An Analysis of Node Sharing on HPC Clusters using XDMoD/TACC_Stats

Joseph P. White[1], Robert L. DeLeon[1], Thomas R. Furlani[1], Steven M. Gallo[1], Matthew D Jones[1], Amin Ghadersohi[1], Cynthia D. Cornelius[1], Abani K. Patra[1], James C. Browne[2], William L. Barth[3], John Hammond[3]

[1]Center for Computational Research, University at Buffalo, SUNY, Buffalo, NY

[2]Department of Computer Science, University of Texas, Austin, TX 78758

[3]Texas Advanced Computing Center, University of Texas, Austin, TX 78758

ABSTRACT

When a user requests less than a full node for a job on XSEDE's large resources - Stampede and Lonestar4 -, that is less than 16 cores on Stampede or 12 cores on Lonestar4, they are assigned a full node by policy. Although the actual CPU hours consumed by these jobs is small when compared to the total CPU hours delivered by these resources, they do represent a substantial fraction of the total number of jobs (~18% for Stampede and ~15% for Lonestar4 between January and February 2014). Academic HPC centers, such as the Center for Computational Research (CCR) at the University at Buffalo, SUNY typically have a much larger proportion of small jobs than the large XSEDE systems. For CCR's production cluster, Rush, the decision was made to allow the allocation of simultaneous jobs on the same node. This greatly increases the overall throughput but also raises questions whether the jobs that share the same node will interfere with one another. We present here an analysis that explores this issue using data from Rush, Stampede and Lonestar4. Analysis of usage data indicates little interference.

Categories and Subject Descriptors

C.4 [Performance of Systems]: *Design studies, Fault tolerance, Measurement techniques, Modeling techniques, Performance attributes, Reliability, availability, and serviceability.*

General Terms

Management, Measurement, Documentation, Performance, Design, Reliability, Verification.

Keywords

XDMoD, TACC_Stats, node sharing, SUPReMM, HPC

XSEDE '14, July 13 - 18 2014, Atlanta, GA, USA

http://dx.doi.org/10.1145/2616498.2616533

1. INTRODUCTION

Node sharing, in which multiple jobs share a given compute node, in HPC clusters has been shown to increase throughput and energy efficiency by 10-20% [1-4]. As the number of cores per node grows for a variety of architectures of modern HPC clusters, it becomes more important to consider allowing small jobs to share nodes rather than simply allocating a full node to each job regardless of the actual requested resources. Although at the current number of cores per node the actual CPU-hours devoted to these small jobs on the large XSEDE HPC clusters such as Stampede and Lonestar4 is small, there are still a large number of such jobs and the scheduling and resource waste issues will increase as systems are brought on line that have more cores for each node. There is an obvious advantage to sharing nodes from both a scheduling and a resource efficiency perspective. However, the question arises - are there consequences that offset this advantage?. This paper presents an analysis of node sharing on the Center for Computational Research's (CCR) production cluster (Rush) based on metrics from XDMoD/TACC_Stats. We also compare with TACC_Stats metrics collected on Stampede and Lonestar4, which do not share nodes.

The rest of the paper includes an overview of XDMoD/TACC_Stats, a brief description of the three HPC clusters, XSEDE's Stampede and Lonestar4 and CCR's Rush, and results from a detailed usage analysis comparing the consequences of sharing nodes.

2. ANALYSIS TOOLS

2.1 XDMoD Framework

Here we present a brief overview of XDMoD, a more detailed description can be found in references [5-8]. XDMoD was originally targeted at providing the analyses required for effective overall management of the computational resources of the XSEDE organization, although an open source version is now available [14]. XDMoD ingests and organizes data on computer system performance and then maps that data into metrics required for overall system management. The XDMoD portal provides a rich set of features accessible through an intuitive graphical interface, which is tailored to the role of the user. Metrics provided by XDMoD include: number of jobs, CPUs consumed, wait time, and wall time, with minimum, maximum and the average of these

metrics, in addition to many others. These metrics can be broken down by: field of science, institution, job size, job wall time, NSF directorate, NSF user status, parent science, person, principal investigator, and by resource. Performance and quality of service metrics of the HPC infrastructure are also provided, along with application code specific performance metrics (flops, IO rates, network metrics, etc) for all applications running on a given resource (through TACC_Stats). Another key feature is the Usage Explorer that allows the user to make a custom plot of any metric or combination of metrics filtered or aggregated as desired. For example, Figure 1, which was created using the Usage Explorer, shows CPU hours and wait time versus job size on a local university-based HPC resource at the Center for Computational Research at SUNY-Buffalo.

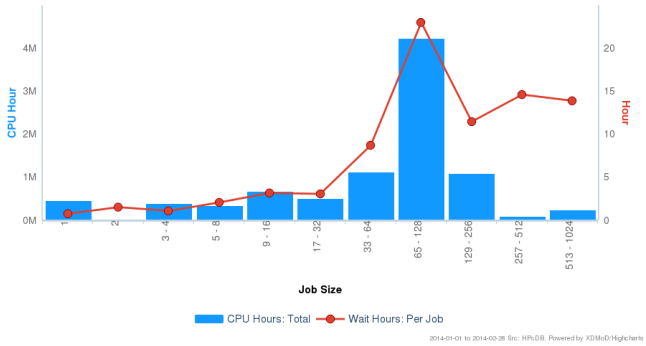


Figure 1 Plot of CPU hours consumed and job wait time versus job size for the January to February 2014 time period at the University at Buffalo Center for Computational Research.

The XDMoD tool is also designed to preemptively identify underperforming hardware and software by deploying customized, computationally lightweight "application kernels" that continuously monitor HPC system performance and reliability from the application users' point of view [5-8]. The term "application kernel" is used in this case to represent micro and standard benchmarks that represent key performance features of modern scientific and engineering applications, and small but representative calculations carried out with popular open-source high performance scientific and engineering software packages. The term "computationally-lightweight" is used to indicate that the application kernel runs for a short period (typically less than 10 min) on a small number of processors (fewer than 128 cores) and therefore requires relatively modest resources for a given run frequency (say once or twice per week). Accordingly, through XDMoD, system managers have the ability to proactively monitor system performance as opposed to having to rely on users to report failures or underperforming hardware and software. The detection of anomalous application kernel performance is being automated through the implementation of process control techniques. In addition, through this framework, new users can determine which of the available systems are best suited to address their computational needs.

In addition, metrics that focus on scientific impact, such as publications, citations and external funding, are now being developed and incorporated into XDMoD to help quantify the role modern cyberinfrastructure plays in advancing research.

Taking advantage of the great similarity of XSEDE and a typical HPC center, we have developed Open_XDMoD, the open source version of XDMoD, which leverages the same code base. Accordingly, as XDMoD continues to be developed, Open_XDMoD will also benefit from this work. Both versions share many of the same metrics and functionality (e.g., Summary, Usage/Usage Explorer, and Report Generator) and differ mainly in support of elements specific to XSEDE. XSEDE maintains a centralized infrastructure (XSEDE central database) for storing job accounting records, users, and allocations/projects while a typical HPC center may not have this data in a centralized location. Open_XDMoD provides its own data warehouse with support for parsing and loading of resource manager log files and spreadsheets containing user information including departmental affiliations. In addition, future versions will support allocations, and integration with local LDAP services, and application kernels for monitoring Quality of Service.

2.2 TACC Stats

The Linux sysstat package is a comprehensive collection of performance monitoring utilities, each of which reports resource statistics of specific components of a system in its own format. TACC_Stats [9] enhances sysstat/sar for the open source software based HPC environment in many ways. It is a single executable binary that covers all performance measurement functions of sysstat and outputs in a unified, consistent, and self-describing plain-text format. It is batch job aware: Performance data are tagged with batch job id to enable offline job-by-job profile analysis. It supports newer Linux counters and hardware devices. Its source code [10] is also highly modular and can be easily extended to gather new kinds of performance metrics.

Currently TACC_Stats can gather core-level CPU usage (user time, system time, idle, etc.), socket-level memory usage (free, used, cached, etc.), swapping/paging activities, system load and process statistics, network and block device counters, interprocess communications (SysV IPC), software/hardware interrupt request (IRQ) count, filesystems usage (NFS, Lustre, Panasas), interconnect fabric traffic, and CPU hardware performance counters. For a complete list of the data acquired by TACC_Stats, see the TACC_Stats web site [10].

Different CPU chips have different sets of performance counters. Therefore the set of performance counter derived metrics available may vary across systems. TACC_Stats utilizes CPU performance counters as follows. At the beginning of the job, TACC_Stats is invoked by the batch scheduler prolog to reprogram performance counters to record a fixed set of events: On AMD Opteron, the events are FLOPS, memory accesses, data cache fills, SMP/NUMA traffic. On Intel Nehalem/Westmere, the events are FLOPS, SMP/NUMA traffic, and L1 data cache hits. On Intel Sandy Bridge, the events are SSE and AVX FLOPS, memory traffic, and cache traffic. In order to not interfere with user's own profiling and instrumentation activities, at periodic invocations (currently every 10 minutes), TACC_Stats only reads values from performance registers, without reprogramming them, to avoid overriding measurements initiated by users. In addition to the data gathered by TACC_Stats, data from the system scheduler and the job management system are collected and coordinated with TACC_Stats data.

3. NODE-SHARING ANALYSIS

The case studies reported in this paper were carried out on the HPC cluster Rush at CCR and the Stampede and Lonestar4 supercomputers at the Texas Advanced Computing Center (TACC). Stampede is a Linux cluster comprising 6400 primary compute nodes each of which has two Intel Xeon eight –core E5-2680 2.7GHz processors and 32 GB of memory. The filesystem is Lustre, and the interconnect is QDR InfiniBand. Lonestar4 is also a Linux cluster with 1088 Dell PowerEdgeM610 compute nodes.

Each compute node has two Intel Xeon 5680 series 3.33GHz hexa-core processors and 24 GB of memory. Lonestar4 has two filesystems: Lustre and NFS and its interconnect is InfiniBand (NFS is connected via Ethernet). CCR's x86_64 Linux cluster, Rush, is a heterogeneous system containing 8, 12, 16 and 32 core nodes. Specifically it consists of 372 Dell C6100 servers, each with two Intel "Westmere" Xeon six-core 2.40GHz (E5645) processors and 48GB of memory, 128 IBM iDataPlex dx360 M2 servers, each of which has two Intel "Nehalem" Xeon quad-core 2.26GHz (L5520) processors and 24GB of memory, 10 Dell servers each with four eight-core Intel Xeon E7-4830 processors eight of which have 256 GB of memory and two have 512 GB memory, 8 IBM servers, each of which has four eight-core AMD Opteron 6132 HE processors and 256 GB of memory and 128 Dell C6100 servers containing two Intel "Westmere" Xeon quad-core 2.13GHz (L5630) processors and 24GB of memory and 32 Dell R620 servers with 2 Xeon 8-core E5-2660 2.26 GZ and 128GB of memory. All are interconnected with QDR InfiniBand and gigabit Ethernet. Rush has two global filesystems Panasas PAS8 Parallel Storage for the global shared parallel scratch directory and Isilon IQ36000x Storage Arrays for general network file system access.

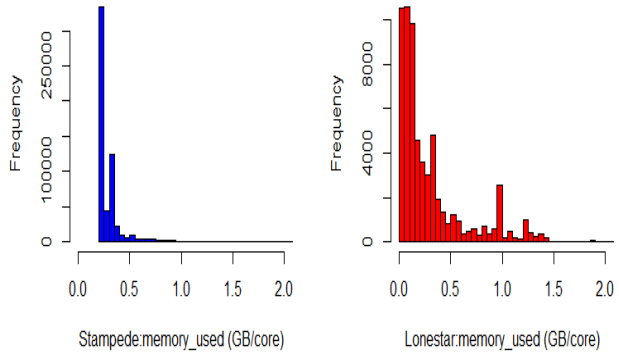
For TACC's Stampede and Lonestar4, user jobs are assigned entire nodes regardless of the number of requested cores, while for CCR's Rush, nodes are shared between jobs by default unless the user requests exclusive use. Rush uses SLURM [11] as the resource manager and has the cgroup task plugin enabled. The cgroup plugin uses the Linux kernel cgroup subsystem to confine jobs to their allocated cpuset and limit the memory available to them. This prevents most jobs from using CPU and memory resources that have been allocated to other jobs sharing the node.[1]

The analysis below looks at the ramifications of sharing versus not sharing nodes. We analyzed TACC_Stats data collected during January and February of 2014 on Rush, Stampede and Lonestar4. In the following discussion, a job is defined as shared if at least one other job ran on its assigned nodes while it was running. An exclusive job had no other jobs running concurrently on its nodes.

Approximately two thirds of jobs on Rush were shared node jobs. Of the jobs that were shared, 98% of them ran on a single node. 91% of the exclusive jobs ran on a single node and 9% ran on multiple nodes. The most common job size on Rush is the single core job (34%) with the majority (64%) of jobs running on one or two cores.

3.1 Memory Usage

Probably the most severe potential problem for sharing nodes is the competition for memory between jobs running on the same node. Figure 2 shows a histogram of the memory per core usage on Stampede and Lonestar4, which do not share nodes. The TACC_Stats metric displayed is the average OS memory usage minus the slab and kernel page cache[2] per node divided by the number of cores on the node. The frequency is the number of jobs with the recorded memory usage. Recall that Stampede has 32 GB per node or 2 GB per core and Lonestar4 has 24 GB per node or 2 GB per core. Obviously, both Stampede and Lonestar4 have more than sufficient memory for their general job-mix. Figure 3 shows the comparable memory usage on Rush where now the usage has been shown for exclusive node (non-shared) jobs and shared node jobs. The distribution of memory usage for the exclusive nodes peaks early, well below 1 GB, with a tail out to larger memory usage. The plot of shared node jobs shows similar distribution but with a broader primary peak at a slightly higher memory usage. Based on the very sensitive Kolmogorov-Smirnov test, the two distributions are statistically significantly different. However for Figure 3 and the other figures comparing the exclusive and shared nodes, the standard that we will apply is only that they are qualitatively similar not identical. Note that the plot does not indicate that the shared jobs themselves used more memory, rather that the nodes that ran shared jobs have a higher memory usage per core than exclusive nodes. On shared nodes there are multiple different jobs running different processes that are less likely to be using the same shared libraries. By comparison, the exclusive jobs that run multiple copies of the same process will reuse shared objects and so have a smaller total memory footprint. Also, since the memory usage is the average per node divided by the number of cores per node, the exclusive jobs that run processes on a subset of the cores will have a lower average memory value than when multiple jobs are running.[3] Figure 4 shows the average kernel page cache usage per core for the exclusive and shared jobs. The cache usage is slightly higher for the shared jobs compared to exclusive jobs. On the shared nodes the different processes are more likely to be accessing different files, which will tend to increase the page cache utilization. Recall that the majority of nodes on Rush have 3 or 4 GB of memory per core and the large memory nodes have 8 or 16 GB of memory per core depending on the node. Even though the shared node jobs show a higher average memory usage, the majority of the nodes with shared jobs have a memory usage below 1GB. Given the memory capacity of the CCR HPC nodes this does not appear to be a problem.



[1] It is possible for jobs to escape from the cgroup restrictions by running processes on the nodes using ssh rather than via the slurm srun command. ssh is enabled on Rush because it required by some of the user and commercial applications.

[2] The slab cache is a cache of kernel data objects. The page cache is a cache of physical pages. The most common data stored in the page cache are the contents of disk blocks.

[3] A single-core job that uses 2GB on an 8 core machine will have and average memory usage of 2/8 whereas two single-core jobs each using 2GB on the same node will each show a usage of (2+2)/8 (i.e. double) even though each job used the same amount of memory.

Figure 2 Memory Usage on Stampede (blue) and Lonestar4 (red).

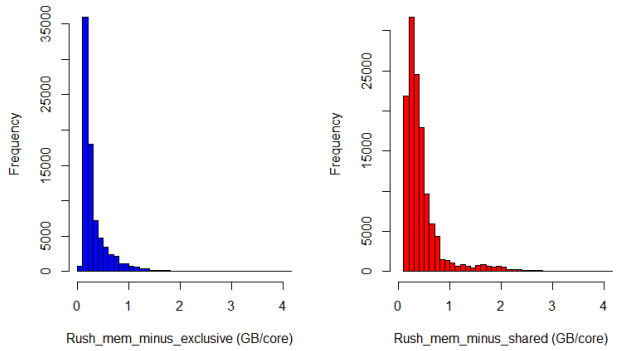


Figure 3 Memory Usage in GB per core on Rush for the exclusive node jobs (blue) and the shared node jobs (red).

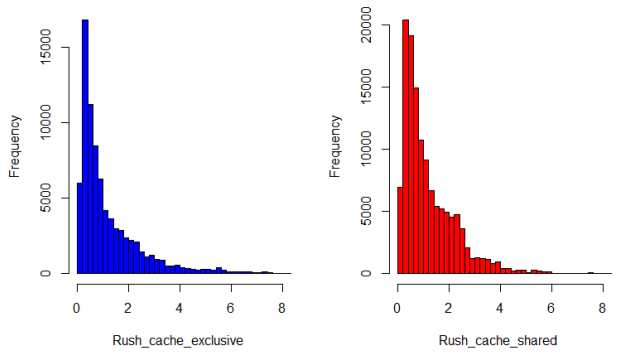


Figure 4 Kernel page cache usage in GB per core on Rush for the exclusive jobs (blue) and shared jobs (red).

The plots in Figure 5 and Figure 6 show the average memory usage per core for all active memory on the node (i.e. including the kernel page cache and slab cache). Figure 5 shows the active memory for exclusive and shared jobs on the Rush nodes that have 4 GB per core. The shared node jobs show a higher overall memory usage compared to the exclusive node jobs. The reasons for this are the increased memory usage due to the user processes as discussed previously and increased page cache usage. The plots in Figure 6 show the average memory usage for 3GB nodes. The shared nodes show an increase in memory usage compared to the exclusive nodes and there is a peak in the distribution at ~2.8 GB, near the total memory on the node. Analysis of the raw data for the cluster of jobs with the high memory usage shows that the high memory usage is due to the contribution of the page cache. It is not *a priori* a problem if the memory usage on the machine is near maximum since the Linux kernel is designed to use all of the available memory for the page cache. However it is possible that the high cache usage is an indicator that the jobs running on the nodes are performing lots of disk I/O and could potentially be interfering with each other. We examine the I/O usage in Section 3.3.

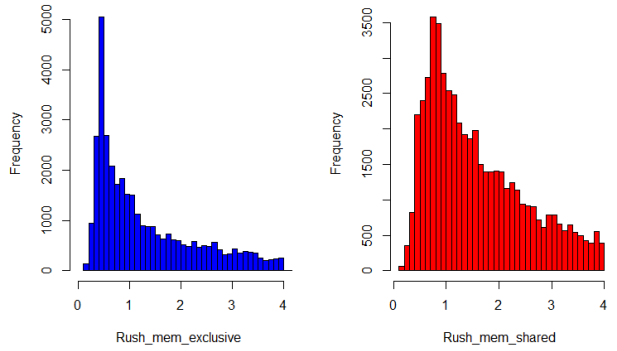


Figure 5 Total memory usage on Rush for jobs that ran on nodes populated with 4 GB memory per core. Exclusive node jobs (blue), shared node jobs (red).

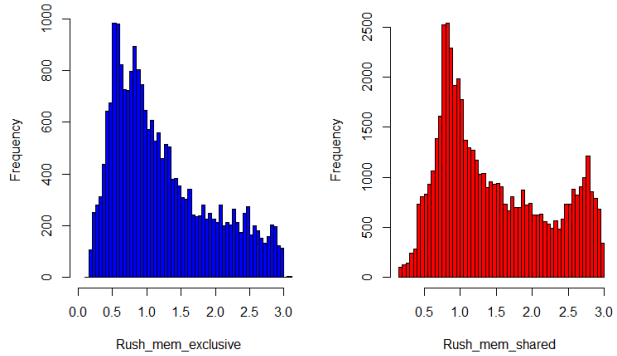


Figure 6 Total memory usage on Rush for jobs that ran on nodes populated with 3 GB memory per core. Exclusive jobs left, shared jobs right.

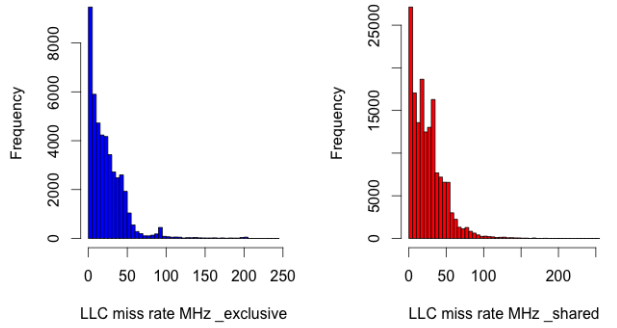


Figure 7 Last level cache read miss rates in MHz per socket for the nodes on Rush with Nehalem and Westmere chipsets. Exclusive node jobs (blue), shared nodes jobs (red).

The plot in Figure 7 shows the average miss rate per socket for last level cache (LLC) reads. The data are obtained from the UNC_LLC_MISS hardware performance counter, which is available on most of the Intel Nehalem and Intel Westmere nodes

on Rush. The rates are computed with respect to wall clock time. The plots show that the shared nodes have a modest increase in the miss rate, which is consistent with the observed increase in memory usage. The number of jobs with high average cache miss rates are low suggesting that LLC interference is not an issue for the majority of jobs. The LLC miss rate allows us to calculate an estimate of the upper bound of memory bandwidth, since an LLC miss results in either a read from main memory or a read from the LLC on another socket. The cache width on the Nehalem and Westmere architectures is 64 bytes. An LLC read miss rate of 100 MHz means that the main memory read rate is not greater than ~ 6 GiB/s per socket..The maximum memory bandwidth is 32 GiB/s per socket for the Westmere nodes and 25.6 GiB/s for the Nehalem nodes. The bulk of the jobs have a miss rate below 50 MHz (~ 3 GiB/s per socket). The plots therefore also indicate that very few jobs have sustained high memory bandwidth usage.

Based on this data, and given the over-provision of memory in Stampede and Lonestar4, see Figure 2, we speculate that allowing node sharing for the TACC HPC clusters would not overtax the memory capacity. There is obviously a trade-off in determining how much memory to provide for each node and whether there is enough memory to share nodes, as well as site and XSEDE policies on capacity versus capability computing needs.

3.2 Exit Codes

We also looked at job failure rates as determined by exit codes on Rush to determine if there was a systematically higher failure rate for the shared node jobs compared to the exclusive node jobs. Figure 8 shows the fractional distribution of jobs for the different exit codes. The exit codes are of the form N:M where N is the error return code from the job script and M is the signal number sent to the job by the scheduler. An exit code of 0:0 indicates the job ran successfully and all other codes indicate a failure. The exit code 0:1 corresponds to jobs that were killed by the scheduler with an interrupt signal: either as a result of a timeout or a cancel command by the job owner.

It is slightly surprising that the overall failure rate is lower for the shared node jobs, however, some of the exclusive node "failures" may be simply time-outs for long jobs. Failure code 1:0 which is an unknown failed job that generally happens quickly (24% of these jobs exit within the first five minutes) is actually lower for the shared node jobs. For the jobs that exit with code 0:1, most of these are timeouts with the job durations clustered around 0.5 hour, 1 hour and 72 hours. The 1 hour and 72 hours values correspond to the time limits on the debug and general compute partitions on Rush. There are twice as many exclusive jobs that timeout after 3 days than shared jobs. It is not unexpected that the exclusive node jobs have a higher 0:1 failure code rate since these are predominantly the long jobs that have timed out and the exclusive jobs have a much higher overall average value of cpu-hours. The other, rarer exit codes generally show a similar rate for exclusive node and shared node jobs. There certainly does not seem to be any systematic trend for the shared node jobs to show a higher failure rate. Note that jobs can still fail even if a success code is returned to the scheduler. For example if the underlying software does not propagate the error. These cases are not captured in this analysis.

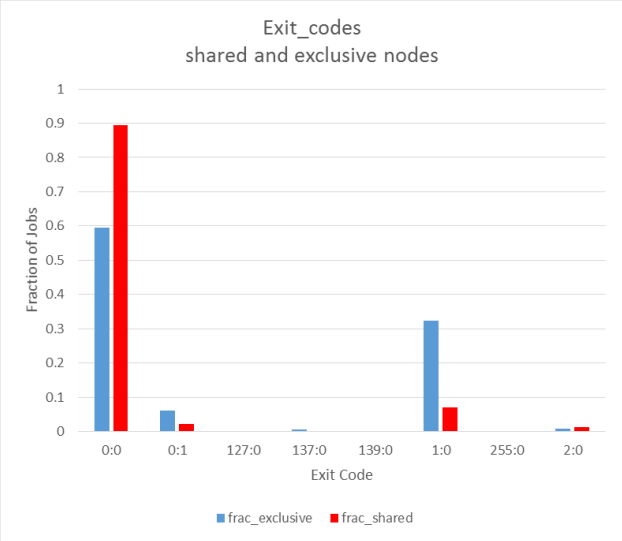


Figure 8 Exit codes on Rush for the exclusive node jobs (blue) and the shared node jobs (red).

3.3 I/O Usage

We examined the relative I/O usage of the exclusive and the shared node jobs including reads and writes to the Panasas file system on Rush. Figure 9 shows the average write rate per node in bytes/s for exclusive and shared jobs. 42% of the jobs on Rush did not use the Panasas file system so the frequency of these zero I/O usage jobs has been truncated on the plot to show the detail of the jobs that did perform I/O. The theoretical maximum write bandwidth per node for the file system is approximately 100 MiB/s. Although the shared nodes show substantially greater I/O activity, the majority of the jobs have an average write rate below 2 MiB/s. This average rate is low enough that it should not cause any difficulty for the shared jobs.

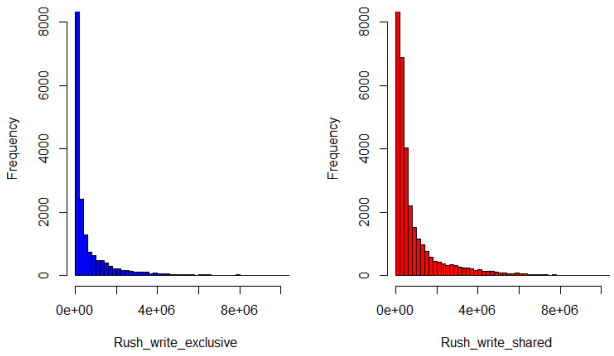


Figure 9 Average data rate in byte/s for writes to the Panasas file system on Rush for the exclusive node jobs (blue) and the shared node jobs (red). Note: The y-axis has been truncated; the 0 bytes/s bin has a frequency of ~49000 and ~40000 for exclusive and shared jobs respectively.

3.4 Network Usage

Finally, we examined the InfiniBand (IB) network usage of the exclusive and the shared node jobs including network receive and transmit of data and packets. For reference, the IB usage on

Stampede is given in Figure 10. The plot shows the average write rate per node in bytes/s and the $\log_{10}$ of the data are plotted due to the large range of data rates. The majority of jobs have average IB write rates between 10^4 and 10^6 bytes/s per node. Figure 11 shows the average IB write rate for all jobs on Rush. The peaks at ~1.70 and ~1.87 (corresponding to ~50 and ~74 bytes/s) are caused by background traffic on the IB bus, rather than job-specific data usage. This shows that the majority of jobs on Rush do not use the IB fabric and is consistent with the observation that, in terms of the total number of jobs (as opposed to total CPU time) most of the jobs on Rush are short duration one or two core jobs. In terms of total CPU time delivered however, jobs requiring 64-128 cores are the most prominent. Figure 12 shows the average IB write rates per node for jobs on Rush for the exclusive and shared jobs. The IB usage distribution is similar for the shared and exclusive node jobs.

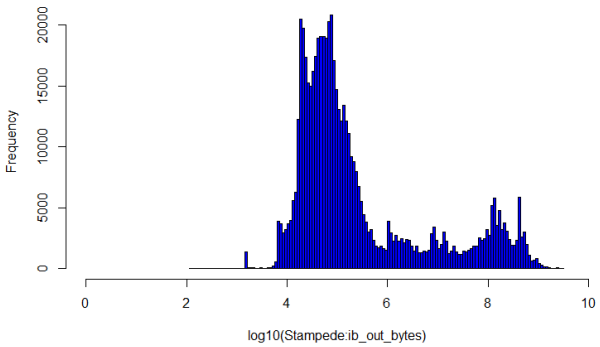


Figure 10 Average InfiniBand write rate per node in bytes/s for jobs on Stampede. The x-axis is a log base 10 scale.

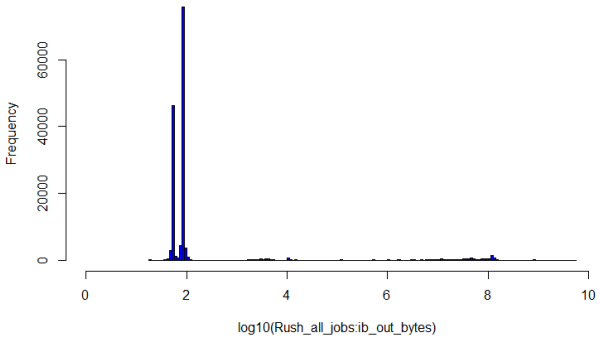


Figure 11 Average InfiniBand write rate per node in bytes/s for jobs on Rush. The x-axis is a log base 10 scale.

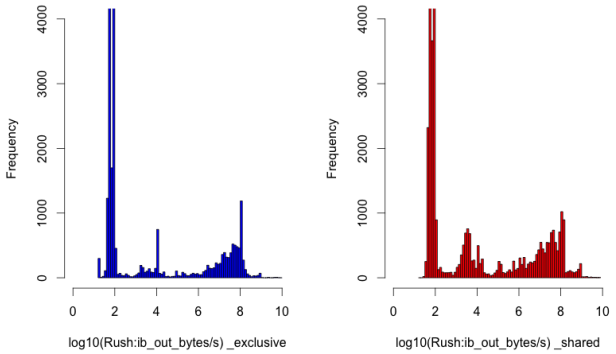


Figure 12 Average InfiniBand write rate per node in bytes/s for Rush for the exclusive jobs (blue) and shared jobs (red). Note: The y-axis has been truncated. The two truncated bins for the exclusive jobs have counts of ~15,000 and ~30,000 and the shared jobs have counts of ~30,000 and ~50,000.

3.5 CPU Usage

While each job on Rush is confined to the allocated CPU(s) regardless of whether or not it is shared, it is interesting to look at the CPU usage on shared and unshared nodes and compare the Rush distributions with the TACC XSEDE clusters. Figure 13 shows the cpu_user fraction for Stampede and Lonestar4. The cpu_user fraction is the ratio of the cpu time spent in user mode to wall clock time averaged over all cores that the job was allocated. Both plots in Figure 13 show that most jobs are relatively efficient with greater than 0.9 cpu_user fractions. There are a group of jobs at the left hand side of the distribution that are relatively inefficient. For Stampede small spikes at 0.5 0.25 and 0.0625 are produced by 8-way, 4-way and 1-way jobs respectively on the 16-core nodes. 18% of the jobs requested a single node and had less than 50% average CPU usage. For Lonestar4 small peaks at 0.67, 0.16 are 8-way and 4-way jobs; the very large peak at 0.083 is from 1-way jobs which are apparently much more prevelant on Lonestar4 than on Stampede. 15% of Lonestar4 jobs used a single node and had less than 50% average CPU usage.

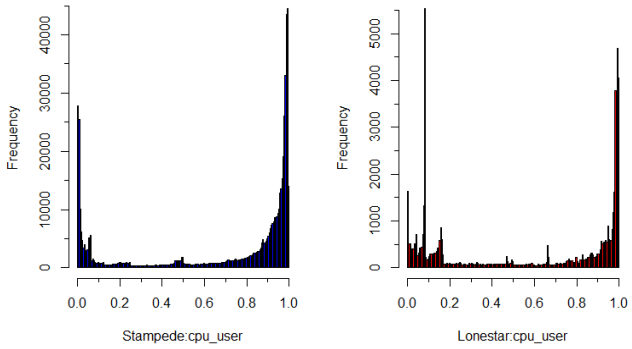


Figure 13 CPU Usage on Stampede (blue) and Lonestar4 (red).

The CPU usage on Rush is shown in Figure 14 for both the exclusive and shared node jobs. Also shown are the cpu_idle and cpu_system fractions which are the average ratio of cpu time to wall time for which the cpu was idle and processing in kernel mode respectively. The average cpu usage is computed over all cores on

the nodes that a job ran on. This means that, for the shared node jobs, the cpu usage statistic for a job includes the contributions for all jobs that run concurrently on the same nodes. This leads to the complicated distribution for the plots for the shared jobs. The distribution is also complicated because Rush is a heterogeneous node cluster with 8, 12, 16 and 32 core nodes. The cpu_user distribution for the exclusive jobs shows a peak near 1 and a secondary peak near 0.75. There are also several peaks on the left hand side of the plot notably peaks at 0.08 and 1.2, these correspond to 1-way jobs on 12 core nodes and 1-way jobs on 8 core nodes respectively. The peak in the user_cpu near 0.75 and peak in cpu_system near 0.25 corresponds to a single user that ran a large number of similar jobs that had high total cpu usage, but had a much higher system to user cpu ratio than a typical job. When these jobs are removed from the analysis, the distribution looks much more like those in Figure 13. That is a peak near ~1 cpu_user, and another smaller group of jobs near 0 with small secondary peaks corresponding to jobs using only a fraction of the cores.

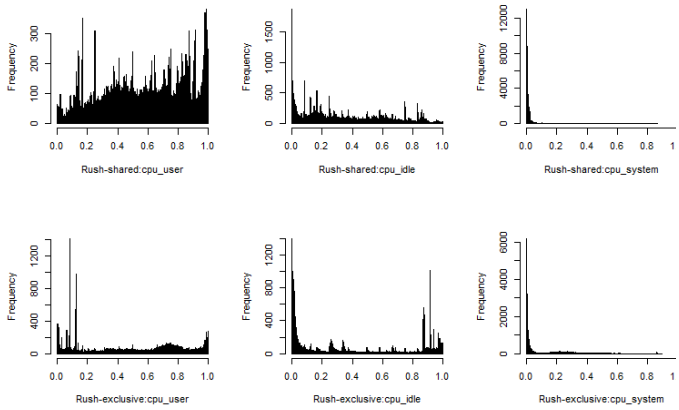


Figure 14 CPU usage on Rush for the exclusive node jobs (bottom) and the shared node jobs (top).

4. RELATED WORK

The advantages of allowing for shared nodes was discussed in reference [1-4] where quantitative advantages were given dependent on the job mix in the context of developing a fair pricing algorithm for shared node jobs. Some issues on node sharing that apply to disk cache and memory bandwidth have also been discussed [12-13].

5. DISCUSSION & FUTURE WORK

The present analysis has examined the consequences of sharing nodes on the CCR job mixture. Overall the data show little difference between the shared and exclusive jobs, suggesting that, for the job mix on CCR's Rush production cluster, job interference is not a significant issue. As newer architectures increase the number of cores per node from the present level to hundreds, the total number of jobs that can effectively share nodes will increase dramatically and job sharing may go from an option to a necessity on large supercomputers. This is likely especially relevant for academic HPC centers, where the job mix will be very much similar to that for CCR, with many users with modest computing requirements in terms of number of processors per job and a smaller pool of users with more demanding multi-node parallel computations. As a secondary objective, the present work has shown the value of XDMoD/TACC_Stats to provide system designers with the data that they need to design upgrades and replacements for major HPC resources.

One of the limitations of the current analysis is that the metrics for the shared jobs are averaged over the entire nodes, rather than having the per-job values. For certain metrics it is not possible to determine the per-job values because the hardware does not have the capability to distinguish the source of the request (for example, IB interface metrics or memory writes from the LLC). However, for many metrics it is possible to identify the associated job. We plan to improve the data collection and analysis utilities to extract per-job information for shared jobs rather than only generating the values averaged over the nodes that the job ran on. As an additional benefit we will also be able to distinguish between jobs that shared the same physical CPU socket and the jobs that shared a node but ran on cores from different sockets.

Another limitation of this work is that we have only presented the mean values of the various metrics with the mean computed over time and data sources. We have observed that these mean values do not differ greatly between the shared and exclusive jobs, but is possible that the average values of the metrics are not a good indicator for job interference, that is, the information loss in the averaging procedure may have thrown useful fine details away. We have the fine-grained time-dependent metrics collected by TACC_stats and we are actively researching the most effective way of processing this large quantity of data.

We have only presented a small set of the available metrics collected by TACC_stats. We have not yet done a detailed analysis of the various hardware counter statistics. Previous studies have shown that contention for the CPU caches and memory bandwidth is a significant source of job interference. With this in mind, we plan to look at the CPU cache miss rates, CPU stall cycles and average FLOPs to see what, if any, differences can be observed between shared and exclusive jobs. These data are collected on CCR Rush, but the available hardware counters differ on the different nodes, which complicates the analysis. The data can still be effectively analyzed by making sure to only compare jobs that ran on the same hardware or by comparing carefully chosen normalized metrics for each job (such as % of available memory bandwidth rather than absolute value).

We have presented all the jobs running on CCR Rush in the same plots, however, not all jobs have the same performance characteristics and different categories of jobs likely interact with each other in different ways. We collect information about the running processes for each job so we are able to determine the applications being run. With this information we should be able compare different applications and see how they interact with each other if they share nodes. This information could be used to feed into scheduler algorithm design.

The metrics presented here do not provide information about how the job wall clock time is impacted by sharing. One way of estimating this is to run multiple identical jobs on both shared exclusive nodes and compare the duration. The existing "application kernels" could be used for this purpose. The applications kernels were designed to monitor the system performance and so are configured to run exclusively by default, however, the configuration could be modified to run more frequently and to allow node sharing for select runs.

Although many of the analyses presented in this paper were an *ad hoc* processing of TACC_Stats data, ultimately we are just completing the process of validating and ingesting this TACC_Stats data into the XDMoD data warehouse. In the near future, this analytical capability will be available to the data center director or users automatically through the XDMoD framework.

6. ACKNOWLEGMENT

This work is supported by the National Science Foundation under grant number OCI 1203560 for SUPReMM and grant number OCI 1025159 for the technology audit service for XSEDE.

REFERENCES

[1] Breslow, A.D., Tiwari, A., Schulz, M., Carrington, L., Tang, L., Mars, J., 2013 "Enabling Fair Pricing on HPC Systems with Node Sharing", 2013, *SC '13: International Conference for High Performance Computing, Networking, Storage and Analysis.*

[2] M. J. Koop, M. Luo, and D. K. Panda. Reducing network contention with mixed workloads on modernmulticore, clusters. In Cluster Computing and Workshops, 2009. CLUSTER'09. IEEE International Conference on. IEEE, 2009.

[3] C. Iancu, S. Hofmeyr, F. Blagojevic, and Y. Zheng. Oversubscription on multicore processors. In Parallel Distributed Processing (IPDPS), 2010 IEEE International Symposium in, April 2010.

[4] A. D. Breslow, L. Porter, A. Tiwari, M. Laurenzano, L. Carrington, D. M. Tullsen, and A. E. Snavely. The case for colocation of hpc workloads. Concurrency and Computation: Practice and Experience, 2013.

[5] Furlani, T.R., Jones, M.D., Gallo, S.M., Bruno, A.E., Lu, C.-D., Ghadersohi, A., Gentner, R.J., Patra, A., DeLeon, R.L., von Laszewski, G., Wang, F., and Zimmerman, A., "Performance metrics and auditing framework using application kernels for high performance computer systems," Concurrency and Computation: Practice and Experience, Vol 25, Issue 7, p 918, (2013), [Online]. Available: http://dx.doi.org/10.1002/cpe.2871

XDMoD: http://xdmod.ccr.buffalo.edu;

[6] Furlani, T.R., Schneider, B.I., Jones, M.D., Towns, T., Hart, D.L., Gallo, S.M., DeLeon, R.L., Lu, C-D., Ghadersohi, A., Gentner, R.J., Patra, A.K., von Laszewski, G., Wang, F., Palmer, J.T., Simakov, N., 2013. "Using XDMoD to facilitate XSEDE operations, planning and analysis", In *Proceedings of the Conference on Extreme Science and Engineering Discovery Environment: Gateway to Discovery* (XSEDE '13). ACM, New York, NY, USA, , Article 46 , 8 pages.DOI=10.1145/2484762.2484763

[7] C. D. Lu, J. Browne, R. L. DeLeon, J. Hammond, W. Barth, T. R. Furlani, S. M. Gallo, M. D. Jones, and A. K. Patra. 2013. "Comprehensive job level resource usage measurement and analysis for XSEDE HPC systems", In Proceedings of the Conference on Extreme Science and Engineering Discovery Environment: Gateway to Discovery (XSEDE '13). ACM, New York, NY, USA, Article 50, 8 pages. DOI=10.1145/2484762.2484781 http://doi.acm.org/10.1145/2484762.2484781

[8] . C. Browne, R. L. DeLeon, C. D. Lu, M. D. Jones, S. M. Gallo, A. Ghadersohi, A. K. Patra, W. L. Barth, J. Hammond, T. R. Furlani, R. T. McLay, "Enabling Comprehensive Data-Driven System Management for Large Computational Facilities", In Proceedings of SC13: International Conference for High Performance Computing, Networking, Storage and Analysis (SC '13). ACM, New York, NY, USA, Article 86, 11 pages. DOI=10.1145/2503210.2503230 http://doi.acm.org/10.1145/2503210.2503230

[9] Hammond, J. "TACC_stats: I/O performance monitoring for the intransigent" In *2011 Workshop for Interfaces and Architectures for Scientific Data Storage (IASDS 2011)*

[10] http://github.com/TACCProjects/tacc_stats

[11] Morris A. Jette, Andy B. Yoo, Mark Grondona, 2002 SLURM: Simple Linux Utility for Resource Management –In Lecture Notes in Computer Science: Proceedings of Job Scheduling Strategies for Parallel Processing (JSSPP) 2003

[12] D. Eklov, N. Nikoleris, D. Black-Scha_er, and E. Hagersten. Cache pirating: Measuring the curse of the shared cache. In Parallel Processing (ICPP), 2011 International Conference on. IEEE, 2011.

[13] D. Eklov, N. Nikoleris, D. Black-Scha_er, and E. Hagersten. Bandwidth bandit: Understanding memory contention. In Performance Analysis of Systems and Software (ISPASS), 2012 IEEE International Symposium on. IEEE, 2012.

[14] Open XDMoD: http://xdmod.sourceforge.net/