CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2014; **26**:2191–2209
Published online 6 March 2014 in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/cpe.3245

SPECIAL ISSUE PAPER

Comprehensive, open-source resource usage measurement and analysis for HPC systems[‡,§]

James C. Browne[1], Robert L. DeLeon[2,*,†], Abani K. Patra[3], William L. Barth[4], John Hammond[4], Matthew D. Jones[2], Thomas R. Furlani[2], Barry I. Schneider[5], Steven M. Gallo[2] Amin Ghadersohi[2], Ryan J. Gentner[2], Jeffrey T. Palmer[2], Nikolay Simakov[2], Martins Innus[2], Andrew E. Bruno[2], Joseph P. White[2], Cynthia D. Cornelius[2], Thomas Yearke[2], Kyle Marcus[2], Gregor von Laszewski[6] and Fugang Wang[6]

[1]*Department of Computer Science, University of Texas, Austin, TX 78758, USA*
[2]*Center for Computational Research, University at Buffalo, State University of New York, 701 Ellicott Street, Buffalo, NY 14203, USA*
[3]*Mechanical and Aerospace Engineering Department, University at Buffalo, State University of New York, Amherst, NY 14260, USA*
[4]*Texas Advanced Computing Center, University of Texas, Austin, TX 78758, USA*
[5]*National Institute of Science and Technology, Gaithersburg, MD, USA*
[6]*Pervasive Technology Institute, Indiana University, 2719 East 10th Street, Bloomington, IN 47408, USA*

SUMMARY

The important role high-performance computing (HPC) resources play in science and engineering research, coupled with its high cost (capital, power and manpower), short life and oversubscription, requires us to optimize its usage – an outcome that is only possible if adequate analytical data are collected and used to drive systems management at different granularities – job, application, user and system. This paper presents a method for comprehensive job, application and system-level resource use measurement, and analysis and its implementation. The steps in the method are system-wide collection of comprehensive resource use and performance statistics at the job and node levels in a uniform format across all resources, mapping and storage of the resultant job-wise data to a relational database, which enables further implementation and transformation of the data to the formats required by specific statistical and analytical algorithms. Analyses can be carried out at different levels of granularity: job, user, application or system-wide. Measurements are based on a new lightweight job-centric measurement tool 'TACC_Stats', which gathers a comprehensive set of resource use metrics on all compute nodes and data logged by the system scheduler. The data mapping and analysis tools are an extension of the XDMoD project. The method is illustrated with analyses of resource use for the Texas Advanced Computing Center's Lonestar4, Ranger and Stampede supercomputers and the HPC cluster at the Center for Computational Research. The illustrations are focused on resource use at the system, job and application levels and reveal many interesting insights into system usage patterns and also anomalous behavior due to failure/misuse. The method can be applied to any system that runs the TACC_Stats measurement tool and a tool to extract job execution environment data from the system scheduler. Copyright © 2014 John Wiley & Sons, Ltd.

Received 28 January 2014; Revised 3 February 2014; Accepted 4 February 2014

KEY WORDS: TACC_Stats; XDMoD; XSEDE; system management; performance analysis; usage analysis; SUPReMM; HPC resource management

*Correspondence to: Robert L. DeLeon, Center for Computational Research, University at Buffalo, State University of New York, 701 Ellicott Street, Buffalo, NY 14203, USA.
[†]E-mail: rldeleon@buffalo.edu
[‡]Categories and Subject Descriptors.
[§]C.4 [Performance of Systems]: *Design studies, Fault tolerance, Measurement techniques, Modeling techniques, Performance attributes, Reliability, availability and serviceability.*

1. INTRODUCTION

High-performance computing (HPC) systems are complicated combinations of software, processors, memory, networks and storage systems, which evolve rapidly with technology changes. The nature of the workload for many HPC systems is also rapidly expanding as the spectrum of disciplines utilizing HPCs grows. For systems using open-source software stacks with Linux as the operating system, the capabilities for measurement of resource use have been inadequate to support effective, comprehensive system management. HPC centers and their users and managers, at least those HPC centers with open-source software stacks, have been, to some extent, 'flying blind', without comprehensive information on system, application or job behavior. Anomalous behavior is usually identified only if there is an egregious impact on system resources and has to be manually diagnosed and remedied with incomplete and sparse data. Many if not most HPC centers lack knowledge of the distribution, performance and resource use characteristics of the applications running on their systems and therefore have no way of knowing if the applications are well tuned and matched to the resource. This is especially worrisome because most HPC systems are purchased based on performance for a projected job mix, which may in fact be significantly different from their actual job mix. It is also difficult for users to assess the effectiveness of their HPC resource use from this sparse information. System managers also need reliable ways to flag low performing jobs that can then be investigated, and performance improvements are made.

In the past, the data available for system-level analyses come from multiple sources and in disparate formats (from Linux 'sysstat' and several other Linux kernel utilities and accounting and scheduler/kernel logs). Most of these measurements are not resolved to the job level. 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 describes and illustrates application of a new and comprehensive resource management system, *S*ystems *U*sage and *P*erformance of *Re*sources *M*onitoring and *M*odeling (SUPReMM). SUPReMM is based on integration of TACC_Stats [1, 2], a comprehensive monitoring and measurement system, with the XDMoD platform [3, 4] for system data analytics.

TACC_Stats is a job-oriented and logically structured enhancement to the conventional Linux 'sysstat' system-wide performance monitor [5]. In addition to the information gathered by systat, TACC_Stats records hardware performance counter values, parallel filesystem metrics and high-speed interconnect usage, resolving the measurements by job and core. The basic 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). Data collection takes place in the background, incurs very low overhead and requires no user intervention. TACC_Stats is open-source and is freely available for download [2]. XDMoD [3, 4] 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. XDMoD ingests and organizes data on computer system behavior and then maps that data into metrics required for all-over system management. XDMoD's capabilities for analysis and reporting are primarily limited by the available data.

The innovations and contributions reported in this paper include the following:

- design of a monitoring system, TACC_Stats, which measures use of all of the resources of an HPC system with resolution to the job and node level with minimal overhead;
- an open-source implementation of this monitoring system customizable to different system configurations;
- integration of the data generated by TACC_Stats into the XDMoD analysis and reporting system to create the SUPReMM system management tool; and
- illustration of resource use analyses by comparison of resource use across systems at job, application, discipline and full system levels. These analyses have not heretofore been available for HPC systems based on open-source software.

Section 2 presents a discussion of previous related work. Section 3 gives an overview of XDMoD and detailed description of resource measurements by TACC_Stats. Section 4 shows the utility of the SUPReMM system management tool through a series of case studies, including analyses at the system, application and job level across three different HPC systems operated by the Texas Advanced Computing Center and the Center for Computational Research. Examples demonstrating its utility to enhance the performance of user jobs are also included. Section 5 discusses the use of the analyses and future work.

2. RELATED WORK

Del Vento *et al*. [6] adopted a similar approach to tackling inefficient HPC resource utilization. The goal in [6] 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 requirements of all stakeholders. Additionally, the monitoring and analysis system in [6] is based on proprietary systems (IBM POWER/AIX), while the system reported here is for HPC clusters based on Linux-based open-source software. Similarly, the OVIS [7, 8] measurement and monitoring system designed for machines from Cray incorporates a lightweight distributed metric service to collect a variety of resource use data, which is then streamed to a remote database for analysis. Analysis comprises largely of outlier detection and failure prediction. Preliminary work on application usage of these data in real time is also reported.

Reference [6] reports, that once problems were identified by their monitoring software, they were, with collaboration with 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 clusters: Tau [9–11], HPCToolkit [12, 13], IPM [14], Open SpeedShop [15], Scalasca [16], VTune [17], Paradyn [18] and PerfExpert [19, 20].

The most nearly comparable open-source system to SUPReMM is the HOPSA [21] project. The goal of HOPSA is to create an integrated system for application and system tuning. It targets generation of performance information for both applications/job and system administrators. HOPSA is based on an integrated tool chain beginning with the LWM^2 monitoring system, which collects data on MPI communication, I/O activity, some metrics for sequential chip-level performance and for heterogeneous nodes with accelerator chips, data on time spent executing CUDA and traffic between the host and device. LMW^2 gathers its data through instrumentation of the application and periodic sampling of performance counters. The data gathered by LMW^2 are stored in a data base. The HOPSA tool chain may then invoke updated versions of several performance optimization tools including ThreadSpotter [22], Scalasca [23], Paraver [24] and Vampir Systems [25]; administrators may directly access the database of performance data.

3. RESOURCE MEASUREMENT AND ANALYSIS TOOLS

3.1. XDMoD framework

Here, we present a brief overview of XDMoD, a more detailed description can be found in References [3, 4]. 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, National Science Foundation (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).

A context-sensitive drill-down capability is available for many charts allowing users to access additional related information simply by clicking inside a plot and then selecting the desired metric to explore. 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.

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 [3, 4]. 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 (less that 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.

While the XDMoD framework was initially designed to meet the needs of XSEDE, much of its functionality is applicable to HPC centers in general. Both XSEDE and other HPC centers typically employ the concept of users, compute jobs, project groups, help desk tickets, scientific publications and one or more HPC resources (e.g., compute cluster and storage pool). Users also fall into a similar hierarchical organizational structure differing only in the terms used to describe nodes in the hierarchy. On XSEDE, users are associated with one or more projects or allocations, each with their own principal investigator. An individual project is associated with an NSF directorate and field of science; an academic HPC center may group users by discipline unit and department; an industrial HPC center may group users by project and department. Similarly, the job-level information collected can include a job name, wall-clock time, start and end dates, memory utilization and other metrics.

Taking advantage of the great similarity of XSEDE and a typical HPC center, we have developed an initial release of Open_XDMoD, the open-source version of XDMoD, which leverages the same code base. In fact, development of the open-source version has led to the refactoring of several parts of

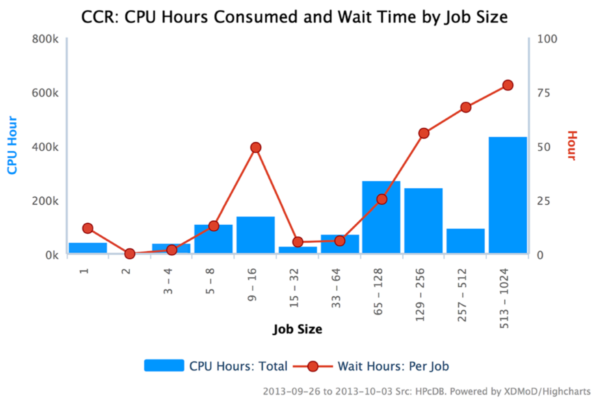

Figure 1. Plot of CPU hours consumed and job wait time versus job size for a 1-week period at the University at Buffalo Center for Computational Research.

XDMoD in order to make it more flexible, easier to maintain and simpler to add new functionality in the future. 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 these 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. Open_XDMoD will be able to inspect the data available in its warehouse and disable the display of metrics that it is not able to reliably generate. For example, if a center does not choose to provide a departmental or project hierarchy for users, then metrics including these will not be displayed.

3.2. *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 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 [2] 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 [2].

Different 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 and SMP/Non-Uniform Memory Access (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 min), 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.

4. CASE STUDIES

In this section, we present the results of several case studies that demonstrate SUPReMM's (XDMoD/TACC_Stats) utility for comprehensive HPC system management. The first case study looks at the ability of XDMoD's application kernels to measure quality of service and help identify underperforming hardware and system software. The second case study demonstrates the utility of TACC_Stats to provide detailed resource utilization metrics for all jobs across a given HPC system. The third case study shows how SUPReMM can be used to automatically identify underperforming applications, which can subsequently be tuned to improve performance.

4.1. Optimizing HPC system operation through application kernels

The HPC center directors have system-related diagnostics, such as network bandwidth utilized, processing loads, number of jobs run and local usage statistics available to characterize their workloads and audit the infrastructure performance. However, this does not provide them with the means to determine how well the computing infrastructure is operating with respect to the actual scientific and engineering applications for which these HPC platforms are designed and operated. Benchmarks cannot fill this gap because in reality, benchmarks are so intrusive that they are not run very often (see, for example, Reference [26] in which the application performance suite is run on a quarterly basis), and in most cases, only when the HPC platform is initially deployed or significantly upgraded. In addition, benchmarks are typically run by a systems administrator on an idle system under preferred conditions and not as a user in a normal production operation scenario and therefore do not necessarily reflect the performance that a user will experience. Modern HPC infrastructure is a complex combination of hardware and software environments that are continuously evolving, so it is difficult at any one time to know if optimal performance of the infrastructure is being realized. Indeed, as the following examples illustrate, it is more likely than not that performance is less than optimal, resulting in diminished productivity (CPU cycles and failed jobs) on systems that are typically oversubscribed. Accordingly, the key to a successful and robust science and engineering-based HPC technology audit capability lies in the development of a diverse set of computationally lightweight application kernels that will run continuously on HPC resources to monitor and measure system performance, including critical components such as the global filesystem performance, local processor and memory performance, and network latency and bandwidth. The application kernels are designed to address this deficiency, and to do so from the perspective of the end-user applications.

We use the term ‘kernel’ in this case to represent micro and standard benchmarks that represent key performance features of modern scientific and engineering applications, as well as small but representative calculations carried out with popular open-source high-performance scientific and engineering software packages. Details can be found in References [3, 4]. We have distilled lightweight benchmarking kernels from widely used open-source scientific applications that are designed to run quickly with an initially targeted wall-clock time of less than 10 min. However, we also anticipate a need for more demanding kernels in order to stress larger computing systems subject to the needs of HPC resource providers to conduct more extensive testing. While a single application kernel will not simultaneously test all of these aspects of machine performance, the full suite of kernels will stress all of the important performance-limiting subsystems and components. Crucial to the success of the application kernel testing strategy is the inclusion of historical test data within the XDMoD system. With this capability, site administrators can easily monitor the results of application kernel runs for troubleshooting performance issues at their site. Indeed, as the following cases illustrate, early implementation of application kernels has already proven invaluable in identifying underperforming and sporadically failing infrastructure that would have likely gone unnoticed, resulting in frustrated end-users and wasted CPU cycles on machines that are already oversubscribed.

Application Kernels have already successfully detected runtime errors on popular codes that are frequently run on XSEDE resources. For example, Figure 2 shows the execution time over the course of 2 months for an application kernel based on NWChem, a widely used quantum chemistry program that is run daily on the large production cluster at the Center for Computational Research at SUNY-Buffalo. While the behavior for eight cores (one node) is as expected, calculations on 16 cores (two nodes) in May showed wildly sporadic behavior, with some jobs failing out right and others taking as much as seven times longer to run. The source of the performance degradation was eventually traced to a software bug in the I/O stack of a commercial parallel file system, which was subsequently fixed by the vendor, as evidenced by the normal behavior in the application kernel after June 6. Indeed, the software patch to fix this problem is now part of the vendor’s standard operating system release. It is important to note that this error was going on unnoticed by the administrators and user community for some time and was only uncovered as a result of the suite of application kernels run with results being carefully monitored.

Figure 3 shows a sudden decrease in file system performance on Lonestar4 as measured by three different application kernels (IOR, MPI-Tile-IO and IMB). The IOR and MPI-Tile-IO both show a

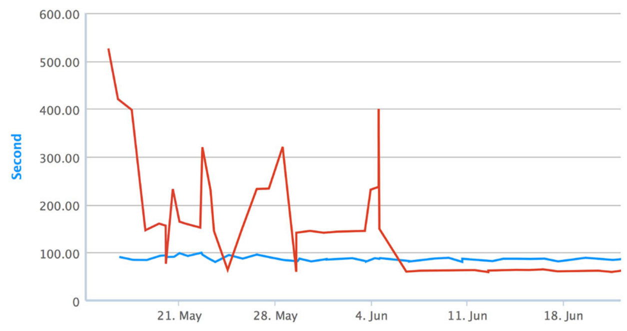

Figure 2. Plot of execution time of the NWChem application kernel on eight cores (blue line) and 16 cores (red line) over a several month period. Calculations on 16 cores show wildly sporadic performance degradation until early June when a patch to a bug in a parallel file system was installed.

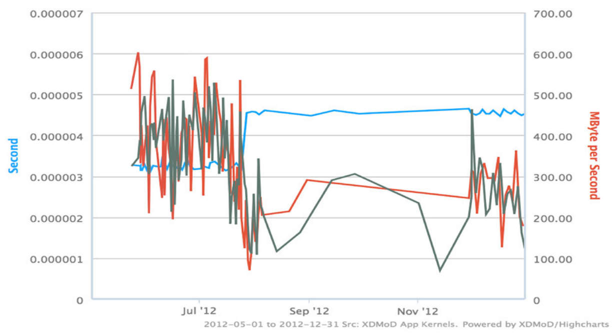

Figure 3. Application kernel data for IMB (blue), IOR (red) and MPI-Tile-IO (black) on Lonestar4. IOR and MPI-Tile-IO data show sudden drop of aggregate write throughput bandwidth, and the IMB data show an increase in latency starting on July 24–25, 2012.

sudden decrease in the aggregate write throughput bandwidth, while IMB, which measures latency, shows an equally sudden increase in latency. The degradation in performance was related to a system-wide upgrade and was subsequently brought back to normal performance level after notification of the degradation. Once again, without application kernels periodically surveying this space to uncover quality of service issues, the loss in performance would has gone unnoticed.

One of the most problematic scenarios entails a single node posing a critical slowdown in which the cumulative resources for a job (possibly running on thousands of processing elements) are practically idled because of an unexpected load imbalance. It is very difficult for system support personnel to preemptively catch such problems, with the result that the end-users are the ‘canaries’ that report damaged or underperforming resources, often after investigations that are very expensive both in terms of computational resources and personnel time. An active monitoring capability designed to automatically detect such problems is therefore highly desirable. For example, Figure 4 shows the results of a log file analysis of CCR’s large production cluster consisting of more than 1000 nodes. By examining only the size of the log files generated on each node (large log file size is indicative of errors), we were able to detect a loose cable on one node and a job scheduler error on another node, both of which resulted in failed jobs. Without such analysis, the loose cable and job scheduler errors would have likely gone undetected, resulting in many failed jobs, frustrated users and underperformance of the resource. While analysis of system log files is not currently included within the XDMoD framework, it is anticipated that future versions will implement this analysis, given its utility in identifying faulty hardware. The Ancor [27] system uses TACC_Stats data to identify jobs with notable resource use anomalies and then analyzes the system logs to determine if the job developed a fault or there was a system failure as a result of the resource use anomaly. Ancor has been demonstrated to deliver accurate analyses of the causes of many system failures.

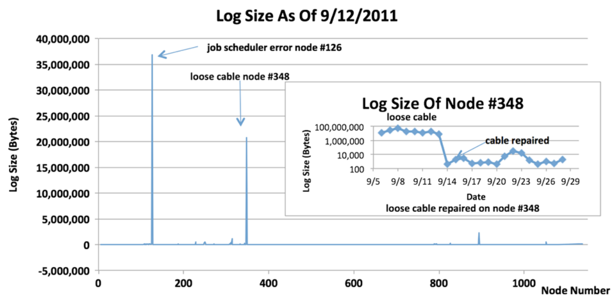

Figure 4. Plot of log file analysis for each node in CCR's production cluster. Two nodes produce very large log files. One node was found to have a loose cable and the other a job scheduler error, both resulting in failed jobs.

4.2. *Measuring resource utilization from TACC_Stats data*

The case studies reported in this subsection were carried out on the Ranger, Lonestar4 and Stampede supercomputers at the Texas Advanced Computing Center (TACC). Ranger (decommissioned as of February 2013) was a Linux cluster comprising of 3936 nodes, each of which had four 2.3 GHz AMD Opteron quad-core processors (16 cores in total) and 32 GB of memory. The filesystem was Lustre, and the interconnect was InfiniBand. Lonestar4 is also a Linux cluster with 1088 Dell PowerEdgeM610 compute nodes. Each compute node has two Intel Xeon 5680 series 3.33 GHz hexa-core processors and 24 GB of memory. The Lonestar4 has two filesystem types: Lustre and NFS, and its interconnect is InfiniBand (NFS is connected via Ethernet). Stampede is also a Linux cluster comprising 6400 primary compute nodes each of which has two Intel Xeon E5-2680 2.7 GHz processors and 32 GB of memory. The filesystem is Lustre, and the interconnect is QDR InfiniBand. TACC_Stats has been deployed on Ranger for 20 months, Lonestar4 for 14 months and Stampede for 9 months. On Ranger, for example, it generated 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. For Stampede, jobs from January 2013 to October 2013 were generally used. For some calculations involving recently added metrics, data only from the period of September 19 to October 27 were used. The jobs included in this study are those longer than the default TACC_Stats sampling interval of 10 min. We ingested both the raw TACC_Stats output files and job accounting information into the XDMoD data warehouse (currently utilizes a MySQL database).

The following analyses will mainly utilize seven key TACC_Stats metrics and closely related quantities: cpu_user (fraction of the CPU utilized by the job in user space), mem_used (per node memory used), cpu_flops (floating-point operations per second produced on a job), io_scratch_write, io_work_write (writing to the scratch and work file systems), net_ib_rx (received messages in bytes per second on the InfiniBand network), and numa_hit (NUMA, successful access of main memory closest to a particular CPU core by a process running on that core) and numa_miss where a process accessed memory from main memory connected through another CPU socket/memory interface. Based on a correlation analysis over *all* of the measured metrics, we have selected the smallest independent set of metrics that describe the execution behavior of the job mix on each system. We also define 1 – cpu_user as the approximate fraction of the CPU that is idle.

4.2.1. Resource use for all jobs across a system. The Lonstar4, Ranger and Stampede job duration distributions are given in Figure 5. Figure 5 (as well as Figures 6–9) shows the kernel density [28] (produced by the R statistical software environment) rather than a histogram in order to avoid making binning choices. As evident in Figure 5, on all three systems, most jobs execute in less than 2.8 h (10,000 s) with the density of short jobs being more pronounced on Stampede. (For the present discussion, we arbitrarily classify a job of less than 2.8 h as short.) The Lonestar4 distribution shows a main peak of short jobs with a secondary maximum at 24 h. For Ranger, the distribution for the first 24 h is similar, but the tail is more prominent. There is also, for Ranger, another set of jobs

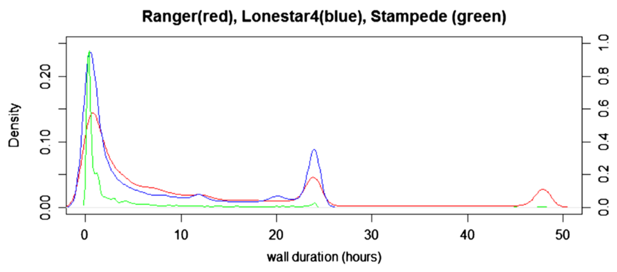

Figure 5. Distribution of Ranger (red, left axis), Lonestar4 (blue, left axis) and Stampede (green, right axis) jobs by wall time in hours.

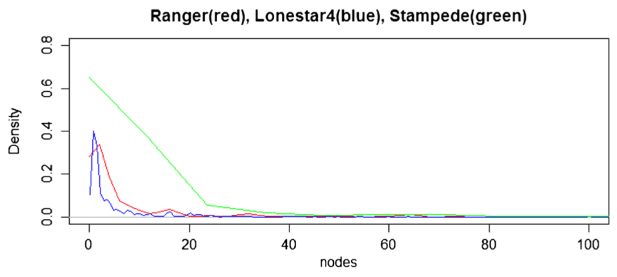

Figure 6. Lonestar4 (blue), Ranger (red) and Stampede (green) job size distribution in number of nodes.

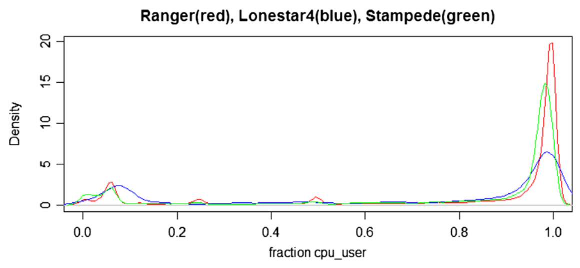

Figure 7. Distribution of user mode fraction of CPU for Ranger (red), Lonestar4 (blue) and Stampede (green).

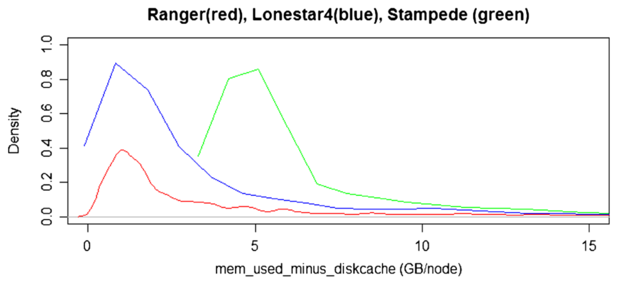

Figure 8. Distribution of memory used minus disk_cache in Gbytes per node for Stampede (green), Lonestar4 (blue) and Ranger (red) jobs.

running out to a final peak at 48 h. These jobs were those run from Ranger's special 'long' job queue. The 24 and 48-h jobs are commonly jobs that can restart from checkpoint data. The secondary peaks are a result of the job queuing policy. The peak at 24 h is much smaller for Stampede than for Ranger or LoneStar4, and there is no appreciable peak at 48 h – even though closer examination reveals that the number of such jobs is reasonable for the short time span of data used here. The lack of pronounced peaks for Stampede is because, while the job mix is similar, Stampede's

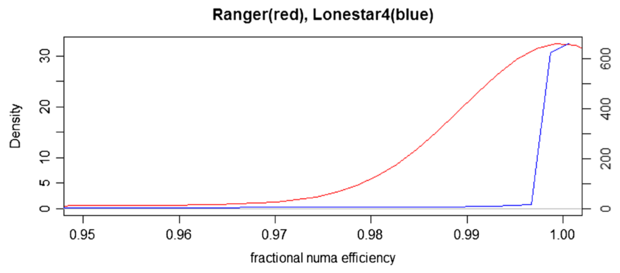

Figure 9. Distribution of Non-Uniform Memory Access accesses of jobs on Ranger (red, left axis) and Lonestar4 (blue, right axis).

scheduling policies are different from Ranger and Lonestar4, and Stampede executes many jobs a factor of 4 or 5 faster than Ranger. Figure 6 shows the distribution of the number of nodes employed by the jobs on Lonestar4, Ranger and Stampede. While Ranger and Lonestar4 show a very similar distribution, jobs on Stampede have a bit of a different distribution but a similar average value of nodes used.

Figure 7 shows that most jobs on Stampede, Lonestar4 and Ranger have a high CPU user busy fraction. A program with a high user mode CPU fraction may or may not be using the CPU efficiently. A low CPU user mode fraction does suggest the application's performance may not be efficient. Most jobs are clustered in the 0.95–1.00 fractional CPU use range. Note the clustering of jobs on vertical lines corresponds to particular values of cpu_user. These are jobs that use only a fraction of the cores available on the compute nodes; 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 (eight cores), 0.25 (four cores) and 0.0625 (one core). For Lonestar4, each node is equipped with 12 cores. The peaks appear at 0.667 (eight cores), 0.50 (six cores), 0.333 (four cores) and 0.083 (one core). For Lonstar4, there is also a very strong cluster at 0.042 corresponding to 1/2 core. For Stampede, the only strong peaks are at 0.0625 (1 core out of the 16) and ~0.032 (1/2 core).

Memory usage varies greatly across jobs on all three systems, as shown in Figure 8. However, most jobs use only a small fraction, ~5–15%, of the available memory (for each node) of 24 GB on Lonestar4 and 32 GB on Ranger and Stampede. *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

$$\text{numa\_eff} = \text{numa\_hit}/(\text{numa\_hit} + \text{numa\_miss}).$$

Using this definition, we can look at the distribution of numa_eff of Lonestar4 and Ranger jobs (Note: This metric was not available for Stampede at the time the data used for this analysis were gathered but is now available.) The numa_efficiency for Ranger and Lonestar4 numa_eff are plotted in Figure 9. The curve for Lonestar4 is very sharply peaked near unity indicating generally good NUMA performance. On Ranger, most jobs are also close to 100% of chip-local memory accesses, but the distribution is broader with a tail of less efficient jobs.

The five metrics shown in Figures 5–9 are only a few examples of the data tracked by TACC_Stats. Similar analyses for all other resource classes are also available. These analyses can also be obtained separately for different days of the week and different hours of the day. *The availability of these data gives a basis for systematic formulation of scheduling policies based on reliable data, which meet policy goals without engendering resource bottlenecks*.

4.2.2. CPU and memory usage by top applications. The CPU and memory use for the most frequently used applications based on node-hours is also of considerable interest because improvements in the resource use characteristics of these systems produces benefit for many users. The application information is captured by the MPI job launcher 'ibrun' on the three TACC systems. It records the job id, the executable binary's full path name, the checksum of the binary and 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 many 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 all three systems are shown in Figures 10–12. The memory use is calculated as the memory used minus disk cache divided by available memory. In general, the CPU user mode fraction is very high, with the memory usage fraction much lower as expected from Figure 8 with memory use on Stampede typically higher. Very few of the largest consumers of node-hours have low CPU use fractions.

In general, the molecular dynamic codes show high CPU user fractions while requiring only small to moderate amounts of memory. On Lonestar4, the only widely used codes with relatively low CPU fractional use are AEROSOFT GASP, GADGET and NWCHEM all of which use less than 0.7 of the CPU. On Ranger, AMBER and GADGET show the lowest CPU and memory use, each using ~0.7 of the CPU and less than 1/3 of the memory. For Stampede, CACTUS, STELLARBOX and WRF show the lowest CPU and memory use, all using less than 2/3 of the CPU and 1/3 of the memory.

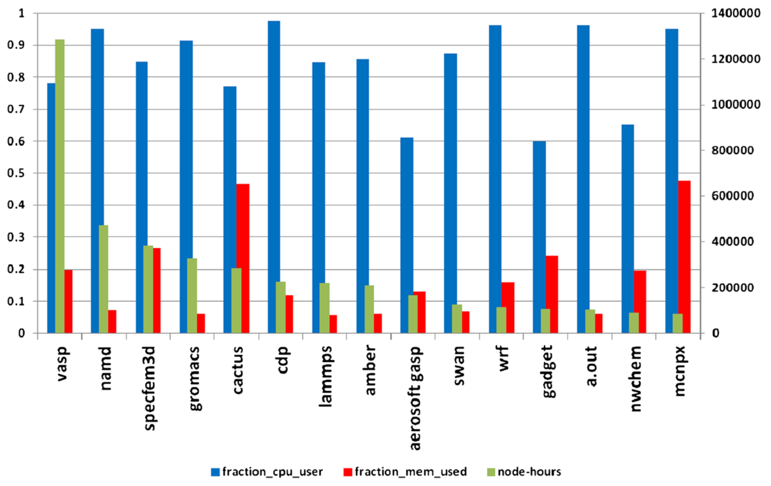

Figure 10. Application's fraction of time in CPU user mode and fraction of memory used on Lonestar4 sorted by node-hour. The right Y axis is the accumulated node-hours of the application.

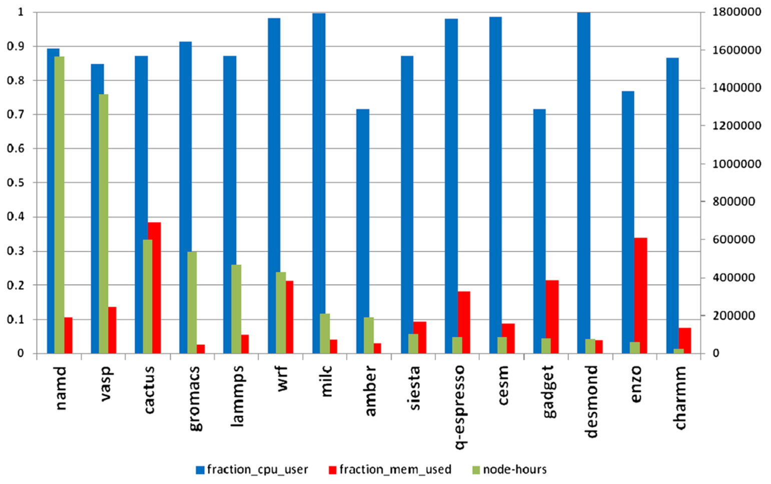

Figure 11. Application's fraction of time in CPU user mode and fraction of memory used on Ranger sorted by node-hour. The right Y axis is the accumulated node-hours of the application.

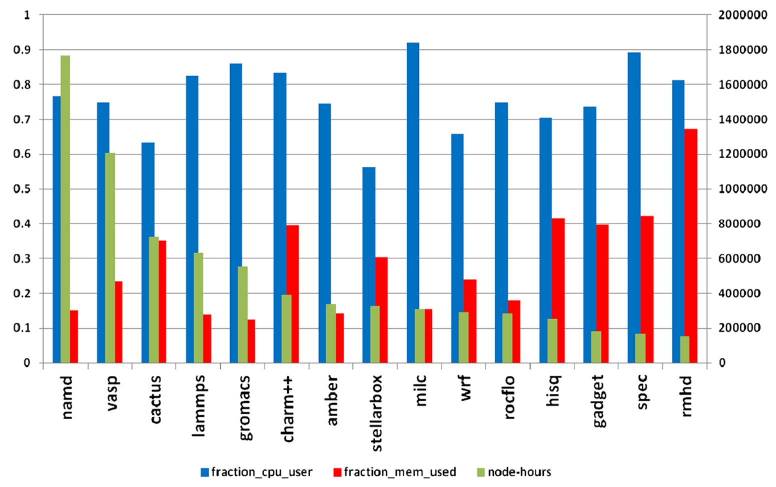

Figure 12. Application's fraction time in CPU user mode and fraction of memory used on Stampede sorted by node-hour. The right Y axis is the accumulated node-hours of the application.

4.2.3. Comparison of job resource use across systems. In this section, the resource use characteristics of the machines on all jobs, long jobs and short jobs are compared across systems. We have chosen to display four selected metrics. Figures 13–15 are bar charts comparing Lonestar4, Ranger and Stampede for all jobs, short jobs ($<$2.8 h) and long jobs using four of the seven TACC_Stats metrics (fraction_cpu_user, fraction_mem_used, scratch_write and work_write) that describe the CPU use, network use and memory use across the three systems.

FLOPS cannot be compared directly across the systems because the counters used record different events on different architectures (Intel vs. AMD). For all jobs, see Figure 13, the comparison shows that memory use fraction and scratch_write traffic are significantly larger on Stampede than on the other two systems. Scratch_write is substantially greater on all systems than work_write for all three systems. The sharp increase in scratch_write for Stampede probably arises because (1) most users did not change their codes when migrating to Stampede, (2) many jobs do output either at synchronization points or periodically at fixed intervals and (3) Stampede runs many jobs four or five times faster than Ranger or Lonestar4. Figures 14 and 15 show that the greater volume of scratch_write I/O is even more pronounced for small jobs than for all jobs, but there is less difference for long jobs.

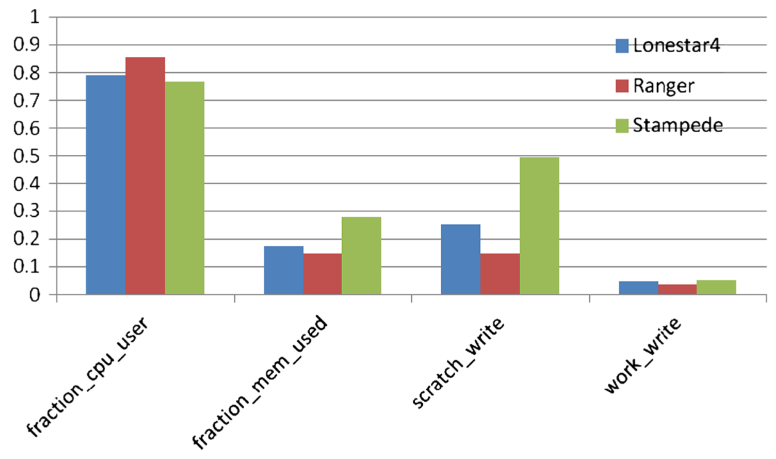

Figure 13. Comparison of all jobs (jobs $>$ 0.1 h) on Lonestar4 (blue), Ranger (red) and Stampede (green) using key tacc_stats metrics. Fraction_cpu_user and fraction_mem_used range from 0 to 1. Scratch_write and work_write are in MB.

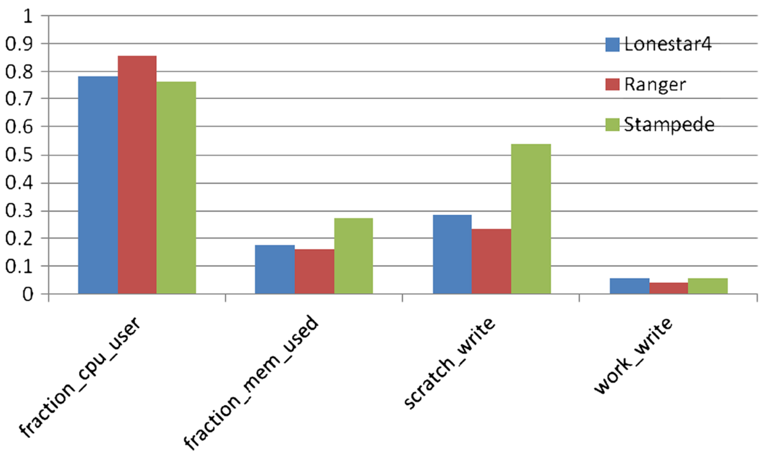

Figure 14. Comparison of short jobs (0.1 h < jobs < 24 h) on Lonestar4 (blue), Ranger (red) and Stampede (green) using key tacc_stats metrics. Fraction_cpu_user and fraction_mem_used range from 0 to 1. Scratch_write and work_write are in MB.

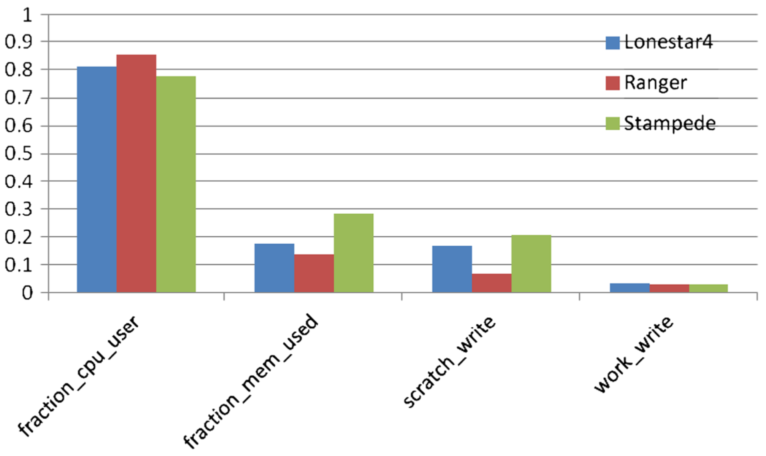

Figure 15. Comparison of long jobs (jobs > 24 h) on Lonestar4 (blue), Ranger (red) and Stampede (green) using key tacc_stats metrics. Fraction_cpu_user and fraction_mem_used range from 0 to 1. Scratch_write and work_write are in MB.

The jobs have similar user CPU fractions across all three systems and relatively little variation in user CPU fractions across long and short jobs. Short and long jobs have similar memory use per node but long jobs do considerable less write I/O. The substantially greater average memory per node use on Stampede over Ranger and Lonestar4 is, in view of the similarity of other resource use metrics, an interesting question for which we have no explanation and which merits further analysis.

4.2.4. Resource use patterns by science domain across systems. We have compared the resource use across Lonestar4, Ranger and Stampede for jobs in different fields of science using the same four metrics that were used for the job comparison. The field of science is based on the allocation name of each job. On Stampede, 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 16–19 show the resource use characteristics of Physics, Chemistry, Molecular Biology and Atmospheric Science.

There are significant differences in resource use both within a discipline across machines and across fields of science. The amount of memory used by physics codes is a factor of 4 higher on Stampede than on Ranger. The volume of scratch write of Chemistry codes is much higher on Stampede than on Ranger or Lonestar4, although work_write volume is more similar. Atmospheric Science usage is

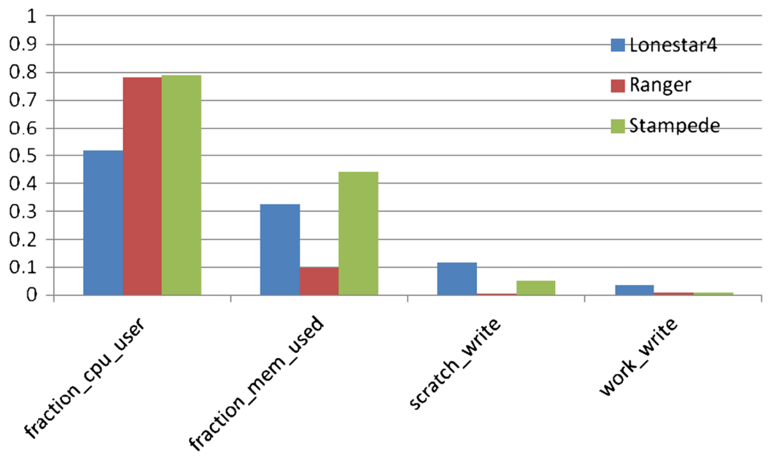

Figure 16. Comparison of Physics usage on Ranger(red), Lonestar4(blue) and Stampede (green) using key tacc_stats metrics. Fraction_cpu_user and fraction_mem_used range from 0 to 1. Scratch_write and work_write are in MB.

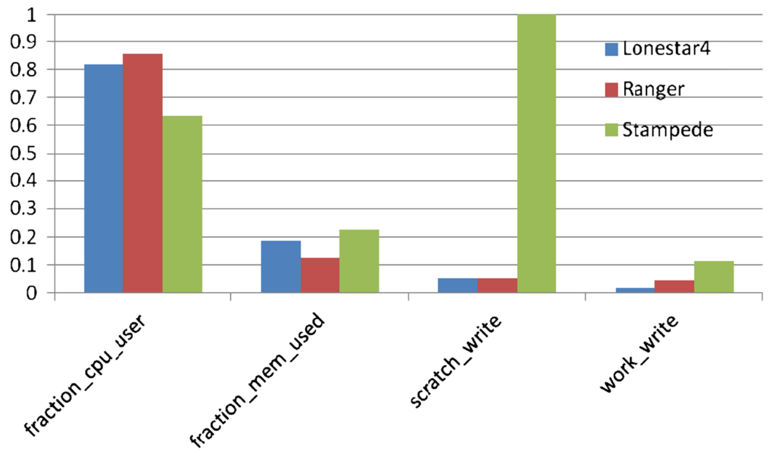

Figure 17. Comparison of Chemistry usage on Ranger(red), Lonestar4(blue) and Stampede(green) using key tacc_stats metrics. Fraction_cpu_user and fraction_mem_used range from 0 to 1. Scratch_write and work_write are in MB.

very similar to that of Chemistry. Physics has a lower scratch write across all three systems. Molecular Biology has a considerably lower user CPU fraction on Stampede (~0.57) than on Lonestar4 or Ranger (0.85). *While user CPU fractions are in general a little lower on Stampede than the other two systems, this magitude of difference suggests further study of the Molecular Biology codes.*

4.3. Analyzing end-user application performance

This section gives the first results from a systematic program to develop a suite of metrics and analyses to identify jobs with performance anomalies, which suggest correctable inefficient resource use. These metrics are computed daily for the jobs executed in the previous day, and jobs that meet the criteria for anomalous resource use are flagged for review by the TACC User Consulting Staff. The great utility of SUPReMM lies in its ability to not only detect these poorly performing jobs but also diagnose them to suggest potential remedies. Not only do the users of the improved application benefit by faster turnaround time but also all center users benefit through the recovery of wasted CPU cycles.

The first metric identifies nearly idle hosts in a parallel job. For the metric, we compute the maximum L1 cache bandwidth and maximum floating-point operation rate at each time sample

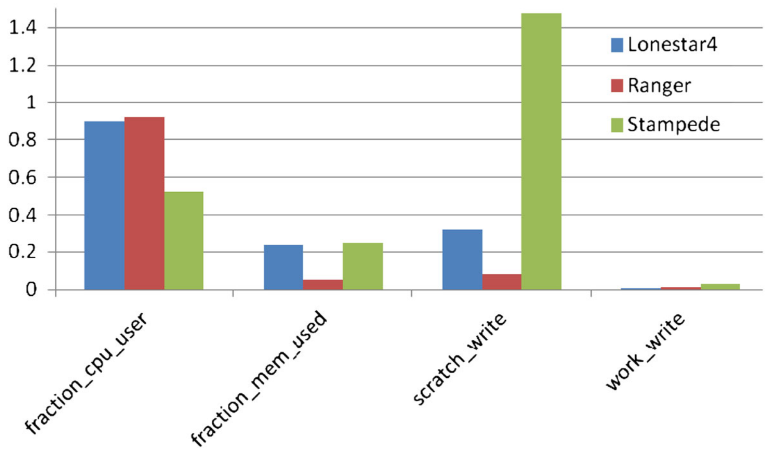

Figure 18. Comparison of Molecular Biology usage on Ranger(red), Lonestar4(blue) and Stampede (green) using key tacc_stats metrics. Fraction_cpu_user and fraction_mem_used range from 0 to 1. Scratch_write and work_write are in MB.

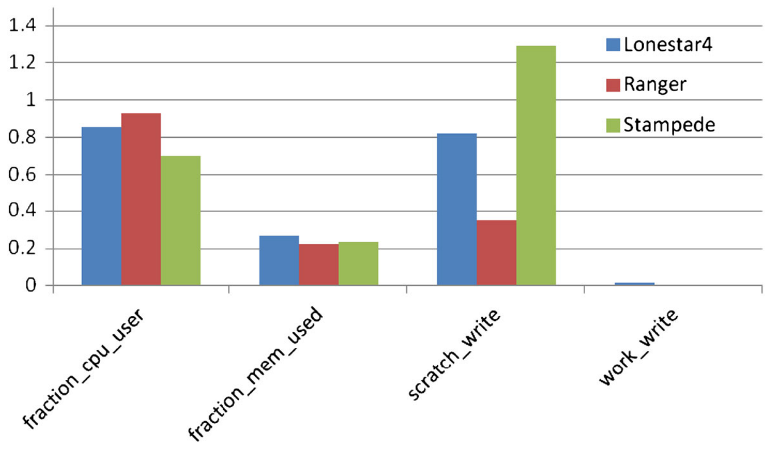

Figure 19. Comparison of Atmospheric Science usage on Ranger(red), Lonestar4(blue) and Stampede (green) using key tacc_stats metrics. Fraction_cpu_user and fraction_mem_used range from 0 to 1. Scratch_write and work_write are in MB.

across hosts in a job. We compute the ratio of the individual rate for each host in the job to its respective maximum. If the time average of this rate is less than 0.1% for either L1 cache rate or floating-point operations, we flag the job for further evaluation. Using this approach on Ranger, we identified a user with a two-node GROMACS job with an apparently idle host. Communication with the user determined that they had incorrectly started GROMACS leading to runs with no MPI parallelism. After being given the correct syntax, the user was able to execute GROMACS correctly and achieve 50–60% performance improvement over their initial approach.

The second metric we have defined looks for jobs with steep, sustained drops in their apparent level of work. Again using L1 cache bandwidth as a proxy for useful work, we fit a step function to the performance of each host in the job and look for hosts with average performance drop greater than 99.9%. For each host, we divide the time into two intervals on either side of a chosen sample and compute the time average in the left and right intervals, thereby fitting a step function to the data with the jump at the chosen sample point. If for any host in a job the ratio between the left and right fitting constants exceeds 99.9%, we flag the job for further evaluation.

Based on this performance drop metric, we automatically flagged a job on Stampede on October 12, 2013. The L1 cache performance for this job is plotted in Figure 20. It is clear from these data that the job suffered from a large change in performance near the 22 h mark. Further investigation showed that

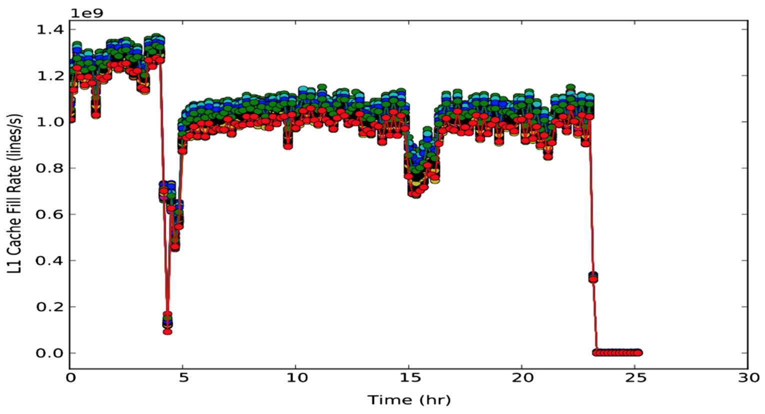

Figure 20. L1 cache performance for a job on Stampede flagged by the performance drop metric. Note the change in performance around the 22 h mark.

a single host also initiated a slow data transfer at this time lasting a little more than 2 h (Figure 20). Contact with the user revealed that they were serially copying local data from each hosts' temporary storage area to the scratch file system on Stampede. After further consultation, the user was able to replace this serial computation with a parallel transfer simultaneously from each host in the job, reducing the total run time of this previously serial section remarkably from 124 to 4 min, as demonstrated by the follow up job in Figure 21.

It is also interesting to study the fractional CPU usage, in terms of cores per node, as an additional measure of efficiency. For example, an analysis was carried out on CCR's production cluster in which the fractional usage of eight-core nodes was calculated based on the SUPReMM data. A histogram of these data, shown in Figure 22, indicates that a small but distinct group of jobs are using only one of the eight cores. From Figure 22, CCR staff identified the poorly performing jobs and then carried out a more detailed analysis of each job's operational characteristics to determine the cause of the high idle CPU fraction. For those jobs for which there appeared to be no valid reason for the poor CPU utilization (i.e., high IO bandwidth or large memory requirements), CCR staff reached out to the users to see what if any improvements could be obtained. For a number of users, the modifications were straightforward and easily implemented.

The poorly performing jobs tended to fall into one of two cases. The first case involved complex computational pipelines, where incorrect resources were requested for the jobs, specifically, requesting two nodes when the pipeline no longer contained a distributed parallel component. The computation could not make use of the cores and memory on the second node. Serial processing tended to dominate the pipeline, along with high memory requirements.

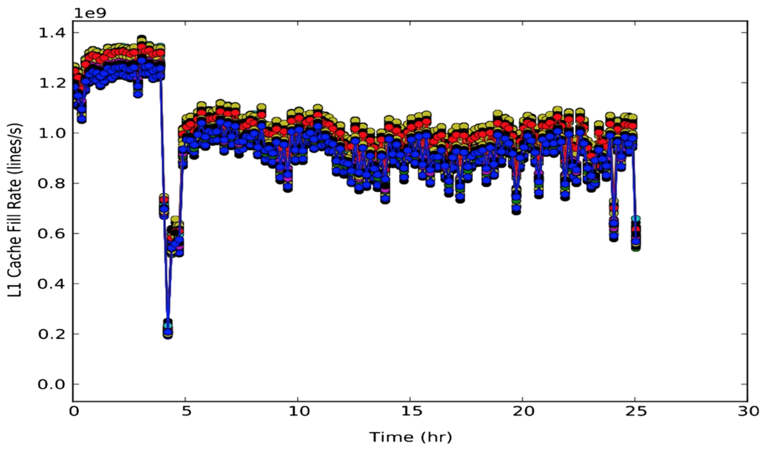

Figure 21. L1 cache performance for a job on Stampede flagged by the performance drop metric after the job was fixed. Note the improvement in performance compared to Figure 20 at 22 h.

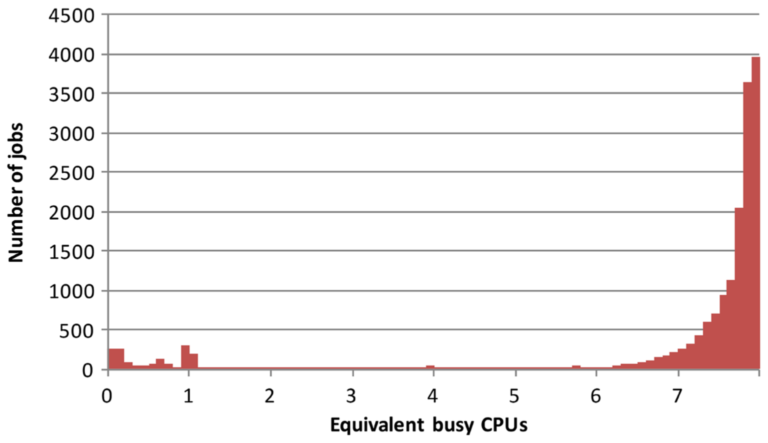

Figure 22. Histogram of equivalent busy cores for CCR eight-core nodes. A value near 8 indicates an efficient code with the eight cores busy most of the time. The secondary peak at ~1 indicates jobs utilizing only one of the eight available cores in a node.

Application parallelism was at the heart of the second case. Users assumed that the application would take advantage of multiple cores, which was not necessarily the case for specific computations. We found that these users tended to rely on the default amount of memory, rather than specifying the actual memory required for the computations.

In summary, the poorly performing jobs identified from Figure 22 were under requesting memory for the computations and assuming tasking parallelism that was not necessarily present in the application or code – resulting in wasted CPU cycles on a cluster that is significantly oversubscribed. CCR's cluster is not unique in this regard, and clearly, the ability for HPC center personnel to characterize the workload on their machines to identify underperforming application codes is highly desirable.

5. DISCUSSION AND FUTURE WORK

The primary focus of this paper has been to demonstrate the ability of the SUPReMM resource management system to provide detailed job and system-level information and analytics of the workload running on multiple HPC resources. While characterizing resource utilization at the job, application and system levels are useful in its own right; the benefit of the analyses based on which can be generated by SUPReMM lies in the ability to tune system and application performance, to configure systems for optimal cost/return, to plan new systems and 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 briefly described initial efforts in contacting the owners of particularly poorly running jobs to better understand their workflow and determine if steps can be taken to mitigate the underperformance. In the future, 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 10–12), 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 10–12 of CPU and memory efficiency can be carried out 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 (Figures 16–19), NSF, DOE and other national sources of cyberinfrastructure 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 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 have presently incorporated TACC_Stats data summaries from Stampede, Lonestar4 and Ranger into a developmental version of XDMoD; future public XDMoD releases will feature a TACC_Stats data realm, and the data will therefore be widely available.

The SUPReMM project is working with XSEDE to install the system on all XSEDE computational resources and ultimately on other HPC systems through the open-source version of XDMoD. 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, plans are underway to streamline the installation and setup of the SUPReMM system.

ACKNOWLEDGEMENTS

This work is supported by the National Science Foundation under grant number OCI 1203560 for SUPReMM, grant number OCI 1025159 for the development of technology audit service for XSEDE, NSF grant number 1203560, and NSF MRI-R2 grant number 0959870 for Data Intensive Supercomputing. We also acknowledge the help of Charng-Da Lu in processing some of the data used here.

REFERENCES

1. Hammond J. TACC_stats: I/O performance monitoring for the intransigent. *2011 Workshop for Interfaces and Architectures for Scientific Data Storage (IASDS)*, 2011.
2. Available from: http://github.com/TACCProjects/tacc_stats [accessed on 21 Febuary 2014].
3. Furlani TR, Jones MD, Gallo SM, Bruno AE, Lu C-D, Ghadersohi A, Gentner RJ, Patra A, DeLeon RL, von Laszewski G, Wang F, Zimmerman A. Performance metrics and auditing framework using application kernels for high performance computer systems. *Concurrency and Computation: Practice and Experience* 2013; **25**(7):918. [Online]. Available from: http://dx.doi.org/10.1002/cpe.2871 XDMoD: http://xdmod.ccr.buffalo.edu; Open XDMoD: http://xdmod.sourceforge.net/ [accessed on 21 Febuary 2014].
4. Furlani TR, Schneider BI, Jones MD, Towns T, Hart DL, Gallo SM, DeLeon RL, Lu C-D, Ghadersohi A, Gentner RJ, Patra AK, von Laszewski G, Wang F, Palmer JT, Simakov N. Using XDMoD to facilitate XSEDE operations, planning and analysis. *Proceedings of the Conference on Extreme Science and Engineering Discovery Environment: Gateway to Discovery* (XSEDE '13), ACM, New York, NY, USA, 2013; Article 46, 8. DOI: 10.1145/2484762.2484763
5. Available from: http://sebastien.godard.pagesperso-orange.fr [accessed on 21 Febuary 2014].
6. Del Vento D, Engel T, Ghosh S, Hart D, Kelly R, Liu S, Valent R. System-level monitoring of floating-point performance to improve effective system utilization. *International Conference for High Performance Computing, Networking, Storage and Analysis,* 2011.
7. Brandt J, Gentile A, Thompson D. Develop feedback system for intelligent dynamic resource allocation to improve application performance. SAND2011-6301, 2011.
8. Brandt J, Debusschere B, Gentile A, Mayo J, Pebay P, Thompson D, Wong M. OVIS-2: a robust distributed architecture for scalable RAS. *22nd IEEE International Parallel and Distributed Processing Symposium, 4th Workshop of System Management Techniques, Processes, and Services*, 2008.
9. Huck KA, Malony AD, Shende S, Morris A. Knowledge support and automation for performance analysis with PerfExplorer 2.0. *Large-Scale Programming Tools and Environments, Special Issue of Scientific Programming* 2008; **16**(2-3):123–134.
10. Shende S, Malony A. The Tau parallel performance system. *International Journal of High Performance Computing Applications* 2006; **20**(2):287–311.
11. Tau. Available from: http://www.cs.uoregon.edu/research/tau/home.php [accessed on 21 Febuary 2014].
12. Tallent NR, Mellor-Crummey JM, Adhianto L, Fagan MW, Krentel M. HPCToolkit: performance tools for scientific computing. *Journal of Physics: Conference Series* 2008; **125**:012088.
13. HPCToolkit. Available from: http://www.hpctoolkit.org/ [accessed on 21 Febuary 2014].
14. Djoudi L, Barthou D, Carribault P, Lemuet C, Acquaviva JT, Jalby J. Exploring application performance: a new tool for a static/dynamic approach. *The Sixth Los Alamos Comp. Science Institute Symp.*, 2005.
15. Open|SpeedShop. Available from: http://www.openspeedshop.org/wp/ [accessed on 21 Febuary 2014].

16. Geimer M, Saviankou P, Strube A, Szebenyi Z, Wolf F, Wylie BJN. Further improving the scalability of the Scalasca toolset. *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, Springer, 2012; 463–474.
17. VTune. Available from: http://software.intel.com/intel-vtune-amplifier-xe [accessed on 21 Febuary 2014].
18. Miller BP, Callaghan MD, Cargille JM, Hollingsworth JK, Irvin RB, Karavanic KL, Kunchithapadam K, Newhall T. The Paradyn parallel performance measurement tool. *IEEE Computer* 1995; **28**:37–46.
19. Burtscher M, Kim BD, Diamond J, McCalpin J, Koesterke L, Browne J. PerfExpert: an easy-to-use performance diagnosis tool for HPC applications. *SC2010 Int. Conference for High-Performance Computing, Networking, Storage and Analysis*, November 2010.
20. PerfExpert. Available from: http://www.tacc.utexas.edu/perfexpert/ [accessed on 21 Febuary 2014].
21. HOPSA-Holistic Performance System Analysis. Available from: http://www.vi-hps.org/projects/hopsa/overview [accessed on 21 Febuary 2014].
22. Threadspotter. Available from: http://www.roguewave.com/products/threadspotter.aspx [accessed on 21 Febuary 2014].
23. Scalasca. Available from: http://www.scalasca.org/ [accessed on 21 Febuary 2014].
24. Paraver. Available from: http://www.bsc.es/computer-sciences/performance-tools/paraver [accessed on 21 Febuary 2014].
25. Vampir. Available from: http://www.vampir.eu/ [accessed on 21 Febuary 2014].
26. Bennett PM. Sustained systems performance monitoring at the U.S. Department of Defense High Performance Computing Modernization Program. *State of the Practice Reports*, ser. SC '11, New York, NY, USA, ACM, 2011; 3:1–3:11. [Online]. Available from: http://doi.acm.org/10.1145/2063348.2063352
27. Chuah E, Jhumka A, Narasimhamurthy S, Browne JC, Hammond J, Barth W. Linking resource usage anomalies with system failures from Cluster Log Data. *Proc. Symposium on Reliable Distributed Systems*, September 2013.
28. Scott DW. Multivariate Density Estimation. Wiley: New York, 1992.