Understanding Application and System Performance Through System-wide Monitoring

R. Todd Evans, James C. Browne, William L. Barth
Texas Advanced Computing Center, Texas 78758
Email: rtevans@tacc.utexas.edu, browne@cs.utexas.edu, bbarth@tacc.utexas.edu

***Abstract*—TACC Stats is a continuous monitoring tool for HPC systems that collects data at the core and process level for every job executing on a monitored system. That data can be aggregated at the system, group, user, application, job, node, or core level. TACC Stats has been in production use for about 5 years and is now used by numerous HPC systems around the world. This paper reports on a major new version of TACC Stats and the additional analyses which can now be accomplished. The data collected is now a truly comprehensive range of metrics spanning all system resources including energy consumption, vectorization, I/O activity and network activity as well as a full set of computationally oriented metrics. TACC Stats also includes a new capability which enables online monitoring of the resource use data which is gathered. TACC Stats automatically customizes itself for different chip architectures and has been extended to execute on Cray systems. In additional to describing the new capabilities, we also describe several analyses, some incorporating the new data such as I/O behavior. These analyses and reports can give insights to identify performance issues with jobs and applications, diagnose system and job errors, and understand the resource needs of users.**

***Keywords*-continuous monitoring; performance analysis; HPC systems;**

I. Introduction

A. Motivation and Problem

The large scale high performance computing systems which are used for computational science and engineering research are scarce and important resources. These systems have thousands of architecturally complex compute nodes, complex software stacks, thousands of users, and run hundreds of thousands of jobs. In this context, applying workload characterization to identify the opportunities to enhance performance of jobs and applications to obtain best return on investment of scarce resources, and to maximize user productivity, is a challenging but important problem. In the past, acquiring all data required to satisfy the needs for workload characterization across a wide spectrum of interests from different classes of stakeholders has typically required use of multiple data sources that are sometimes of different granularity and not simple to integrate. This situation has recently changed with several resource use monitors (Section II gives a brief listing of currently active resource use monitors) now available.

B. Overview of TACC Stats

TACC Stats [1], [2], [3] is a continuous monitoring tool for HPC systems that collects data at the core and process level for every job executing on a monitored system. That data can be aggregated at the system, group, user, application, job, node, or core level. TACC Stats has been in production use for about 5 years on several different systems at TACC including Ranger, Stampede, Lonestar 4 and 5 and is now used by numerous HPC systems around the world. This paper reports on a major new version of TACC Stats and the additional analyses which can now be accomplished. The data collected is now a truly comprehensive range of metrics spanning all system resources including energy consumption, vectorization, I/O activity and network activity as well as a full set of computationally oriented metrics. TACC Stats now also enables online monitoring and analyses of the resource use data which is gathered, automatically customizes itself for different chip architectures and has been extended to execute on Cray systems. TACC Stats is most commonly applied with samples always taken at the start and end of every job and at 10 minute intervals during job execution with an overhead estimated to be 0.02%. TACC Stats is capable of sub-second sampling depending on the level of overhead which is acceptable. Even at relatively coarse intervals insights into workflow and performance can be made from the appropriate metrics and their interpretation.

TACC Stats also includes capabilities for generating several different reports including a report giving a resource use profile for every job run on Stampede and Lonestar 5. These reports are available to the consulting staff of TACC to assist in diagnosing problems which may have occurred during execution of a job and will soon be available to users on a routine basis.

TACC Stats has been deployed on multiple chip architectures on multiple HPC systems around the world including all of the major HPC systems of the NSF XSEDE consortium. The data generated by TACC Stats is a principal foundation for the XDMOD reporting and auditing system which is used on all XSEDE systems. Open XDMOD, which incorporates TACC Stats as a principal source of data, has been downloaded by 340 HPC sites. Numerous examples of the reports (spanning from reports on individual jobs to reports for funding agencies) which can be generated by XDMOD using using TACC Stats data are available in the literature [4], [5], [6], [7], [8].

C. Contributions

The recent enhancements to TACC Stats enable a major extension of the analyses and reports which can be generated.

For example, TACC Stats can now be used for identification and analysis of I/O bound and network bandwidth bound applications. Analyses of energy use broken down by socket, process and dram components are now available. Mapping the data directly to a relational database enables very easy design and implementation of analyses. Examples of some of these analyses are given in Sections 4 and 5. The newly implemented capability enabling online resource use analyses with soft real time feedback to systems administrators and users can help avoid system slowdowns and failures and also help with timely diagnoses of failures which do occur. Implementation on the Cray execution environment enables TACC Stats to be utilized on another of the most widely used HPC computational systems.

D. Contents of Paper

Section II has related research. Section III gives details of the current functionality of TACC Stats. Section IV describes some of the analyses which can be accomplished using TACC Stats data. Section V gives illustrations of these analyses and Section VI sketches further work.

II. Related Work

Continuous, system wide job profiling seems to have begun about 30 years ago with Morph [9] and DCPI [10] for Digital's UNIX. Morph used low overhead system-wide sampling and binary rewriting to continuously evolve programs to adapt to their host architectures. DCPI gathered much more robust profiles, such as precise stall times and causes, and focuses on reporting information to users. OProfile [11], a DCPI-inspired tool, collects and reports data much in the same way as DCPI for multiple architectures, but offers less sophisticated analyses. The book "Systems Performance: Enterprise and the Cloud" by Brendan Gregg [12] gives full coverage to many of the issues faced in system wide job profiling.

Continuous system wide job profiling or resource use accounting has two major branches: High performance computing systems and cloud or data center systems. High-performance computing (HPC) shares many profiling challenges with cloud computing, because profilers must gather representative samples with minimal overhead over thousands of nodes. It was formerly the situation that cloud computing had additional challenges–multiple machine configurations, and a wide spectrum of applications, workloads, and machine configurations. As HPC systems have become more complex with heterogeneous system configurations and many different system configurations requiring monitoring, HPC monitoring and cloud/data center monitoring have grown more alike. Since HPC systems are the main concern for this paper the principal focus for this section will be systems developed for continuous system wide monitoring of HPC systems. We will only briefly mention some important cloud/data center continuous system wide monitoring tools.

The tool most similar to TACC Stats, both in terms of approach and data gathered, is LDMS (The Lightweight Distributed Metric Service) [13] developed at Sandia National Laboratory as a part of the Ovis [14] project. LDMS has a similar approach and collects from data sources similar to those used by TACC Stats. Applications of LDMS data are reported in [15] and [16]. It is has been deployed on a variety of HPC platforms.

Del Vento, et. al. [17] describe a similar approach to system-wide, job-level monitoring and used the data for identifying poorly performing jobs and diagnosis of code failures. The approach adopted by Del Vento et. al. relied on a command, `hpmstats`, that is available only on AIX systems to collect hardware counter data.

Performance Co-Pilot (PCP) [18] is another open source tool which can be adapted to continuous system wide resource use profiling.

Another tool which could potentially be adapted to continuous system wide profiling is the HOPSA [19] project's LWM^2 [20] performance screening tool. LWM^2 gathers its data though instrumentation of each application, and by periodic sampling of performance counters. The data gathered by LWM^2 is stored in a database. The HOPSA tool chain may then invoke updated versions of several performance optimization tools.

Cray provides a monitoring system, Resource Use Reporting [21], for which plugins can be used to extend the range of monitoring data that is gathered, or to generate reports and analyses.

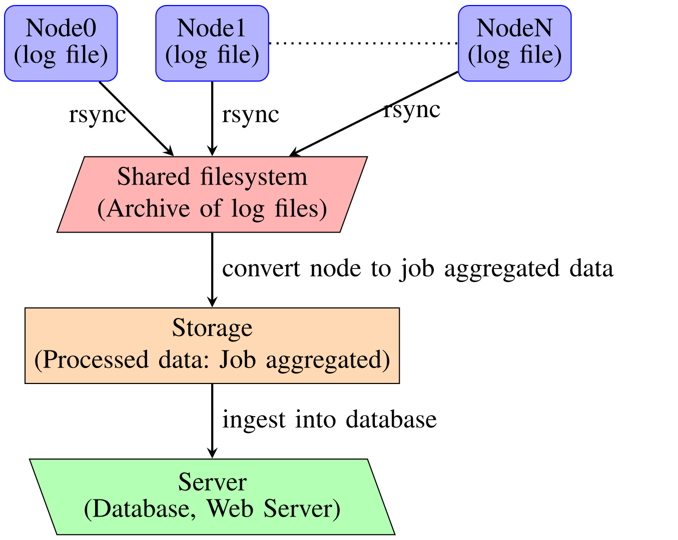
There are many other open source and commercial resource usage monitoring tools including: CLUMON [22], Ganglia [23], and Nagios [24]. Only one of these, CLUMON, which uses data from PCP, is capable of resolving the data by job at the core level. CLUMON and PCP do not, however, collect data from hardware performance counters. Ganglia and Nagios do not resolve data by jobs by users. None of these systems is capable of resolving data at the application and/or project level

Perhaps the most widely known system for continuous system wide profiling for clouds/data centers is GWP (Google-Wide profiling) [25]. Ref. [25] combines system wide, continuous profiling spanning all of Google's compute resources and applications with a set of analyses, presentations and optimization tools. It is, however, proprietary to Google. A somewhat related tool called P-Tracer [27] has been developed at Alibaba to support performance optimization and problem diagnosis on Alibaba's cloud systems. P-Tracer is based on selective tracing and trace analyses. In fact, all of the major operators of cloud systems are known to have monitoring tools but little has been disclosed about most of them, NetFlix [26] being an exception.

III. Current Collection Functionality

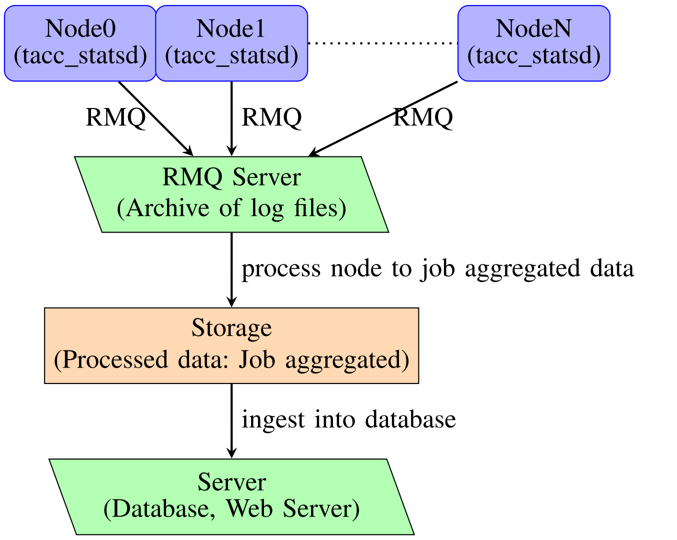
A. Operation modes

At the begin and end of every job TACC Stats is executed by a job scheduler in order to obtain at least 2 data points per job and provide TACC Stats with a job id. The exact implementation of this step may vary by job scheduler but generally a single statement is added to the prolog and epilog



Node0 (log file)
Node1 (log file)
NodeN (log file)
rsync
rsync
rsync
Shared filesystem (Archive of log files)
convert node to job aggregated data
Storage (Processed data: Job aggregated)
ingest into database
Server (Database, Web Server)

Figure 1. The cron-based mode of TACC Stats stores collected data on the compute node on which it is running. It requires a daily rsync copy to make this data accessible.



Node0 (tacc_statsd)
Node1 (tacc_statsd)
NodeN (tacc_statsd)
RMQ
RMQ
RMQ
RMQ Server (Archive of log files)
process node to job aggregated data
Storage (Processed data: Job aggregated)
ingest into database
Server (Database, Web Server)

Figure 2. The daemon-based version of TACC Stats is operated as a Linux service daemon. The compute nodes send data over the network directly to a RabbitMQ server, where it is immediately processed into job aggregated data.

scripts. In addition to these two data points, data is collected at regular intervals using either of the following two modes of operation.

The original operation mode of TACC Stats relies on the `cron` job scheduler in order to run the collection executable `tacc_stats`. This mode of operation appends the collected data to a log file, local to the compute node on which it is running, that is created during a daily log rotation triggered by `cron`. A copy of this log file is later made to a central location on a shared filesystem. In order to avoid undue stress on the filesystem the data is centralized once a day at a different random time per node when the system utilization is low (e.g. early morning). This is still the principle mode of operation on most systems where TACC Stats is installed. This operation mode introduces a time lag between when the data is collected and when it is accessible. The time lag prevents real time action on data and introduces the possibility that a node failure will result in data loss. A schematic of this mode is sketched in Fig. 1.

The new mode of operation resulted from a request from a prospective TACC Stats deployment site. The site requested a version that did not involve the filesystem in its operation and reported data in real time. The open source message broker software RabbitMQ [28] was identified as a viable option to help address this request. A TACC Stats daemon, `tacc_statsd`, was implemented that runs on each node and relies on the system call `sleep()` to induce data collection and RabbitMQ to send data directly over the Ethernet network to a RMQ server. A data consuming executable was implemented to consume this data from the RMQ server as soon as it is available and output the data to raw stats files. This mode was first tested on TACC's 132 node Maverick system, then deployed on SDSC's 1984 node Comet system, and most recently deployed on TACC's 1278 node Lonestar 5 Cray system. A schematic of this mode is shown in Fig. 2.

B. Collected Data

In addition to the new real-time mode of operation, data from many new devices are collected. For a list and description of devices that have been supported since 2013 see TABLE I in Ref. [3]. The new devices monitored by TACC Stats are:

1) Nehalem, Westmere, Ivy Bridge, and Haswell processors including both the core counters collected from `msr` files and uncore counters (QPI, L3 cache, memory controller etc.) collected from PCI config space
2) Xeon Phi data accessed from the host
3) Running average power limit (RAPL) counters that tracks the power consumption separately of all cores + LLC cache, all cores, and DRAM
4) collection of process information from the `procfs` filesystem such as
   - executable names
   - Size and high water mark of virtual memory
   - Size of locked memory
   - Size and high water mark of physical (RSS) memory
   - size of data, stack, and text segments
   - number of threads per process
   - cpu affinities per process
   - memory affinity per process

In order to simplify the installation procedure and enable deployment across systems composed of multiple architectures TACC Stats has been modified to identify the processor architecture and uncore devices automatically at runtime. It also will detect the topology of a node and modify it's collection procedure appropriately for processors with and without hardware threading. Currently only 3 hardware configuration options for a given system are specified at build time: whether Infiniband is supported, whether a Xeon Phi coprocessor is present on a node, and whether a Lustre filesystem is present. In fact, if any of these are not present on a node TACC Stats will execute successfully at run time. These options are only used at compile time to determine whether to look for dependencies required for Infiniband, Xeon Phi, and Lustre monitoring.

IV. Current Analysis Framework

A. Metrics Computed

After data collection TACC Stats maps the raw output from each node to job ids. Metadata describing each job along with a set of computed metrics are then ingested into a PostgreSQL database [29]. With this new release of TACC Stats, a variety of metrics related to a job's Lustre I/O performance are now computed in addition to the original metrics described in Ref. [3]. A summary of all computed metrics is listed in Table I with a short description of each. The device data the metrics are computed from can be naturally grouped into four categories: Lustre, Network, Processor, and Operating System.

There are two main types of metrics computed for each job. *Average* metrics listed in this table correspond to the Average Rate of Change (ARC) of the relevant statistics: they are computed by first averaging the relevant data over time and then over nodes. The *Maximum* metrics listed are computed by first computing the relevant data's delta over each time interval for each node, then summing over nodes and taking the maximum resulting delta. In the case of ratios the averages are computed before the ratio is formed. All counters used to compute the metrics in Table I, aside from those used to derive MemUsage, are cumulative. Therefore infrequent (e.g. 10m) sampling intervals over the lifetime of a job does not prevent an accurate calculation of the ARC. Maximum metrics are computed over finite time intervals and must be interpreted as an approximation to the maximum instantaneous rate of change.

A few of the listed metrics require additional explanation:

- The MemUsage metric is unique in that it is a snapshot of memory usage at a given instance in time. This snapshot may miss memory usage spikes. However, we can now validate results derived from this metric with the collection of per-process data from `procfs`, where a true memory high water mark for each process is recorded by the OS.
- The VecPercent metric is the ratio of the number of vectorized floating point instructions issued to non-vectorized floating point instructions issued over the lifetime of a job.
- CPU_Usage is the fraction of time a job spent in user space.
- idle is the ratio of the node's CPU_Usage with the smallest value to the node's CPU_Usage with the largest value. It measures work imbalance across nodes.
- catastrophe is the ratio of the time window with smallest CPU_Usage, summed over all nodes, to the time window with the largest CPU_Usage. It measures work imbalance across time.

All of the metrics are stored in the database in the same record as the job metadata. The database can be searched across the computed metrics returning, for example, jobs with metric values that exceed thresholds. TACC Stats uses these metrics to automatically flag jobs that meet certain criteria, such as high metadata request rates. Queries can be performed from either the PostgreSQL command-line or the web portal framework described in the next section.

B. Web Portal

TACC Stats data is accessible via a web portal built with the Django web framework [30] backed by the PostgreSQL server. The front page of the web portal to Stampede is shown in Fig. 3. Jobs may be browsed by date, or searched along any combination of metadata and up to three *Search fields*, where a Search field consists of one of the metric names from Table I plus a modifying suffix to indicate the comparison operator to use against a threshold value entered in the *Value field*. A detailed view of any job can be accessed directly via the *Job ID* field in the upper right-hand corner.

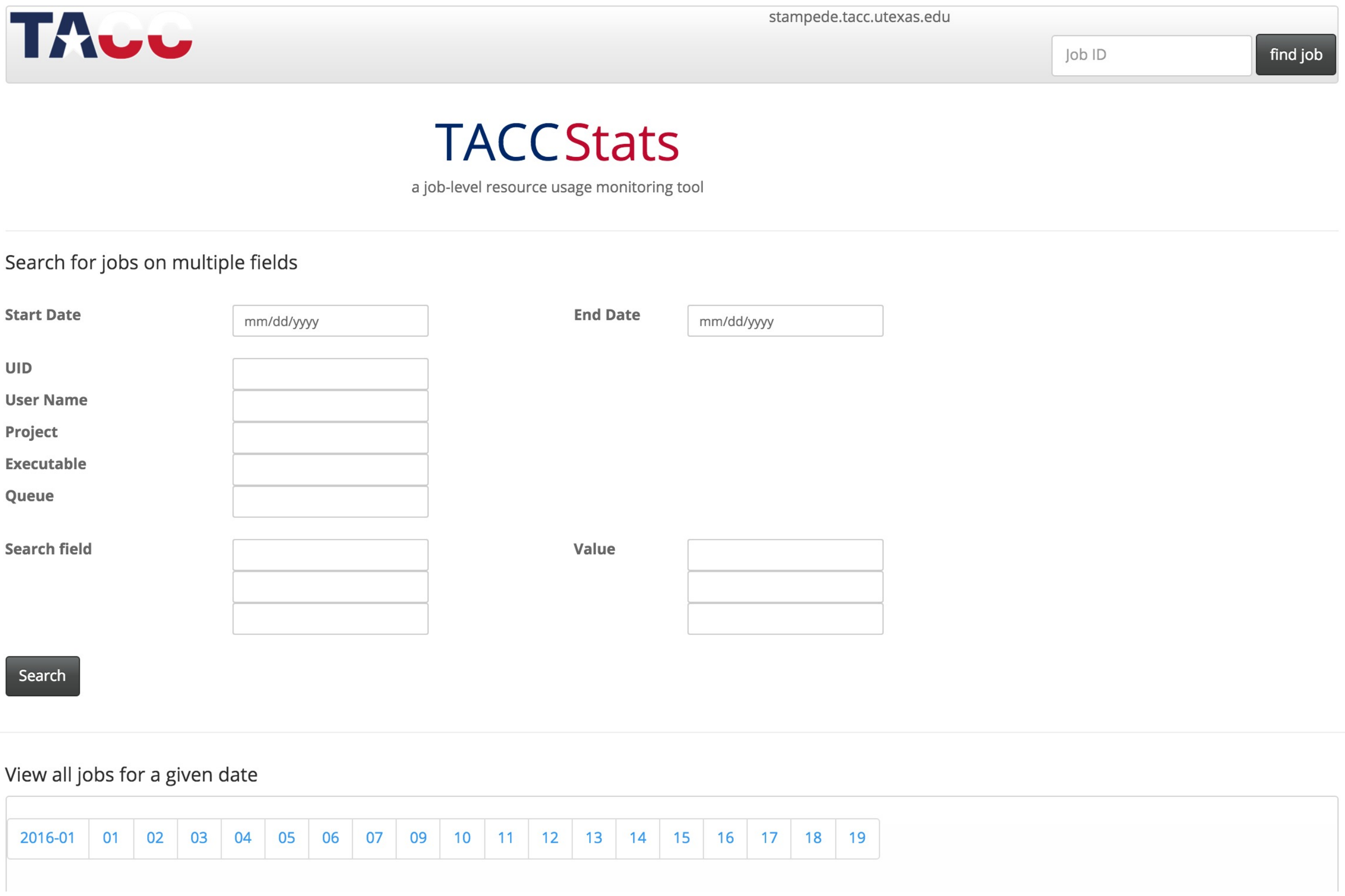
A query from this page will return a list of all jobs that satisfy the specified search criteria. This list displays the metadata for each job including Job ID, username, executable, start time, end time, run time, queue, job name, job completion status, node wayness, number of reserved nodes, and node hours consumed. A link to any of the jobs in this list can be followed to see the detailed view of the job including metadata, performance plots, executable paths, working directories, and links to the logs relevant to that job. More detailed information such as individual processes and their memory usage, cpu affinities, and thread count can be accessed from this detailed view page, along with a report indicating which of the computed metrics passed or failed comparison tests, and which modules were loaded and libraries were linked to at runtime. Note the modules and libraries are only available if the XALT plugin is enabled [31], [32].

V. Analyses Illustrations

A. Web Portal-based Analyses

It is straightforward to use the web portal to explore data. For example, the search for all jobs running the WRF executable `wrf.exe` [33] (a commonly run weather forecasting code) on Stampede from the dates Jan 1, 2016 to Jan 14, 2016 over 10 minutes in runtime returns 558 jobs. A histogram, shown in Fig. 4, of jobs versus runtime, nodes, queue wait time, and maximum metadata requests is automatically generated for these searches along with the job list. In this figure outliers are visible in the Metadata Reqs histogram. These outliers can be attributed to a particular user and will be examined in the next subsection.

Every search also returns a sublist of jobs that have been flagged for metric values that exceed thresholds such as high metadata rates, excessive use of the GigE network, running in the largemem queue but using little memory, idle nodes, sudden performance increases or drops, and a high average cycles per instruction. Identifying jobs based on these characteristics


stampede.tacc.utexas.edu

Job ID find job

TACCStats

a job-level resource usage monitoring tool

Search for jobs on multiple fields

Start Date	mm/dd/yyyy	End Date	mm/dd/yyyy
UID			
User Name			
Project			
Executable			
Queue			
Search field		Value	

Search

View all jobs for a given date

2016-01 01 02 03 04 05 06 07 09 10 11 12 13 14 15 16 17 18 19

Figure 3. The web portal enables queries along the show metadata fields and Search fields based on metric name and threshold value. All jobs completed on a given data can also be browsed.

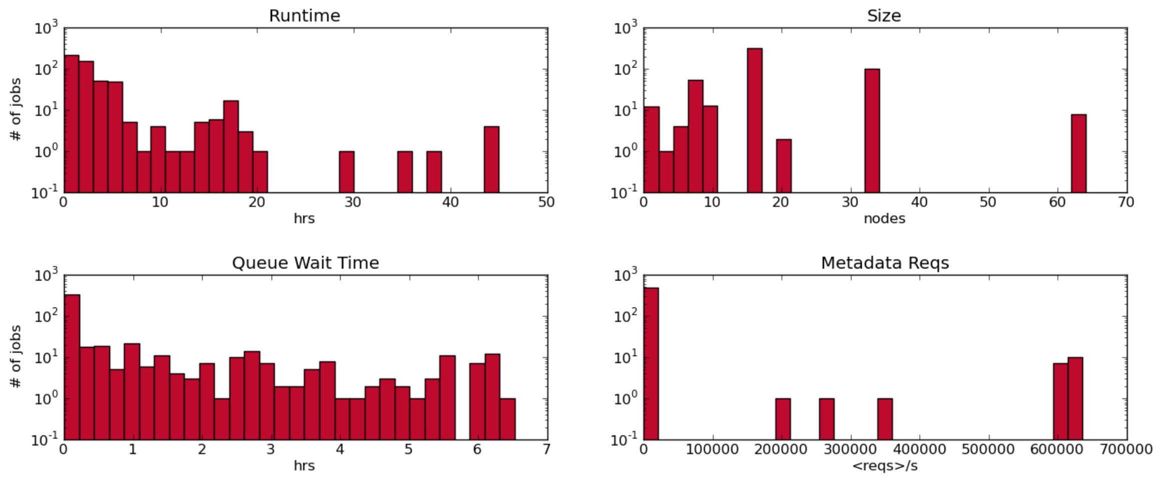


Figure 4. The web portal generates a histogram of jobs versus runtime, nodes, queue wait time, and maximum metadata requests after every query. This histogram is a result of querying all jobs running `wf.exe` on Stampede from the dates Jan 1, 2016 to Jan 14, 2016 over 10 minutes in runtime.

TABLE I
SET OF METRICS COMPUTED FOR EVERY JOB

Label	Definition
	Lustre Metrics
MetaDataRate	Maximum Metadata server operation rate
MDCReqs	Average Metadata server operation rate
OSCReqs	Average Object Storage server operation rate
MDCWait	Average time required to complete Metadata server operations
OSCWait	Average time required to complete Object storage server operations
LLiteOpenClose	Average file open/close rate
LnetAveBW	Average Lustre bandwidth
LnetMaxBW	Maximum Lustre bandwidth
	Network Metrics
InternodeIBAveBW	Average Infiniband Bandwidth between compute nodes (typically MPI)
InternodeIBMaxBW	Maximum Infiniband Bandwidth between compute nodes (typically MPI)
Packetsize	Average Infiniband Package Size
Packetrate	Average Infiniband Package Rate
GigEBW	Average Bandwidth over the GigE network
	Processor Metrics
Load_All	Average Cache load rate from any cache level
Load_L1 Hits	Average L1 cache hit rate
Load_L1Hits	Average L2 cache hit rate
Load_LLCHits	Average Last-level cache hit rate
cpi	Average Ratio of Cycles to Instructions
cpld	Average Ratio of Cycles to L1 data cache loads
flops	Average FLOPs
VecPercent	Ratio of vectorized versus unvectorized instructions
mbw	Average Memory bandwidth
	OS Metrics
MemUsage	Maximum memory usage
CPU_Usage	Average CPU utilization
idle	Ratio of maximum to minimum CPU_Usage over nodes
catastrophe	Ratio of maximum to minimum CPU_Usage over time
MIC_Usage	Average CPU Utilization of the Intel Xeon Phi Coprocessor

was motivated by input from both system administrators and consultants. For example, high metadata request rates are always cause for concern to system administrators because they can adversely affect the Lustre filesystem's performance. High GigE traffic indicates users running their own MPI builds over the Ethernet rather than the more performant Infiniband fabric. The largemem queue is composed of expensive 1 TB nodes and is a scarce resource for job scheduling. Running jobs in this queue that do not require the extra memory delays the running of jobs that do require it. Idle nodes indicate a misconfiguration of a job submission script or lack of understanding of an application's parallel behavior on the users part. Entirely idle nodes represent a definite waste of resources for those jobs which reserve more nodes than they utilize. This is a surprisingly common occurrence with dozens of jobs with idle nodes identified daily and over 2% of jobs in the last quarter of 2015. Sudden performance increases suggest a job that consists of a compilation step before it runs, while sudden drops indicate application failure. High cycles per instruction may indicate a variety of performance issues such as a memory layout or I/O pattern that is not performant.

Aside from automatically identifying jobs that misuse resources or are performing less than optimally, these search capabilities allow us to characterize which resources are important to our users and inform user education priorities and future system acquisition decisions. For example, a search over all jobs that ran in the last quarter of 2015 on Stampede, 404,002 in total, that used more than 1% of cpu time on the MIC results in 1.3% of jobs. This suggests our user community is having difficulty taking advantage of the Xeon Phi Coprocessors and additional instruction may be of value.

A search for Stampede jobs with greater than 1% of their floating point operations vectorized returns 52% of jobs, while searching for jobs with greater than 50% vectorization returns 25%. This suggests a quarter of our applications are effectively vectorized, while almost half are not. An examination of jobs with low vectorization shows that many applications were not compiled with the most advanced vector instruction set available. This may be addressed through targeted documentation.

A search for Stampede jobs that used more than 20GB of

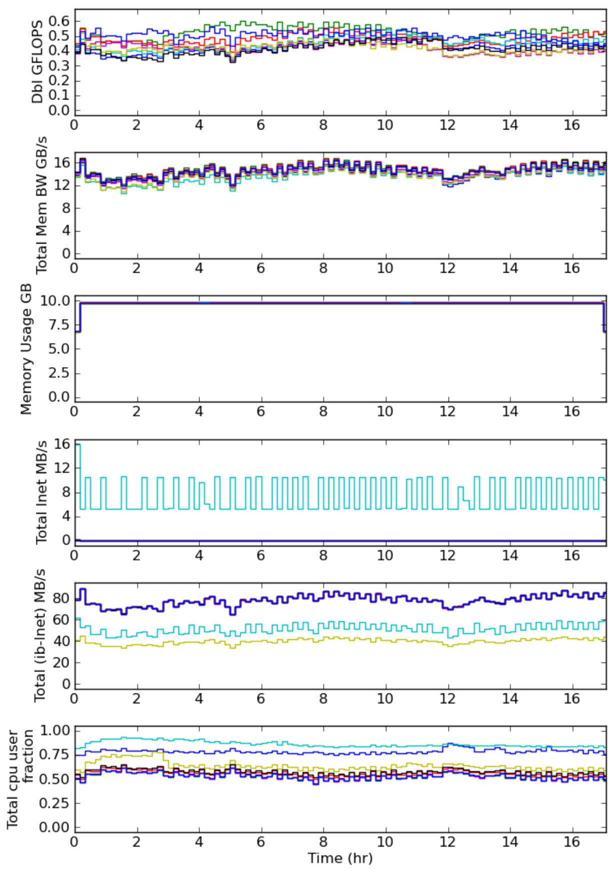


Figure 5. These plots show performance data over time for a WRF-based job that generates a large number of metadata requests. Every line on each plot corresponds to an individual node. The small bandwidth to the Lustre filesystem, show in the fourth plot from the top, suggests these requests are unnecessary. The CPU User fraction, shown in the bottom plot, is low for WRF jobs.

the possible 32GB on every node shows only 3% of jobs in this time period. This suggests that for the vast majority of users and applications larger amounts of memory are not required.

These web portal searches require no knowledge of SQL or the underlying database structure upon which the queries are made. They are thus accessible to a variety of stake holders. It is, however, possible to generate more customized or complex queries with Python scripts using Django's Object-relation mapping (ORM) interface to the PostgreSQL database as we will see in the next subsection.

B. Lustre I/O Analyses

In the previous section several jobs were identified running the WRF application that generated almost more than twice the number of metadata requests to the Lustre MDS than any other WRF jobs. This not only adds significant load to the filesystem, but is likely impacting the performance of the user's jobs. The Django ORM to the PostgreSQL database provides a variety of aggregation functions including averaging a metric field over a returned job list. We use this functionality to investigate this user's job performance.

An indication of the performance of this user's jobs can be seen in Fig. 5, which is generated automatically on the detailed view page for any job. The plots in this figure from top to bottom are the following quantities plotted over time for each node reserved for the job

- Gigaflops
- Memory Bandwidth in GB/s
- Memory Usage in GB
- Lustre Filesystem Bandwidth in MB/s
- Internode Infiniband traffic due to MPI in MB/s
- CPU User fraction: the fraction of time the job spent in user space

The CPU User fraction appears poor and varies greatly from node to node. The actual Lustre bandwidth utilization is small and restricted to a single node.

A simple query shows this user's 105 WRF jobs run in the last quarter of 2015 (of which the job illustrated in Fig. 5 is one) averaged 67% CPU_Usage and an average MetaDataRate of 563,905 requests per second. We can compare this to the average 80% CPU_Usage and 3,870 MetaDataRate for the entire WRF population of 16,741 jobs in this time period. This suggests the user's application is suffering a performance penalty for these metadata requests. We can also compare the average value for LLiteOpenCLose, the average rate of file opens and closes for a job, of the general WRF population of 2 per second to this user's rate of 30,884 per second. A quick inspection of the user's code found a loop where they inserted a statement that opened and closed a file every iteration in order to read a single parameter. We are currently working with this user to correct the issue.

In an ongoing study related to this analysis we've identified the principal predictor for poor CPU utilization: Lustre I/O. Of the 110,438 *production* jobs (jobs run in production queues that completed successfully and ran for more than an hour) that ran in the last quarter of 2015, there is a correlation coefficient of -0.11 between CPU_Usage and MDCReqs, one of -0.20 between CPU_Usage and OSCReqs, and -0.19 between CPU_Usage and LnetAveBW. CPU_Usage is used in our study because of it's ease of interpretation–any fraction of time a job's nodes are not in user space is time the job is not performing computation. Other metrics such as mbw (Memory Bandwidth) and flops are more difficult to interpret as indicators of application performance. The performance characteristics of applications vary too widely.

While it may seem intuitively obvious that applications performing more I/O are performing less computation in a given time interval, it is not obvious that large request rates to Lustre servers are necessary for these applications. I/O could be performed with fewer requests to these servers by using more performant file access patterns such as avoiding redundant file operations, moving files to local disk at the start of the job, and/or collective I/O utilities. Performance could also be improved by modifying Lustre stripe sizes and counts.

We are currently investigating methods to characterize a job's I/O performance so that targeted advice may be offered to the user without manual inspection of their application.

VI. Future Work

A. Time-series Analysis

Our work with TACC Stats has been focused on characterizing individual jobs. We recognize that the interactions between jobs can severely impact performance, particularly when interference occurs over shared resources like the Lustre filesystem. Simultaneously running jobs may individually use modest filesystem's resources but in aggregate overwhelm the managing servers. Identifying these interactions between jobs is possible with this latest version of TACC Stats but is a laborious and time-consuming process as the time-series data is organized into a separate file per job.

With this in mind we are importing data into the time-series database OpenTSDB [34]. The data in this database is organized into time-series with each series labeled by a tuple of tags, where a tag in our setup consists of a host name, device type, device name, and event name. The time-series can be aggregated along any subset of these tags and their values. Combining this data with the job metadata available in the PostgreSQL database will allow time-series data to be extracted by job, user, or application. The time-series data can then be aggregated or correlated efficiently. For instance, a particular user's metadata requests in a particular time interval from multiple jobs could be related to other users' increased Lustre operation wait times.

B. Automated Real-time Analysis

Combining this time-series analysis capability with the real time reporting recently enabled in TACC Stats will allow problem jobs to be quickly identified and suspended before they create system-wide slowdowns or crashes. This identification process could be automated and a system administrator notified immediately upon identification of problematic behavior. Ideally, this would result in fewer slow downs and more uptime for the system, resulting in higher productivity.

C. Shared nodes

Many HPC centers operate with shared nodes, where a single node may have multiple jobs running on it simultaneously. While it is impossible to definitively attribute all the data TACC Stats collects to specific jobs on shared nodes (consider memory bandwidth in a configuration where sockets are shared), we do have an approach to disentangling some of the data. A description of this scheme follows:

1) A variable containing a list of jobs currently running on the node is maintained.
2) Every *process* start up and shutdown triggers a data collection.
3) Each data collection is labeled by the list of currently running jobs
4) The `procfs` data, gathered as part of each data collection, provides a list of active processes along with their owners and cpu affinities.

This scheme guarantees at least two data points per process are taken regardless of process runtime.

We implement this scheme by loading a shared-object at process runtime using the `LD_PRELOAD` environmental variable. This library signals the `tacc_statsd` daemon at the startup and shutdown of every process using two functions, one with the gcc `constructor` attribute and one with the gcc `destructor` attribute. A function with the `constructor` attribute is run after the process begins but before the process's `main` function is run. A function with the `destructor` attribute is run after the process's `main` function is complete but before the process ends. This scheme ensures every process launched on a particular node will be tracked regardless of which job launched it, with a minimum of two data collections performed per process. If jobs are pinned to cores or sockets, such as through the use of `cgroups`, core-level and process-level data can be reliably extracted.

To perform a collection and transmit data off the node TACC Stats requires a single core for $\sim$0.09s on a system such as Lonestar 5 . Thus if large numbers of processes are continually started and ended the overhead will naturally increase from the 0.02% level that is typical for 10 minute sampling intervals. Multiple long running processes will not significantly increase the overhead because the data for all processes on a node are captured in a single collection.

There are two issues we've identified with this approach. First, there is the potential for race conditions if multiple processes start at nearly the same instant on the same node. A policy needs to be developed to handle this scenario in a consistent manner. At present, up to one signal can be captured while another signal is still being processed. This means that two processes starting simultaneously can be handled correctly. If additional processes are launched in that 0.09s runtime interval then they will be missed until the next data collection is performed. Second, an approach to extracting reliable node-level data such as network usage or I/O due to MPI and Lustre may not be possible without code instrumentation. Not withstanding these issues, many of the measurements of resource use within nodes is valid.

VII. Conclusion

We've summarized the latest capabilities introduced into TACC Stats. These new capabilities include real-time data collection, automatic configuration to chip architecture, and the collection of data from the Xeon Phi, RAPL power counters, and `procfs` filesystem. In addition to new collection capabilities, many new metrics related to Lustre I/O are now computed for every job. We described how to perform simple analyses relevant to performance and resource usage characterization through the web portal. We also apply customized analyses using Django's ORM to a Lustre I/O case study. This case study is part of a broader exploration of I/O data that has turned our focus to Lustre I/O, which appears to be the most important factor contributing to sub-optimal job performance on Stampede.

VIII. Acknowledgements

The work in this project has been supported by the SUPReMM project funded by NSF grant ACI-1445806, and has utilized and enhanced the NSF-funded systems Ranger (OCI-0622780), Lonestar4 (OCI-1042124), and Stampede (ACI-1134872).

References

[1] J. Hammond, "Tacc_stats: I/O performance monitoring for the instransigent," in *Invited Keynote for the 3rd IASDS Workshop, 2011*, 2011, pp. 1–29.

[2] J. C. Browne, R. L. DeLeon, C.-D. Lu, M. D. Jones, S. M. Gallo, A. Ghadersohi, A. K. Patra, W. L. Barth, J. Hammond, T. R. Furlani, and R. T. McLay, "Enabling comprehensive data-driven system management for large computational facilities," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC '13. New York, NY, USA: ACM, 2013, pp. 86:1–86:11. [Online]. Available: http://doi.acm.org/10.1145/2503210.2503230

[3] T. Evans, W. L. Barth, J. C. Browne, R. L. DeLeon, T. R. Furlani, S. M. Gallo, M. D. Jones, and A. K. Patra, "Comprehensive resource use monitoring for hpc systems with tacc stats," in *Proceedings of the First International Workshop on HPC User Support Tools*, ser. HUST '14. Piscataway, NJ, USA: IEEE Press, 2014, pp. 13–21. [Online]. Available: http://dx.doi.org/10.1109/HUST.2014.7

[4] T. R. Furlani, M. D. Jones, S. M. Gallo, A. E. Bruno, C.-D. Lu, A. Ghadersohi, R. J. Gentner, A. Patra, R. L. DeLeon, G. von Laszewski, F. Wang, and A. Zimmerman, "Performance metrics and auditing framework using application kernels for high-performance computer systems," *Concurrency and Computation: Practice and Experience*, vol. 25, no. 7, pp. 918–931, 2013. [Online]. Available: http://dx.doi.org/10.1002/cpe.2871

[5] T. R. Furlani, B. L. Schneider, M. D. Jones, J. Towns, D. L. Hart, S. M. Gallo, R. L. DeLeon, C.-D. Lu, A. Ghadersohi, R. J. Gentner, A. K. Patra, G. von Laszewski, F. Wang, J. T. Palmer, and N. Simakov, "Using xdmod to facilitate xsede operations, planning and analysis," in *Proceedings of the Conference on Extreme Science and Engineering Discovery Environment: Gateway to Discovery*, ser. XSEDE '13. New York, NY, USA: ACM, 2013, pp. 46:1–46:8. [Online]. Available: http://doi.acm.org/10.1145/2484762.2484763

[6] E. Chuah, A. Jhumka, S. Narasimhamurthy, J. Hammond, J. C. Browne, and B. Barth, "Linking resource usage anomalies with system failures from cluster log data," in *Proceedings of the 2013 IEEE 32Nd International Symposium on Reliable Distributed Systems*, ser. SRDS '13. Washington, DC, USA: IEEE Computer Society, 2013, pp. 111–120. [Online]. Available: http://dx.doi.org/10.1109/SRDS.2013.20

[7] J. C. Browne, R. L. DeLeon, A. K. Patra, W. L. Barth, J. Hammond, M. D. Jones, T. R. Furlani, B. I. Schneider, S. M. Gallo, A. Ghadersohi, R. J. Gentner, J. T. Palmer, N. Simakov, M. Innus, A. E. Bruno, J. P. White, C. D. Cornelius, T. Yearke, K. Marcus, G. Laszewski, and F. Wang, "Comprehensive, open-source resource usage measurement and analysis for hpc systems," *Concurr. Comput. : Pract. Exper.*, vol. 26, no. 13, pp. 2191–2209, Sep. 2014. [Online]. Available: http://dx.doi.org/10.1002/cpe.3245

[8] J. P. White, R. L. DeLeon, T. R. Furlani, S. M. Gallo, M. D. Jones, A. Ghadersohi, C. D. Cornelius, A. K. Patra, J. C. Browne, W. L. Barth, and J. Hammond, "An analysis of node sharing on hpc clusters using xdmod/tacc_stats," in *Proceedings of the 2014 Annual Conference on Extreme Science and Engineering Discovery Environment*, ser. XSEDE '14. New York, NY, USA: ACM, 2014, pp. 31:1–31:8. [Online]. Available: http://doi.acm.org/10.1145/2616498.2616533

[9] X. e. a. Zhang, "System support for automatic profiling and optimization," in *SOSP '16: Proceedings of the Sixteenth ACM Symposium on Operating Systems Principle*. New York, NY, USA: ACM Press, 1997, pp. 15–26.

[10] J. e. a. Anderson, "Continuous profiling: Where have all the cycles gone?" in *SOSP '16: Proceedings of the Sixteenth ACM Symposium on Operating Systems Principle*. New York, NY, USA: ACM Press, 1997, pp. 357–390.

[11] "OProfile." [Online]. Available: http://oprofile.sourceforge.net

[12] B. Gregg, *Systems Performance: Enterprise and the Cloud*, 1st ed. Upper Saddle River, NJ, USA: Prentice Hall Press, 2013.

[13] A. Agelastos, B. A. Allan, J. M. Brandt, P. Cassella, J. Enos, J. Fullop, A. C. Gentile, S. Monk, N. Naksinehaboon, J. Ogden, M. Rajan, M. T. Showerman, J. Stevenson, N. Taerat, and T. Tucker, "The lightweight distributed metric service: A scalable infrastructure for continuous monitoring of large scale computing systems and applications," in *SC*. IEEE, 2014, pp. 154–165.

[14] J. Brandt, B. Debusschere, A. Gentile, J. Mayo, P. Pebay, D. Thompson, and Wong, "Ovis-2: A robust distributed architecture for scalable ras. in 22nd ieee international parallel and distributed processing symposium," in *4th Workshop of System Management Techniques, Processes, and Services*, 2008.

[15] J. Brandt, K. Devine, A. Gentile, and K. Pedretti, "Demonstrating improved application performance using dynamic monitoring and task mapping," in *Cluster Computing (CLUSTER), 2014 IEEE International Conference on*. IEEE, 2014, pp. 408–415.

[16] J. Brandt, K. Devine, and A. Gentile, "Infrastructure for in situ system monitoring and application data analysis," in *Proceedings of the First Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization*. ACM, 2015, pp. 36–40.

[17] D. Del Vento, T. Engel, S. Ghosh, D. Hart, R. Kelly, S. Liu, and R. Valent, "System-level monitoring of floating-point performance to improve effective system utilization," in *High Performance Computing, Networking, Storage and Analysis (SC), 2011 International Conference for*, Nov 2011, pp. 1–6.

[18] "Performance Co-Pilot." [Online]. Available: http://www.pcp.io/index.html

[19] "HOPSA-Holistic Performance System Analysis." [Online]. Available: http://www.vi-hps.org/projects/hopsa/overview

[20] "LWM2." [Online]. Available: http://www.vi-hps.org/Tools/LWM2.html

[21] "Managing System Software for the Cray Linux Environment." [Online]. Available: http://docs.cray.com/cgi-bin/craydoc.cgi?mode=Show;q=;f=/books/S-2393-51/html-S-2393-51//chapter.2.RLKS8MHD.html

[22] "CLUMON." [Online]. Available: http://clumon.ncsa.illinois.edu

[23] "GANGLIA." [Online]. Available: http://ganglia.sourceforge.net

[24] "NAGIOS." [Online]. Available: http://www.nagios.org

[25] G. Ren, E. Tune, T. Moseley, Y. Shi, S. Rus, and R. Hundt, "Google-wide profiling: A continuous profiling infrastructure for data centers," *IEEE Micro*, vol. 30, no. 4, pp. 65–79, Jul. 2010. [Online]. Available: http://dx.doi.org/10.1109/MM.2010.68

[26] "Netflix." [Online]. Available: http://www.brendangregg.com/blog/2015-02-27/linux-profiling-at-netflix.html

[27] H. Mi, H. Wang, H. Cai, Y. Zhou, M. R. Lyu, and Z. Chen, "P-tracer: Path-based performance profiling in cloud computing systems." in *COMPSAC*, X. Bai, F. Belli, E. Bertino, C. K. Chang, A. Eli, C. C. Seceleanu, H. Xie, and M. Zulkernine, Eds. IEEE Computer Society, 2012, pp. 509–514. [Online]. Available: http://dblp.uni-trier.de/db/conf/compsac/compsac2012.html#MiWCZLC12

[28] RabbitMQ. [Online]. Available: https://www.rabbitmq.com

[29] "PostgreSQL." [Online]. Available: http://www.postgresql.org

[30] "Django." [Online]. Available: https://www.djangoproject.com

[31] K. Agrawal, M. R. Fahey, R. McLay, and D. James, "User environment tracking and problem detection with xalt," in *Proceedings of the First International Workshop on HPC User Support Tools*, ser. HUST '14. Piscataway, NJ, USA: IEEE Press, 2014, pp. 32–40. [Online]. Available: http://dx.doi.org/10.1109/HUST.2014.6

[32] "XALT." [Online]. Available: https://github.com/Fahey-McLay/xalt

[33] "The Weather Research & Forcasting Model." [Online]. Available: http://www.wrf-model.org/index.php

[34] "OpenTSDB." [Online]. Available: http://opentsdb.net