Comprehensive Job Level Resource Usage Measurement and Analysis for XSEDE HPC Systems

Charng-Da Lu*, James Browne ‡, Robert L. DeLeon*, John Hammond†, William Barth†, Thomas R. Furlani*, Steven M. Gallo*, Matthew D Jones*, Abani K. Patra*

*Center for Computational Research, SUNY at Buffalo, Buffalo, NY

‡Department of Computer Science, University of Texas, Austin, TX

†Texas Advanced Computing Center, University of Texas, Austin, TX

clu2@fastmail.us, {rldeleon, furlani, smgallo, jonesm, abani}@buffalo.edu

browne@cs.utexas.edu

john.hammond@intel.com, bbarth@tacc.utexas.edu

ABSTRACT

This paper presents a methodology for comprehensive job level resource use measurement and analysis and applications of the analyses to planning for HPC systems and a case study application of the methodology to the XSEDE Ranger and Lonestar4 systems at the University of Texas. The steps in the methodology are: System-wide collection of resource use and performance statistics at the job and node levels, mapping and storage of the resultant job-wise data to a relational database which eases further implementation and transformation of data to the formats required by specific statistical and analytical algorithms. Analyses can be carried out at different levels of granularity: job, user, or system-wide basis. Measurements are based on a novel lightweight job-centric measurement tool "TACC_Stats" [1], which gathers a comprehensive set of metrics on all compute nodes. The data mapping and analysis tools will be an extension to the XDMoD project [2] for the XSEDE community. This paper also reports the preliminary results from the analysis of measured data for Texas Advanced Computing Center's Lonestar4 and Ranger supercomputers. The case studies presented indicate the level of detailed information that will be available for all resources when TACC_Stats is deployed throughout the XSEDE system. The methodology can be applied to any system that runs the TACC_Stats measurement tool.

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

TACC_Stats, XDMoD, XSEDE, system management, performance analysis, usage analysis

XSEDE '13, July 22 - 25 2013, San Diego, CA, USA

1. INTRODUCTION

High-performance computing (HPC) systems are a complex combination of software, processors, memory, networks, and storage systems each of which is characterized by frequent disruptive technological advances. In this environment, system managers, users and sponsors find it difficult if not impossible to know if optimal performance of the infrastructure is being realized, or even if all subcomponents are functioning properly. HPC centers and their users, at least those HPC centers with open source software stacks, have been to some extent "flying blind", without complete information on system behavior. Anomalous behavior has to be manually diagnosed and remedied with incomplete and sparse data. In addition, many if not most HPC centers lack knowledge of the distribution and performance of the applications running on their systems and therefore have no way of knowing if the applications are well tuned to the architecture and making efficient use of the resource. This is especially worrisome since most HPC systems are purchased based on performance on a projected job mix, which may in fact be significantly different from what is actually experienced. Furthermore, the information generated by this methodology is useful not only to help ensure that the resources are being used optimally but also for future system purchases, where this information can aide in making design decisions.

It also has been effort-intensive for users to assess the effectiveness with which they are using the available resources. The data available for system level analyses appear from multiple sources and in disparate formats (from Linux "sysstat" [3] and accounting to scheduler/kernel logs). There are also many user-oriented performance instrumentation and profiling tools but most require extensive system knowledge, code changes and recompilation, and thus are not widely used. This paper reports on the use of the TACC_Stats tool to surmount these difficulties and provides a system and user level audit of resource usage.

TACC_Stats is a job-oriented and logically structured enhancement to the conventional Linux "sysstat" system-wide performance monitor. In addition to the information gathered by systat, TACC_Stats records hardware performance counter values, parallel filesystem metrics, and high-speed interconnect usage. Additionally TACC_Stats resolves the measurements by job. The core component is a collector executed on every compute node, both at the beginning and end of each job (via batch scheduler prolog and epilog scripts) and at periodic intervals (via cron). Like the "sar" (System Activity Reporter) program in sysstat package, the performance collection takes place in the background, incurs very low overhead, and requires no user intervention. TACC_Stats is open source and is available for download at: https://github.com/TACCProjects/tacc_stats. The

rest of the paper includes a detailed description of TACC_Stats and results from a usage analysis on two XSEDE resources Lonestar4 and Ranger.

2. RESOURCE MEASUREMENT

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 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 [4] 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 (InfiniBand and Myrinet), and CPU hardware performance counters. For a complete list of the data acquired by TACC_Stats, see the TACC_Stats web site [4].

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. 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.

3. USAGE ANALYSIS

The case studies reported in this paper were carried out on the Ranger and Lonestar4 supercomputers at the Texas Advanced Computing Center (TACC). Ranger (decommissioned as of February 2013) is a Linux cluster comprising of 3936 nodes, each of which has four 2.3GHz AMD Opteron quad-core processors (16 cores in total) and 32 GB of memory. The filesystem is Lustre and the interconnect is 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. The Lonestar4 has two filesystems: Lustre and NFS and its interconnect is InfiniBand (NFS is connected via Ethernet).

TACC_Stats has been deployed on Ranger for 20 months and Lonestar4 for 14 months. On Ranger it generates a raw data file of 0.5 MB per node per day and collectively 60 GB (uncompressed) or 20 GB (compressed) for the entire cluster per month.

We analyzed TACC_Stats data collected on Ranger during June 2011 to January 2013 with a total of 521,010 jobs and Lonestar4 data from November 2011 to January 2013 with a total of 337,011 jobs. The jobs included in this study are those longer than the default TACC_Stats sampling interval of 10 minutes. The job pool for our analysis also contains non-TeraGrid/XSEDE jobs (e.g. from TACC partners and Texas higher educational institutions). We ingested both the raw TACC_Stats output files and job accounting information into IBM Netezza data warehouse appliances and a MySQL database, respectively.

The following analysis will be mainly concerned with seven key TACC_Stats metrics and closely related quantities: cpu_user, mem_used, cpu_flops, io_scratch_read, io_work_read, net_ib_rx, and numa_hit. We have chosen these seven based on a correlation analysis over *all* of the measured metrics. We found that there are many highly correlated or anti-correlated metrics, such as cpu_user is negatively correlated to cpu_idle, or net_ib_rx is positively correlated to net_ib_tx. Therefore, we have selected the the smallest independent set of metrics that describe the execution behavior of the job mix on each system.

The explanation of the seven metrics is as follows. Cpu_user is that fraction of the CPU utilized by the job in user space. Aside from some small overhead CPU usage in kernel space or in IO wait state, 1-cpu_user gives the approximate fraction of the CPU that is idle. Mem_used refers to the per node memory used, including the disk buffer and cache managed by the Linux operating system Mem_used_max refers to the peak observed memory usage over all used nodes and all sampling intervals of a job. Cpu_flops is the floating-point operations per second produced on a job. Io_scratch_read and io_work_read refer to reading from the scratch and work file systems (Lustre). The difference of these arises in the purge policy and quota size. “Scratch” is purged periodically and has a largish quota to the tune of hundreds of TB, and “work” is non-purged space with a 200GB quota. Net_ib_rx records the received messages in bytes per second on the InfiniBand network, which includes both application-induced packets and Lustre filesystem’s traffic. NUMA_hit (NUMA=Non-Uniform Memory Access) refers to the successful access of main memory closest to a particular CPU core by a process running on that core. In contrast, a NUMA_miss is where a process accessed memory from main memory connected through another CPU socket/memory interface.

3.1 Lonestar4 and Ranger Job Duration and Job Size

Although not uniquely specific to TACC_Stats measurements, the Lonstar4 and Ranger job duration distributions are given in Figures 1 and 2. Figure 1 and 2 and several later figures show the kernel density [5] (produced by the R statistical software environment) rather than a histogram in order to avoid making binning choices. On Lonestar4, most jobs fall into the short category at less than 10,000 seconds. There is a distribution with a secondary maximum at 86,400 seconds (24 hours). For Ranger the distribution for the first 86,400 seconds is similar but the tail is more prominent. There is also another set of jobs running out to a final peak at 172,800 seconds (48 hours), and these jobs are those run from Ranger’s special “long” job queue. The 24 and 48-hour jobs are usually those that try to do as much work as possible before the allotted time exceeds and can restart from checkpoint data. Figures 3 and 4 show the distribution of the number of nodes employed by the jobs on Lonestar4 and Ranger, respectively. The great majority of the jobs use less than 10 nodes with only a short tail out to larger job sizes. Clearly, the secondary peaks are a result of policy. Since, most of the jobs are also small

in the number of nodes it raises the question of possible policy adjustments to allow a few long jobs on smaller nodes to alleviate this bottleneck in system usage.

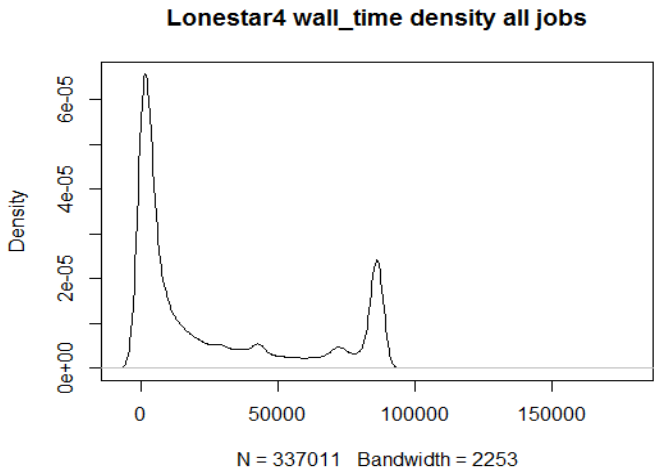

Figure 1: Distribution of Lonestar4 jobs by wall time. Units are seconds. The rightmost peak at 86,400 seconds corresponds to 24 hour jobs.

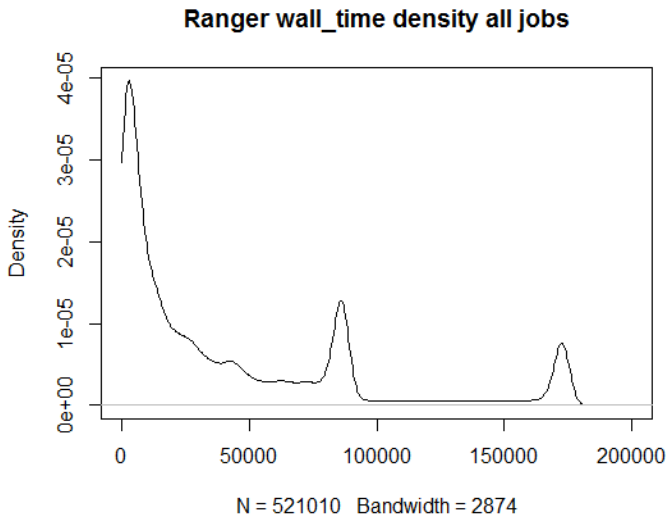

Figure 2: Distribution of Ranger jobs by wall time. Units are seconds. The two rightmost peaks correspond to jobs of 24 and 48 hours, respectively.

3.2 CPU and Memory usage and NUMA efficiency

Figures 5 and 6 show that most jobs on Lonestar4 and Ranger use the CPU efficiently. Most jobs are clustered in the 0.95-1.00 fractional efficiency range. Note the clustering of jobs on vertical lines corresponding to particular values of cpu_user. These are jobs that use only a fraction of the cores; each peak indicates a specific number of cores used per node. For Ranger each node is equipped with 16 cores. The peaks appear at 0.75 (12 cores), 0.50 (8 cores), 0.25 (4 cores), and 0.0625 (1 core). For Lonestar4 each node is equipped with 12 cores. The peaks appear at 0.667 (8 cores), 0.50 (6 cores), 0.333 (4 cores) and 0.083 (1 core). For Lonstar4 there is also a very strong cluster at 0.042 corresponding to ½ core.

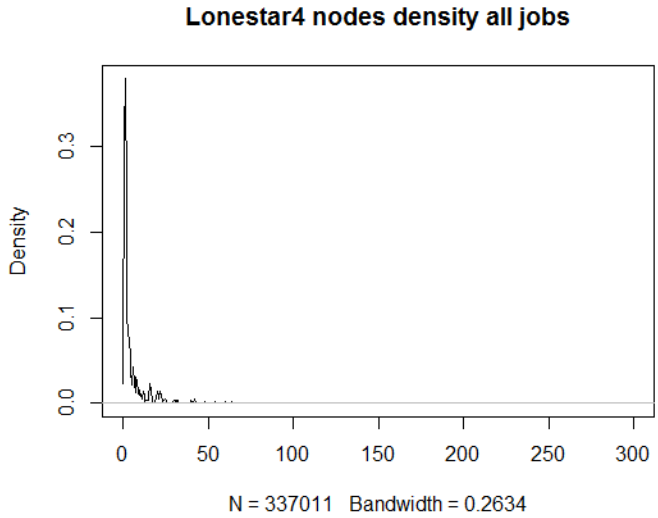

Figure 3: Lonestar4 job size distribution in number of nodes.

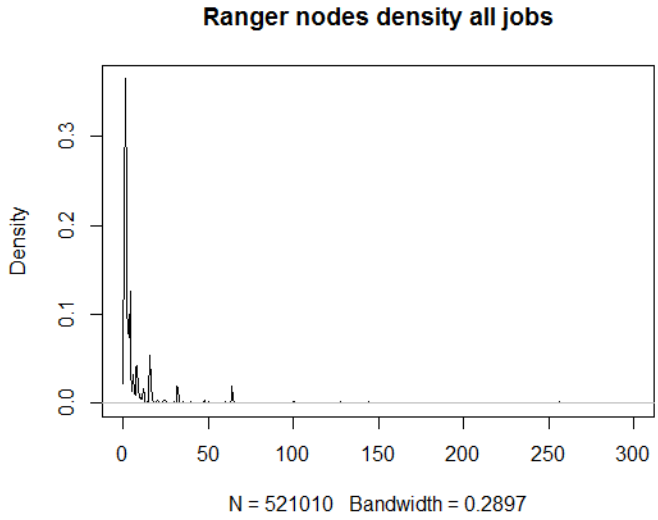

Figure 4: Ranger job size distribution in number of nodes**.**

Memory usage varies greatly by job on both Lonstar4 and Ranger, as shown in Figures 7 and 8. However, most jobs use only a small fraction of the node's available memory of 24 GB on Lonestar4 and 32 GB on Ranger. Given the total cost of memory for large clusters, information such as this would be useful during the design phase of system purchase.

A NUMA efficiency can be defined as:
numa_eff = numa_hit/(numa_hit + numa_miss).

Using this definition, we can look at the distribution of numa_eff of Lonestar4 and Ranger jobs. The Lonestar4 numa_eff, see Figure 9, is very sharply peaked near unity indicating generally good NUMA performance. On Ranger, see Figure 10, most jobs are also close to unit efficiency but the distribution is broader with a small tail of less efficient jobs.

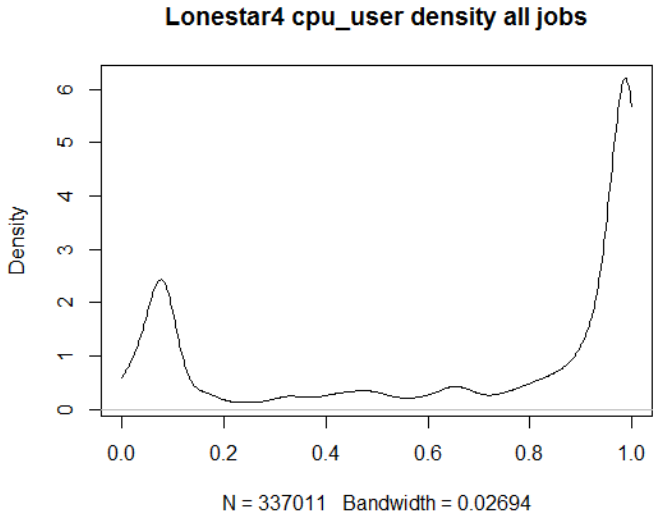

Figure.5 Fraction of CPU used by Lonestar4 jobs.

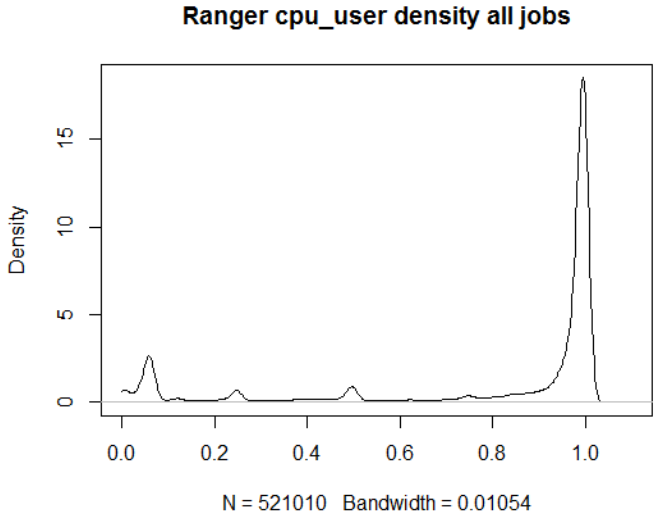

Figure 6. Fraction of CPU used by Ranger jobs.

3.3 CPU and Memory usage by top applications

For Lonestar4 and Ranger we also analyzed the CPU and memory efficiency for the most prominently used applications based on node-hours. The application information is captured by the MPI job launcher "ibrun" on the two systems. It records the job id, the executable binary's full path name, checksum of the binary, as well as all of its dependent dynamic linked libraries. We map the executable binaries to a list of well-known HPC applications using simple regular expression matching on their full path names. It should be noted that for most jobs the executable binaries are the user's own code and cannot be identified as known HPC applications.

The top 15 applications by node-hour for Lonestar4 and Ranger are shown in Figures 11 and 12, respectively. The memory efficiency is calculated as the peak used memory divided by available memory. Note, however, the used memory includes Linux-managed disk buffers and cache, so there could be overestimation. In general, the CPU usage efficiency is very high, with the memory usage much lower as expected from Figures 7 and 8. Very few of the largest consumers of node-hours have low CPU use fractions.

In general, the molecular dynamic codes show very good CPU efficiency while requiring only moderate amounts of memory. On Lonestar4, the only widely used codes with relatively low CPU fractional use are CHARMM [6] and NWCHEM [7] both of which use less than 0.7 of the CPU. On Ranger, SIESTA [8] and CHARMM show the lowest CPU and memory use, each using less than 2/3 of the CPU and less than 1/3 of the memory.

3.4 Seven key TACC_Stats metrics on Lonestar4 and Ranger

In this section the performance characteristics of the two machines under real usage are compared. We have chosen to display seven key metrics on radar charts. In general, the charts have been formulated such that a comparison of these metrics for similar

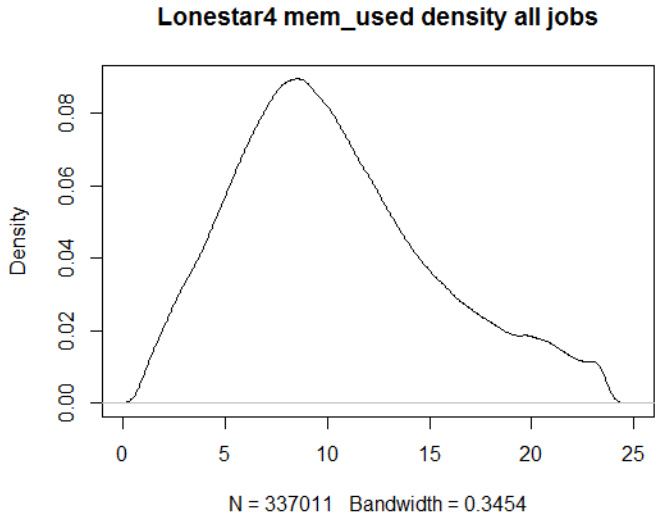

Figure 7. Distribution of memory used in Gbytes for Lonestar4 jobs.

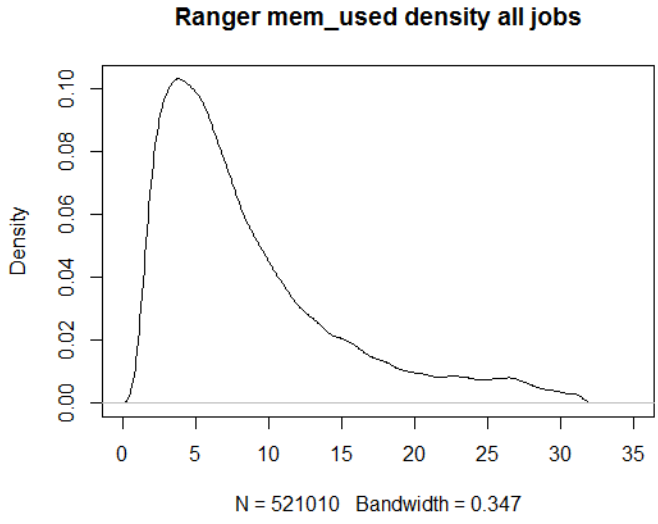

Figure 8. Distribution of memory used in Gbytes for Ranger jobs.

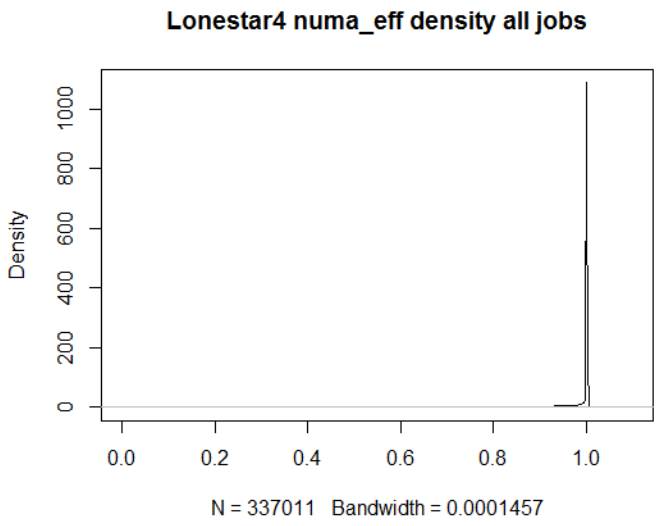

Figure 9. NUMA efficiency of Lonestar4 jobs.

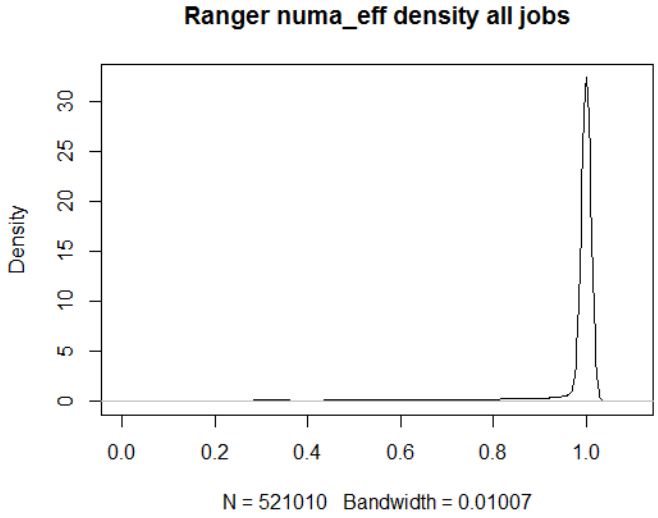

Figure 10. NUMA efficiency of Ranger jobs.

cases would produce a regular polygon with all metrics equal to one. Numbers much greater than or less than one indicate a sizable difference for that metric in the comparison. Figure 13 is a radar chart comparing Lonestar4 and Ranger using six of the seven TACC_Stats metrics (cpu_user, io_scratch_read, io_work_read, mem_used_max, net_ib_rx and numa_hit) that describe the CPU usage, network usage and memory usage on the two systems. Unfortunately, FLOPS on Lonestar4 and Ranger cannot be compared directly since the counters used record different events on different architectures (Intel vs. AMD). Aside from the much smaller InfiniBand usage and the slightly larger io_scratch read and io_work_read on Lonestar4, the two systems are comparable for the other metrics. The remainder of the plots in this section show the ratios of all seven metrics.

Figures 14 and 15 show a comparison of the short jobs (<10,000 seconds) on Ranger to the long and very long jobs. The long jobs and very long jobs are very similar but differ significantly from

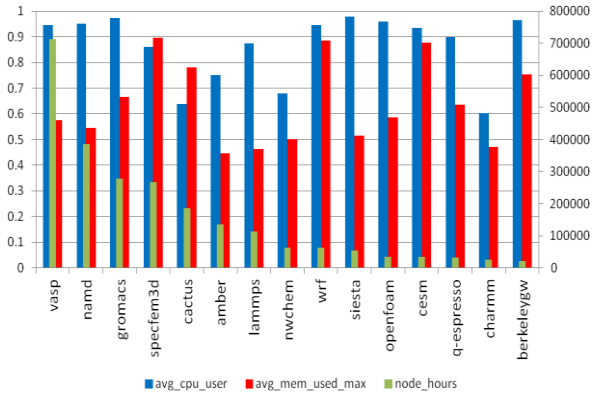

Figure 11. Application's CPU and memory efficiency on Lonestar4 sorted by node-hour. The right Y axis is the accumulated node-hours.

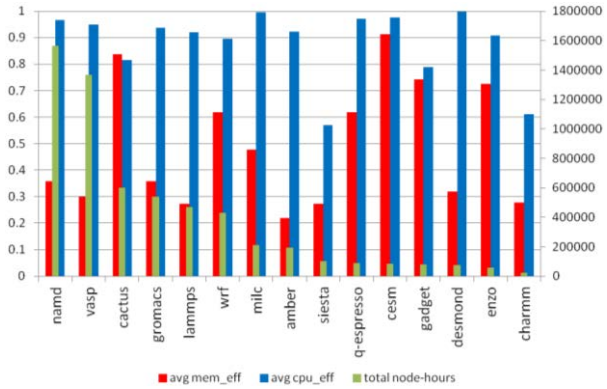

Figure 12. Application's CPU and memory efficiency on Ranger sorted by node-hour. The right Y axis is the accumulated node-hours.

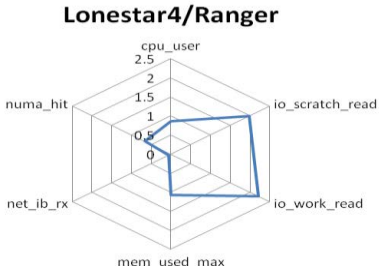

Figure 13 Comparison of Lonestar4 and Ranger using key tacc_stats metrics. A value of 1.0 on the radar chart for a given metric indicates identical performance for that metric on Lonestar4 and Ranger.

the short jobs. Compared to the short jobs, they tend to produce 50% more FLOPS, be slightly more efficient on CPU usage, have similar memory usage and have 50% less io_scratch usage, io_work read and numa_hits. Comparing Figures 14 and 16, showing long vs short jobs on Ranger and Lonestar4, shows a substantial difference in FLOPS. Lonestar4 long jobs produce significantly less FLOPS than the short jobs. Like Ranger, the

numa_hits are lower on the large jobs on Lonestar4 than the small jobs.

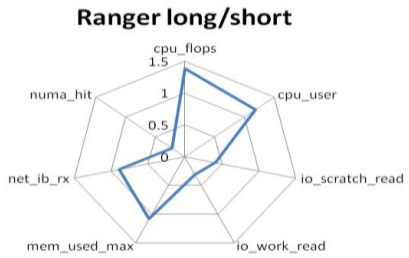

Figure 14. Comparison of the short jobs (<10,000 seconds) to the long jobs (10,000-84,600 seconds) on Ranger. Long jobs tend to produce higher FLOPS and be slighly more efficient in CPU usage.

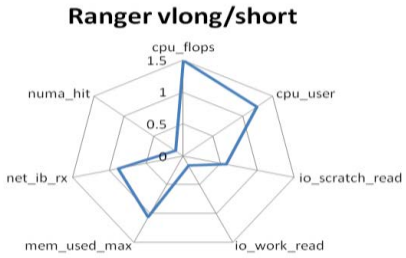

Figure 15. Comparison of the short jobs (<10,000 seconds) to the very long jobs (>84,600 seconds) on Ranger.

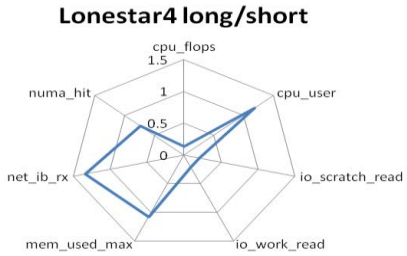

Figure 16. Comparison of the short jobs (<10,000 seconds) to the long jobs (>10,000 seconds) on Lonestar4.

Figures 17 and 18 show a comparison of the large jobs (greater than 20 nodes) with the small jobs (less than 20 nodes) on Ranger and Lonestar4, respectively. The patterns are quite different on the two systems with the large jobs on Lonestar4 doing much more io_work_read, but this is more a reflection of the large variance in io_work_read caused by a few large IO usage jobs rather than a difference between the two systems. For both systems the large jobs have more numa_hits, less FLOPS lower CPU usage and comparable memory usage.

3.5 Field of Science comparison on Lonestar4 and Ranger

We have compared the Lonestar4 and Ranger usage for jobs in different fields of science using the same set of seven key

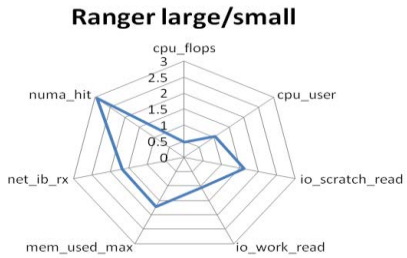

Figure 17. Comparison of the small jobs (≤20 nodes) to the large jobs (>20 nodes) on Ranger.

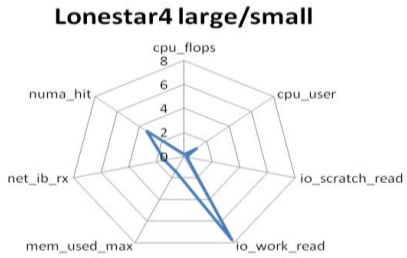

Figure 18. Comparison of the small jobs (≤20 nodes) to the large jobs (>20 nodes) on Lonestar4.

TACC_Stats metrics. The field of science is based on the allocation name of each job. On Ranger and Lonestar4, TeraGrid/XSEDE jobs have a prefix of TG-XXX in the allocation name, and XXX is the short code for NSF-designated field of science. Figures 19 to 22 show the comparison of Physics, Chemistry, Molecular Biology and Atmospheric Science with the total all usage on Ranger. Figures 23 to 26 show comparable plots for Lonestar4.

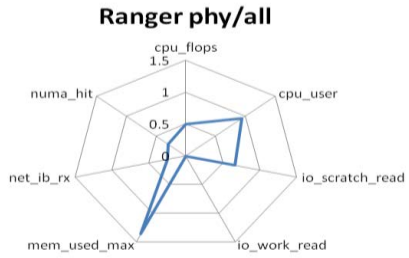

Figure 19. Comparison of Physics usage with total Ranger XSEDE usage for seven metrics.

Physics usage is similar on the two systems showing lower flops, low io_work_read, less IB network uasge and more memory usage. They differ in that the CPU usage is very low on Lonestar4 and the numa_hits are lower on Ranger. Chemistry usage is similar on both systems in that the outstanding feature is the large IB network usage compared to all other jobs. Molecular biology usage on Ranger and Lonestar both show low io_scratch usage and low numa_hits. On Lonestar4 the io_work_read is also low and the FLOPS are very much lower, in stark contrast to the Ranger data. Atmospheric science usage on Lonestar4 and Ranger share low FLOPS, low io_work_read, low numa_hits and low IB network usage. They only differ in that io_work_read is higher on Lonestar4 and lower on Ranger.

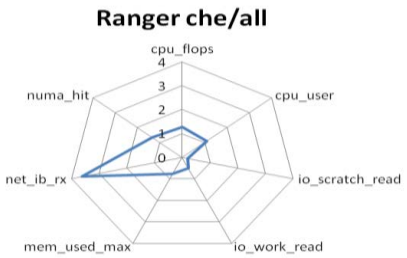

Figure 20. Comparison of Chemistry usage with total Ranger XSEDE usage for seven metrics.

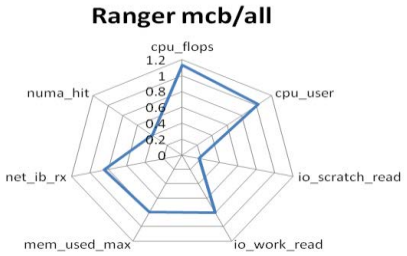

Figure 21. Comparison of Molecular Biology usage with total Ranger XSEDE usage for seven metrics.

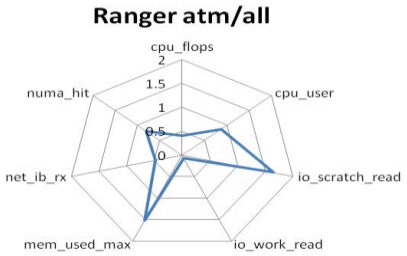

Figure 22. Comparison of Atmospheric Science usage with total Ranger XSEDE usage for seven metrics.

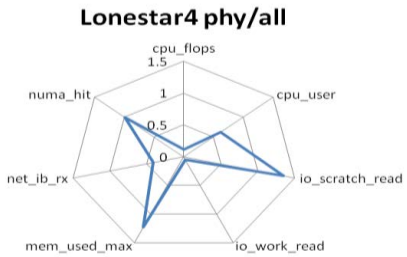

Figure 23. Comparison of Physics usage with total Lonestar4 XSEDE usage for seven metrics.

4. RELATED WORK

Del Vento *et al.* [9] adopted a similar approach to tackling inefficient HPC resource utilization. The goal in [9] was primarily optimization of user codes and applications. The system reported here does support identification of resource use anomalies, but it has much broader goals and capabilities, targeting information [9] reports, that once problems were identified by their monitoring requirements of all stakeholders. Additionally, the monitoring and analysis system in [9] is based on proprietary systems (IBM POWER/AIX) while the system reported here is for HPC clusters

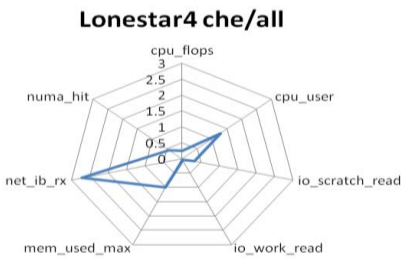

Figure 24. Comparison of Chemistry usage with total Lonestar4 XSEDE usage for seven metrics.

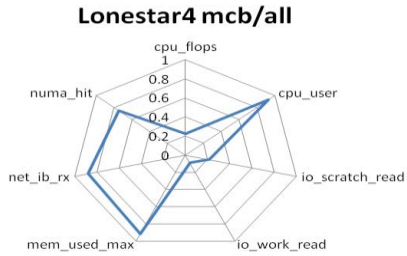

Figure 25. Comparison of Molecular Biology usage with total Lonestar4 XSEDE usage for seven metrics.

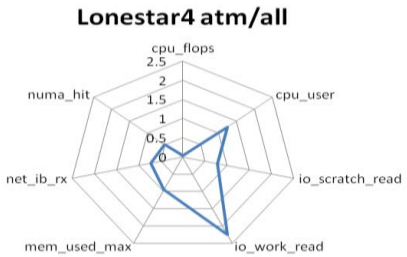

Figure 26. Comparison of Atmospheric Science usage with total Lonestar4 XSEDE usage for seven metrics.

based Linux-based open source software. [9] reports, that once problems were identified by their monitoring software, they were, with collaboration for users, able to improve CPU usage (e.g. correct process affinity/CPU binding) and troubleshoot thorny issues (e.g. memory leak). Once a job or application with a pattern of inefficient use of one or more resources has been identified by our analyses and reports, we recommend that the user or application developer (with collaboration from consulting staff) apply one of the many performance optimization tools available for open source cluster, for example: Tau[10,11,12], HPCToolkit[13,14], Scalasca[15] and PerfExpert[16,17].

5. DISCUSSION & FUTURE WORK

The primary focus of this paper has been to demonstrate the ability of a TACC_Stats based resource analyses to provide detailed job and system level information and analytics of the workload running on a given HPC resource. While characterizing resource utilization at the job level and system levels is useful in its own right, the real power of the analyses based on TACC_Stats lies in the ability to tune system and application performance based on this data to improve overall resource utilization. Given the oversubscription of most if not all HPC resources, this capability is particularly desirable. Accordingly, future work will

center on a detailed investigation of some of the application and system inefficiencies uncovered by the study carried out here.

For example, we are contacting the owners of particularly poorly running jobs to better understand their workflow and determine if steps can be taken to mitigate the underperformance. These steps are likely to include a wide range of remedies, some simple to implement and others requiring a more substantial effort. For example, given the high operational efficiency of the molecular dynamics package NAMD relative to other widely used MD packages (as shown in Figures 11 and 12), HPC centers might choose to encourage users to consider NAMD for their simulations. Furthermore, although it is hardly surprising to learn that some applications run considerably better on certain machine architectures, with TACC_Stats we can easily identify those applications and provide incentives for users to run on architectures best suited for their application. Additionally, a very similar analysis to that shown in Figures 11 and 12 of CPU and memory efficiency can be done for individual users, as opposed to applications, to determine which users are using the resources efficiently and which are not. Such quantitative information can be input for a more appropriate allocation for such users.

The data, especially the comparative analysis across architectures and usage classes raise interesting questions. For example one could argue that given the very different demands placed on machines by users from different fields of science (Fig. 19-26) XSEDE should consider providing a "bouquet" of machines tuned to different user groups rather than the monolithic general purpose machines of today.

Although we only briefly alluded to the challenges of analyzing the data generated by a TACC_Stats, we are assessing various technologies (e.g. NoSQL) to quickly process, store, and query massive TACC_Stats data. This is critical, as it is a key step to developing a capability to rapidly import TACC_Stats data into XDMoD, which will greatly expand its access to end users, systems administrators, and center directors. As a first step, we are presently incorporating the Lonestar4 and Ranger data into XDMoD; the next release at XSEDE13 will feature a TACC_Stats data realm and will therefore be widely available.

TACC_Stats will soon be deployed on TACC's Stampede and ultimately on all XSEDE resources. This will require identification of an "equivalent" set of performance counters on each architecture and extraction of parameters from some software systems such as the scheduler and file systems used in each system. Finally, in order to have a broad impact on the efficient operation of HPC resources throughout the U.S., including academic and industrial HPC centers, plans are underway to streamline the installation of TACC_Stats and the ingestion of its data files into XDMoD.

6. ACKNOWLEGMENT

This research is supported by the National Science Foundation under award numbers 1203560, 1203604, 0959870, and 1025159.

REFERENCES

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

[2] 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, pp.n/a–n/a, 2012. [Online]. Available: http://dx.doi.org/10.1002/cpe.2871

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

[3] http://sebastien.godard.pagesperso-orange.fr

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

[5] Scott, D. W. "Multivariate Density Estimation". Wiley, New York, 1992.

[6] Brooks, B., *et al*. "CHARMM: The biomolecular simulation program." J. Comput. Chem. Vol 30, pages 1545-1614, 2009

[7] Valiev, M., *et al*. "NWChem: a comprehensive, scalable open-source solution for large scale molecular simulations." Comput. Phys. Commun. Vol 181, page 1477, 2010

[8] Soler, J. M., *et al*. "The SIESTA method for ab initio order-N materials simulation." J. Physics: Condensed Matter. Vol 14, No 11, 2002

[9] Del Vento, D., Engel, T., Ghosh S., Hart, D., Kelly, R., Liu, S. and Valent, R. "System-level monitoring of floating-point performance to improve effective system utilization." In *2011 International Conference for High Performance Computing, Networking, Storage and Analysis* (*SC11*)

[10] K. A. Huck, A. D. Malony, S. Shende, and A. Morris. "Knowledge Support and Automation for PerformanceAnalysis with PerfExplorer 2.0." Large-Scale ProgrammingTools and Environments, Special Issue of Scientific Programming, vol. 16, no. 2-3, pp. 123-134. 2008.

[11] S. Shende and A. Malony. "The Tau Parallel Performance System." International Journal of High Performance Computing Applications, 20(2): 287-311.

[12] Tau: http://www.cs.uoregon.edu/research/tau/home.php.

[13] N. R. Tallent, J. M. Mellor-Crummey, L. Adhianto, M.W. Fagan, and M. Krentel. "HPCToolkit: performance tools for scientific computing." Journal of Physics: Conference Series, 125. 2008.

[14] HPCToolkit: http://www.hpctoolkit.org/.

[15] M. Geimer, P. Saviankou, A. Strube, Z. Szebenyi, F. Wolf, B. J. N. Wylie: Further improving the scalability of the Scalasca toolset. In Proc. of PARA 2010: State of the Art in Scientific and Parallel Computing, Part II: Minisymposium Scalable tools for High Performance Computing, Reykjavik, Iceland, June 6–9 2010, volume 7134 of Lecture Notes in Computer Science, pages 463–474, Springer, 2012.

[16] M. Burtscher, B.D. Kim, J. Diamond, J. McCalpin, L. Koesterke, and J. Browne. "PerfExpert: An Easy-to-Use Performance Diagnosis Tool for HPC Applications." SC2010 Int. Conference for High-Performance Computing, Networking, Storage and Analysis. November 2010.

[17] PerfExpert: http://www.tacc.utexas.edu/perfexpert/.