CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2015; **27**:5238–5260
Published online 20 September 2015 in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/cpe.3564

Application kernels: HPC resources performance monitoring and variance analysis

Nikolay A. Simakov*,†, Joseph P. White, Robert L. DeLeon, Amin Ghadersohi, Thomas R. Furlani, Matthew D. Jones, Steven M. Gallo and Abani K. Patra

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

SUMMARY

Application kernels are computationally lightweight benchmarks or applications run repeatedly on high performance computing (HPC) clusters in order to track the Quality of Service (QoS) provided to the users. They have been successful in detecting a variety of hardware and software issues, some severe, that have subsequently been corrected, resulting in improved system performance and throughput. In this work, the application kernels performance monitoring module of eXtreme Data Metrics on Demand (XDMoD) is described. Through the XDMoD framework, the application kernels have been run repetitively on the Texas Advanced Computing Center's Stampede and Lonestar4 clusters for a total of over 14,000 jobs. This provides a body of data on the HPC clusters operation that can be used to statistically analyze how the application performance, as measured by metrics such as execution time and communication bandwidth, is affected by the cluster's workload. We discuss metric distributions, carry out regression and correlation analyses, and use a PCA study to describe the variance and relate the variance to factors such as the spatial distribution of the application in the cluster. Ultimately, these types of analyses can be used to improve the application kernel mechanism, which in turn results in improved QoS of the HPC infrastructure that is delivered to the end users. Copyright © 2015 John Wiley & Sons, Ltd.

Received 16 December 2014; Revised 3 April 2015; Accepted 7 May 2015

KEY WORDS: performance monitoring; HPC; application kernels

1. INTRODUCTION

Here, a brief overview of XD metrics on demand (XDMoD) is presented; a more detailed description can be found in references [1–6]. XDMoD was originally designed to provide the analyses required for effective overall management of the computational and operational resources of the Extreme Science and Engineering Discovery Environment (XSEDE) [7] as part of the XSEDE Technology Audit Service. Because the operation of high performance computing (HPC) centers are conceptually very similar to XSEDE (i.e., they provide researchers with access to high-end computational resources), an open source version of XDMoD (Open XDMoD) that shares much of the functionality of XDMoD has also been developed and is widely in use at academic and industrial HPC centers and can be downloaded from http://xdmod.sourceforge.net/.

The XDMoD tool ingests and organizes data on computer system usage and performance and then maps that data into metrics required for overall system management. Conceptually, it can be considered to consist of three primary components: (1) the XDMoD portal that provides the user with access to a wide variety of usage and performance metrics through a GUI; (2) application kernels, a method for

*Correspondence to: Nikolay A. Simakov, Center for Computational Research, State University of New York at Buffalo, Buffalo, NY,USA.
†E-mail: nikolays@buffalo.edu

measuring the Quality of Service (QoS) provided by the HPC infrastructure; and (3) integrated HPC systems usage and performance of resources monitoring and modeling (SUPReMM)/TACC_Stats with Lariat, a method for measuring performance data for all jobs running on the HPC infrastructure. We briefly describe each one of these components.

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. Figure 1 shows a screenshot of the XDMoD portal. For Open XDMoD, which is designed for academic and industrial HPC centers as opposed to XSEDE, metrics are organized by a customizable hierarchy appropriate for these organizations including group, faculty member, department, and decanal unit. Performance and QoS metrics of the HPC infrastructure are also provided, along with application code-specific performance metrics (flops, I/O rates, network metrics, etc.).

In addition to providing usage data, the XDMoD tool is also designed to measure QoS and preemptively identify underperforming hardware and software by deploying customized, computationally lightweight 'application kernels' that are run frequently (daily to several times per week) to continuously monitor HPC system performance and reliability from the application users' point of view [3, 5, 6]. The term 'computationally lightweight' is used to indicate that the application kernel requires relatively modest resources for a given run frequency. 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 third primary component of the XDMoD tool is centered around monitoring the performance of all user jobs running on a given HPC resource with the goal of identifying inefficient resource use and using the collected metrics to both identify the areas of poor performance and provide information that can be utilized to remedy the inefficiency. This is accomplished through the SUPReMM program [1, 8, 9] that utilizes TACC_Stats and Lariat to collect detailed job and node level performance data without requiring recompilation of end user codes and with low system overhead (less than 0.1% on the Texas Advanced Computing Center (TACC) Stampede system). For example, TACC_Stats can monitor the CPU utilization of a user's job and determine if the CPUs are being efficiently used. Identifying and correcting poorly running codes are important, as it will allow more efficient utilization of these oversubscribed resources.

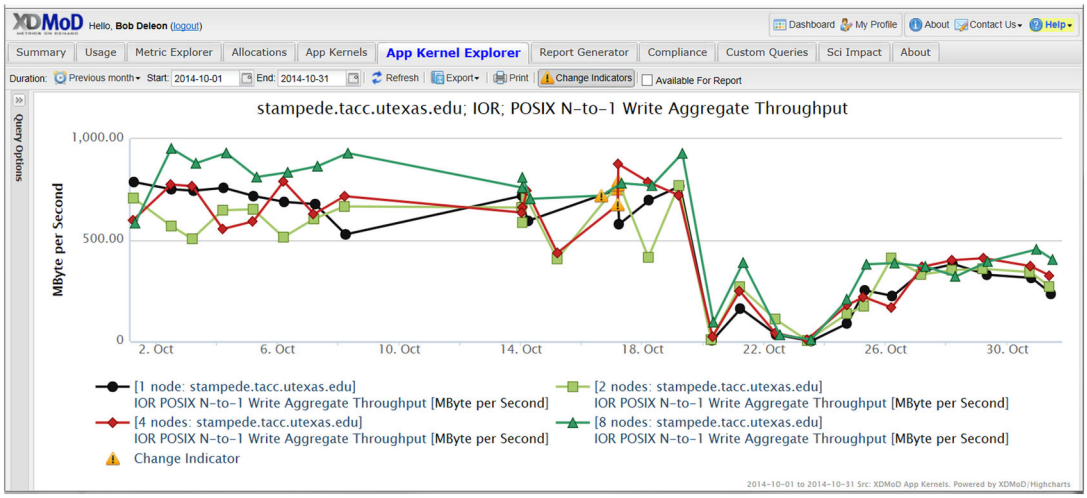


Figure 1. Screen shot of the XDMoD portal. Note the tab-based navigation. Shown is a plot of the interleave or random (IOR) file system benchmark application kernel running on Stampede. For the date range shown, the Lustre file system experienced serious problems. Texas Advanced Computing Center (TACC) made repairs that restored most of the bandwidth.

DOI: 10.1002/cpe

Taken together, these three components of XDMoD provide an effective and much needed tool for the comprehensive management of HPC systems. While there have been numerous publications related to XDMoD and SUPReMM, to date, we have not yet described the implementation and analysis of the application kernels in great detail. Accordingly, this paper focuses on application kernels and their utility in providing QoS metrics and preemptively identifying underperforming hardware and software.

The remainder of this paper is organized as follows. Related work is presented in Section 2. An overview of how application kernel monitoring is implemented is presented in Section 3. Section 3 also describes our data sources and lists examples where the use of application kernels helped identify and diagnose QoS issues on production HPC resources. The main body of the paper, Sections 4 and 5 present an analysis of variance of application kernel data run on TACC's Stampede and Lonestar4 facilities including distributions of the data, an analysis of outliers, an analysis of network latency as a function of the number of hops between the nodes, application performance as a function of Lustre system loading, a regression/correlation analysis of the various metrics, and a principle components analysis of the metrics. The final section, Section 6, provides a brief discussion of the value of using application kernels for performance analysis.

2. RELATED WORK

A number of open source and commercial monitoring tools are available to track HPC performance, and some network monitoring systems can be adopted for HPC performance monitoring purposes. Some of the more commonly used open source tools are performance co-pilot [10], Ganglia [11, 12], lightweight distributed metric service (LDMS) [13], Zabbix [14], and TACC_Stats [8, 9]. Nearly all of these tools keep track of the hardware and operating system performance counters. They are mainly concerned with the performance of the computational nodes. However, they cannot directly address the scientific software performance quantitatively, that is, the application running in a degraded mode or compare performance across different computational resources. This is partially because some of the mentioned tools do not track by application and even for those which do, per node or per job statistics cannot reveal the absolute application performance as it is related to the users' problem sizes and types. XDMoD's application kernel module addresses this by repeatedly running the same set of application kernels with the same input parameters.

Synthetic benchmarks and real-world application benchmarks are commonly used to assess the performance of hardware and software infrastructure. The most popular synthetic benchmark for HPC systems is high performance Linpack [15] that is used for ranking the top 500 most powerful computer systems. The HPC challenge (HPCC) benchmark [16] targets the overall system performance and includes multiple subtests such as Linpack, fast Fourier transform, matrix multiplication, and others. Many synthetic benchmarks target a specific subsystem: Intel message-passing interface (MPI) benchmarks (IMB) [17] (Intel, Santa Clara, CA, USA) measure node-to-node communication, interleave or random (IOR) [18], and MPI-Tile-IO [19] target the performance of high-performance parallel file systems. The purpose of the synthetic test is to show the peak performance; however, they have a very limited predictability on the performance of real-life applications [20]. To address that question, some benchmarks like Standard Performance Evaluation Corporation MPI2007 [21] extract the computationally intense routines from various applications and use them for benchmarking. Of course, applications themselves can be used as a benchmark. However, as opposed to the nearly turn-key solution of synthetic benchmarks, the interested party will need to identify a proper test problem.

Benchmarks are also often used to fine-tune the performance of HPC resource during initial deployment. During production, it is considered a good practice to regularly rerun them especially after major changes in the infrastructure. Usually, these benchmarks are run and analyzed manually or using custom local 'in-house' scripts. The application kernel module of XDMoD provides automatic tools for the regular execution of application kernels and performance analysis. Some of

DOI: 10.1002/cpe

the application kernels use 'real-world' applications, preferably installed resource wide. The historic data allow comparisons among the wide range of XSEDE resources. After the initial setup, the interested system manager can receive periodic reports on their HPC resource performance or choose to receive notification only when problems are detected.

3. APPLICATION KERNELS PERFORMANCE MONITORING MODULE

As mentioned earlier, the main goal of the application kernels module is to monitor the performance of HPC resources to help ensure QoS and preemptively identify underperforming or failed hardware and software. When a new resource is introduced, a great deal of work is typically expended fine-tuning its computational performance. Unfortunately, substantially less work is carried out to ensure its continued operational quality. However, during the lifetime of the resource, often substantial changes happen to the software stack and hardware configuration. Therefore, there is a need for an automatic tool that will be able to identify underperforming software and hardware and inform interested parties about problems that arise prior to impacting users and causing wasted cycles on resources that are typically oversubscribed. The application kernels performance monitoring module accomplishes this goal by a periodic execution of computationally lightweight application kernels and an analysis of their performance metrics (for example, wall time).

3.1. Operation overview

The application kernel module of XDMoD consists of three parts. (1) The application kernel remote runner (AKRR) executes the scheduled jobs, monitors their execution, processes the output, extracts performance metrics, and exports the results to the database. (2) The application kernel process control identifies poorly performing individual jobs. (3) The application kernel automatic anomaly detector analyzes the performance of all application kernels executed on a particular resource and automatically recognizes poorly performing application kernels.

3.1.1. Application kernel remote runner. Application kernel remote runner executes application kernels on HPC resources using the same mechanism as a regular user, for example, it uses secure shell to access the system and submits job scripts through the system scheduler. This allows for not only monitoring the performance of the application kernels themselves but also testing the whole workflow that regular users employ in order to carry out their work.

Application kernel remote runner was designed to automatically execute a large number of jobs on a number of HPC resources according to a user-defined schedule. To achieve high reliability, a multi-process design was chosen where the master process dispatches a small self-contained subtask to the child processes. This allows the master process code to be relatively simple and moves the more complicated code to the child processes. This way, a severe error on one of the child processes does not cause the whole system to collapse.

Although application kernels are computationally lightweight and their pure execution time lies between minutes to half an hour, the total time from job script creation to loading the results to the database can easily take several days, due to potentially long-queue wait times. In order to manage a large number of jobs and to be able to recover from critical failures, the entire application kernel execution task is split into small self-contained subtasks (see Table I for the subtasks). Each subtask is executed by a child process and should take only a few seconds. At the end of each subtask, the current job state is dumped to the file system. This allows us to recover to the last known state of AKRR in the case of a critical failure.

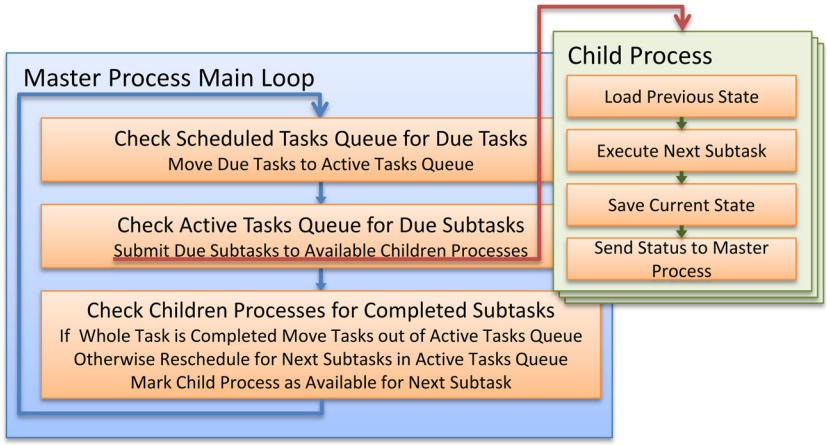
The AKRR master process utilizes two queues for job tracking: The first one is named 'scheduled tasks' and contains the jobs scheduled for execution in the future, and the second one is named 'active tasks' and contains jobs that are currently being executed (Figure 2). When the scheduled time occurs, the job is moved from the scheduled tasks queue to the active tasks queue with the current due time. When a job in the active tasks queue is due, the master process dispatches a subtask of this job to a child process. The child process executes the subtask, and in the case of a successful execution, it requests the master process to schedule the next subtask for execution;

DOI: 10.1002/cpe

Table I. Subtasks of the application kernel jobs.

Number	Subtask/step description
1	Create a job script.
2	Copy it to HPC resource and submit to queue. Reschedule to repeat until success or allowed number of attempts is exceeded.
3	Check job status on HPC resource. Reschedule to repeat until job complete or allowed in-queue time is exceeded.
4	Collect the job output if it is present.
5	Process the output if it was collected.
6	Load results to database. Reschedule to repeat until success or allowed number of attempts is exceeded.

HPC, high performance computing.



Scheduled Tasks Queue

Due Time	Resource	App.Kernel	N_{nodes}	Repeat in
2014-08-10 01:15	Stampede	NAMD	8	1 day
2014-08-10 01:15	Trestles	NWCHEM	4	1 day
2014-08-11 05:45	Gordon	GAMESS	4	3 day
2014-08-11 01:15	Lonestar	HPCC	16	1 day

Active Tasks Queue

Due Time	Resource	App.Kernel	N_{nodes}	Status	Next Step
2014-08-10 01:10	Stampede	NAMD	4	New Task	Create and Submit to Queue
2014-08-10 01:12	Gordon	NWCHEM	8	Still in Queue	Check the Queue

Figure 2. Illustration of the application kernel remote runner (AKRR) master process. The 'scheduled tasks' queue contains the jobs scheduled for execution in the future. When the scheduled time occurs, the job is moved to the 'active tasks' queue, and at the due time, the master process dispatches a subtask of this job to a child process. The child process executes the subtask, and in the case of a successful execution, it requests the master process to schedule the next subtask for execution; otherwise, the same subtask is rescheduled for future execution. When all subtasks are completed or the allowed number of rescheduling attempts is exceeded, the job is moved out of the active tasks queue.

otherwise, the same subtask is rescheduled for future execution. When all subtasks are completed or the allowed number of rescheduling attempts is exceeded, the job is moved out of the active tasks queue.

3.1.2. Application kernel process control. Because the same input file is used each time for a given application kernel, the execution time or other metric that the application kernel is designed to measure (for example, I/O rate) should be relatively constant from one run to the next. Indeed, the

DOI: 10.1002/cpe

very basis for the use of the application kernels is to provide QoS metrics. Because there are a substantial number of application kernels and there can be several metrics associated with each application kernel, it is desirable to have an automated process for determining if each application kernel is performing within its normal bounds. We refer to this scheme as process control.

Process control monitors the performance of each individual application kernel metric and automatically identifies underperforming regions. Figure 3 illustrates the identified performance regions for the ping bandwidth metric of the IMB kernel. Each data point is categorized as either in control, out of control, or better than control. The control region is defined to be the normal operating envelope of the application kernel. In the beginning, the control region is automatically selected and is often associated with the installation of a new application kernel. The process, in this case, the performance of a given application kernel, is assumed to be nominally in control in this region. If the five-point running average at a given point beyond the control region exceeds a specified tolerance (based upon the data range in the control region), the process is flagged as out of control. In Figure 3, the region where the network PingPing bandwidth metric of the IOR application kernel drops substantially is automatically evaluated to be out of control. The control region is automatically readjusted to account for software environment updates. The updates are determined by the change in application signature [22]. The five-point running average used to determine the baseline for a given

Figure 3. Application kernel process control. The time history for the network benchmarking application kernel, Intel message-passing interface (MPI) benchmarks (IMB), demonstrates an underperforming region. The solid blue line is the data, the dashed black line is a five-point average, and the blue shading indicates the control zone range. The red zones indicate that the process is out of control in an unfavorable sense, while the green zones indicate superior performance compared to the in-control performance.

DOI: 10.1002/cpe

metric measured by a specific application kernel argues in favor of running the application kernels fairly frequently (daily or several times a week at least). For example, if they are only run once per week, then it may be several weeks before enough runs have occurred for the process control algorithm to automatically identify a poorly performing application kernel. The end result being that the HPC resource may have been running in a degraded mode for that entire time.

3.1.3. Application kernel automatic anomaly detector. The automatic anomaly detector creates a performance map for all application kernels executed on a particular HPC resource (Figure 4). The performance map is a discrete heat map depicting the dependency of the application kernel performance on its execution on a given day. The single day performance of an application kernel on a given resource is summarized as a discrete variable that can take three values. These values are color coded as follows: Green means the application kernel ran successfully (was in-control) at least once on that day, yellow indicates that the application kernel ran but was out of control, and red indicates the failure of the application kernel to run for all runs that day. This discrete summary is

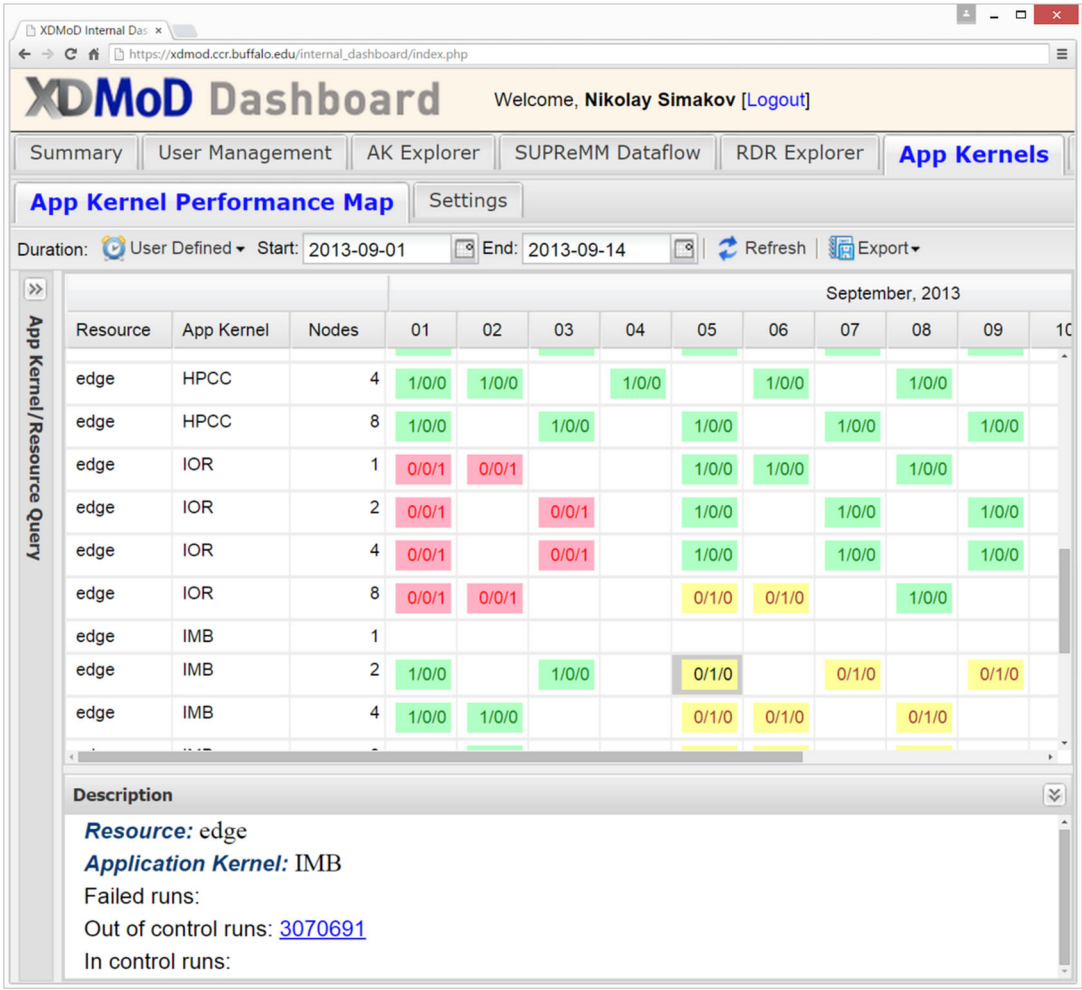


Figure 4. Portion of the application kernel performance map for the Edge cluster at Center for Computational Research (CCR) showing which kernels ran normally (green), which ones ran sub-optimally (yellow), and which ones failed to run (red). The three integers in each colored box, which are referred to as a triplet, are respectively the number of runs of the application kernel that was in-control, the number out of control, and the number of times the application kernel failed to run on that particular day. For example, 1/0/1 would refer to an application kernel that ran two times, and one of the times was in-control, and the other time failed to run. Similarly, 2/1/0 would indicate that the application kernel ran three times, and two of those times were in-control, and one time was out of control.

DOI: 10.1002/cpe

used for the creation of a performance heat map for automatic anomaly detection. Using pattern analysis, the following problems are detected: (1) All application kernels failed/underperformed on a given day or several days in a row; (2) specific application kernels failed/underperformed on all node counts on a given day or several days in a row; and (3) specific application kernels failed/underperformed on certain node count on a given day or several days in a row. Later on, the identified problems are sorted according to severity and sent to interested personnel via the execution summary report (Figure 5). In addition, the report also presents application kernel execution statistics and the performance map. The reports can be delivered by e-mail to subscribed users on a regular basis (daily, weekly, or monthly) or in case of degraded performance. The application kernel process control and the application kernel automatic anomaly detector allow site administrators to easily monitor application kernel run failures for troubleshooting performance issues at their site.

3.2. Application kernels

In order to provide a useful characterization of HPC resource performance, a set of application kernels should test critical HPC resource components and should closely resemble the calculations performed by the real users. Our selection of application kernels can be separated into two categories: benchmark-based and real application-based application kernels. The first set addresses the performance of HPC resource components, while the second set addresses the performance of real applications. Furthermore, because different science fields can utilize similar numerical methods, the performance of some applications and benchmarks can, to some extent, be transferred to other applications (for more details, see the Berkeley 'dwarfs' classification of numeric methods [23] and its overlap with application areas [24]). Currently, eight application kernels are used for performance monitoring: Northwest computational chemistry program (NWChem) [25], general atomic and molecular electronic structure system (GAMESS) [26], nanoscale molecular dynamics program (NAMD) [27], Enzo [28], Graph500 [29], HPCC [16], IMB [17], and IOR [18]. Figure 6 shows the top 20 applications executed on Stampede during the 2013 year. As can be seen, molecular dynamics simulation dominates the resource. Electronic structure calculations and applications that heavily rely on PDE solving are also among the top applications. Our current selection of application kernels provides a good coverage for this job mixture. In the rest of this section, we briefly describe each application kernel to characterize the type of information that each provides on HPC operations; more details on the application kernel input parameters can be found in APPENDIX A.

Northwest computational chemistry program (NWChem) [25] is a heavily used computational chemistry code that spans the gamut from molecular mechanics and molecular dynamics to full *ab initio* calculations. Its design goal was to handle a wide range of problems in quantum chemistry and dynamics. It was designed to be scalable to take advantage of the computational capability of large HPC clusters to address large problem sizes.

General atomic and molecular electronic structure system code (GAMESS-US) [26] is an *ab initio* computational chemistry code.

Nanoscale molecular dynamics program (NAMD) [27] is a parallel molecular dynamics code that was specifically designed for high performance simulations on large HPC clusters. It is routinely one of the most heavily run applications on the large XSEDE compute resources. It is primarily used for large biochemical problems. It has been scaled to thousands of cores.

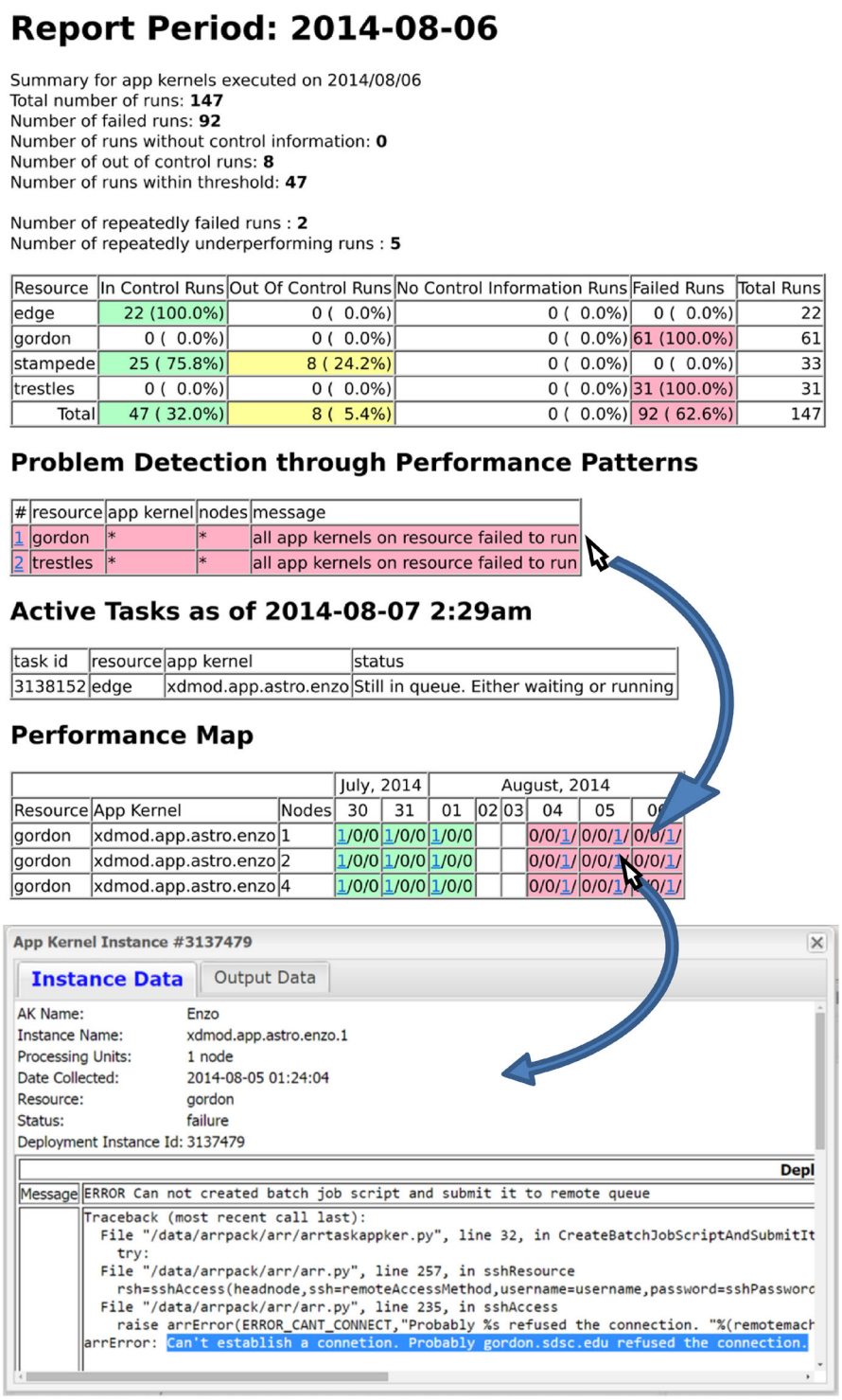
Graph500 [29] is a benchmark that measures the performance of breadth-first search on a graph. There are two kernels; the first generates a graph and compresses it, and the second does a parallel search over random vertices.

The HPCC benchmark [16] is a benchmark that consists of seven tests: (1) Linpack; (2) matrix multiplication; (3) memory bandwidth; (4) parallel matrix transpose; (5) random memory access; (6) fast Fourier transform; and (7) bandwidth and latency. The goal is to measure a wide range of HPC operations.

Intel MPI benchmark (IMB) [17] is an Intel benchmark that measures MPI node-to-node communication.

Interleave or random benchmark [18] is used to test parallel file systems using portable operating system interface (POSIX), MPI-IO, and hierarchical data format 5 (HDF5) interfaces.

DOI: 10.1002/cpe



Report Period: 2014-08-06

Summary for app kernels executed on 2014/08/06
Total number of runs: **147**
Number of failed runs: **92**
Number of runs without control information: **0**
Number of out of control runs: **8**
Number of runs within threshold: **47**

Number of repeatedly failed runs : **2**
Number of repeatedly underperforming runs : **5**

Resource	In Control Runs	Out Of Control Runs	No Control Information Runs	Failed Runs	Total Runs
edge	22 (100.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	22
gordon	0 (0.0%)	0 (0.0%)	0 (0.0%)	61 (100.0%)	61
stampede	25 (75.8%)	8 (24.2%)	0 (0.0%)	0 (0.0%)	33
trestles	0 (0.0%)	0 (0.0%)	0 (0.0%)	31 (100.0%)	31
Total	47 (32.0%)	8 (5.4%)	0 (0.0%)	92 (62.6%)	147

Problem Detection through Performance Patterns

#	resource	app kernel	nodes	message
1	gordon	*	*	all app kernels on resource failed to run
2	trestles	*	*	all app kernels on resource failed to run

Active Tasks as of 2014-08-07 2:29am

task id	resource	app kernel	status
3138152	edge	xdmod.app.astro.enzo	Still in queue. Either waiting or running

Performance Map

			July, 2014		August, 2014					
Resource	App Kernel	Nodes	30	31	01	02	03	04	05	06
gordon	xdmod.app.astro.enzo	1	1/0/0	1/0/0	1/0/0			0/0/1/	0/0/1/	0/0/1/
gordon	xdmod.app.astro.enzo	2	1/0/0	1/0/0	1/0/0			0/0/1/	0/0/1/	0/0/1/
gordon	xdmod.app.astro.enzo	4	1/0/0	1/0/0	1/0/0			0/0/1/	0/0/1/	0/0/1/

App Kernel Instance #3137479

Instance Data | Output Data

AK Name: Enzo
Instance Name: xdmod.app.astro.enzo.1
Processing Units: 1 node
Date Collected: 2014-08-05 01:24:04
Resource: gordon
Status: failure
Deployment Instance Id: 3137479

Depl

Message ERROR Can not created batch job script and submit it to remote queue

```
Traceback (most recent call last):
  File "/data/arrpack/arr/arrtaskappker.py", line 32, in CreateBatchJobScriptAndSubmitIt
    try:
  File "/data/arrpack/arr/arr.py", line 257, in sshResource
    rsh=sshAccess(headnode,ssh=remoteAccessMethod,username=username,password=sshPasswor
  File "/data/arrpack/arr/arr.py", line 235, in sshAccess
    raise arrError(ERROR_CANT_CONNECT,"Probably %s refused the connection. "%(remotemach
arrError: Can't establish a connetion. Probably gordon.sdsc.edu refused the connection.
```

Figure 5. Fragment of an e-mail report generated by the automatic anomaly detector. The e-mail contains hypertext markup language (HTML) links that will redirect to areas of interest within e-mail or to the XDMoD website to view a detailed report on the individual application kernel run. Throughout the report, color coding is used: green for normally performing jobs, yellow for underperforming, and red for completely failed runs. In this report, it was determined that all jobs submitted to the Gordon high performance computing (HPC) resource failed to execute. The performance map shows that given application kernels were performing properly at the end of July and suddenly failed to execute. Clicking on particular jobs will bring a detailed job execution report, which indicates an inability to access the resource. Further investigation revealed the change in resource access method from GlobusSSH to OpenSSH.

DOI: 10.1002/cpe

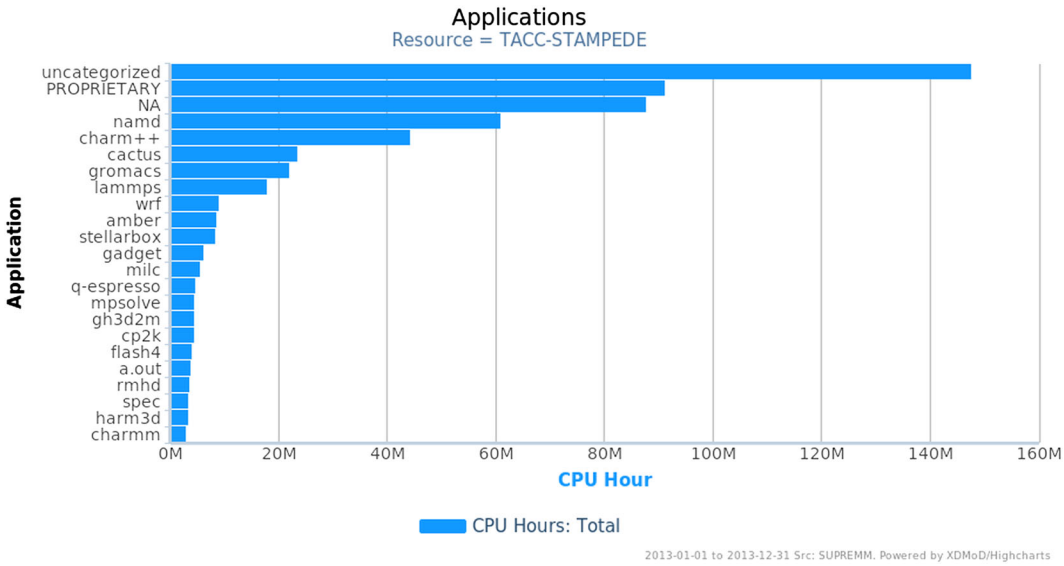


Figure 6. Applications ranking on Stampede for the year 2013. TACC, Texas Advanced Computing Center.

Enzo [28] is an adaptive mesh refinement code for astrophysical simulation of cosmological structure formation. The purpose of this code in our application kernel collection is to address the performance of scientific applications that utilize adaptive mesh refinement in solving PDEs.

The proper choice of application kernel input parameters is critical for keeping total execution time small but still being large enough to be representative of a real-life workload. Further, complicating these requirements, it is also desirable to use the same problem for all node counts in order to monitor the parallel scaling. Furthermore, these input parameters eventually need to be readjusted because of growth of computational power and advances in the user applications. For NWChem, GAMESS, NAMD, Graph500, and Enzo, we choose to use the same problem size, and for HPCC, IMB, and IOR, the problem size is proportional to the node count. The input files used our suite of application kernels is distributed with the application kernel toolkit, which is part of the release of Open XDMoD. The parallel scaling of application kernels with chosen inputs is shown in Figure 7. The total run-time for the majority of application kernels lies in the 1–30-min range.

3.3. The use of application kernels to assure QoS

The application kernels deployed as part of XDMoD have demonstrated their utility in helping to maintain the QoS on XSEDE resources. As one example, see Figure 8 taken directly from an XDMoD plot. Poor performance of the most recent version of NWChem 6.3 was identified through the application kernels on the Kraken supercomputer at the National Institute for Computational Sciences (NICS). Initially, it was discovered through the NWChem application kernel that NWChem version 6.1.1 had stopped running. Our analysis showed that this version had been recompiled by the service provider and had broken a dependency on the optimized linear algebra library if the default launching scheme was used (i.e., 'module load NWChem'). In addition, as shown in Figure 8, the most recent version of NWChem (NWChem 6.3) suffered a significant (three to eight times) decrease in performance and inverted scaling behavior. The service provider was informed about both issues, and as a result, NICS issued a notification e-mail informing users about the poor performance of NWChem 6.3 and reverted to a prior version of NWChem (version 6.1). The degraded performance of version 6.3 was later traced to drop support for the interconnect used in Kraken.

Another example, this time uncovering a system-wide performance degradation in the parallel file system used for scratch space on the HPC resource at the Center for Computational Research – University at Buffalo (CCR), is shown in Figure 9. Here, the IOR application kernel clearly

DOI: 10.1002/cpe

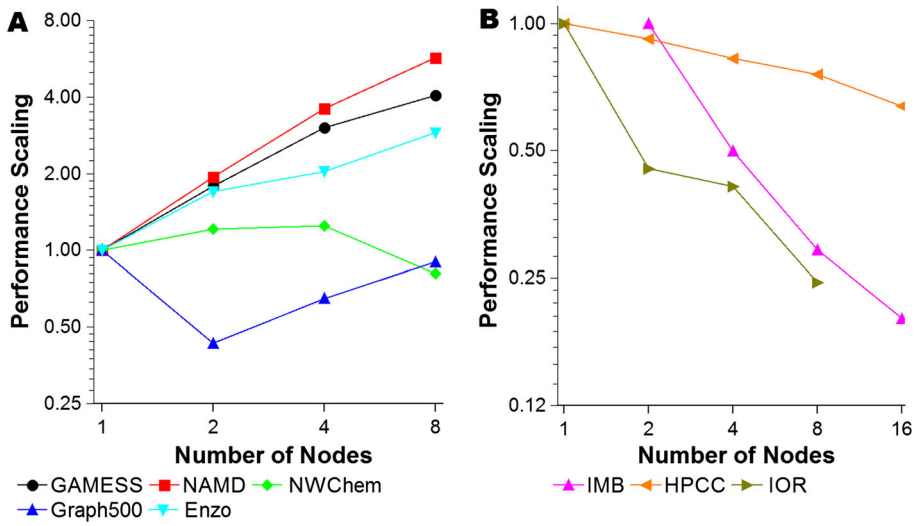


Figure 7. Application kernels parallel performance scaling on Stampede. (A) Application kernels with constant problem size. (B) Application kernels for which problem size grows as the number of nodes. Performance scaling is defined as the ratio of execution wall time on single nodes to wall time on multiple nodes. In case of nanoscale molecular dynamics program (NAMD) and Graph500, reverse simulated Newton second per day and reverse traversed edges per seconds (TEPS) were used instead of wall time. In our implementation of the Graph500 benchmark, on a single node, we use the pure OpenMP version as it is substantially faster than the hybrid OpenMP/message-passing interface (MPI) version utilized for multi-node configurations. GAMESS, general atomic and molecular electronic structure system; NWChem, Northwest computational chemistry program; IMB, Intel message-passing interface (MPI) benchmarks; HPCC, high performance computing challenge; IOR, interleave or random.

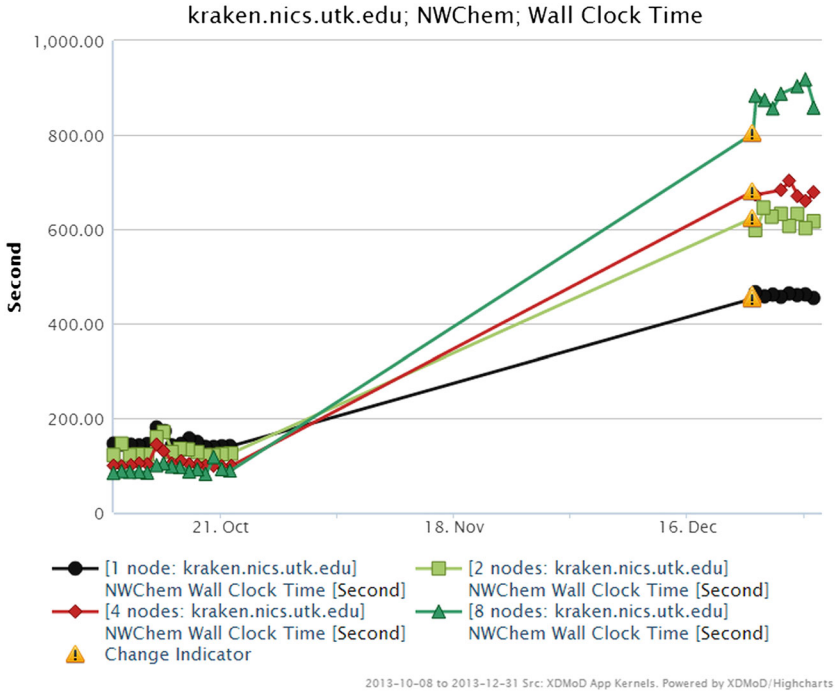


Figure 8. Northwest computational chemistry program (NWChem)-based application kernel discovers performance degradation on Kraken caused by a new version of NWChem. Performance is measured as total wall time. The points on the left correspond to earlier compilation of NWChem 6.1.1 and on the right to current version of 6.3. The missing points in the middle correspond to the failure of the 6.1.1 version. Version 6.3 is 3–8 times slower than 6.1.1 and does not exhibit any parallelism.

DOI: 10.1002/cpe

shows a dramatic and sudden drop-off in write performance in January 2014. Because the application kernels are run frequently (daily in this case), CCR support staff were able to associate the drop-off in performance with a routine firmware upgrade of the center's core network switch. As shown in Figure 9, the file system performance was eventually returned to normal after working closely with the switch manufacturer and the vendor of the parallel file system.

Additional application kernel success stories include (1) detection of a 25% degradation of NAMD performance after an 'upgrade' [3]; (2) detection of a software bug in the I/O stack of a commercial parallel file system that caused sporadic behavior in NWChem [5]; and (3) a sudden decrease in file system performance on TACC Lonestar4 as measured by three different application kernels (IOR, MPI-Tile-IO, and IMB) [6].

4. VARIANCE ANALYSIS OF APPLICATION KERNEL DATA

This section of the paper includes a detailed description of variance analysis of the performance metrics for application kernels that ran on two XSEDE resources, TACC Lonestar4, and TACC Stampede. Because each application kernel is by definition the exact same application run on the same problem, in principle, each run on a given resource should take the same time and have identical metrics. This of course is true only for the unrealistic ideal case that all of the other jobs concurrently running on the resource have no effect. In reality, rather than a single repetitive value, there is a distribution for each metric that is characteristic of how the other jobs interact with the application kernel performance. Therefore, an analysis of variance of the application kernel data for the various metrics gives us information on how the jobs in the job mixture interact with one another on a given resource. The different nature of the various application kernels and different metrics make them the ideal probes of this interaction.

4.1. Data sources

4.1.1. High performance computing systems. The case studies given as examples in this paper were carried out on the Stampede and Lonestar4 supercomputers at the TACC. Stampede has 6400 nodes each of which resides in a Dell PowerEdge C8220z chassis (Dell, Round Rock, TX, USA) and contains two 8-core Intel Xeon E5-2680 processors, 32 GB of memory and one Intel Xeon SE10P Phi coprocessor (Intel, Santa Clara, CA, USA). The file system is Lustre, and the interconnect is

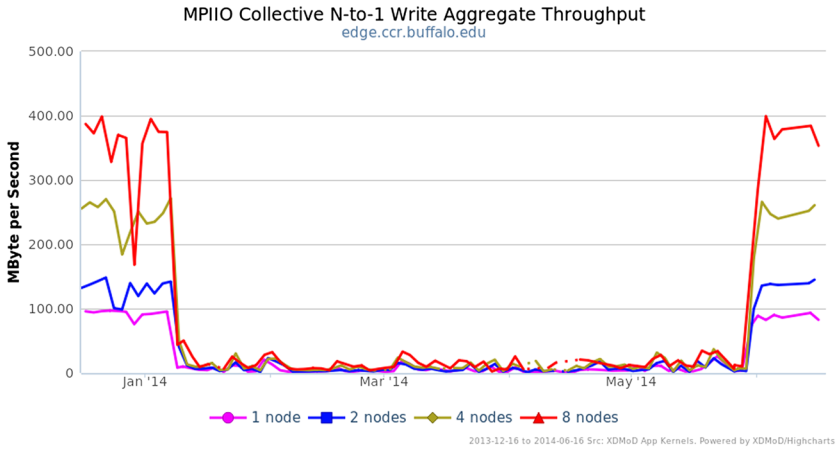


Figure 9. Interleave or random (IOR)-based application kernel that shows the degradation in the parallel file system at Center for Computational Research (CCR) that occurred suddenly in January 2014. The date of the performance degradation, which was identified because of the periodic running of the application kernels, was tied to an upgrade to the core network switch. Message Passing Interface parallel Input/Output (MPI-IO) library was used for parallel file access.

DOI: 10.1002/cpe

Fourteen Data Rate (FDR) InfiniBand. Lonestar4 is also a Linux cluster with 1888 Dell PowerEdgeM610 compute nodes. Each compute node has two Intel Xeon 5680 series 3.33 GHz hexa-core processors and 24 GB of memory. Lonestar4 has two file systems: Lustre and network file system (NFS), and its interconnect is Quad Data Rate (QDR) InfiniBand (NFS is connected via Ethernet).

4.1.2. Augmentation with systems usage and performance of resources monitoring and modeling/TACC_Stats data. Data from the TACC_Stats tool have recently been added to the XDMoD data warehouse. Therefore, it is now possible to enhance the application kernels performance metrics with hardware performance counter data.

TACC_Stats [8, 9, 30, 31] is a tool for HPC resource performance measurement and analysis on the job level. TACC_Stats executes and collects the values from performance registers at the beginning of a job, periodically during the job (currently every 10 min) and at the end of the job. For each executed job, TACC_Stats can gather core-level CPU usage (user time, system time, idle time, 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 inter-process communication), software/hardware interrupt request count, file systems usage (NFS, Lustre, and Panasas), interconnect fabric traffic (InfiniBand and Myrinet), and CPU hardware performance counters. For a complete list of the data acquired by TACC_Stats, see the TACC_Stats web site [9].

TACC_Stats has been deployed on Stampede since early 2013 and on Lonestar4 since November of 2011. We analyzed TACC_Stats application kernel data collected on Stampede during July 2013 to June 2014 and on Lonestar4 from March 2013 to June 2014.

4.2. Metric distributions

The wall time is certainly a key metric for the application kernels. Upon analyzing the complete data from the eight application kernels over the two resources and the ~14,000 jobs, it was found that in general, the application kernel wall times were positively skewed distributions. That is, the distribution has a long tail on the right side (to longer run times). For example, see Figures 10 and 11 that give the wall time distribution of NWChem and NAMD running on Stampede and Lonestar4. The NWChem and NAMD wall time distributions shown in Figures 10 and 11 are typical of all of the eight application kernels in that they all exhibit this positive skew. These positively skewed distributions are caused by their being a minimal time that it takes to run the application kernel and a long tail that is a characteristic of the variations that typically occur. For example, the long tail to larger wall times shows that other jobs running concurrently on the same resource typically cause an increase in execution time.

We also examined the distributions of various other metrics. Overall FLOPS, average memory bandwidth, and average InfiniBand transmit and receive rates have mainly negative skew coefficients (that is, the distribution has a long tail on the left side), whereas the CPU idle percent and the CPU imbalance are characterized by positive skew coefficients. CPU imbalance is defined as the (max–min)/max of the average CPU user per cent over all cores that the job was assigned. For most metrics, the skew coefficients have the same sign and similar magnitude for the Stampede and Lonestar4 data. For most metrics, the direction of the skew goes in the direction of poorer performance. That is, metrics where poorer performance is to the higher side (such as wall time) tend to be positively skewed and those where poorer performance is to the lower side (such as memory bandwidth) tend to be negatively skewed.

4.3. Outliers analysis

We define an outlier as a job for which the wall time value is more than three standard deviations above or below the mean value for that application. As shown in Table II, the outlier rate for all application kernels are less than 3% on both Stampede and Lonestar4. Another measure of how consistently these applications run is given by the coefficient of variation (the sample standard deviation/sample mean). Fairly broad distributions are observed for IMB and IOR on Lonestar4 and NWChem on both systems.

DOI: 10.1002/cpe

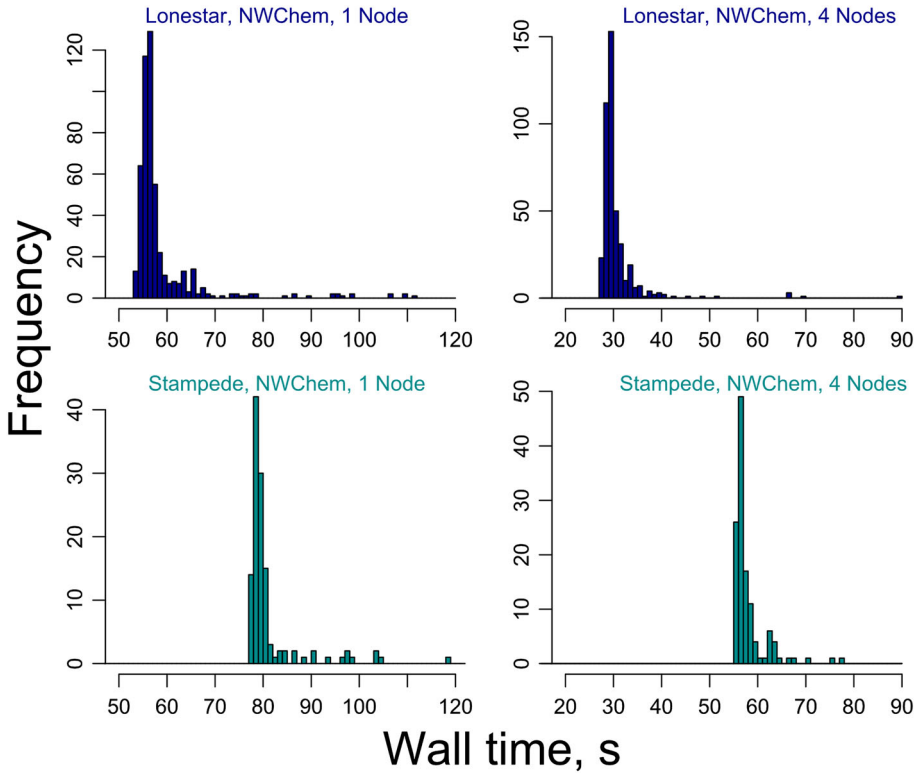


Figure 10. Wall time distributions for the northwest computational chemistry program (NWChem) application kernel on Lonestar4 (top) and Stampede (bottom) on one and four nodes.

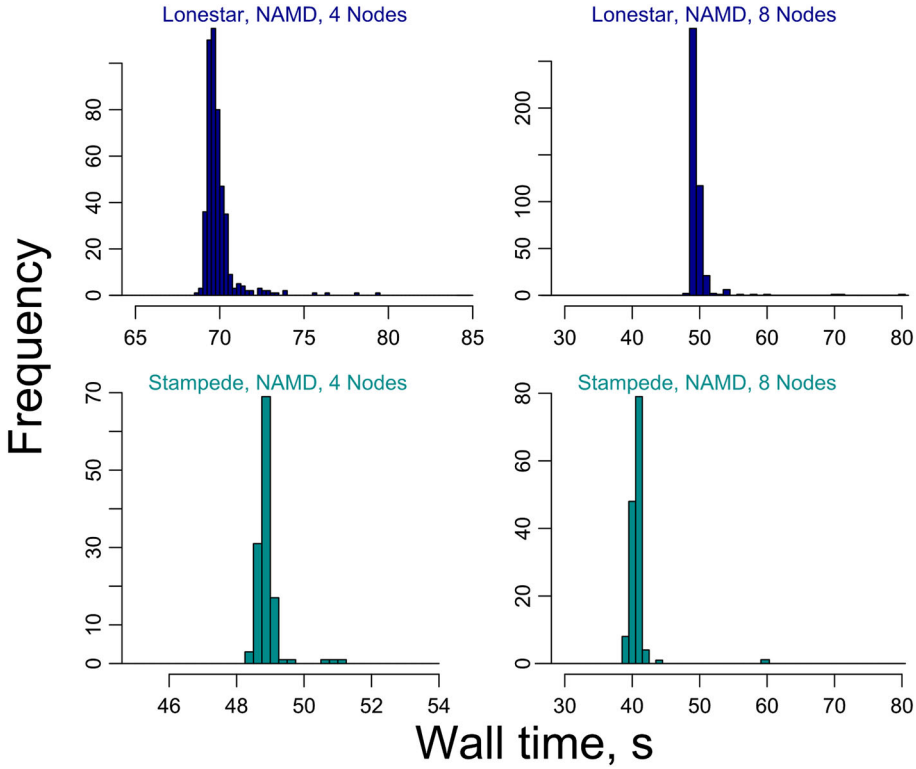


Figure 11. Wall time distributions for the nanoscale molecular dynamics program (NAMD) application kernel on Lonestar4 (top) and Stampede (bottom) on four and eight nodes.

DOI: 10.1002/cpe

The broad distributions for these applications are likely because they are dependent on the performance of shared machine resources. IOR and NWChem performance depends on the parallel file system performance, which in turn depends on the other jobs running concurrently. The MPI benchmark IMB strongly depend on the interconnect performance. We discuss the file system and interconnect performance in more details in Sections 4.4 and 4.5.

4.4. Influence of system nodes distribution on application kernel performance

The majority of large parallel jobs executed on supercomputing resources are characterized by significant network traffic. Network bandwidth and latency can significantly affect the performance of scientific applications [32, 33]. Poor choices of interconnect can result in weak to non-existent application parallel scaling. Therefore, it is not surprising that a substantial fraction of the supercomputer cost comes from the interconnect. Here, we use application kernel data to analyze how the distribution of the job dedicated nodes on the system network topology affects its performance.

The relative location of two hosts in the network can be characterized by hop count, which is the number of intermediate devices (e.g., switches and routers) separating the two hosts. The number of intermediate switches directly affects the network latency, and the bandwidth can be substantially reduced because of the traffic from other nodes. Thus, the expectation is that a higher hop count should result in poorer performance for network traffic intensive applications. To characterize the job's positioning on the network for multi-node jobs (two nodes and more), we utilize an average hop count obtained by averaging over all possible node pairs. Other hop count averaging schemes show qualitatively similar results. Figure 12 shows the dependency of wall time on average hop count (over InfiniBand). IMB's sole purpose is to determine the highest possible interconnect performance; therefore, it is not surprising that it exhibits the strongest dependency on average hop count. The real world-based application kernels such as GAMESS, NWChem, NAMD, and Enzo do not show much dependency on the hop count. This is not too surprising because both Lonestar4 and Stampede are high-end supercomputing resources with fast interconnect and a great deal of effort was expended in developing these applications to optimize the inter-node communication. There are two main differences between Lonestar4 and Stampede. First is that on Lonestar4, jobs are concentrated more towards the high hop count, whereas on Stampede, they

Table II. Application kernel wall time outliers. Coefficient of variation is defined as sample standard deviation divided by sample mean.

Resource	Application kernel	Number of jobs	Number of outliers	Fraction of outliers	Coefficient of variation
Stampede	IMB	544	8	0.015	0.110
Stampede	GAMESS	539	14	0.026	0.032
Stampede	NAMD	522	3	0.006	0.019
Stampede	NWChem	513	8	0.016	0.719
Stampede	HPCC	661	18	0.027	0.025
Stampede	IOR	364	5	0.014	0.195
Stampede	Graph500	629	9	0.014	0.200
Stampede	Enzo	269	0	0.000	0.064
Lonestar4	IMB	1648	17	0.010	0.433
Lonestar4	GAMESS	1672	28	0.017	0.099
Lonestar4	NAMD	1879	17	0.009	0.074
Lonestar4	NWChem	1826	22	0.012	0.340
Lonestar4	HPCC	2194	20	0.009	0.113
Lonestar4	IOR	757	10	0.013	0.833
Lonestar4	Graph500	480	1	0.002	0.013
Lonestar4	Enzo	241	1	0.004	0.059

IMB, Intel message-passing interface (MPI) benchmarks; GAMESS, general atomic and molecular electronic structure system; NAMD, nanoscale molecular dynamics program; NWChem, northwest computational chemistry program; HPCC, high performance computing challenge; IOR, Interleave or random.

DOI: 10.1002/cpe

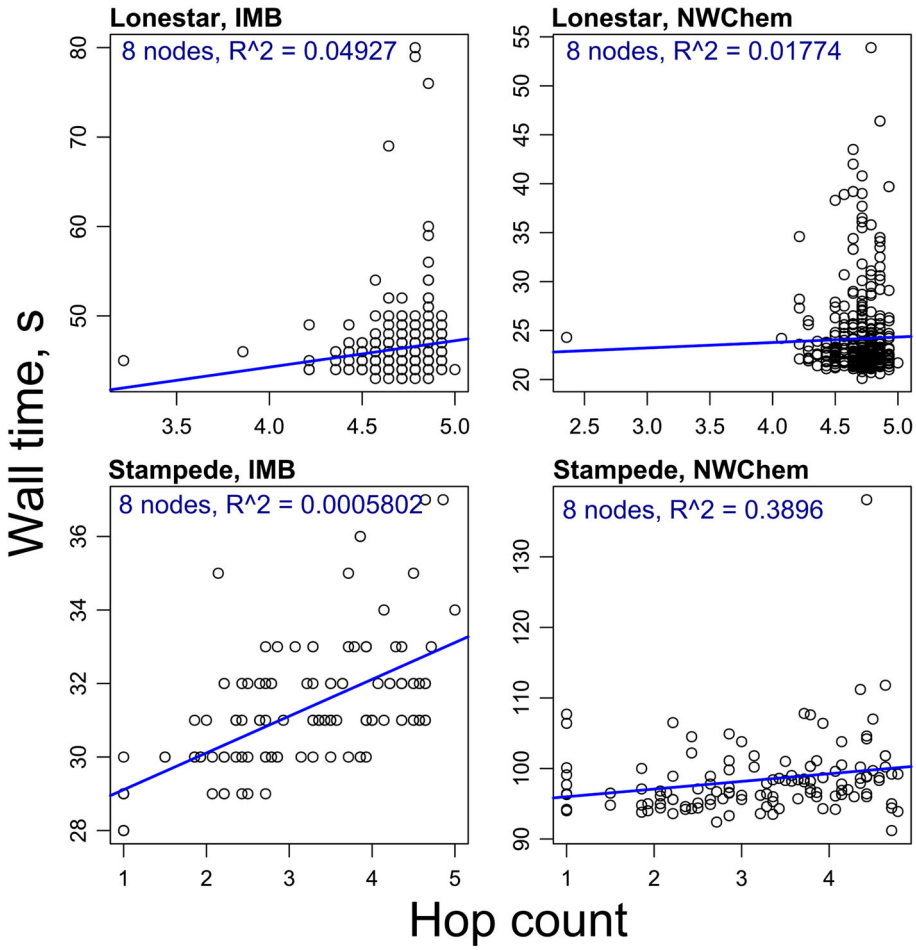


Figure 12. Dependency of execution wall time on average hop count on eight nodes for Intel message-passing interface (MPI) benchmarks (IMB) and northwest computational chemistry program (NWChem) application kernels.

are more equally distributed between lowest and highest hop counts. Second, the distributions of job wall times on Lonestar4 are wider towards high hop count, whereas on Stampede, it stays about the same. Although using different schedulers, both resources do not utilize topology aware scheduling and have similar network topology. The greater population of low hops count on Stampede can be traced to the higher number of nodes on endpoint switches (20 vs. 12), to different schedulers (simple Linux utility for resource management (SLURM) versus Portable Batch System (PBS)) and to differences in load patterns (e.g., the larger number of nodes may result in a higher probability of the job scheduler finding a block of space for the job to run on neighboring nodes). The wider distribution towards high hop count on Lonestar4 can be explained by worse distribution of the jobs' nodes across the system (thus, more traffic through core switches) and a slower fabric (QDR versus Stampede's FDR). Although Stampede has five-fourth oversubscription of the core switches, both systems use four links between leaf and core switches. Assuming that other user's jobs exhibit a similar distribution of hop count, the higher hop counts would lead to a higher probability of interference from the traffic on other nodes and thus widen the wall time distribution as the hop count grows. Therefore, the increased number of ports and improved bandwidth and latencies significantly reduces the influence from other jobs even without utilization of a topology-aware scheduler.

4.5. Influence of overall Lustre file system load

All scientific applications have some interaction with the file system because they have a need to read the input parameters and write the calculation's results. Some applications have only minimal file

DOI: 10.1002/cpe

system I/O utilization (for example, sparse trajectory recording for a molecular dynamics simulation), while others cause a significant stress on the file system (for example, large scale quantum chemistry applications utilize the file system to store intermediate results and extend available memory). Parallel file systems like Lustre are a shared resource, and thus simultaneous usage by multiple users can decrease their performance. In many cases, this can be overcome by implementing asynchronous I/O. In this section, the dependence of application performance on the overall Lustre file system load is discussed.

The overall Lustre file system load is calculated as the total number of bytes read and written from all computational nodes to all Lustre nodes. Because the calculated overall Lustre file system load metric does not distinguish the traffic to the individual Lustre nodes, this metric is not very sensitive and is prone to a high noise level. The overall Lustre file system load was only calculated for Stampede.

Figure 13 shows the wall time dependency on the overall Lustre file system load for the IOR file system benchmark. As expected, this file system application kernel exhibits the strongest correlation with the total file system load. Most of the other application kernels have very minor file system utilization by design as they read fairly small input files and produce small output files. Besides IOR, only NWChem produces sufficient file system traffic, due to its need for temporary storage, and exhibits a weak dependency on the overall file system load. The coefficient of variation reaches nearly 30% for IOR. Therefore, data driven but computationally lightweight applications can suffer severely when the shared file system is heavily used.

4.6. Regression and correlation analysis

A full correlation matrix was computed for the eight application kernels running on Stampede having 13 metrics. We tracked:

1. wall time (seconds),
2. FLOPS – floating point operations per second (average per core, Giga FLOPS),

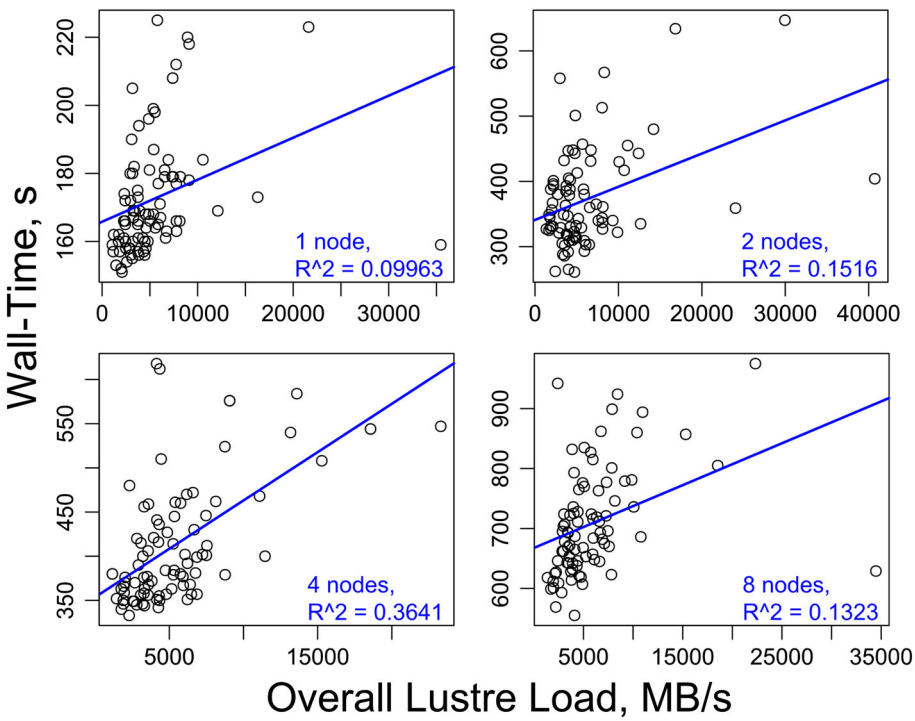


Figure 13. Dependency of interleave or random (IOR) execution wall time on overall file system load for Texas Advanced Computing Center (TACC) Stampede on one, two, four, and eight nodes.

DOI: 10.1002/cpe

3. mem_trans main-memory transfer rate (average per node, GB/s),
4. cpu_idle – CPU idle fraction (average per core),
5. cpu_system – CPU system fraction (average per core),
6. cpu_user – CPU user fraction (average per core),
7. cpi – clock ticks per instruction (average per core),
8. cpld – clock ticks per L1D load (average per core),
9. cpu_imbal – CPU imbalance (measured as (max–min)/max of the cpu_idle of the processors used on the job, average per node, %),
10. lustre_tx – average transmit bandwidth by Lustre network drive (average per node, MB/s),
11. lustre_rx – average receive bandwidth by Lustre network drive (average per node, MB/s),
12. ib_tx – average transmit bandwidth by the InfiniBand interface (average per node, MB/s), and
13. ib_rx – average receive bandwidth by InfiniBand interface (average per node, MB/s).

Corregrams for NAMD and NWChem for four nodes on Stampede are shown in Figure 14. The NWChem wall time anti-correlates with average FLOPS, file system, and InfiniBand bandwidth (i.e., higher bandwidth leads to shorter runs). As opposed to NWChem, the NAMD wall time does not correlate with most of the metrics besides average CPU imbalance. NAMD dynamically balances the load between MPI proccesses, and in our short run, the balancing stage is dominant. Therefore, it is not surprising that shorter imbalanced periods result in faster runs. The correlations between different metrics for NAMD and NWChem have many similarities. For example, average FLOPS correlate with memory transfer rate and average network and file-system bandwidth, because the data need to be delivered to the CPU arithmetic units. Overall, most of the metrics tended to be positively correlated or anti-correlated with wall time as expected. There are groups of metrics that tend to correlate with each other suggesting that a PCA might be useful.

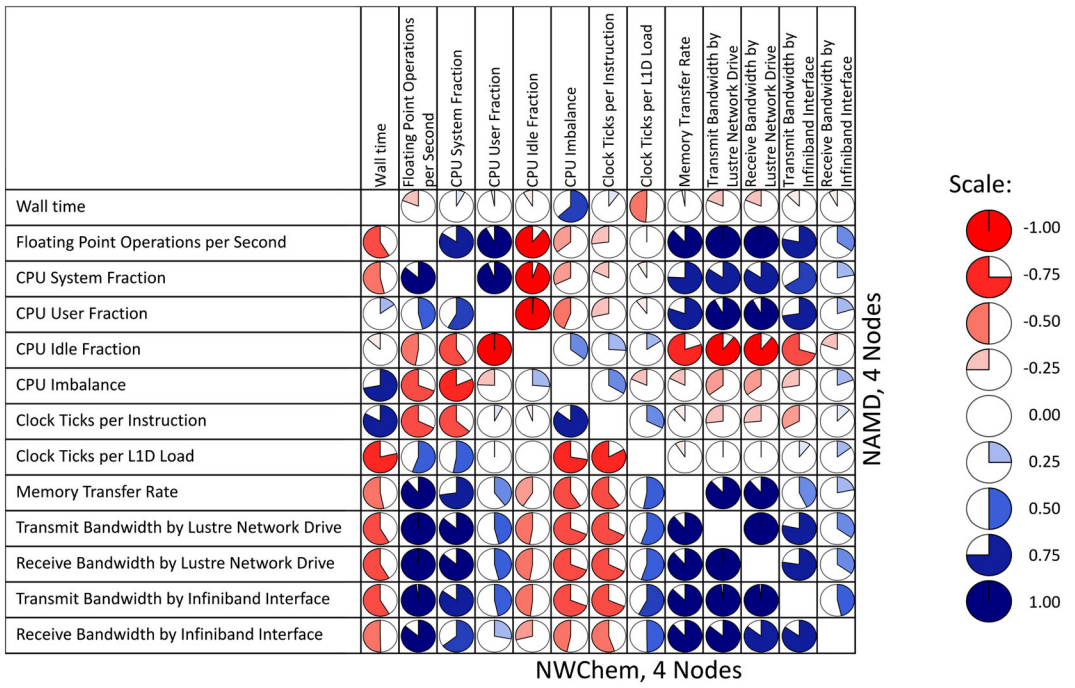


Figure 14. Correlation matrix between 13 performance metrics for northwest computational chemistry program (NWChem) (lower triangle) and nanoscale molecular dynamics program (NAMD) (upper triangle) executed on Stampede. Pearson correlation coefficients are shown as a pie diagram; the scale is shown on the right. For example, a full blue pie chart corresponds to total positive correlation, an empty pie chart corresponds to no correlation, and full red pie chart corresponds to total negative correlation. Because the correlation coefficients matrix is symmetric, the matrixes for NWChem and NAMD are shown on the same plot for the sake of saving space and not for cross comparison.

DOI: 10.1002/cpe

Although corregrams for different applications have many similarities, they are quantitatively different. Therefore, in addition to the metrics themselves, their correlation can be used for users' application classification. This can be useful for correlating users application performance to the performance of application kernels.

Regressions to fit the wall time of the application kernels based on the other metrics were not particularly successful. For example, NAMD running on four nodes of Stampede was fit by cpu_imbal with a p value < 0.0001 and $R^2 = 0.41$. Adding cpu_idle and block_write improved the fit marginally. For NWChem running on four nodes of Stampede, the best fit was cpi + cpld with $R^2 = 0.24$ and both parameters with $p < 0.0001$. Adding cpu_imbal marginally improved R^2, but the p value was >0.05.

5. PRINCIPAL COMPONENT ANALYSIS (PCA)

The blocks of correlated metrics in Figure 14 suggest that a PCA might provide some insight to understanding the relationship between these job metrics and aid in understanding the variation in the way such repeated jobs run on the two HPC clusters. Therefore, a PCA was performed for the application kernels running on Stampede and Lonestar4. We will present the results of the PCA of NWChem on Stampede as an example.

For the Stampede PCA, we included the following metrics: wall_time (seconds), ib0_tx (InfiniBand bytes transmitted), cpu_idle (fraction of time that the CPU spends in idle mode), cpi (cycles of the reference clock divided by the number of instructions), cpld (cycles of the reference clock divided by the number of L1 cache loads), scratch_write (bytes), memory (memory used per core in megabyte), lustre_tx (bytes transmitted on Lustre file system), eth_tx (bytes transmitted on Ethernet), block_write (bytes), flops, and cpu_imbal (standard deviation/mean of the cpu_idle of the processors used on the job). Table III gives the loadings for the first four components of the PCA of NWChem running on four nodes of Stampede. Figure 15 shows the scree plot, and Figure 16 plots the scores for the first three components of this PCA.

PCA is generally used to reduce the number of variables in highly dimensional data. As such, it can often simplify the data interpretation by reducing the number of variables, although the interpretation of the new variables can be more complex. The results of the present PCA are rather interesting. Note from Table III that the metrics split neatly into two sets. Component 1 has ib0_tx, cpu_idle, scratch_write, lustre_tx, eth_tx, block_write, and FLOPS. Component 2 has wall_time, cpi, cpld, memory, and cpu_imbal. Only the small coefficient of cpu_imbal in component 1 breaks a perfect partition. Together, as seen in Figure 15, these two components account for 0.76 fraction of the variance. This represents progress because now the original 12 variables can, to a certain extent, be represented by the two main components 1 and 2. Furthermore, the metrics present in components 1

Table III. PCA of northwest computational chemistry program (NWChem) running on four nodes on Stampede.

Loadings	Comp_1	Comp_2	Comp_3	Comp_4
wall_time	–	+0.524	–	–
ib0_tx	+0.370	–	–	+0.169
cpu_idle	−0.353	–	+0.426	–
cpi	–	+0.515	−0.158	+0.264
cpld	–	−0.500	+0.207	−0.239
scratch_write	+0.399	–	–	–
memory	–	0.318	−0.151	−0.903
lustre_tx	+0.391	–	–	–
eth_tx	+0.350	–	+0.165	–
block_write	−0.384	–	–	–
Floating point operations per second (FLOPS)	+0.353	–	+0.193	−0.133
cpu_imbal	−0.118	+0.299	+0.800	–
cum_variance	0.507	0.761	0.842	0.906

DOI: 10.1002/cpe

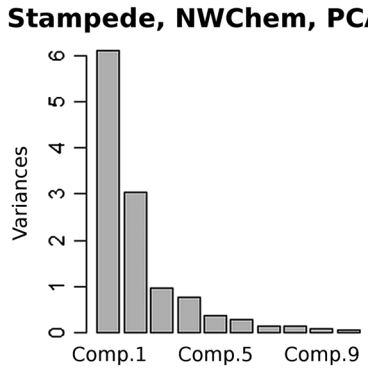


Figure 15. Scree plot for the PCA of data of northwest computational chemistry program (NWChem) four nodes running on Stampede.

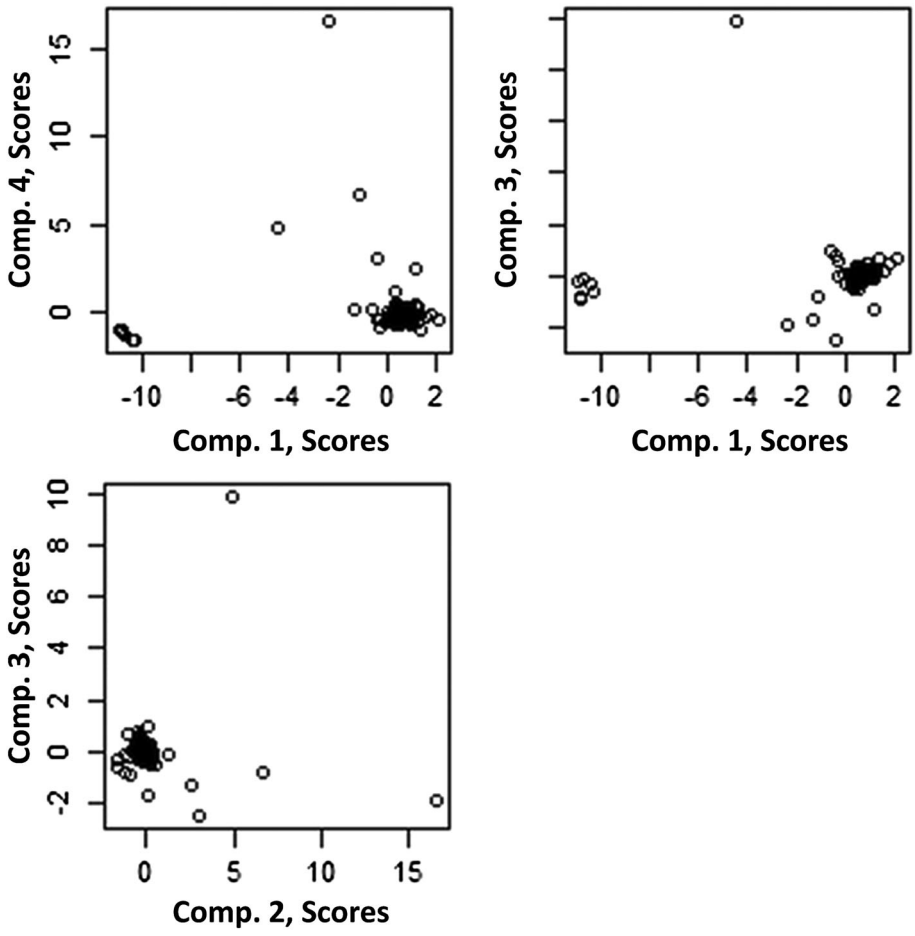


Figure 16. Plot of scores for the first three components of the PCA of data of northwest computational chemistry program (NWChem) four nodes running on Stampede.

and 2 are related and can be characterized. Component 1 mainly relates to external data transfer, and component 2 carries the remainder of the principally internal operations. Components 3 and 4 have a much smaller effect and tend to mix the two sets of metrics. We can see that the best way to view the collection of metrics is that they are divided into internal operations that are conducted on machine hardware that is dedicated to the job (such as CPU cores and the main memory bus) and external operations that are conducted on shared resources (such as the parallel file system and the

DOI: 10.1002/cpe

InfiniBand interconnect). The results of other PCA computations on other Stampede applications and on Lonestar4 applications are qualitatively similar to the example case presented.

6. CONCLUSIONS

As demonstrated in Section 3.3 and the references cited therein, running the application kernels continuously on the XSEDE (or other HPC) resources provides a powerful tool to help assure QoS on HPC systems. However, these application kernels now generate a tremendous amount of performance data that makes manual oversight of the data to identify underperforming hardware or software impractical. Accordingly, as demonstrated in Section 3.1, the development of automated process control systems and automated notification systems to monitor application kernel performance will provide the results to resource managers in a simple, timely fashion. These make a strong addition to the XDMoD framework application kernel system.

The variance analysis presented in the present work suggests a number of conclusions about the operation of the subject HPC resources and likely HPC resources in general. Generally, the application kernels ran very well with few outliers. Metrics such as the wall time exhibited a skewed distribution characterized by an ideal value and a long tail to poorer performance. Because IOR measures I/O performance, its wall time exhibited the largest variance and showed a definite dependence on the volume of concurrent file system usage. The hop count analysis shows a surprisingly small dependence on the network topology for all kernels based on real world applications. The PCA of the metrics shows that the first two components split neatly into two sets, one generally related to external data transfer and the other carrying the remainder of the metrics. This detailed analysis will aid in using application kernels as a mechanism to help to maintain QoS.

APPENDIX A: EXECUTION PARAMETERS OF APPLICATION KERNELS

Northwest computational chemistry program (NWChem) [25] application kernel performs semi-direct MP2, fully direct MP2, and CCSD(T) energy calculation of gold ion (Au^+) with explicitly defined basis set (using Gaussian basis function). Calculations are performed using one, two, four and eight nodes.

General atomic and molecular electronic structure system (GAMESS-US) [26] application kernel performs MP2 energy calculation of 1,3,5,7-octatetraene (C_8H_{10}) with cc-pVTZ basis set. Calculations are performed using one, two, four, and eight nodes.

Nanoscale molecular dynamics program (NAMD) [27] application kernel performs short (2.4 ps duration) NVE molecular dynamics simulation of apolipoprotein A1 dimer in water solution. This simulation is based on ApoA1 NAMD benchmark (http://www.ks.uiuc.edu/Research/namd/performance.html). System consists of 92,224 atoms. The 12 Å cut-off is used for short-range interactions, and PME is used for long-range interaction. Calculations are performed using one, two, four, and eight nodes.

Graph500 [29] application kernel performs breadth-first search (BFS) on a graph. The graph consists of 10^{23} vertices. During each run, BFS searches for 64 vertices on the graph. The benchmark is executed on one, two, four, and eight nodes.

High performance computing challenge (HPCC) [16] application kernel performs operation on matrixes size of $(20000*N^{1/2}) \times (20000*N^{1/2})$, where N is number of nodes. The benchmark is executed on one, two, four, eight, and 16 nodes.

Intel message-passing interface (MPI) benchmarks (IMB) [17] application kernel measures MPI node-to-node and collective communication latencies and bandwidth on a large number of MPI operations (PingPong, PingPing, Sendrecv, Exchange, Allreduce, Reduce, Reduce_scatter, Allgatherv, Gather, Gatherv, Scatter, Scatterv, Alltoall, Alltoallv, Bcast, Barrier, Window, Unidir_Get, Unidir_Put, Bidir_Get,Bidir_Put, and Accumulate) on messages with size up to 4 MiB. The benchmark is executed on two, four, eight, and 16 nodes.

Interleave or random (IOR) [18] application kernel performs write/read operations on file system using four APIs: portable operating system interface (POSIX), MPI-IO, hierarchical data format 5, and NetCDF. For each API, three file access modes are tested: (1) each MPI process writes to and reads

DOI: 10.1002/cpe

from a separate file; (2) all MPI process writes to and reads from a single shared file; and (3) similar to previous but collective I/O calls are used where available (all except POSIX). Each MPI process writes/reads a portion of the file equal to 200 MiB. For example, in the case of a single shared file mode executed on eight nodes of 12 cores per node machine, the size of the file is 19,200 MiB (18.75 GiB). The benchmark is executed on one, two, four, and eight nodes.

Enzo [28] application kernel is based on one of the Enzo tests. It performed a reionization simulation using radiation hydrodynamics and radiating star particles. The top level grid is of size 128^3 with up to two mesh refinement levels with a refinement factor of 2. Calculations are performed using one, two, four, and eight nodes.

ACKNOWLEDGEMENTS

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

REFERENCES

1. White JP, Barth WL, Hammond J, *et al*. An analysis of node sharing on HPC clusters using XDMoD/TACC_stats. Proc. 2014 annu. Conf. Extrem. Sci. Eng. Discov. Environ. – XSEDE'14. ACM Press, New York, New York, USA, 2014; pp. 1–8
2. Browne JC, Furlani TR, McLay RT, *et al*. Enabling comprehensive data-driven system management for large computational facilities. Proc. Int. Conf. High Perform. Comput. Networking, Storage Anal. - SC'13. ACM Press, New York, New York, USA, 2013; pp. 1–11.
3. Furlani TR, Jones MD, Gallo SM, *et al.* Performance metrics and auditing framework using application kernels for high-performance computer systems. *Concurrency and Computation Practice and Experience* 2013; **25**:918–931. doi:10.1002/cpe.2871.
4. Lu C-D, Browne J, DeLeon RL, *et al*. Comprehensive job level resource usage measurement and analysis for XSEDE HPC systems. Proc. Conf. Extrem. Sci. Eng. Discov. Environ. Gatew. to Discov. - XSEDE'13. ACM Press, New York, New York, USA, 2013; p 1.
5. Furlani TR, Gentner RJ, Patra AK, *et al*. Using XDMoD to facilitate XSEDE operations, planning and analysis. Proc. Conf. Extrem. Sci. Eng. Discov. Environ. Gatew. to Discov. - XSEDE'13. ACM Press, New York, New York, USA, 2013; p. 1
6. Browne JC, DeLeon RL, Patra AK, *et al.* Comprehensive, open-source resource usage measurement and analysis for HPC systems. *Concurrency and Computation Practice and Experience* 2014; **26**:2191–2209. doi:10.1002/cpe.3245.
7. Towns J, Cockerill T, Dahan M, *et al.* XSEDE: accelerating scientific discovery. *Computing in Science & Engineering* 2014; **16**:62–74. doi:10.1109/MCSE.2014.80.
8. Hammond J. TACC_stats: I/O performance monitoring for the intransigent. 2011 Work. Interfaces Archit. Sci. Data Storage (IASDS 2011), 2011.
9. TACC_stats. http://github.com/TACCProjects/tacc_stats [Accessed 11 Aug 2014].
10. Performance Co-Pilot. http://www.performancecopilot.org/index.html [Accessed 11 Aug 2014].
11. Ganglia Monitoring System. http://ganglia.info/ [Accessed 11 Aug 2014].
12. Massie ML, Chun BN, Culler DE. The Ganglia distributed monitoring system: design, implementation, and experience. *Parallel Computing* 2004; **30**:817–840. doi:10.1016/j.parco.2004.04.001.
13. Lightweight Distributed Metric Service (LDMS). https://ovis.ca.sandia.gov/mediawiki/index.php/CRAY-LDMS-EXTRA [Accessed 13 Aug 2014].
14. Zabbix. http://www.zabbix.com/ [Accessed 11 Aug 2014].
15. Dongarra JJ, Luszczek P, Petitet A. The Linpack benchmark: past, present and future. *Concurrency and Computation Practice and Experience* 2003; **15**:803–820. doi:10.1002/cpe.728.
16. Luszczek PR, Bailey DH, Dongarra JJ, *et al*. The HPC challenge (HPCC) benchmark suite. Proc. 2006 ACM/IEEE Conf. Supercomput. 2006; p. 213
17. Intel MPI Benchmarks. https://software.intel.com/en-us/articles/intel-mpi-benchmarks [Accessed 11 Aug 2014].
18. IOR HPC benchmark | free system administration software downloads at SourceForge.net. http://sourceforge.net/projects/ior-sio/ [Accessed 11 Aug 2014].
19. MPI-Tile-IO: Parallel I/O Benchmarking Consortium. http://www.mcs.anl.gov/research/projects/pio-benchmark/ [Accessed 11 Aug 2014].
20. Gray A, Bethune I, Kenway R, *et al.* Mapping application performance to HPC architecture. *Computer Physics Communications* 2012; **183**:520–529. doi:10.1016/j.cpc.2011.11.013.
21. SPEC MPI2007. http://www.spec.org/mpi2007/ [Accessed 11 Aug 2014].
22. Lu C-D. Automatically mining program build information via signature matching. Proc. 11th ACM SIGPLAN-SIGSOFT Work. Progr. Anal. Softw. Tools Eng. - PASTE'13. ACM Press, New York, New York, USA, 2013; pp. 25–32
23. Dwarf Mine. 2015. http://view.eecs.berkeley.edu/wiki/Dwarf_Mine

DOI: 10.1002/cpe

24. Ahern S, Alam S, Fahey M, *et al*. Scientific application requirements for leadership computing at the exascale. Proc. 2008 Cray User Gr. Meet. Helsinki, Finland, May 5–8, 2008. Also available as ORNL/TM-2007/238
25. Valiev M, Bylaska EJ, Govind N, *et al.* NWChem: a comprehensive and scalable open-source solution for large scale molecular simulations. *Computer Physics Communications* 2010; **181**:1477–1489. doi:10.1016/j.cpc.2010.04.018.
26. Schmidt MW, Baldridge KK, Boatz JA, *et al.* General atomic and molecular electronic structure system. *Journal of Computational Chemistry* 1993; **14**:1347–1363. doi:10.1002/jcc.540141112.
27. Phillips JC, Braun R, Wang W, *et al.* Scalable molecular dynamics with NAMD. *Journal of Computational Chemistry* 2005; **26**:1781–802. doi:10.1002/jcc.20289.
28. Norman ML, Bryan GL, Harkness R, *et al*. Simulating cosmological evolution with Enzo. eprint arXiv:0705.1556, 2007.
29. Graph 500. http://www.graph500.org/ [Accessed 11 Aug 2014].
30. Chuah E, Jhumka A, Narasimhamurthy S, *et al*. Linking resource usage anomalies with system failures from cluster log data. 2013 IEEE 32nd Int. Symp. Reliab. Distrib. Syst. IEEE, 2013; pp. 111–120.
31. Hammond JL, Minyard T, Browne J. End-to-end framework for fault management for open source clusters. Proc. 2010 TeraGrid Conf. - TG'10. ACM Press, New York, New York, USA, 2010; pp. 1–6.
32. Martin R, Vahdat A, Culler D, Anderson T. Effect of communication latency, overhead, and bandwidth on a cluster, 1998.
33. Liu W, Lo V, Windisch K, Nitzberg B. Non-contiguous processor allocation algorithms for distributed memory multicomputers. 1994; 227–236.

DOI: 10.1002/cpe