface[2] functionality were installed. Some of these are quite old (before major OS version upgrades) and the naming scheme is inconsistent. Since none of these were installed with slurm support, we had to write a bash script to run the tests. The results are presented in the figure-abc.

In total, 42% of the available mpi libraries were faulty with 28% being nonfunctional and the rest being nonperformant.

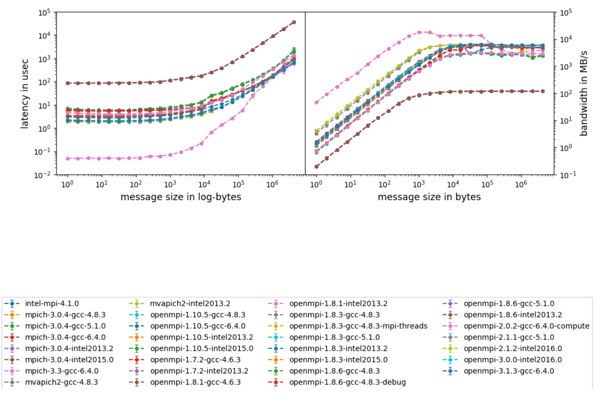

Figure 1: Benchmark of currently available mpi builds

2.1 Improvements

Spack[3] 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. Consistent with the literature we decided to use the "UCX" transport layer for all mpi libraries and to enable the "PMIx" plugin in slurm for better performance.

What is UCX ? How do we want to install it ? What is PMIx ? How do we want to install it ?

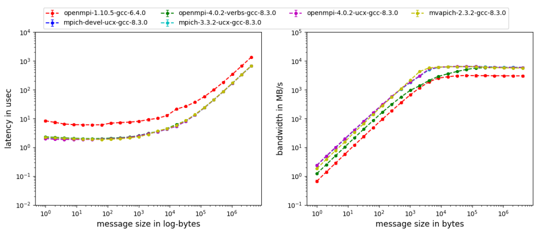

Figure 2: Benchmark of new mpi builds

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 NODE ARCH DEPENDENT SOFTWARE

3.1 benchmarks

Table with benchmarks for LAMMPS/GROMACS.

Elaborate on the benchmark probelm, add citation.

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. We plan to build each software package optimzed for all possible architectures and configure a task prolog script that sets the modulepath based on *SLURM_NODELIST* as shown below.

```
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
```

This was tested on a virtual slurm cluster provisioned by four docker containers tied together via docker-compose. [cite].

ACKNOWLEDGMENTS

To Alex from NUiT for help, to various mailing lists and forums including but not limited to mpich-discuss, slurm-info, spack-users.

REFERENCES

[1] 2019. *Quest.* Retrieved May 1, 2019 from https://www.it.northwestern.edu/research/user-services/quest/specs.html

[2] Message Passing Interface Forum. 2015. *MPI: A Message-passing Interface Standard, Version 3.1 ; June 4, 2015.* High-Performance Computing Center Stuttgart, University of Stuttgart. https://books.google.com/books?id=Fbv7jwEACAAJ

[3] 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

[4] 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.