**JMU Cluster Performance Variation**

By: Dani Bowser, Thomas Shattuck, Bryan Secoy, and Obi Chukwuezi

**Background:**

Our project consisted of running performance analysis tests on every node of the JMU cluster as a continuation of a previous years work. Their work focused on individual node analysis as well as the time of communications between two nodes. The majority of our work focused on rerunning the single-node analysis tests they used and running new analysis tests on the cluster. We used one test to see how nodes communicate with each other during MPI collective operations.

Our goal was to use the results to come up with any viable conclusions as to how and why the cluster performances the way it does. We also wanted to see if there was any variation between performance over a year or between hardware from the same manufacturer and same specifications.

**Methods:**

Continuing with single-node analysis, we ran two NAS parallel benchmarks, EP (Embarrassingly Parallel) test and the UA (Unstructured Adaptive mesh) test, as well as the HPL (High Performance Linpack) benchmark with two different compilers, MPICH and OMPI, on the entire cluster. The EP test provides an estimate of the upper achievable limits for floating point performance without significant interprocessor communication. The UA test gauges performance of scientific applications whose memory access patterns are irregular and continually change and the HPL test solves a random dense linear system in double precision arithmetic. Our last test was one created by us that used MPI collective calls to move a large array across nodes.

For the EP, UA, and HPL tests, we created an sbatch script that would create and then execute other scripts that ran the specified test on each individual node at the same time. The EP and UA master script ran twenty trials of the tests on each core of each node. The HPL master script was simpler, by running a single HPL test on each of the node. Many of the HPL parameters were defined and manipulated in the HPL.dat file, so we had to be careful to verify all parameters in that file before launching the test script.

The HPL.dat file controlled many of the parameters of the HPL tests. This file had to be modified to set the output method, the input value “N”, block size “NB”, and dimensions of grid (P & Q). After researching what each parameter’s purpose was, we found an HPL calculator, which aided us in calculating the values needed based on the cluters’ specifications. We were not able to devise a method to run tests with different parameters without manually updating the HPL.dat file, so we were limited to launching a set of tests, then modifying the file, then launching another round of tests.

Test results were stored for each test in separate output files. We were able to use “grep "e+" test14\_8192.txt | tr -s ' ' | cut -d" " -f7” to parse through the files and copy/paste the recorded Gflops values in to Excel spreadsheets. This allowed us to quickly create statistical analysis data and generate box graphs. For EP, UA, and MPI Collective tests, we used box-whisker plots since they were ran multiple times. For HPL, each test was only ran once, so we used bar graphs.

**Experiments:**

When it came down to the EP and UA tests the previous years team had results for the original 12 nodes of the cluster (compute01-12). Our team ran these tests on the newer 4 nodes (compute13-16) and found that results were around 2 seconds faster in total. Each tests in total for one node with all cores and 20 trials per core took about 40 minutes which meant these four tests took 2.6 hours consecutively. To ensure that time of execution and other jobs on the cluster didn’t interfere with the EP and UA tests we alter the previous groups batch script to create other batch scripts that execute and ran across all nodes at the same time. This also saved time in our project by reducing the total time completion for the EP and UA tests from roughly 10.6 hours to 40 minutes. There was no massive data change from the four nodes that we did consecutively vs the four that ran concurrently. However, to derive conclusive data with these tests we rerun them on the cluster to see how or if the data changed over a year specifically for the older 12 nodes. It was found that these tests ran faster than previously recorded and we will discuss this is more detail in the results section.

Our initial concern with running the HPL tests was the amount of time required to perform them. Based on our calculated parameters, we attempted to run a test with an input size of 52,000 which would use an estimated 80% of the nodes memory. After a single test ran for about four hours, it was clear to us that we needed to scale back. We decided to start small, and double the input value and then look up the run time using “sacct -j <jobID> --format=Elapsed”. We found that as the input size doubled, the run time doubled as well. This allowed us to predict that an input of 4,096 would take about 18 minutes. The actual test ran in 18:21 or ~3.7 sec for each input. Unfortunately, that result also meant that running with the recommended input of 52,000 would take ~4 hours. At this time we were running tests serially, so it would take ~64 hours to run the test on all 16 nodes. That was prohibitively long, so we decided to test using inputs of 8,192 and 16,274.

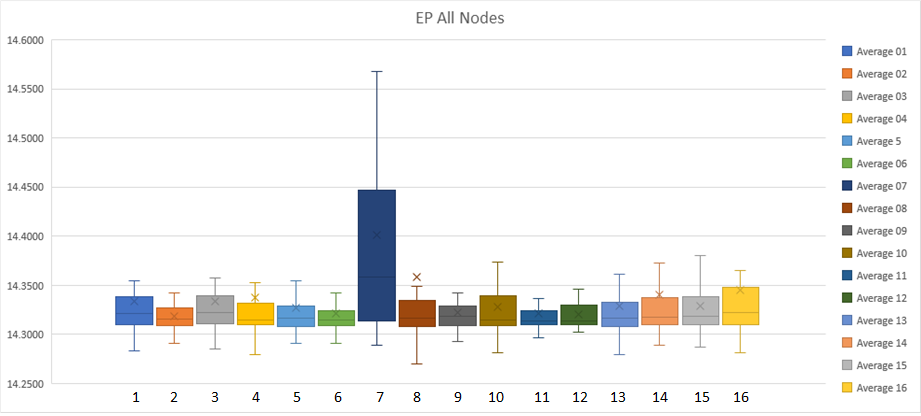
In an effort to speed up the testing, we began researching how we might run multiple tests simultaneously. This led us to creating an sbatch script that would create individual jobs for each node, and run them at the same time. It was too late to run test with 52,000 to include it in our poster, but we did attempt to run it immediately after. Unfortunately, after 5 hours of running, the HPL test failed with with a generic error. This forced us to scale back, so we attempted to use 32,765. That test failed as well. So we scaled back the input value to 24,576. This tests was finally successful.

The other experiment we performed with HPL was to compare the results of HPL tests compiled with MPICH vs those compiled with OMPI. We hypothesized MPICH would run faster because on our cluster it supports MPI\_THREAD\_MULTIPLE, while OMPI does not.

**Results:**

Our results for the EP test compared to last years results showed that it performed much better than in the past, however, the UA tests performed very similar except for node 7 (last year node 11 had a similar display of erratic timing in both tests). Specifiically for the EP test, the run time decreased for all previous 12 nodes and surprisingly some of them performed faster than the newer nodes 13-16. One possibility for why the EP test ran with different times is because last years group ran the tests with srun while we simply used a batch script. There is a bug report that srun executed OPEN MPI applications slower than mpirun in OPEN MPI version 1.10.1. We currently use MPI version 1.10.6 on the cluster. They were able to consistently report this finding with mpi applications like the NAS benchmarks that the EP and UA tests we use are. They did not see these occurrences in OPEN MPI version 1.8 but under 1.10 they also found that the user gets little resources because it spent more time cycling in pthread handling. Research suggests that this bug has been fixed in newer versions of OPEN MPI which is likely why the timing from last year does not vary drastically but still gives us a reason to think it altered the time a few seconds.

In our results, node 7 has very poor performance compared to the others. Without physical access to the servers, we have to hypothesize this could be a result of a manufacturer defect or heat related problem. If it is indeed a heat problem it could explain why node 11 had similar differences in testing results last year as the cluster has been moved since then. If a heat source was near node 11 then node 7 could have been moved to be facing a similar issue. We do not believe it is due to excessive use, because the lower numbered nodes perform many more jobs than the higher, due to the way slurm schedules the jobs. The newer nodes have the least amount of use but do not perform much different or even better than the older nodes. We ensured that this data was not some sort of testing inconsistency by rerunning the EP test on node 7 and we did not see any difference in the results.



*Figure 1: EP Results Across All Nodes*

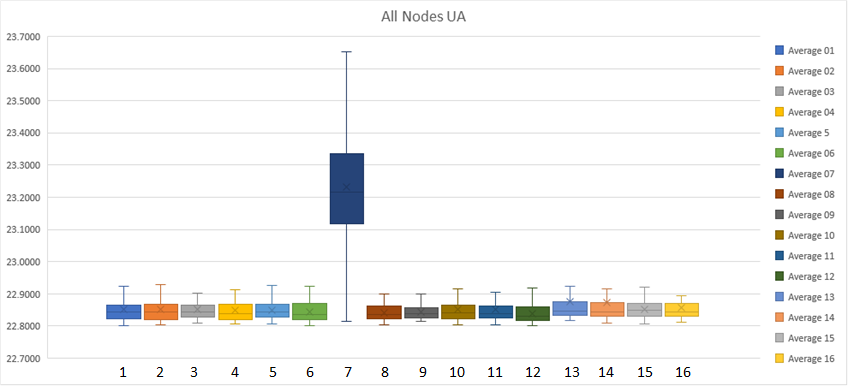
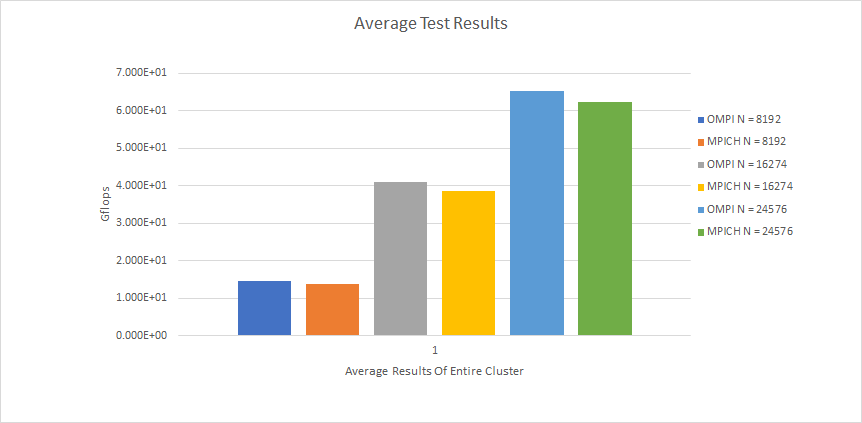


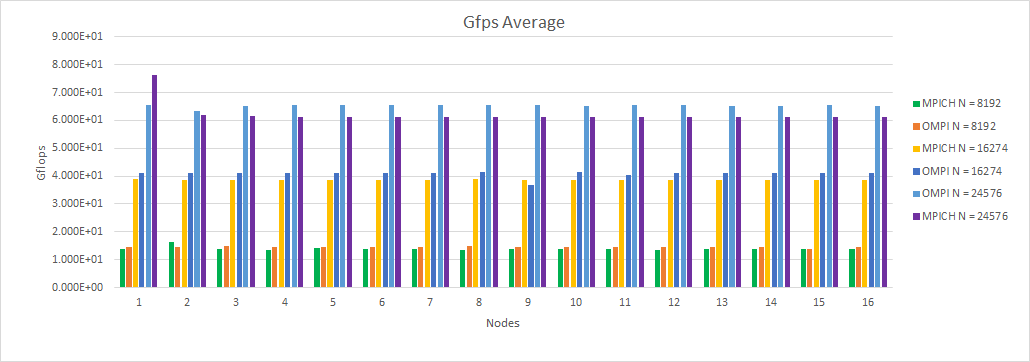
Figure 2: UA Results Across All Nodes

The HPL test results are very interesting. Even though we hypothesized MPICH would have performed faster than OMPI, it actually ran about 5% slower across all nodes and for all input sizes. Overall, the difference in performance doesn’t necessarily lend itself to choosing one implementation over the other, as the difference is negligible.



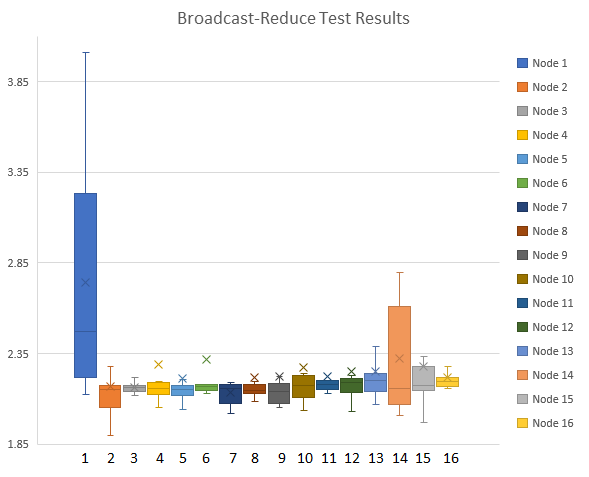
*Figure 3: Minimal difference in speed between MPICH and OMPI*

The much more significant difference in performance came from increasing the number of inputs. We tested input sizes of 8,192, 16,274, and 24,576. The increase in speed between 8,192 and 16,274 was 178%. The difference between 16,274 and 24,576 was 62%. This shows the speedup is quite large when scaling up from small input values, but is reduced as the input size grows.



*Figure 4: Overall performance of all nodes (HPL)*

MPI Collective call tests. Need more info added (will be added tomorrow along with the other data for single internode comms).



*Figure 5: Overall performance of all 16 nodes*

**Conclusion:**

For EP and UA tests, we were able to identify that node seven runs significantly slower than all other nodes. Without lower level access to the cluster, we can only hypothesize why this is happening. Overall, all other nodes performed consistently well.

For HPL tests, the simple conclusions were that MPICH vs. OMPI made very little difference in performance, and that as the input size increased, speed increased as well, but eventually began to drop when using very large inputs. The speed variation across the nodes was very small, and we didn’t identify and particular nodes that ran significantly slower or faster than the other nodes. We hypothesize that when we used an input size of > 24,576 the tests began to fail due to running out of memory. The HPL tests was not designed to test single CPU’s, but is intended for entire clusters. I believe if we had performed the test that way, we would have been able to run it with much higher input values, but we would still encounter an issue with the test being very time consuming.

With the MPI collective calls tests, we identified nodes 1 and 14 run the slowest and have the highest variation in their speed. We believe this is likely being caused by the network in some way, such as a bad ethernet cable, poor connection, or even a bad port on the switch. Node one is the most used node, so we could hypothesize this is being caused by using it more often than the other nodes, but this can not be said for node 14.

**Future Work:**

There are several areas in which we would like to expand the cluster performance variation project. First, we would like to run the HPL tests with different combinations of nodes, such as all even nodes, all odd nodes, the first 8 nodes, the last 8 nodes, etc. This would allow for larger input values and more accurate results. Comparing the different combinations could help identify faster or slower combinations due to locality, network overhead, or even heat distribution. It would also be very insightful if logs were taken of CPU temperatures while HPL tests were being performed, so the performance results could be compared to operating temperature. In this case, it might prove that node seven performed poorly due to heat saturation.

Secondly, we would like to run other commercial benchmark suites on the cluster. Some of these benchmarks might include Coremark and many of the original eight NAS benchmarks (such as IS, CG, MG). Comparing tests results of different NAS classes of tests would also help provide more insightful results.

Third, getting lower-level access to the cluster would allow for more accurate and/or interesting conjecture as to what could be causing the performance variance among the nodes of the cluster, despite them being of similar hardware architecture. Some of the factors that could be making a difference in the performance of the nodes are things such as: amount of dust in the air, physical distance from one node to another, temperature of the node and/or room that houses the cluster, amount of moisture in the air, and the quality/age of the thermal paste used for the CPUs on each node.

Lastly, it would be interesting to compare the JMU’s cluster to distributed systems at other universities and maybe even larger clusters that are used commercially.

**References:**

<https://www.nas.nasa.gov/publications/npb.html>

<http://hpl-calculator.sourceforge.net/>

<http://www.netlib.org/benchmark/hpl/>

<https://w3.cs.jmu.edu/lam2mo/cs470/cluster.html>

<https://bugs.schedmd.com/show_bug.cgi?id=2414>

<http://www.crc.nd.edu/~rich/CRC_Summer_Scholars_2014/HPL-HowTo.pdf>