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, and Joel Stevenson
Sandia National Laboratories
Albuquerque, NM.
Email: (amagela|baallan|brandt|gentile|slefant|smonk|jbogden|mrajan|josteve)@sandia.gov

***Abstract*—A detailed understanding of HPC application's resource needs and their complex interactions with each other and HPC platform resources is 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.

***Keywords*-High Performance Computing; Monitoring**

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

Herein 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 1,232 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 Sections II and III we present our motivation and requirements for monitoring systems and applications in SNL's HPC environment. Next in Sections IV and V 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. We discuss related work in Section VI and address future work and conclude in Section VII.

II. Motivation

The following HPC-centric roles are typically hampered due to a lack of sufficient information: system administrators trying to troubleshoot application/system performance and variation issues as well as maximize system efficiency and throughput; application users striving to minimize time to solution; code developers needing to know if their latest enhancements positively or negatively impacted production simulations; and system architects. The lacking information is insight, at an appropriate fidelity, into how applications are utilizing resources and what resources are being over-subscribed/under-subscribed by how much and why.

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

We are utilizing our Lightweight Distributed Metric Service (LDMS) [1] at SNL to continuously monitor HPC resources and provide insight into resource utilization. LDMS is particularly suitable for this task because it does not significantly skew the resource utilization measurements and it provides synchronized sampling, i.e., data over the same time window across the entire system. On Chama, we monitor with temporal resolutions ranging from 1 to 20 seconds where 20 seconds is the default. The data generated by LDMS can provide insight for many areas/roles including administrative debugging, user code optimization, and HPC hardware architects and procurement planners.

Administrative debugging: 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.

User code optimization: In order to assist users to better understand their application's behavioral characteristics and optimize/debug where needed, for each user job we provide time history plots, such as those presented here, and usage summary text files in a canonical location with appropriate file permissions. Users can then reference the appropriate files for any of their jobs.

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 SNL system architects data to use as requirements in the design and procurement of their future HPC systems.

III. Sampling Fidelity

In order to provide meaningful information on resource utilization, a monitoring system must sample at fidelities commensurate with the features it is necessary to resolve since one cannot know apriori 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 discussed herein. This is illustrated in Figure 1, which shows a resource utilization profile of a 64-node Sierra/SolidMechanics (Adagio) job; this run will be examined in more detail in Section V-B3. Here the time spent in IO wait over sampling intervals at 1 sec (top) and 60 sec (bottom) are considered for the same job. Values during the job execution are shown on the white background. Each node in the job is an individual line. Job Id and time range are given at the top of the figure. This format is used for the figures throughout the rest of the paper.

Insight into the behavior of the application is 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. This detail is lost at larger sampling intervals.

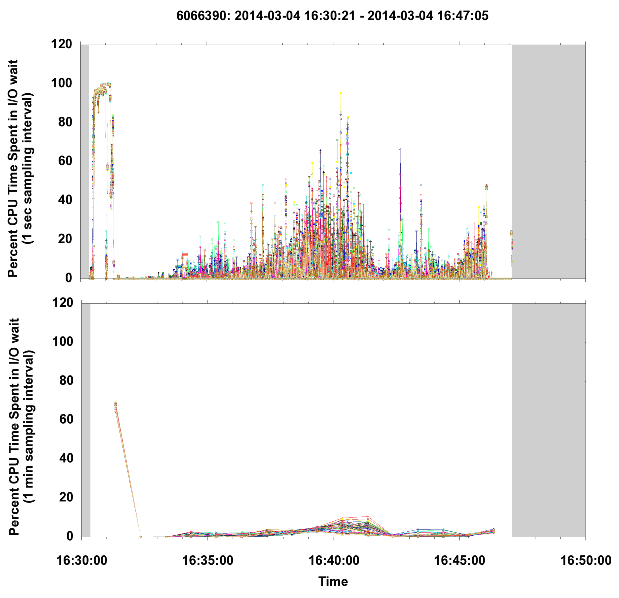


Figure 1. IO wait profiles for a 64-node job: 1 sec interval (top), 60 sec interval (bottom). Features are lost at longer sampling intervals. Each line is a single node's data. The legend of lines colored by nodes have been suppressed. The gray background shows times pre- and post-job. These job presentation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

IV. System Profiling Using Whole-System Data

Collecting full-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. An important attribute of our monitoring process in this case is that data collection is synchronized in time across all nodes of a system. This synchronization has the effect of producing a "snapshot" across the whole system of the state metrics that are being collected. System-wide maps and plots are presented in this section; these can be used to easily visualize aggregate use of resources as well as how individual jobs are contributing to that utilization.

A. *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-hour period are shown in Figures 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 second interval is over an order of magnitude greater than the maximum number of bytes written (example near-maximum magnitude read occurrences are marked in Figure 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 resource limits are being hit. In Figure 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, parameters) tuple, schedulers and resource managers can be more intelligent about placement and thus increase overall machine throughput.

B. *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 the result is the need for a different task decomposition that spreads the job over more nodes this means that the terminated job's resources over the time until termination were 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 an efficient system. Note that long term memory profiling provides real-use statistics that can be utilized in next generation platform design and procurement requirements.

Additionally providing the information that enables an administrator to quickly identify the reason for a job running

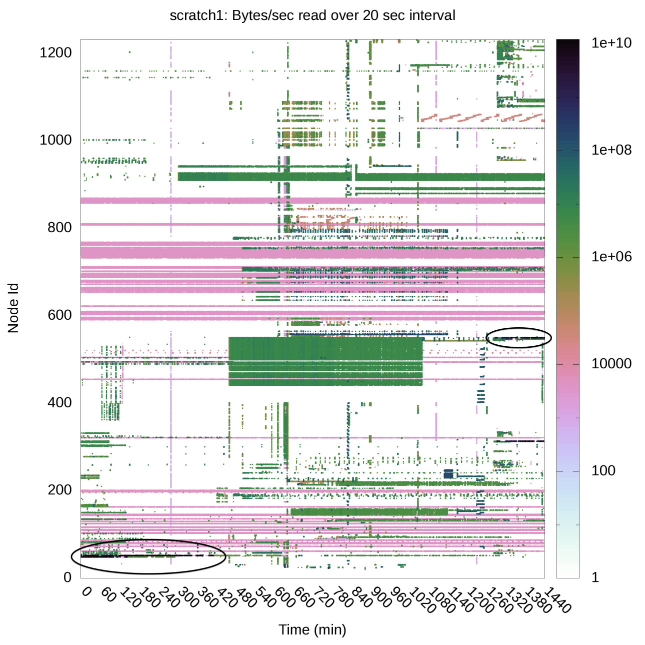
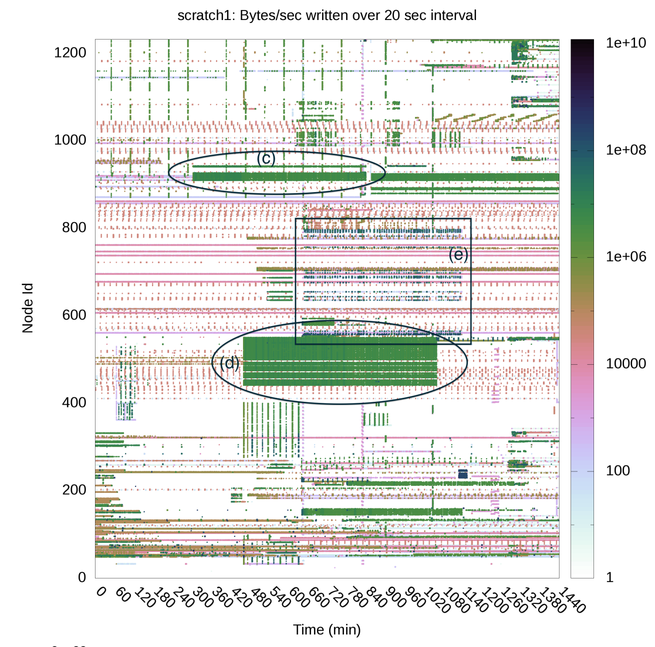


Figure 2. System map of bytes read/sec over the 1,232-node cluster over a 24 hr. period. Hot spots and long-lived features are identifiable.

out of memory is key to getting the problem resolved quickly and not having needless repeats. Heat maps for active memory utilization, similar to Figure 2, can 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. Section V-B4 describes a use case where identification of a user application with low memory utilization has led to runtime changes and improved throughput.

V. Application Profiling Using Whole System Data

Appropriate mapping of an application to system resources is key to maximizing throughput. Even if a user has profiled their code with typical profiling tools, the results may not capture the performance of the application under production conditions where other applications are competing for shared resources. Additionally, when new features are added to an application or enhancements to the platform occur, rigorous re-profiling may not take place. Finally, since the overall performance is subject to a number of application needs, complete profiling of an application, including changes in resource demands with a wide variety of application input, may not be feasible. Instrumenting



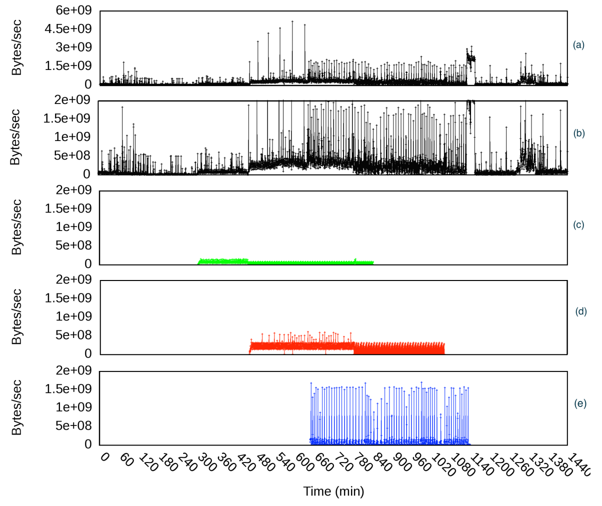


Figure 3. Continuous, high-fidelity monitoring can be used to identify jobs contending for the same shared resources in addition to an indication of their contention. System map of bytes written/sec over the 1,232-node cluster for the same 24 hr. period as Figure 2 (top). Total bytes written/sec for 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).

a particular application can provide insight but is done on a per-application basis, and access to system-specific information (beyond the interfaces to quantities in `/proc`) would require re-instrumentation for every platform.

As a result, in many cases, users apply rules-of-thumb to determine the resource requests and mapping (e.g., X mesh elements per node, Y processors per node) without knowing the actual resource demand numbers. Further, system administrators, who may have understanding of a specific sub-system (e.g., network or storage) have limited access to the application's usage requirements and thus are missing key information that can be used for early problem detection. Similarly purchasers lack this information and are limited in their ability to match acquisitions to needs.

This section shows how the same type of data used for system profiling in Section IV, 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. We first introduce the concept of application resource utilization scoring. We then 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 discuss profiles of active memory and Lustre I/O for the applications presented. 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.

Note that profiles provided to users include the job information and node legends as in Figure 10. 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.

A. *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.

Our currently defined scores focus on: 1) determining the fractional utilization of target resources relative to some upper value based on hardware and/or configuration limitations and 2) determining how well-balanced the utilization of the target resources is across all nodes of a job. We use deciles to score *usages* and the skewed scale in Table I to convert standard deviation into a *balance* score. On both scales, 1 is the "poor" end of the scale. The skewed balance scale conversion is chosen empirically to emphasize poor performers since σ is unbounded.

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	$\sigma <$	∞	35	30	25	20	15	10	7	4	1

Table I
BINS CONVERTING μ & σ TO SCORES.

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.

To illustrate this basic scoring method, we first show it applied to the metric "Active memory as a % of physical memory" for the applications described in Section V-B. The results are tabulated in Table II and cross-referenced to the Active Memory plots (Section V-B) in the final table column.

Fig.	%Peak	μ %	σ %	Bal	Usage	P.U.
4.1(top)	36	22	7.3	7	3	4
5.1	26	25	0.1	10	3	3
7.1	18	2.4	3.6	9	1	2
8.1	17	15	0.1	8	2	2
10	61	12	108.4	1	2	7

Table II
JOB MEMORY USAGE CONVERTED TO SCORES.

In Table II % Peak is the maximum memory usage seen on any node within the job, μ is the time averaged memory usage percentage averaged across all nodes, and σ is the average deviation of single nodes from the job average μ expressed as relative percentage: $100 \times \sigma/\mu$. Both time averaged (μ) and single node peak (P.U.) are scored.

This simple single-metric scoring scheme is effective at highlighting persistent memory imbalances (e.g., Figures 4 and 10). However, in a large job where transient single-node peak usage causes an out-of-memory failure, that peak may be washed out in the standard deviation calculation.

Table III 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., 80GB/second) 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 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 III Peak is the maximum link fair share bandwidth % 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

Fig.	Peak %	μ %	σ %	Bal	Usage	Act. %
READ						
4.3	135.18	11	7.3	7	2	1.44
5.2	65.29	42	9.7	7	5	1.00
7.4	38.79	3.0	10	6	1	22.14
8.2	0.10	0.05	94	1	1	0.01
11.1	29.92	0.50	200	1	1	11.92
WRITE						
4.4	0.01	0.00	2263	1	1	0.02
5.3	122	15.1	14	6	2	2.2
7.5	9.3	2.6	5	8	1	37.7
8.3	7.4	0.05	140	1	1	0.52
11.2	8.1	0.22	200	1	1	10.1

Table III
LUSTRE BANDWIDTHS CONVERTED TO SCORES.

write activity, and σ is the average deviation of single nodes from the job average μ expressed as relative percentage: $100 \times \sigma/\mu$. Column Act. gives the portion of intervals across all nodes with active (non-zero) read or write values.

Before examining the continuous monitoring time-series data, we can reasonably anticipate some I/O features: (1) the balance scores for Figures 8 and 11 suggest I/O occurring from only a single node of the job, (2) the Activity and Usage scores for Figure 4 suggest that checkpoint and output were disabled for the run, and (3) the high Balance and Activity scores suggest Figure 7 may reveal something interesting to account for the low Usage score.

B. Application Use Cases

In this section, we examine the continuous monitoring data and scoring as it applies to some of the most significant applications in our workload.

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.

1) Sierra Low Mach Module: Nalu: The one-second profiles for one of the 8,192 processing element (PE) simulations are presented in Figure 4. 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 Figure 4. The scores from Table II suggest Nalu [2] 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-minute 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% of a node's total memory (16 GB). So, this spread is 16$\pm$3.2 GB, or 16$\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.

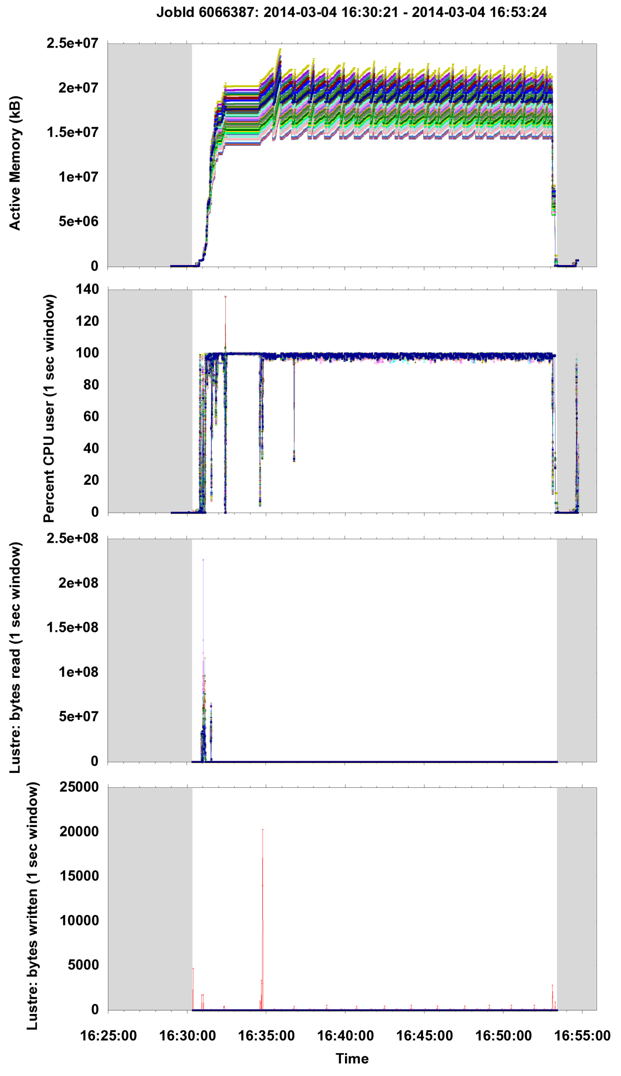


Figure 4. Profiles for a Nalu 512 node job. Active Memory (1st); CPU user (2nd); Lustre reads (3rd); Lustre writes (4th). The CPU utilization value greater than 100 is an artifact due to a missing data point.

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.

2) CTH: The one Hertz profiles for a CTH [3] 7,200 PE simulation are shown in Figure 5, with the two Lustre profiles repeated with annotations in Figure 6. The scores from Tables II and III 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.

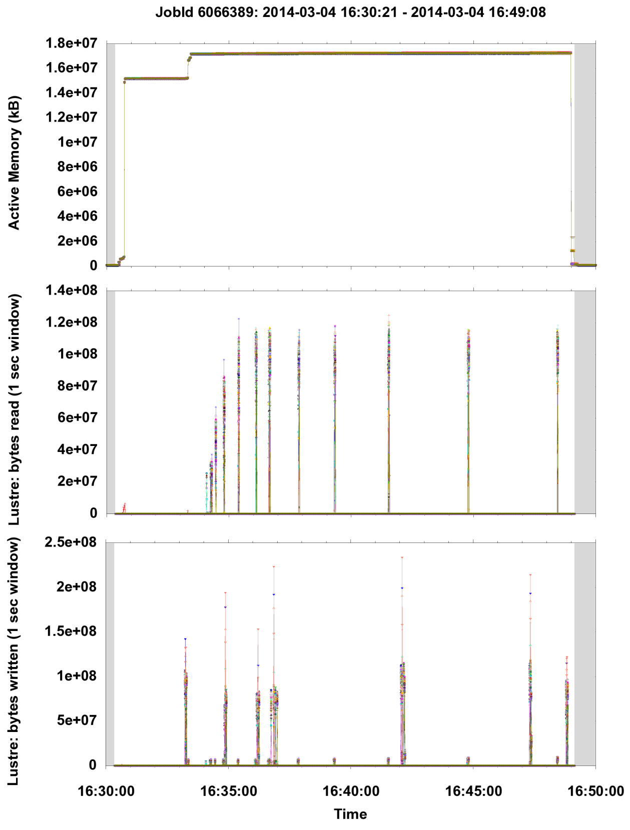


Figure 5. CTH 450 Node Job: Active Memory (1st); Lustre reads (2nd); and Lustre writes (3rd).

CTH uses an N-N I/O pattern (one file per processor) for managing restarts and Spymaster data. In this example, 7,200 new wall-time-based restart files are created and written at every backup time. However, the 7,200 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 are 12 writes of the Spymaster data in the 19 minute 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 7,200 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.

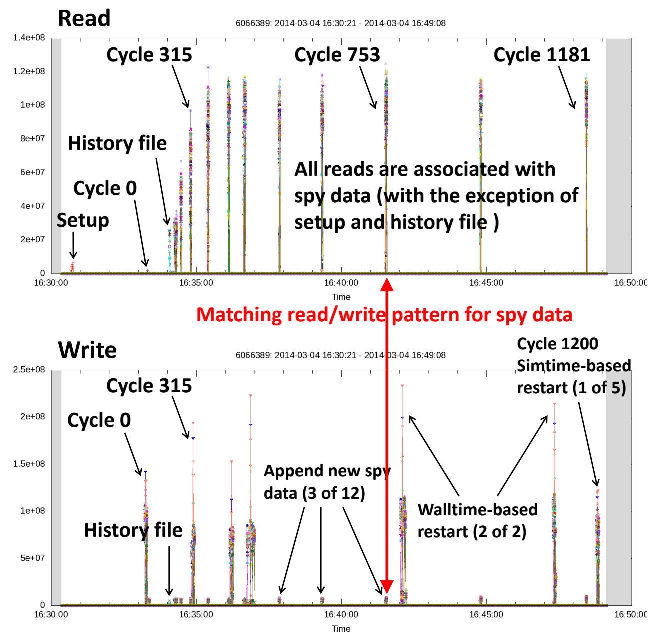


Figure 6. CTH I/O event chronology of Figure 5, annotated. Lustre file system read (top) write (bottom).

3) Sierra/SolidMechanics: The scores for this Adagio [4] run indicated it is well balanced in memory, moderately balanced in I/O and a heavy user of Lustre. One Hertz profiles collected during execution of a 1,024 PE simulation of Adagio are shown in Figure 7. 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 Figure 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 [5] 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 (Figure 7 (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.

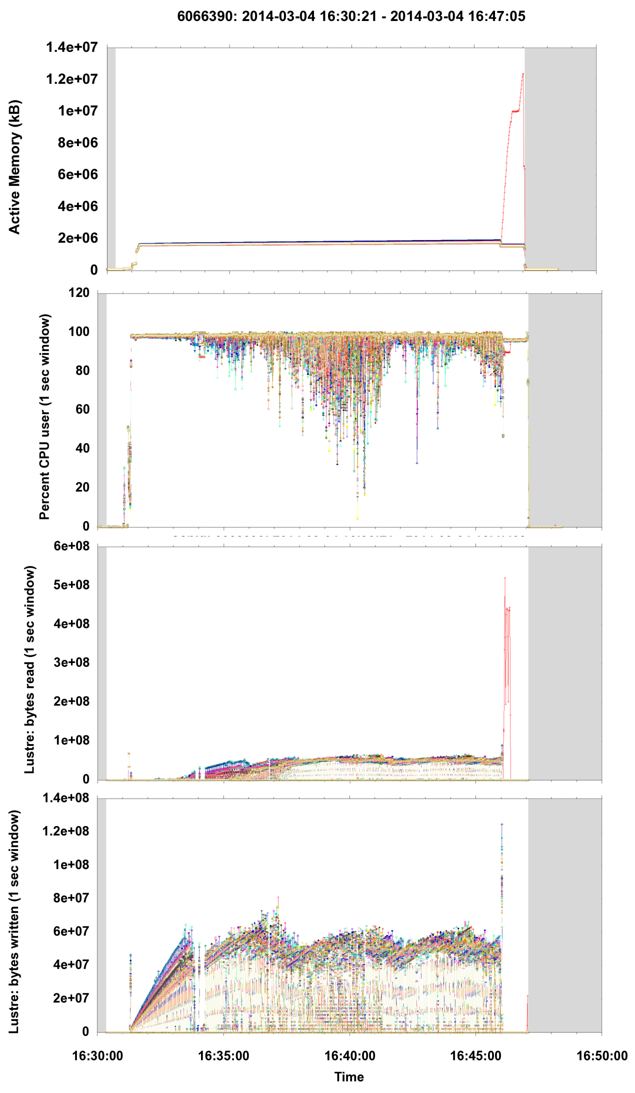


Figure 7. Adagio 64 Node job. Active Memory (1st); CPU user (2nd); Lustre reads (3rd); Lustre writes (4th). One node is responsible for both the memory peak at the end of (1st) and the bytes read at the end of (3rd).

4) 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 1st consideration above 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.

From analysis of the memory utilization of jobs we discovered that a significant number of jobs were not taking advantage of the increased memory. Users often pick the number of nodes (i.e., the degree of parallelism) empirically to minimize wall clock time to solution and complete their investigations expeditiously. Rarely there is such a drastic resource modification that they need to re-evaluate.

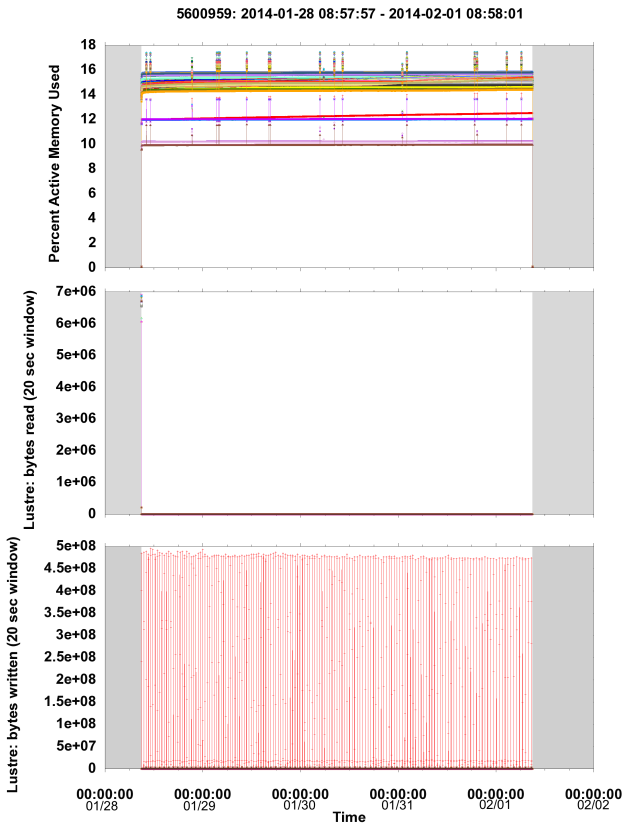


Figure 8. LAMMPS 256 nodes. Memory usage is a small fraction of that available. CPU utilization data (unshown) suggests that the application is CPU bound and hence, while decreasing the number of nodes may not benefit its individual runtime, it is a candidate for investigation of improving overall machine throughput by decreasing the node resource request.

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 Figure 8. 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 versus 8 hours 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) 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 (OOMKiller). System administrators had very limited insight into running applications resource demands including memory before we began our continuous monitoring.

Figure 9 shows two different jobs' memory profiles including run-away memory demands (top), and large imbalance in memory demands and abrupt changes (bottom).

These profiles also have value from the system architects' and buyers' perspectives as well. 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.

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 V-B6.

6) Gaussian: As in Section V-B4, this case also involves top users on the Chama system who were encouraged to experiment with their resource requests. In this case, however, this application frequently terminates early because it runs out of memory.

Local users of Gaussian [6] and VASP [7], 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 Figure 10. 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

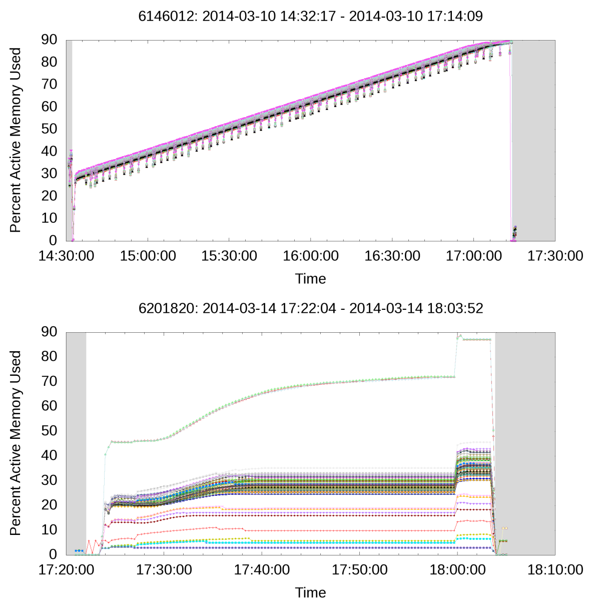


Figure 9. Examples of jobs killed by OOMKiller where profiles may yield important behavioral insights: run-away memory demands (top); imbalance and abrupt change in memory demands (bottom) (both 128 nodes).

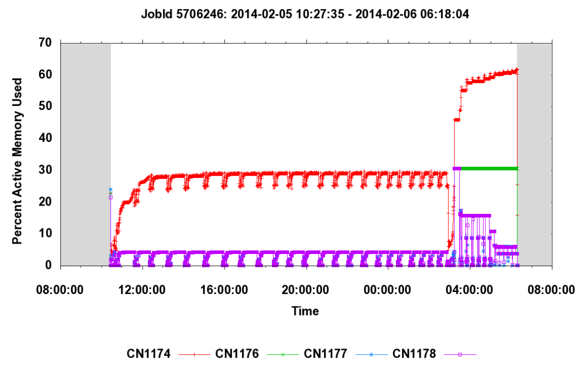


Figure 10. Gaussian 4 Node Job: Active Memory profile shows imbalance across nodes and across time.

demonstrated a qualitatively similar memory profile. There is a clear change in the resource 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

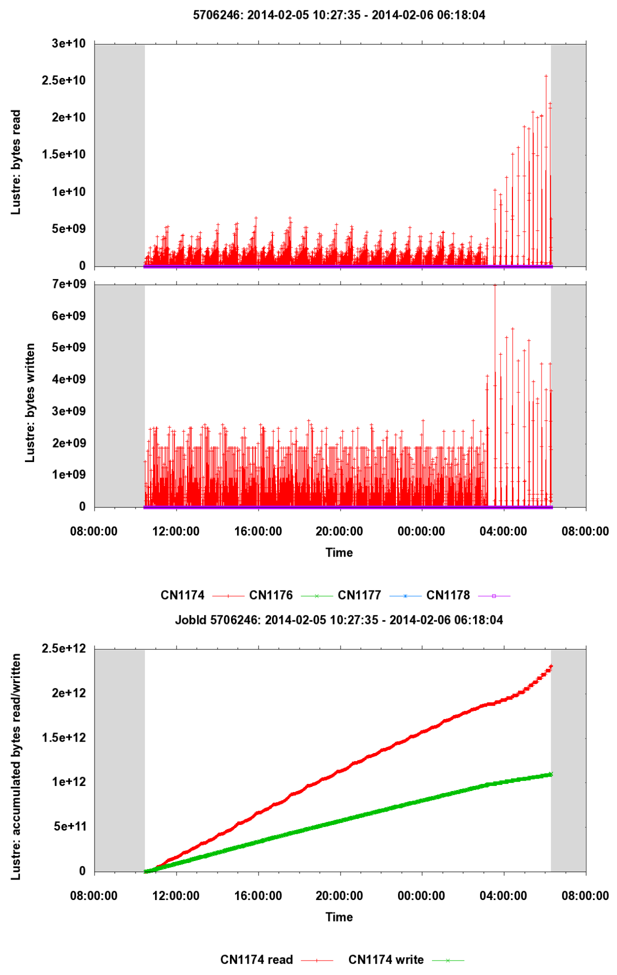


Figure 11. Gaussian: Lustre Reads (1st) Writes (2nd) on a per-node basis. Accumulated Lustre reads and writes (3rd).

same node that exhibits high memory utilization (Figure 11). 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 Figure 11. 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.

VI. Related Work

Many HPC monitoring systems exist. The comparison of LDMS to other systems is provided in [1].

A joint international consortium supports the HOlistic Performance System Analysis project. HOPSA[8] 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 is 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 distinguish individual tasks co-scheduled on the same compute node.

The *TACC Stats/HPC Report Card* [9] project 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 minutes 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 even 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 [10]. 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.

VII. Conclusion

Using data from Chama, a 1,232-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 RM 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 in troubleshooting issues by both the admins and the users.

While we have been providing the user with profiling plots and statistics at the termination of each job, we are working on a web based interface to this same data. The application scoring provides an easy entry point to draw the attention of users to possible areas of improvement for their application to resource matching, to draw the attention of admins to potential problem jobs and inefficiencies on the system, and to eventually be used in an automated fashion by resource management systems. We are currently expanding and refining the application scoring in order to hone in on the sweet spot for the metrics under consideration.

Acknowledgments

The authors thank Ken Lord for feedback on improving the usefulness of profiles to users experiencing job failures due to OOM and Aidan Thompson, Michael Foster, and Dan Spataru for participation in and feedback on the memory usage vs throughput study.

References

[1] 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, and 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.

[2] Sandia National Laboratories, "spdomin/Nalu," https://github.com/spdomin/Nalu.

[3] J. M. McGlaun and S. L. Thompson, "CTH: A Three-Dimensional Shock Wave Physics Code," *International Journal of Impact Engineering*, vol. 10, pp. 351–360, 1990.

[4] SIERRA Solid Mechanics Team, "Sierra/SolidMechanics 4.22 User's Guide," Sandia National Laboratories, Albuquerque, New Mexico 87185 and Livermore, California 94550, Technical report SAND2011-7597, October 2011.

[5] J. Vetter and C. Chambreau, "mpiP," http://mpip.sourceforge.net.

[6] M. J. Frisch *et al.*, "Gaussian 09 Revision D.01," Gaussian Inc. Wallingford CT 2009.

[7] "Vienna Ab initio Simulation Package VASP." [Online]. Available: http://www.vasp.at/

[8] B. Mohr, V. Voevodin, J. Gimnez, E. Hagersten, A. Knpfer, D. Nikitenko, M. Nilsson, H. Servat, A. Shah, F. Winkler, F. Wolf, and I. Zhukov, "The HOPSA Workflow and Tools," in *Tools for High Performance Computing 2012*, A. Cheptsov, S. Brinkmann, J. Gracia, M. M. Resch, and W. E. Nagel, Eds. Springer Berlin Heidelberg, 2013, pp. 127–146.

[9] J. Hammond, B. Barth, and J. McCalpin, "TACC Stats/HPC Report Card," http://www.mcs.anl.gov/events/workshops/iasds11/presentations/jhammond-iasds.pdf.

[10] "Vtune openmp scalability analysis," https://software.intel.com/en-us/intel-vtune-amplifier-xe.