Parallel Computing 58 (2016) 90–106

Contents lists available at ScienceDirect

Parallel Computing

journal homepage: www.elsevier.com/locate/parco

Continuous whole-system monitoring toward rapid understanding of production HPC applications and systems

Anthony Agelastos, Benjamin Allan, Jim Brandt, Ann Gentile*, Sophia Lefantzi, Steve Monk, Jeff Ogden, Mahesh Rajan, Joel Stevenson

Sandia National Laboratories, Albuquerque, NM, USA

ARTICLE INFO

Article history:
Received 29 November 2015
Revised 10 April 2016
Accepted 15 May 2016
Available online 18 May 2016

Keywords:
HPC monitoring
System profiling
Application profiling
Resource utilization scoring

ABSTRACT

A detailed understanding of HPC applications' resource needs and their complex interactions with each other and HPC platform resources are critical to achieving scalability and performance. Such understanding has been difficult to achieve because typical application profiling tools do not capture the behaviors of codes under the potentially wide spectrum of actual production conditions and because typical monitoring tools do not capture system resource usage information with high enough fidelity to gain sufficient insight into application performance and demands.

In this paper we present both system and application profiling results based on data obtained through synchronized system wide monitoring on a production HPC cluster at Sandia National Laboratories (SNL). We demonstrate analytic and visualization techniques that we are using to characterize application and system resource usage under production conditions for better understanding of application resource needs. Our goals are to improve application performance (through understanding application-to-resource mapping and system throughput) and to ensure that future system capabilities match their intended workloads.

1. Introduction

Workload balance across all processes of an application is key to achieving both scalability and performance on today's large scale High Performance Computing (HPC) platforms. Intelligent scheduling and resource-to-application mapping is a crucial component to achieving workload balance in the face of multiple applications competing for shared and possibly oversubscribed resources (e.g., network, filesystems). In recent years, there has been increased emphasis on concurrently considering both applications and their target platforms design and development, referred to as *co-design*. In practice, however, application developers and users are generally at the mercy of the platforms that are available to them and system administrators and buyers have limited insight into the actual resource requirements of the supported applications. While application developers may use performance analysis tools to better tune application performance, such tools are not generally used under typical production conditions with respect to scale and contention with concurrently running applications.

* Corresponding author.
E-mail addresses: amagela@sandia.gov (A. Agelastos), baallan@sandia.gov (B. Allan), brandt@sandia.gov (J. Brandt), gentile@sandia.gov (A. Gentile), slefant@sandia.gov (S. Lefantzi), smonk@sandia.gov (S. Monk), jbogden@sandia.gov (J. Ogden), mrajan@sandia.gov (M. Rajan), josteve@sandia.gov (J. Stevenson).

http://dx.doi.org/10.1016/j.parco.2016.05.009
0167-8191/

Additionally, the types of analyses performed by the application developer may not be those of most benefit to the system administrator.

In this work we demonstrate that continuous, synchronous, high-fidelity, whole-system monitoring can provide meaningful profiling of system and application resource utilization under production conditions. Further we discuss how use of this information can drive more efficient troubleshooting, platform utilization, and procurement decisions. In this paper we present results from our ASC capacity 1232 compute node cluster, Chama, using our lightweight HPC monitoring tools. To the best of our knowledge, we provide the only production HPC environment to continuously monitor at rates suitable for application profiling analysis and to share this level of profiling results with general users.

The outline of this paper is as follows. In Section 2 we present our motivation for whole-system monitoring toward application and system resource utilization understanding in SNL's HPC environment. In Section 3 we present details of the monitoring and characterization approaches we are taking. Next in Sections 4 and 5 we present system (focused on conditions of interest) and application resource utilization profiling data gathered on Chama. We show how the profiling data can be used to: better understand applications and their resource utilization, both alone and in conjunction with other concurrently running applications; guide users in efficiently matching their applications to available resources; provide administrators insight into the system's usage; and provide buyers information on target application resource demands. In Section 6 we discuss data handling and processing considerations. In Section 7 we present a few examples from other systems, which also illustrate the general utility and scalability of our approach. We discuss related work in Section 8 and summarize and address future work in Section 9.

2. Motivation

Performance of applications, as measured by wall time to completion, can have wide variation [1] on a particular HPC platform and across different HPC platforms. Often the causes go undiagnosed or unexplained due to insufficient availability of information. Though the causes can be diverse the end result is longer time to solution for applications, more work for users and administrators due to debugging efforts, and ultimately less science output per invested dollar due to lower system efficiency and throughput.

While application profiling tools such as Open|SpeedShop [2], TAU [3], HPCToolkit [4], Vampir [5], and CrayPat [6] can be utilized to understand relative performance parameters of a particular application, their overheads are high enough that they are only run when trying to tune an application. Conversely typical monitoring tools such as Ganglia [7] and Nagios [8] collect data on such long time scales that using them for diagnosing all but the most glaring problems (e.g. node, network, file system outages) is typically unproductive.

The rest of this article describes, and provides use case examples on, how we utilize synchronized whole system monitoring to enable people in the following HPC centric roles to gain rapid insight into application/system interactions. In the remainder of this section, along with each role, we provide a short description of some of the problems and how information from this type of monitoring can provide benefit.

System administrators. By having access to synchronously collected, system-wide data, about all metrics of interest, system administrators can rapidly debug job slowness due to over-subscription of resources (e.g., file system bandwidth) for a particular user or over a whole cluster. For example, traditional monitoring systems and user reports can show when a site-shared parallel file system has been heavily loaded and caused I/O to take much longer than usual. With a robust data collection tool that exposes file I/O statistics on a per node and job basis, system administrators can quickly detect if it is a particular user, code, or an aggregate over several jobs with high-storage bandwidth needs that is causing file system slowness.

Code users. In order to provide users with a better understanding of their application's behavioral characteristics, and possibly assist in their optimization and debugging where needed, for each user job we provide time history plots, such as those presented in this work, and text usage summary statistics (e.g., min, max, active memory by node and job) in a canonical location with appropriate user file permissions. Users can reference the appropriate files by job ID for any of their jobs.

Application developers. Developers are increasingly concerned with the performance characteristics of their optimized applications preferably divorced from platform interference (e.g., OS noise, resource contention). Moreover, they would like these performance characteristics to be immediately available and part of their periodic performance testing so they can instantly correlate algorithmic changes with performance impact. Resource utilization scoring can provide meaningful, rapid understanding of both application and corresponding system and can be easily integrated into existing test harnesses. Time history plots and system maps of metrics can then be used for deep dives when appropriate.

HPC hardware architects and procurement planners. In order to meet the needs of their user communities, those planning future procurements need to be able to size the breadth (number of nodes), depth (number of compute cores, memory, etc.), and support systems (storage, network, burst buffers, etc.) of their next platforms. Having statistics over time (e.g., memory per core usage, I/O router bandwidth, interconnect bandwidth, latency) gives system architects data to use as requirements in the design and procurement of future HPC systems.

3. Approaches

In this section we describe our approaches to obtaining and analyzing data to provide system and application resource utilization understanding.

3.1. Data collection

Foundational to understanding resource utilization is data collection. We utilize our low overhead, low noise Lightweight Distributed Metric Service (LDMS) [9] at SNL to continuously monitor HPC resources with respect to a large number of performance metrics. LDMS is particularly suitable for this task because it does not significantly skew the resource utilization measurements and because data collection is synchronized in time across all nodes of a system. Details on the assessment of impact of LDMS data collection can be found in [9]; no statistically significant impact was seen. Synchronization has the effect of producing a "snapshot" across the whole system of the LDMS metrics that are being collected. Data collected in this fashion is hereafter referred to as "LDMS data" or "metric data".

Currently LDMS data includes counters and error information for the following:

- Memory from `/proc/meminfo`
- CPU from `/proc/stat`
- Lustre file system from `/proc/fs/lustre/llite/⟨mount point⟩/stats`
- NFS from `/proc/net/rpc/nfs`
- Processes, memory, paging, block IO, traps from `/proc/vmstat`
- Ethernet from `/proc/net/dev`
- Infiniband from `/sys/class/infiniband/.../counters`
- Cray specific Gemini and Aries networks from `gpcd counters`

Data is collected on compute and service nodes and pulled to off-system aggregation points. The number of such aggregation points depends on size of the system (LDMS supports fan-ins of >10,000-to-1), size of metric data sets being collected, and the degree of redundancy desired.

In the work presented here we primarily reference information from a 1232 node Linux cluster "Chama" that is sited at Sandia National Laboratories in Albuquerque NM. Each Chama node is comprised of two 8-core Intel Sandy Bridge processors with 64 GB of shared memory. The Chama nodes are connected via QDR Infiniband in a Fat Tree network topology.

3.2. Data sampling fidelity

In order to provide meaningful information on resource utilization, data must be sampled at fidelities commensurate with the features it is necessary to resolve since one cannot know a priori where and when regions and issues of interest may arise. Typical system monitoring tools are deployed with sampling intervals of minutes. Unsynchronized data at such low fidelity are unsuitable for the type of understanding we desire. This is illustrated in Fig. 1, which shows a CPU utilization profile of a 64-node Sierra/SolidMechanics (Adagio) job; this run will be examined in more detail in Section 5.4. Here the time spent by the CPU in IO wait over sampling intervals at 1 s (top) and 60 s (bottom) are presented for the same job. Values during the job execution are shown on the white background where each node in the job is an individual line. Job Id and time range are included at the top of the figure. This format is used for the figures throughout the rest of the paper. Perceived behavior of the application can be significantly different for the two sampling intervals. At higher fidelity, it is clear that the nodes spend time in I/O wait over a significant portion of the application run time. For Adagio because of the short lived CPU user/IO changes, feature details are lost when moving to larger sampling (lower fidelity) intervals.

Another tradeoff, with respect to fidelity, that we contend with is data volume with respect to our storage capacity. As an example, on NCSA's Blue Waters platform (27,648 nodes), the 200 LDMS metrics per node being collected at 60 s intervals translates into roughly 55 GB data stored per day. Increasing the sampling interval, for the same metric data, to 1 s would result in a 60 fold increase or 3.3 TB per day. On our production systems we typically target 1–20 s sample intervals.

3.3. Resource utilization scoring

In order to help system administrators, analysts, and users quickly identify resource utilization behaviors that *may* imply room for substantial performance improvement, deviate substantially from what is *typical*, or have large run-to-run variation we have begun working on a job-based resource utilization *scoring* methodology. The scores distill detailed utilization information into numerical representation(s) that can be easily parsed for gaining insight and providing automated guidance to resource managers.

Our currently defined scores are job based and focus on: 1) determining the fractional utilization of target resources relative to some upper value based on hardware or configuration limitations and 2) determining how well-balanced the utilization of the target resources is across all nodes of a job. Application-independent platform-wide scores can be calculated in a similar fashion.

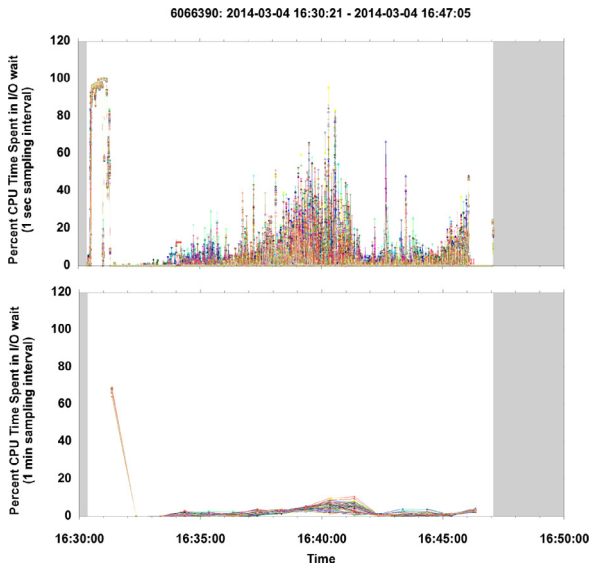

Fig. 1. IO wait profiles for a 64-node job: 1 s interval (top), 60 s interval (bottom). Features are lost at longer sampling intervals. Each line is a single node's data. The legend of lines colored by nodes have been suppressed. The gray background shows times pre- and post-job. These job presentation format details are used in subsequent figures as well.

Table 1
Bins converting μ and *CV%* to scores.

Category	Score	1	2	3	4	5	6	7	8	9	10
Usage	$\mu <$	10	20	30	40	50	60	70	80	90	100
Balance	*CV%* $<$	∞	35	30	25	20	15	10	7	4	1

We use deciles to score usage, peak usage, or activity and the skewed scale in Table 1 to convert standard deviation into a *balance* score Eqs. 1–(4). On both scales, 1 is the "poor" end of the scale. The skewed balance scale conversion is chosen empirically to emphasize poor performers since *CV%* is unbounded.

While the methodologies presented in this paper are applied to single metric examples (e.g., memory utilization and Lustre I/O) it is important to recognize that focusing on single metric scores may be misleading when taken out of the context of the whole. For example, a low memory utilization score taken without the context of memory bandwidth might look glaringly bad whereas coupled with a high memory bandwidth utilization score it may be quite fine. The characteristic *sweet spot* for a given metric on a given application will rarely be at the extreme ends of the scale.

Our scoring formulae in detail are:

$$Activity\ \ score = decile\_bin\_lookup(Activity\%) \tag{1}$$

$$Peak\ \ score = decile\_bin\_lookup(Peak\%) \tag{2}$$

$$Usage\ \ score = decile\_bin\_lookup(\mu\%) \tag{3}$$

$$Balance\ score = deviation\_bin\_lookup(CV\%) \tag{4}$$

$$Peak\% = 100 * max(M_{i,k})/LIMIT_m \tag{5}$$

$$\mu\% = 100 * \mu = 100 * \left(\sum_{i=1}^{F} \sum_{k=1}^{N_p} (M_{i,k} * \Delta t_{i,k}) \right) \Big/ (LIMIT * (t_{i=F} - t_{i=1}) * N_p) \tag{6}$$

$$CV\% = 100 * (\sigma_{nz}/\mu_{nz}) \tag{7}$$

$$Activity\% = count(M_{i,k} \neq 0)/count(M_{i,k}) * 100 \tag{8}$$

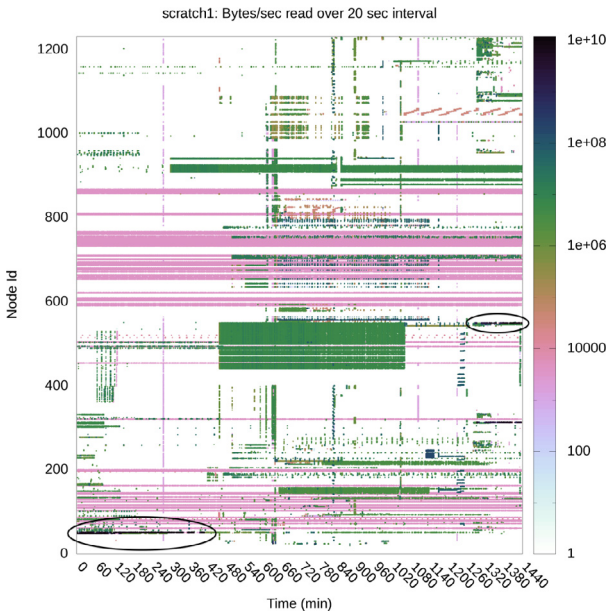

Fig. 2. System map of bytes read/s over the 1232-node cluster over a 24 h. period. Hot spots and long-lived features are identifiable.

- N_p is the number of nodes running the job.
- F is the number of measurements taken on a single node.
- $M_{i,\,k}$ is the measured value of the ith sample on the kth node.
- $\Delta t_{i,\,k}$ is the time between samples $i-1$ and i on node k.
- σ_{nz} is the standard deviation over nodes of A_k, the per-node nonzero time-weighted average measurement: $A_k = \text{average}((M_i * \Delta t_i) \forall M_i \neq 0)_k$.
- *CV%* is 100 $*$ the coefficient of variance of nonzero measurements.
- $LIMIT_m$ is the chosen limit for resource m. For RAM it is physical: 64 GB, and for Lustre bandwidth it is the fair share: $80(\text{GB/s})/\text{N}_\text{p}$.

4. System level profiling

Collecting whole-system profiling data enables system-wide insight as well as insight into an application's performance and system resource utilization characteristics. Additionally, it can shed light on system hot spots due to competition for shared resources both within and between applications. In system profiling, we rely on the synchronous feature of the collection.

4.1. System I/O profiling

Demands on the file system and file system performance are of particular interest on our HPC systems. This is because for many applications poor file I/O translates directly into poor application performance. Thus applications that cause severe congestion in the parallel file system, either on their own or due to aggregate load, can severely impact all I/O sensitive jobs on the system.

System-wide visualizations of Lustre data can give insight into which clients are utilizing significant file system bandwidth. Additionally, time and node information can be associated with scheduler/resource manager (RM) information in order to give administrators insight into resource demands associated with particular jobs which in turn maps into the associated applications and their run-time parameters.

Lustre reads and writes over a 24-h period are shown in Figs. 2 and 3 (top), respectively. Both are plotted on the same scale, however, over the time shown, the maximum number of bytes read in a 20 s interval is over an order of magnitude greater than the maximum number of bytes written (example near-maximum magnitude read occurrences are marked in Fig. 2). Nodes and times with particularly high demand are easily identified. In addition, by summing the values over all nodes for a time window, it is readily apparent if/when aggregate resource limits are being hit. In Fig. 3, this sum is shown as time-history plot (a) beneath the associated heat map. Additionally, the values of individual jobs (c,d,e) can also be broken out of the aggregate (b). Thus, when there are slowdowns due to file system overload it can be easily seen what jobs/nodes are the heaviest users of the file system. Ultimately by associating these characteristics with a (user, application,

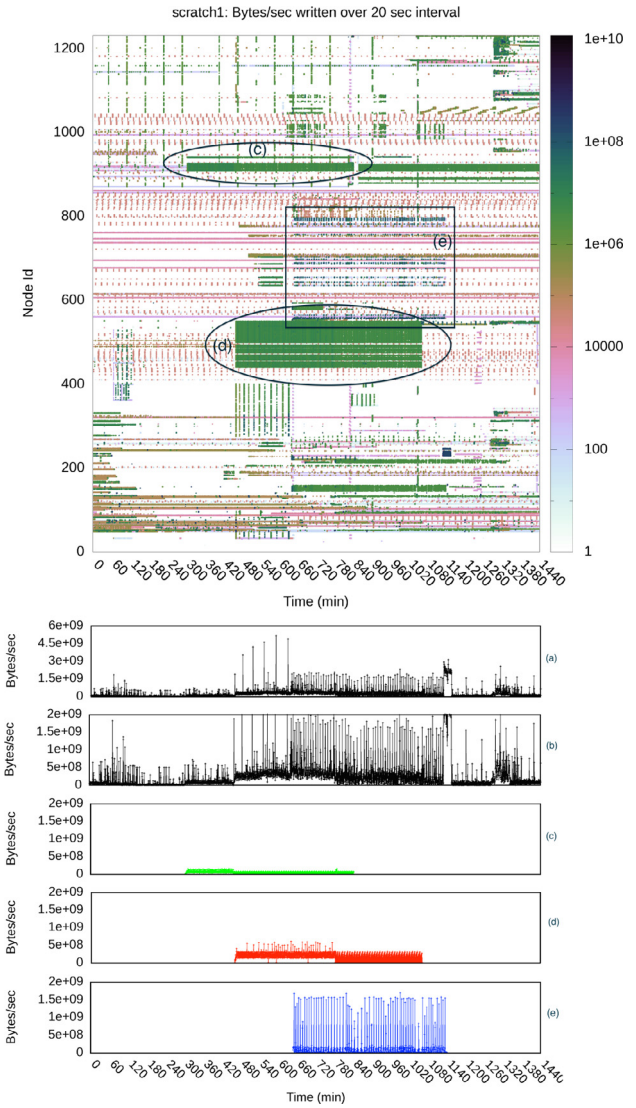

Fig. 3. Continuous, high-fidelity monitoring can be used to identify jobs contending for the same shared resources in addition to an indication of their contention. System map of bytes written/s over the 1232-node cluster for the same 24 h. period as Fig. 2 (top). Total bytes written/s for all nodes (a). Contributions to this total from three jobs overlapping in time (c), (d), and (e); these jobs also marked in (top). To assist in assessing magnitudes, (b) is a slice of (a) limited to the y range of (c)–(e). Coloring in (a)–(e) is for distinguishing plots and is unrelated to the key in (top). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

parameters) tuple, schedulers and resource managers could be more intelligent about placement and thus increase overall machine throughput.

4.2. System memory profiling

While memory is a node-level resource that is only shared by an application's processes, understanding how an application utilizes memory across its distributed processes with respect to total per-node, total concurrently active, and balance quantities are still important. If, for instance, an application is prematurely terminated due to a task decomposition decision that requires more memory per node than is available, then the terminated job's resources over the time until termination may have been wasted, i.e., did not contribute to useful results. Thus, user understanding of their application's memory needs with respect to size and bandwidth and right-sizing memory for the workloads running on a platform is important to efficient system operation.

Additionally providing the information that enables a system administrator to quickly identify the reason for a job running out of memory (e.g., problem decomposition, memory leak, zombie processes from previous jobs) is key to getting the problem resolved quickly and not having needless repeats. Heat maps for active memory utilization, similar to Fig. 2, can

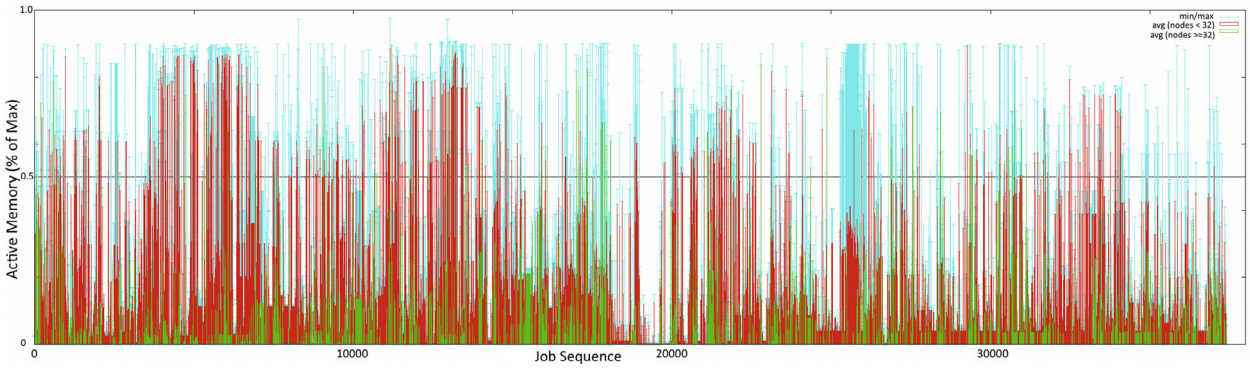

Fig. 4. Global view of system jobs to assess system utilization. Average is in red (job $<$ 32 nodes) and green ($>=$ 32 nodes) with the low and high water marks in blue.

enable a system administrator to easily correlate areas of high utilization with what job was being run and the associated user. Likewise, areas of extremely low utilization, especially over many nodes and long periods of time, can be used to identify candidates for consolidation which could free up resources for other jobs. Note that while low memory utilization could be due to need for higher per-process memory bandwidth it could also be due to a lack of understanding on the part of the user.

Long term memory profiling of a mix of workloads can provide real workflow statistics that can be utilized in next generation platform design and procurement requirements. Fig. 4 shows per-job average, low, and high water marks for memory usage for jobs over a month of normal Chama operation. This view lends itself to getting a qualitative feel for the memory utilization over various sized time frames. In this case, with accompanying statistics, we saw that perhaps more jobs could take advantage of the memory available. This prompted further detailed investigation and identification of some candidate applications. Section 5.5 describes a use case where identification of a user application with low memory utilization has led to runtime changes and improved throughput.

5. Application profiling

In this section, we examine the continuous monitoring data and scoring as it applies to some of the most significant applications in our workload. We present 6 application use cases from SNL's Chama where we utilize a combination of visual analysis and resource utilization scoring to provide understanding of these jobs profiles. We show how the same type of data used for system profiling in Section 4, when combined with scheduler/RM information, can be utilized for coarse profiling of individual jobs which in turn represent applications and their run-time parameters. These profiles also demonstrate that even for significantly large jobs (node counts up to 512 nodes are covered), in many cases, a single graphic per-metric for all nodes can convey important information. Imbalance in time and/or space are easily observed.

The first three cases (Nalu, CTH, Adagio) show how a user can gain insight into their application's behavior from the resource utilization scores and profiles. The out of memory case (OOM) addresses a common job failure mode on our system, and how the profiles can be used to help in identifying and troubleshooting the applications. Two cases (LAMMPS and Gaussian) address how the scores and profiles can be used to identify and gain insight into applications that could better utilize resources and improve overall throughput. In addition, understanding of the workload demands of the final case also drives a recommendation for future procurements.

Note that profiles provided to users include the job information and node legends as in Fig. 11. These are suppressed in some of the figures on this paper due to space constraints at large node counts. Grey areas in the figures distinguish pre- and post-job times in order to inform the viewer of the state of the nodes surrounding the job.

5.1. Resource utilization scoring applied to examples

We first present the scores for the application runs to follow. Note that the scoring is for particular runs of an application and/or workload and is not meant to indicate scores for the application generally. The application name is provided in the tables here only to aid the reader in association with the figures.

Scoring for the metric "Active memory as a % of physical memory" for the application runs next described is tabulated in Table 2 and cross-referenced to the Active Memory plots by the first table column.

In Table 2 both time averaged (μ) and single node peak usage are scored. Peak usage is the maximum memory usage computed with Eq. 5, μ is the time averaged memory usage computed with Eq. 6, and σ is the average deviation of single nodes as normalized with Eq. 7.

This simple single-metric scoring scheme is effective at highlighting persistent memory imbalances (e.g., Figs. 5 and 11). In a long, large job where a transient single node peak usage causes an out-of-memory failure, that peak may be washed

Table 2
Job memory usage and balance converted to scores.

App run	See figure	Peak usage		Average usage		Balance	
		Peak%	score	μ%	score	σ%	score
Nalu	5.1(top)	36	4	22	3	7.3	7
CTH	6.1	26	3	25	3	0.1	10
Adagio	8.1	18	2	2.4	1	3.6	9
LAMMPS	9.1	17	2	15	2	5.0	8
Gaussian	11	61	7	12	2	108.4	1

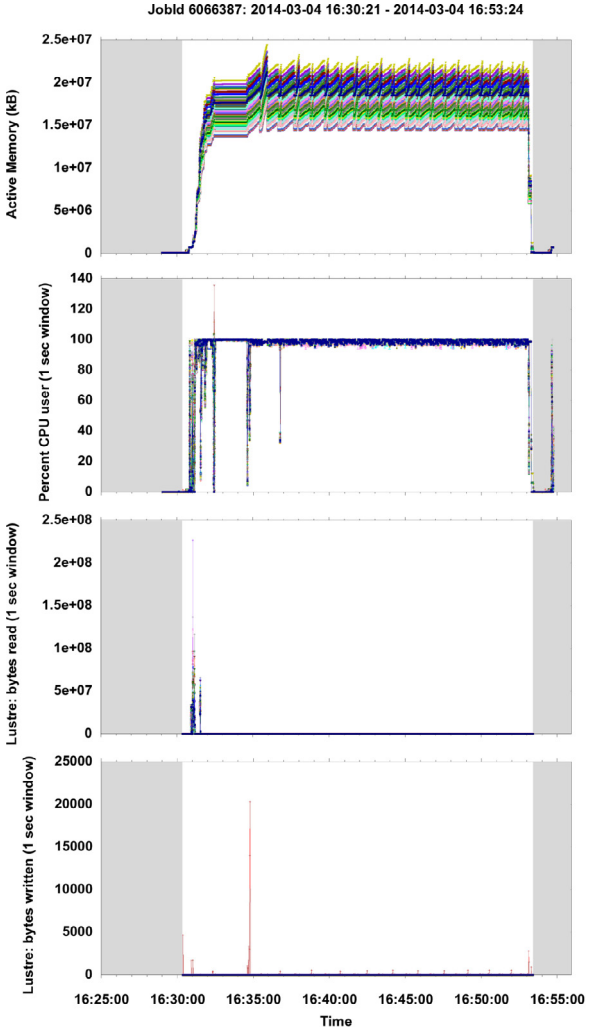

Fig. 5. Profiles for a Nalu 512 node job. Active Memory (1st); CPU user (2nd); Lustre reads (3rd); Lustre writes (4th). The CPU utilization value greater than 100 is an artifact due to a missing data point.

out in the standard deviation calculation. This demonstrates that no individual metric or score is sufficient to characterize rich, unexplored time-series data.

Table 3 applies the scoring method to Lustre read and write bandwidth, illustrating the complexity of creating scores based on single metrics collected from compute nodes. The Lustre store used has 20 QDR IB links connecting it to the compute nodes. The assumptions in the Lustre bandwidth scoring process include the following: (1) the appropriate "fair share" 100% usage value for streaming bandwidth to or from storage for a single compute node is the available store bandwidth (e.g., 80 GB/s) divided by the number of nodes in the job; (2) applications light on Lustre use during computations should not be penalized by including intervals with no traffic in their job-level bandwidth average; (3) no other jobs are contending

Table 3
Lustre bandwidths converted to scores.

		Activity		Peak usage		Average usage		Balance	
App run	See figure	Act. %	Score	Peak %	Score	μ %	Score	σ %	Score
READ									
Nalu	5.3	1.44	1	135	10	11	2	7.3	7
CTH	6.2	1.0	1	65.3	7	42	5	9.7	7
Adagio	8.4	22.1	3	38.8	4	3.0	1	10	6
Lammps	9.2	0.01	1	0.10	1	0.05	1	94	1
Gaussian	12.1	11.9	2	29.9	3	0.50	1	200	1
WRITE									
Nalu	5.4	0.02	1	0.01	1	0.00	1	2263	1
CTH	6.3	2.2	1	122	10	15.1	2	14	6
Adagio	8.5	37.7	4	9.3	1	2.6	1	5	8
Lammps	9.3	0.52	1	7.4	1	0.05	1	140	1
Gaussian	12.2	10.1	2	8.1	1	0.22	1	200	1

for the Lustre filesystem; and (4) there is no need to distinguish fresh and cached bytes. These assumptions could be avoided by performing more complex analyses with data extracted from a Lustre appliance should such data become available.

In Table 3 Activity gives the portion of intervals across all nodes with active (non-zero) read or write values, Peak is the maximum link fair share bandwidth percentage used by Lustre read or write on any node within the job (which due to cache effects and stripe imbalance might be well over the chosen fair share bound), μ is the time averaged fair share bandwidth percentage averaged across all nodes including only those intervals with Lustre read or write activity, and σ is the average deviation as it was for the memory analysis.

Before examining the continuous monitoring time-series data, we can reasonably anticipate some I/O features: (1) the balance scores for Figs. 9 and 12 suggest I/O occurring from only a single node of the job, (2) the Activity and Usage scores for Fig. 5 suggest that checkpoint and output were disabled for the run, and (3) the high Balance and Activity scores suggest Fig. 8 may reveal something interesting to account for the low Usage score.

5.2. Sierra low mach module: Nalu

The one-second profiles for one of the 8192 processing element (PE) simulations are presented in Fig. 5. The CPU and Lustre file system input and output figures look as expected for a simulation that should only exhibit large file I/O during the initial read of the computational mesh and one that shouldn't pause computations. The interesting figure is the active memory profile depicted in the top of Fig. 5. The scores from Table 2 suggest Nalu [10] has plenty of spare memory available across all nodes and that it may not be well balanced in this configuration. The memory profile details the memory spread of the 512 nodes during the approximately 23-min wall time simulation. The spread exhibited accounts for about 10% of a node's memory (6.4 GB). The approximate mean of this spread is 25% (16 GB) of a node's total memory. So, this spread is 16 ± 3.2 GB, or $16 \pm 20\%$ if the percent is with respect to the mean. This spread is larger than expected and no similar memory statistics are provided by Nalu to the user to compare.

For this simulation, the inertial mesh decomposition method was used; this method divides the vertices into sets of equal mass by planes orthogonal to the principal axis and is the default method for Sierra applications. The memory spread may be due to algorithms within Nalu and/or it may be due to the mesh decomposition. The smallest piece of decomposed mesh is approximately 83% of the size of the largest one. Ergo, the spread is likely a combination of both.

Ideally, there would be no memory spread. Its presence indicates the need for additional profiling to better understand its root cause, how to reduce it, and the adverse impact it may have on the time needed to complete the simulation. The active memory profile, in this context, is extremely useful to the user, who may be able to influence it by changing solvers or decomposition methods, and to the developer, who can leverage the profile during development to help optimize its characteristics.

5.3. CTH

The one Hertz profiles for a CTH [11] 7200 PE simulation are shown in Fig. 6 with the two Lustre profiles repeated with annotations in Fig. 7. The scores from Tables 2 and 3 indicate CTH has very well-balanced memory demands and moderately good Lustre usage and balance. There are two types of restart files written: simulation time-based files (RESTART, i.e., `rsct`) and wall time-based files (BACKUP, i.e., `brct`). CTH Spymaster data files (SAVING, i.e., `spct`) are written periodically throughout the run containing pressure, density, etc.

CTH uses an N-N I/O pattern (one file per processor) for managing restarts and Spymaster data. In this example, 7200 new wall-time-based restart files are created and written at every backup time. However, the 7200 CTH Spymaster data files are created only at the beginning of the run (cycle 0) and then appended to during the run. The Spymaster data I/O behavior is considerably different and the Lustre profiles show this nicely, giving very good insight into I/O sizes and patterns. There

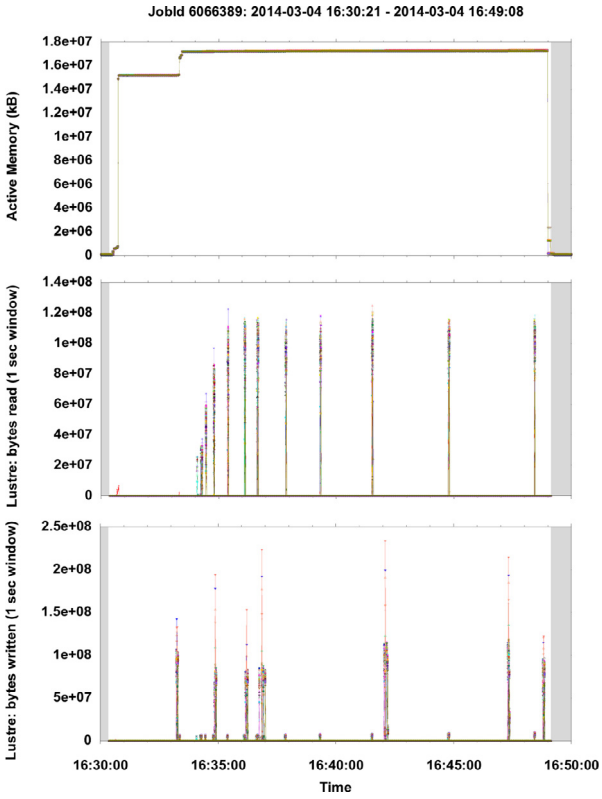

Fig. 6. CTH 450 node job: Active memory (1st); Lustre reads (2nd); and Lustre writes (3rd).

are 12 writes of the Spymaster data in the 19 min run (the write size is small at just a few MB each time). At the next scheduled update of Spymaster data (cycle 240), a read of some information in the 7200 files stored to disk is required in order to append the new data to the existing data. The first three reads (cycle 240, 279, 315) show an increasing trend in the size of data read (30 MB, 60 MB, 90 MB respectively). All of the remaining 8 reads show a stable read size of approximately 120 MB. Without these profiles the user might mistakenly think that the application is just periodically writing a small amount of data and discount the effects of a busy Lustre file system as inconsequential when in fact the reads are much larger than the writes and a busy file system could adversely impact application performance.

5.4. Sierra/SolidMechanics: Adagio

The scores indicate that this Adagio [12] run is well balanced in memory, moderately balanced in I/O and a heavy user of Lustre. One Hertz profiles collected during execution of a 1024 PE simulation of Adagio are shown in Fig. 8. The percent CPU user plot shows a drop halfway through the simulation that correlates well with the increased percent of CPU spent in I/O wait in Fig. 1(top). The Lustre writes of results data in a N-N pattern used by Adagio are impacting CPU utilization. We know from instrumentation of Adagio with the mpiP [13] tool that a large fraction of the run time is spent in MPI due to the complex message exchanges required by contact algorithm computations. The jump in Lustre bytes written (Fig. 8 (bottom)) towards the end is related to the mpiP data that is dumped to the file system at the end of the simulation by MPI task 0. The memory profile shows good load balance except for the jump at the end of the simulation which is attributable to MPI task 0 collecting the mpiP profile from all the other MPI tasks into local buffer for file output.

5.5. Memory and system throughput use case

Sandia initially procured Chama with 32 GB of RAM per node. In 2013 this memory was upgraded to 64 GB per node. The decision to upgrade the memory was based on two considerations: 1) the possibility of significantly improving the job throughput by encouraging users to resize their jobs to require fewer nodes while using more memory on each node, and 2) to minimize the number of jobs that encountered OOM terminations, often seen after substantial amount of node hours already logged by the job. The first consideration is reasonable given that some applications experience higher parallel efficiencies when restructured to use fewer nodes because of their scaling characteristics. The computations-to-communication-time ratio, which impacts parallel efficiency, increases in this case.

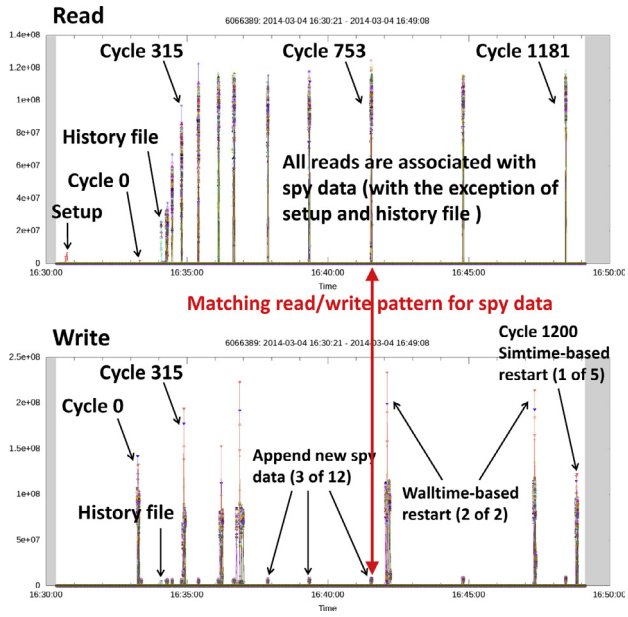

Fig. 7. CTH I/O event chronology of Fig. 6, annotated. Lustre file system reads (top) writes (bottom).

From analysis of the memory utilization of jobs we discovered that a significant number of jobs were not taking advantage of the increased memory (Section 4.2). Top users, in terms of node hours, who were seen to be using relatively little memory were encouraged to experiment with their resource requests and resize their jobs to use fewer nodes. Resource utilization profiles for an instance of one application (LAMMPS) involving 256 nodes (20% of Chama) are shown in Fig. 9. There is significant memory headroom to support a reduced node request (Memory Usage score = 2). Because the application looks CPU bound (unshown) the runtime was expected to increase. Since this user runs many jobs of this size, he and his application were considered good candidates for investigating the possibility of an overall throughput increase. In this case halving the number of nodes resulted in only a modest increase in wall time (10 vs. 8 h per job or a 25% increase). Note that any increase less than 100% is a net savings in the number of available nodes and increases the overall system throughput. Additionally, where simulation studies involve multiple runs of an application, although the run time of an individual job when using fewer nodes may increase, the user may experience faster completion of their ensemble of jobs.

As a result of this investigation the user has revised his resource allocation requests and is utilizing fewer nodes since the increased runtime is more than offset by even the reduced queue time, i.e., waiting for fewer nodes.

5.6. Understanding out-of-memory conditions

A significant cause of job failures on our HPC systems is application processes being killed by the Out-Of-Memory Killer (OOM Killer). Our system administrators had limited insight into running applications' resource demands, including memory, before we began our continuous monitoring. Time history plots such as those in Fig. 10 have provided additional insight that enables both the system administrators and users to better understand how applications are utilizing memory resources and to address the root causes of resulting OOM occurrences. Fig. 10 shows two different jobs' memory profiles including runaway memory demands (left), and both large imbalance and abrupt changes in memory demands (right).

These profiles also have value from the system architects' and buyers' perspectives. Substantial OOMs alone are not a clear sign that increasing memory would be of benefit to the system users. Profiles can help to draw attention to root causes of OOMs and help in evaluating the cost benefit tradeoffs among memory provisioning options.

A case which regularly triggers OOM events with a memory utilization profile that might drive a more sophisticated response to memory demands than just use of larger resource allocations is presented in Section 5.7.

5.7. Gaussian

As in Section 5.5, this case also involves top users on the Chama system who were encouraged to experiment with their resource requests. In this case, however, the application frequently terminates early because it runs out of memory.

Local users of Gaussian [14] and VASP [15,16], both electronic structure codes, account for a substantial fraction of the OOM errors on Chama. A representative active memory profile for Gaussian is shown in Fig. 11. As anticipated from the aforementioned scores, Gaussian is seen to be highly imbalanced in both memory and I/O (practically all I/O goes through the single head node). VASP has demonstrated a qualitatively similar memory profile. There is a clear change in the resource

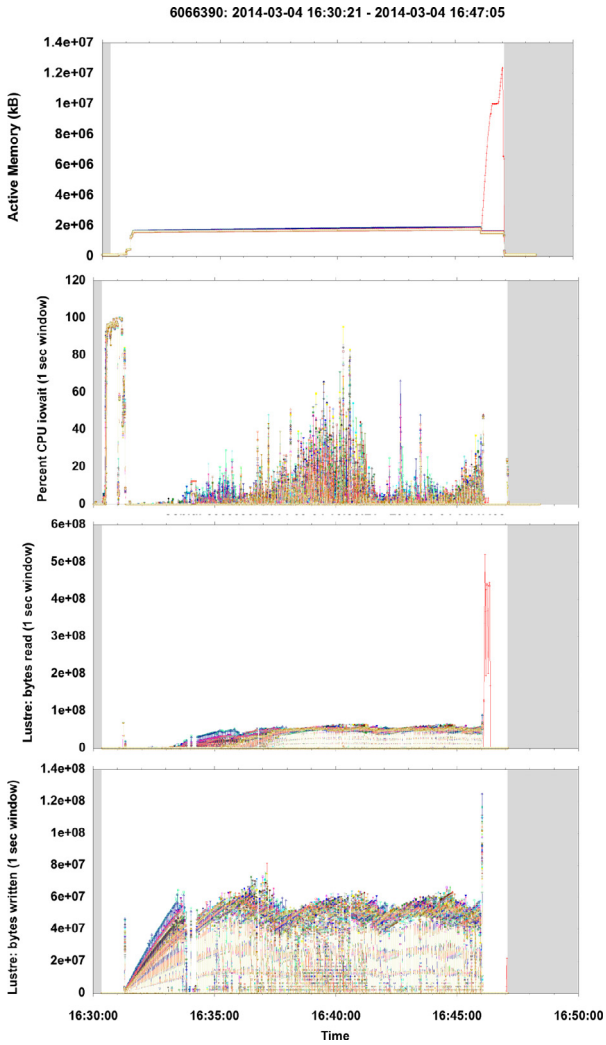

Fig. 8. Adagio 64 node job. Active memory (1st); CPU user (2nd); Lustre reads (3rd); Lustre writes (4th). One node is responsible for both the memory peak at the end of (1st) and the bytes read at the end of (3rd).

demands of the application during its run time. The first phase may run for multiple days, with the job ultimately killed due to the significantly increased memory demand in the second phase. In addition the memory demand is imbalanced across nodes over the whole run. This is observed in the memory scores (Peak Usage of 7, Balance of 1, Usage of 2) and in the plot.

Discussions with the users revealed the memory demand in the second phase drives their allocation request. While they submit the run as a single job, the two phases of the computation (energy levels calculation and DFT) could be separated if there were some benefit in doing so. In addition, they were unaware of the memory imbalance.

File system I/O for the application is dominated by the same node that exhibits high memory utilization (Fig. 12). Reading and writing occurs throughout the run with significantly large values in the second phase. Accumulated reads and writes are shown in the bottom of Fig. 12. Over 2 TB is read over the lifetime of this job.

Imbalanced memory demands over the lifetime of an application and imbalanced I/O demands across the application nodes can inform better options for application-to-resource mapping, both from the point of view of system users and system architects. As a result of this analysis we have recommended the addition of high memory nodes in future purchases to accommodate these needs.

6. Data handling and processing considerations

The visualizations and analyses presented in this work are the results of post processing of LDMS data that has been stored as raw Comma Separated Value (CSV) strings in files or minimally processed, as data was streamed through an aggregator, values stored to a MySQL database. This section describes some of the details and performance bottlenecks related to the various aspects of the data handling.

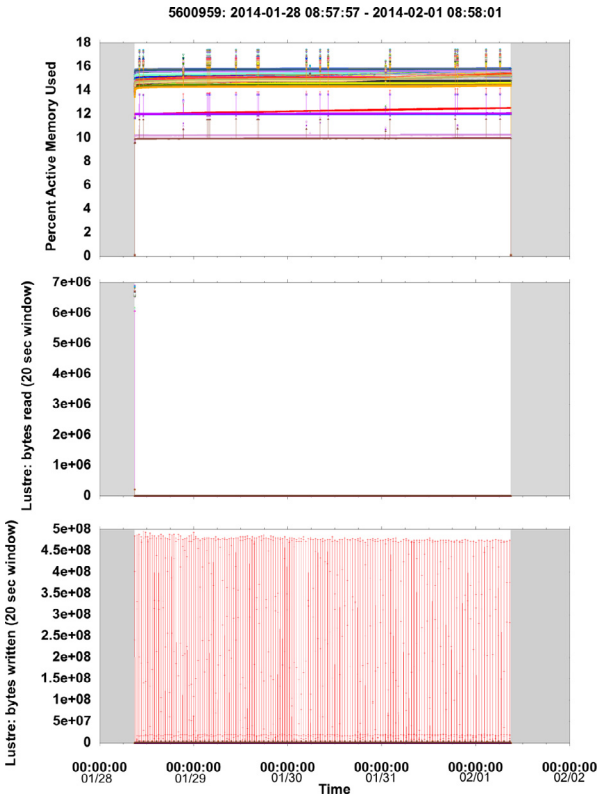

Fig. 9. LAMMPS 256 nodes. Memory usage is a small fraction of that available. CPU utilization data (unshown) suggests that the application is CPU bound and hence, while decreasing the number of nodes may not benefit its individual runtime, it is a candidate for investigation of improving overall machine throughput by decreasing the node resource request.

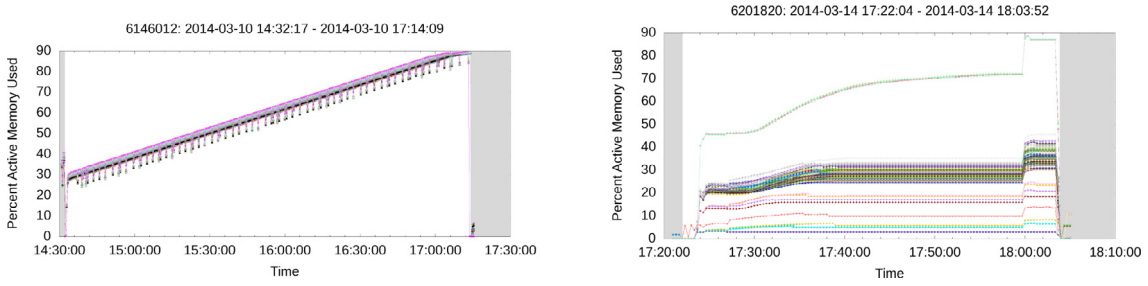

Fig. 10. Examples of jobs killed by OOMKiller where profiles may yield important behavioral insights: run-away memory demands (left); imbalance and abrupt change in memory demands (right) (both 128 nodes).

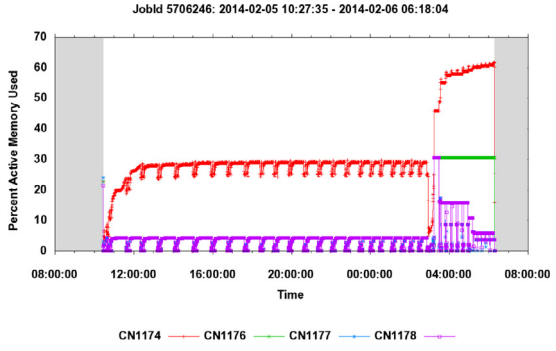

Fig. 11. Gaussian 4 node job: Active memory profile shows imbalance across nodes and across time.

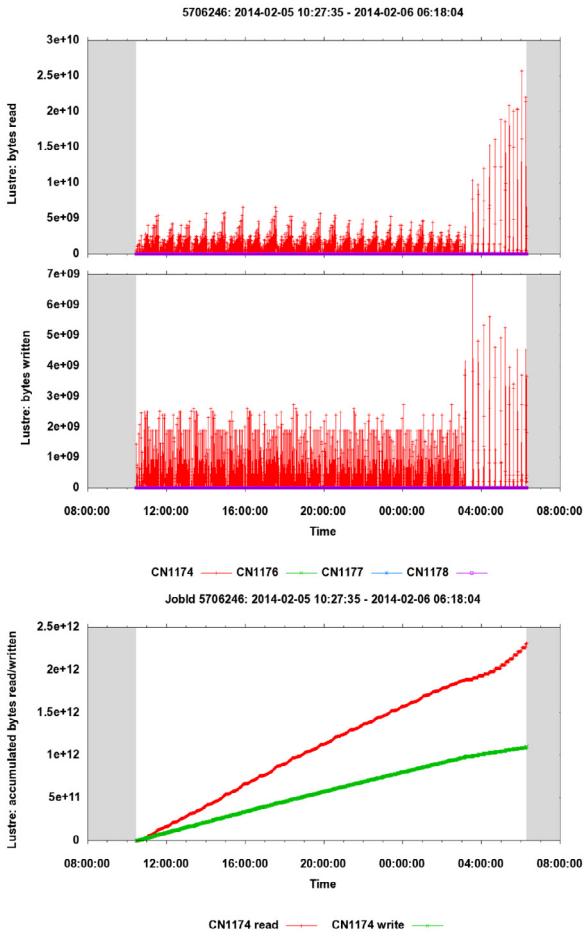

Fig. 12. Gaussian: Lustre reads (1st) Writes (2nd) on a per-node basis. Accumulated Lustre reads and writes (3rd).

Synchronization of data sampling and aggregation provides windowed snapshots of system state. These snapshots provide insight into interactions between applications and the system resources (e.g. correlation of large amounts of Lustre file system reads, writes, or both by an application with sluggish file system performance or with network congestion at I/O nodes. To accomplish this LDMS provides the ability to schedule all data sampling at the same time using each node's real-time clock. Within a single HPC cluster, synchronization of real-time clocks, using NTP, is typically within a millisecond. LDMS data for a single interval of real time has small variations across the cluster in the measured interval limits because the sampling process does not run with elevated priority. These small variations are typically within a few milliseconds [9], so the variations of the sampling interval endpoints for a single interval even at 1 Hz are less than 1%.

Streaming analysis uses data available at the aggregator on a per-set basis as soon as it is collected by the aggregator. If streaming analysis is desired across all the data for an aggregation period, the delay will depend on how timing is set up and possible network congestion but we typically see sets with hundreds of LDMS metrics aggregated from multiple thousands of nodes being available in less than 200 ms. Our streaming analytics as of this writing have been conversion of counts to differences and rates (e.g., bytes Tx/Rx, network bandwidth, error rates) to enable simple use of database functionality for identifying peaks, computing descriptive statistics, and plotting.

Skipped measurements may occur at times as a result of a node's work load, network demands, power state (between jobs), and ldmsd state (e.g. down for collector reconfiguration). In many circumstances other than node reboot, linear interpolation of a cumulative metric to approximate the missing data is justifiable and the possible errors introduced are well bounded. As an example, when scoring of Lustre bandwidth measurements derived from cumulative data transmission counters, the maximum relative error in the rate computed by linear interpolation over K missing intervals is $(K - 1/K)$. When processing active memory measurements, however, interpolation is not justifiable unless we have prior knowledge of the application running on the node.

Missing data (i.e., when no data at all is collected from a node assigned to the application for an extended period or even the entire length of the job) forces us to choose among assumptions to compute our scores and present the results with a warning. The choices are a) the node resource usage is zero, b) the node behaves on a per-interval basis as the average of

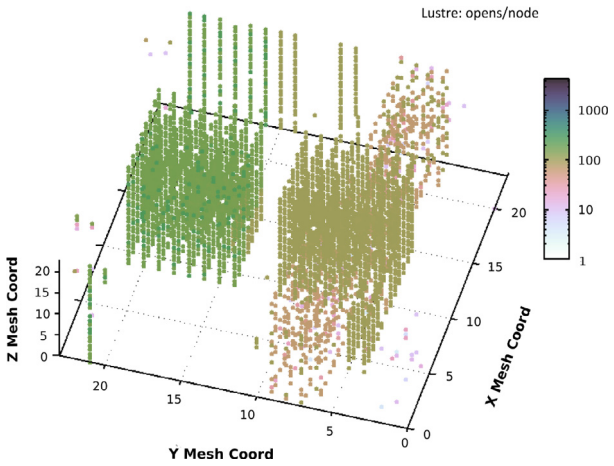

Fig. 13. Time slice of Lustre opens for each node across all 27,648 nodes of NCSA's Blue Waters platform. Nodes are shown in their network topology context. Every two nodes are attached to a Cray Gemini router chip. Router chips are organized in 24 × 24 × 24 cube and interconnected in a 3D torus configuration.

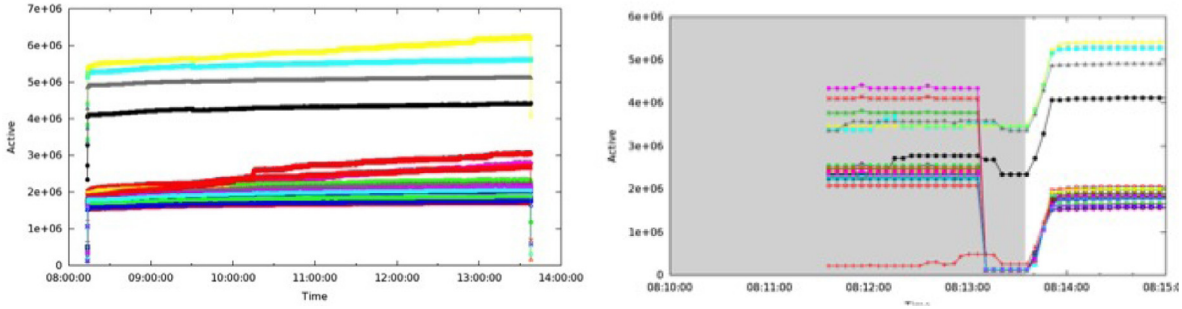

Fig. 14. Whole-system monitoring enables capture and analysis of conditions on nodes before and after the job for discovery of problem conditions.

the other nodes in the job, and c) the node resource usage follows some application-specific model derived from prior runs and the data of other nodes in this run. For the scores we computed, no nodes were absent. When plotting, the time line or points on a heat map simply don't appear.

Scale can affect numbers and placement of LDMS aggregators, latency in full snapshot aggregation and processing, storage technologies, storage volume, and post processing times and techniques. For example, 24 h of 1 Hz LDMS data for 400 metrics on 1232 Chama nodes yields approximately 900 GB of raw data. Loading the entire data set into memory on a single host requires use of specialized large memory hosts and certain analyses, including those from general purpose mathematical analysis packages, may still perform sluggishly due to memory bandwidth limits. In this work we have worked around these issues by performing streaming analysis and database loading of metric data of current focus or processing on small subsets of the data read from raw CSV files. Lossless data reduction methods that do not interfere with analysis algorithms and provide at least the 99% data reduction needed to manage the data on a conventional workstation are the subject of ongoing research.

7. Examples from other platforms

In this section we present information from two other systems monitored in the same fashion as Chama. Fig. 13, shows color mapped values of Lustre file opens for each node over a particular 1 min interval from our data collection [9] on NCSA's 27,648 node Cray XE/XK platform Blue Waters [17]. The visualization includes the location of the nodes within the network topology of the system (Cray Gemini 3D torus). Understanding the job placement in the network topology of the system is necessary for assessing impact both within and external to an application job run. In this case it can be easily seen that there are three distinct file open rate groupings mostly segregated within the network fabric. In addition we have used this type of visualization to construct 3D animations that play through a time series of these slice-of-time visualizations to enable understanding of how network congestion evolves over time within the network fabric.

Fig. 14 shows a visualization that enabled us to discover the root cause of a problem on a 300 node Linux commodity HPC system at SNL. We were using our whole-system monitoring and visualization to investigate why an application that generally ran to completion would sometimes be terminated due to Out Of Memory (OOM) condition even when using

identical input parameters. From the left hand figure it can be seen that, while the application's overall per node memory behaviors are similar in shape, some use much more memory than others. In this case inclusion of pre-job information, shown in gray in the right hand figure, showed that in some cases nodes with unusually high baseline Active memory were allocated. Upon further investigation it turned out that the high baseline conditions were due to zombie processes left on nodes from previous jobs that were consuming significant memory resources. Our current systems, including Chama, now use node checks to prevent such conditions.

8. Related work

Many HPC monitoring systems exist. The comparison of LDMS to other systems is provided in [9].

A joint international consortium supports the HOlistic Performance System Analysis project. HOPSA [18] shares many of our goals (better application and system understanding), but takes a different approach. A combination of light- and heavy-weight *application-level* profiling tools are applied to measure individual application behavior. The data for all applications is aggregated in a database for analysis by the user or administrators. The data collected may vary in frequency and metrics from job to job; this may make analyses aimed at identifying contention for shared resources more difficult. In some cases, the user's binary is modified or rebuilt and data collection overheads may become significant, but it can resolve individual tasks co-scheduled on the same compute node.

The *TACC Stats* [19] project also shares many of our goals. TACC Stats is used by admins for understanding, but the data is aggregated nightly and hence cannot be used for run-time and immediate post-run understanding. The collection interval is on the order of 10 min so the dynamics of resource conflicts are lost.

Allocation and scheduling for non-competition of shared resource demands has been recognized, particularly in cloud environments where multiple virtual machines may share the same node. Characterization of peak and average workload resource demands for this purpose is reported frequently as part of cluster provisioning and VM management. The more general scoring we do here is more in alignment with the style of Intel's VTune OpenMP scalability analysis [20]. We seek higher-level metrics that can also be used for users to gain insight into their simulation configuration, developers to gain insight into the scalability and production uses of their codes, system administrators to gain tactical insight into the applications and their needs across the whole system, and future procurement and development planners to gain insight into the strategic needs of the wide array of applications on all of the supported HPC platforms.

9. Conclusion

Using data from Chama, a 1232-node HPC system, we have demonstrated that continuous, synchronous, whole-system monitoring can provide the ability to do meaningful profiling of system and application resource utilization on production systems. Through examination of system and application profiles we have shown how this information can be used to drive more efficient platform utilization though better matching of applications to resources based on a) user insights into imbalance and resource demands that can be used in addressing performance issues and in allocation requests and b) scheduling and resource allocation decisions that can be made by the resource manager based on knowledge of application resource demands and how those can lead to contention with other concurrent applications. Further, we have shown how knowledge of a site's workload profile can be used for procurement decisions based on evaluation of resources that are under/over provisioned. Finally we have shown how such information can be used by both administrators and users in troubleshooting issues.

While we have been providing the user with profiling plots and statistics at the termination of each job, we are also working on a web based interface to this same data to make browsing easier. We have shown how our application scoring can provide an easy entry point to draw attention to possible areas for improvement. These analyses are new and will need further tuning, including taking into account multiple interacting variables, but they have the potential not only to enable better resource utilization understanding and optimization but to also inform future platform purchases tuned to workflows and even to interact with schedulers and resource managers for making more informed application co-scheduling and allocation decisions.

Acknowledgments

The authors thank Ken Lord for feedback on improving the usefulness of profiles to users experiencing job failures due to OOM; John Noe for prompting the memory usage vs. throughput study; and Aidan Thompson, Michael Foster, and Dan Spataru for participation in that study. Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the United States Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

References

[1] A. Bhatele, K. Mohror, S. Langer, K. Isaacs, There goes the neighborhood: performance degradation due to nearby jobs, in: Proceeding of the International Conference on High Performance Computing, Networking, Storage and Analysis, 2013.

[2] M. Schulz, J. Galarowicz, D. Maghrak, W. Hachfeld, D. Montoya, S. Cranford, Open | SpeedShop: an open source infrastructure for parallel performance analysis, Sci. Program. 16 (2-3) (2008) 105–121, doi:10.3233/SPR-2008-0256.
[3] S.S. Shende, A.D. Malony, The tau parallel performance system, Int. J. High Perform. Comput. Appl. 20 (2) (2006) 287–311.
[4] L. Adhianto, S. Banerjee, M. Fagan, M. Krentel, G. Marin, J. Mellor-Crummey, N.R. Tallent, HPCToolkit: tools for performance analysis of optimized parallel programs, Concurr. Comput. 22 (6) (2010) 685–701, doi:10.1002/cpe.1553.
[5] A. Knüpfer, H. Brunst, J. Doleschal, M. Jurenz, M. Lieber, H. Mickler, M. Müller, W.E. Nagel, The vampir performance analysis tool-set, in: M.M. Resch, R. Keller, V. Himmler, B. Krammer, A. Schulz (Eds.), Tools for High Performance Computing - Proceedings of the 2nd International Workshop on Parallel Tools for High Performance Computing, Springer, 2008.
[6] Cray Inc., Using Cray Performance Analysis Tools, Cray Doc S-2376-52, 2011.
[7] M.L. Massie, B.N. Chun, D.E. Culler, The ganglia distributed monitoring system: design, implementation, and experience, Parallel Comput. 30 (5-6) (2004) 817–840, doi:10.1016/j.parco.2004.04.001.
[8] J. Reams, Extensible monitoring with Nagios and messaging middleware, in: Strategies, Tools, and Techniques: Proceedings of the 26th Large Installation System Administration Conference, LISA 2012, San Diego, CA, USA, 2012, pp. 153–162. December 9-14, 2012.
[9] A. Agelastos, B. Allan, J. Brandt, P. Cassella, J. Enos, J. Fullop, A. Gentile, S. Monk, N. Naksinehaboon, J. Ogden, M. Rajan, M. Showerman, J. Stevenson, N. Taerat, T. Tucker, Lightweight distributed metric service: a scalable infrastructure for continuous monitoring of large scale computing systems and applications, in: Proc. IEEE/ACM International Conference for High Performance Storage, Networking, and Analysis (SC14), 2014.
[10] Sandia National Laboratories,spdomin/Nalu, https://github.com/spdomin/Nalu.
[11] J.M. McGlaun, S.L. Thompson, CTH: a three-dimensional shock wave physics code, Int. J. Impact Eng. 10 (1990) 351–360.
[12] (October 2011), SIERRA Solid Mechanics Team, Sierra/SolidMechanics 4.22 User's Guide, Technical Report SAND2011-7597, Sandia National Laboratories, Albuquerque, New Mexico 87185 and Livermore, California 94550.
[13] J. Vetter, C. Chambreau, MpiP, http://mpip.sourceforge.net.
[14] M.J. Frisch, et al. Gaussian 09 Revision D.01, Gaussian Inc. Wallingford CT, 2009.
[15] Vienna Ab initio Simulation Package VASP, http://www.vasp.at/.
[16] J. Hafner, Ab-initio simulations of materials using VASP: density-functional theory and beyond, J. Comput. Chem. 29 (13) (2008) 2044–2078.
[17] Blue Waters, https://bluewaters.ncsa.illinois.edu.
[18] B. Mohr, V. Voevodin, J. Gimnez, E. Hagersten, A. Knüpfer, D.A. Nikitenko, M. Nilsson, H. Servat, A. Shah, F. Winkler, F. Wolf, I. Zhukov, The HOPSA workflow and tools, in: A. Cheptsov, S. Brinkmann, J. Gracia, M.M. Resch, W.E. Nagel (Eds.), Tools for High Performance Computing 2012, Springer Berlin Heidelberg, 2013, pp. 127–146.
[19] T. Evans, W. Barth, J. Browne, R. DeLeon, T. Furlani, S. Gallo, M. Jones, A. Patra, Comprehensive resource use monitoring for HPC systems with TACC stats, Proceeding of the First International Workshop on HPC User Support Tools, 2014.
[20] Vtune OpenMP Scalability Analysis, https://software.intel.com/en-us/intel-vtune-amplifier-xe.