The Lightweight Distributed Metric Service: A Scalable Infrastructure for Continuous Monitoring of Large Scale Computing Systems and Applications

Anthony Agelastos[§], Benjamin Allan[§], Jim Brandt[§], Paul Cassella[*], Jeremy Enos[†],
Joshi Fullop[†], Ann Gentile[§], Steve Monk[§], Nichamon Naksinehaboon[‡], Jeff Ogden[§],
Mahesh Rajan[§], Michael Showerman[†], Joel Stevenson[§], Narate Taerat[‡], and Tom Tucker[‡]
[*]Cray, Inc. Seattle, WA. Email: cassella@cray.com
[†]National Center for Supercomputing Applications, Univ. of Illinois, Urbana, IL.
Email: (jenos|jfullop|mung)@ncsa.illinois.edu
[‡]Open Grid Computing, Austin, TX. Email: (nichamon|narate|tom)@opengridcomputing.com
[§]Sandia National Laboratories. ABQ, NM. Email: (amagela|baallan|brandt|gentile|smonk|jbogden|mrajan|josteve)@sandia.gov

***Abstract*—Understanding how resources of High Performance Compute platforms are utilized by applications both individually and as a composite is key to application and platform performance. Typical system monitoring tools do not provide sufficient fidelity while application profiling tools do not capture the complex interplay between applications competing for shared resources. To gain new insights, monitoring tools must run continuously, system wide, at frequencies appropriate to the metrics of interest while having minimal impact on application performance.**

We introduce the Lightweight Distributed Metric Service for scalable, lightweight monitoring of large scale computing systems and applications. We describe issues and constraints guiding deployment in Sandia National Laboratories' capacity computing environment and on the National Center for Supercomputing Applications' Blue Waters platform including motivations, metrics of choice, and requirements relating to the scale and specialized nature of Blue Waters. We address monitoring overhead and impact on application performance and provide illustrative profiling results.

Catagories and Subject Descriptors: C.4 [Computer Systems Organization]: Performance of Systems – *Measurement techniques; Performance attributes*; K.6.2 [Management of Computing and Information Systems]: Installation Management – *Performance and Usage Measurement*; C.2.3 [Computer Communication Networks]: Network Operations – *Network monitoring*

General Terms: Management, Monitoring, Performance

Keywords: resource management, resource monitoring

I. Introduction

There exists an information gap between coarse-grained system event monitoring tools and fine-grained (instruction, function, or message level) application profiling tools. Coarse-grained monitoring tools record the status of hardware periodically, typically on an interval of minutes. The data is automatically checked for out-of-normal conditions that are then communicated to system administrators for investigation. Profiling tools typically target fine-grained characterizations of specific applications execution profiles with respect to what code sections take the most time and metrics such as cache misses, memory bus bandwidth, and MPI barrier time so the developer can tune for more efficient or balanced operation. Along with the detailed insight, however, comes significant execution overhead and developer time for analysis and code tuning. Because of these costs, profiling and tuning are usually episodic activities [1] rather than part of normal execution. The normal behavior of the unmodified application across the full span of inputs used on a daily basis thus remains largely unknown. Further, its impact on system behavior and other applications is difficult to infer from such isolated profiling.

Sandia National Laboratories (SNL) and Open Grid Computing (OGC) are jointly developing OVIS, a suite of High Performance Computing (HPC) monitoring, analysis, and feedback tools, to fill this information gap. The long-term goal is to obtain insight into behavioral characteristics of individual applications with respect to platform resource utilization (e.g. memory, CPU, network, power) and how platform resources are being utilized, stressed, or depleted due to the aggregate workload and job placement on the resources. In order to reach this goal, system-wide data must be continuously collected at frequencies suitable for resource utilization analysis. The data collection, transport, and storage component of the OVIS suite is a low overhead monitoring tool called the Lightweight Distributed Metric Service (LDMS). LDMS provides more flexibility and proven scalability with respect to dynamic configuration and variety of both transport and storage support than any monitoring software we have evaluated. LDMS scales well to many thousands of nodes and to data sampling intervals on the order of one second. Since the overhead is low, LDMS can be deployed on a continuous basis across an entire HPC platform.

We elaborate on the motivation and utility of high fidelity, whole HPC system monitoring, describing the LDMS framework and its applicability to this task. Our functional comparison with other system level monitoring tools justifies our claim that LDMS is a better fit for this regime. We present details from production deployments on two significantly different HPC systems: 1) The National Center for Supercomputing Applications (NCSA) Cray XE6/XK7 capability platform *Blue*

[†]This research is part of the Blue Waters sustained-petascale computing project, which is supported by the National Science Foundation (award number ACI 1238993) and the state of Illinois. Blue Waters is a joint effort of the University of Illinois at Urbana-Champaign and its National Center for Supercomputing Applications.

[§]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.

Waters, and 2) one of Sandia's Infiniband connected Linux capacity clusters, *Chama*. For each case we include: motivation for resource utilization metrics being collected; collection overhead in terms of memory footprint, CPU utilization, and network load; and results from application performance impact testing. Additionally we present illustrative characterizations derived from this data which demonstrate both system and application perspectives. This is the first presentation at this size and fidelity of the *Blue Waters* Gemini based High Speed Network (HSN) performance counter data. Finally we summarize and present future work.

II. Motivation and Goals

Large scale tightly coupled scientific application performance is subject to effects of other concurrently running applications that compete for the same shared resources (e.g., network bandwidth, shared file systems) and to application placement effects, even in the absence of such contention. Hence there is a significant body of work dedicated to optimizing application resource allocation for various objectives (e.g., [2]–[5]). Those carrying out such work in shared production environments rarely have access to detailed information of machine state, such as what competing applications are running concurrently on the system and their resource demands.

Cray XE/XK systems, which are widely-used in scientific computing (e.g., NCSA's Blue Waters, NERSC's Hopper and Los Alamos's Cielo), have a shared network architecture [6], [7] in which traffic between nodes dedicated to one application may be routed through Gemini network elements that are directly connected to nodes belonging to other applications. Thus not only may one application's performance be impacted by another application's HSN traffic, but also information about congestion along an application's traffic routes may not be accessible to that application. Bhatele et. al. [8] have observed ranges of execution time of a communication heavy parallel application from 28% faster to 41% slower than the average observed performance on a Cray XE6 system and have attributed this significant performance variation to impacted messaging rates due to contention with nearby applications for the shared communication infrastructure.

The lack of data from a system perspective prevents users from understanding the sometimes large variations experienced by similar application runs and limits their ability to more optimally place or configure jobs. Relatedly, diagnosing system issues when degraded application performance has been experienced is difficult when there is limited data on the state of the system and the expected behavior. At NCSA and Sandia our large scale HPC platforms share both these problems and lack of pertinent information. This has motivated our work in high fidelity monitoring. Information that we deemed important to obtain on a system-wide basis in order to gain insight into system and application performance includes:

- Network related information: Congestion, Delivered Bandwidth (Total), Operating System Traffic Bandwidth, Average Packet Size, and Link Status
- Shared File System information (e.g. Lustre): Opens, Closes, Reads, Writes
- Memory related information: Current Free, Active
- CPU information: Utilization (user, sys, idle, wait)

III. Requirements

A. Blue Waters

NCSA's Cray XE/XK platform *Blue Waters* [9] is comprised of 27,648 nodes. The network is a 3-D torus built on Cray's proprietary Gemini interconnect. Motivation for monitoring began with a desire to be able to better understand the impact from network contention on application performance. It became apparent that we needed a method to gather and analyze more detailed information regarding the usage of individual network links at the administrative level. We decided to concurrently monitor other metrics that could also help us analyze both system and user behaviors. To this end we assembled a team of systems engineers and applications specialists at NCSA to compile the list of desirable metrics.

Blue Water's size, diskless method of booting, with all compute node images being served from a central server, and the desire to have the monitoring come up at boot time required the monitoring libraries and binaries to be included in the boot image and that they be relatively small given the total image size of about 400MB.

Finally we needed to balance the value of the data against both the performance impact on applications and data volume that we would need to store. Though we determined that we could achieve most goals using a one minute collection interval, we decided to also investigate the impact of a one second interval to determine the feasability of higher collection rates using LDMS should higher fidelity observations be desired.

B. SNL Capacity Systems

The tools typically employed on SNL HPC clusters lack the fidelity to gather resource usage data on a per job or per user basis. Thus an extensible tool to aid HPC system administrators, users, and procurement planners to better understand the actual compute, memory, file system and networking requirements of SNL codes was needed.

Additional requirements for a monitoring tool are scalability to thousands of nodes and collection of hundreds of metrics per node on intervals of seconds to minutes. The reason for such high fidelity is to enable attribution of performance degradation to root causes in an environment where jobs can come and go on minute time scales. A group of HPC users at SNL provided bounds on acceptable overhead to be less than 1 percent slowdown and up to 1MB memory footprint per core.

We anticipated immediate benefits from this type of monitoring in the following areas:

User code optimization: Users could optimize or debug the per node footprint of their compute jobs by reviewing data on CPU, memory, and network usage.

Administrative debugging: Administrators could rapidly debug job slowness for a particular user or a whole cluster. For example, diagnosing site shared file system performance degradation could become as simple as looking across all systems sharing the resource for outlier access patterns.

HPC hardware procurement planning: In order to meet the needs of their user communities, those planning future procurements must size the number of nodes, number of compute cores, memory, and interconnect bandwidth of their next platforms. Summary usage statistics gathered over months on memory usage, IO router bandwidth, interconnect bandwidth and latency would give system architects solid data to use in deriving future system design requirements.

C. Similarities and Differences

The end goals for monitoring both NCSA's large scale Cray system and SNL's smaller scale capacity Linux clusters with respect to both metrics of interest and end use of the data are the same. However the difficulties in achieving them vary due to differences in scale, network topologies, network technologies, and availability of network related data. In particular the Cray HSN presented more challenges than Infiniband with respect to implementation of the LDMS RDMA transport and gathering metrics related to network performance.

While we utilize information from the `/sys` and `/proc` file systems to gather information for our commodity cluster interconnects, this was not originally an option for the Cray HSN. In support of the *Blue Waters* needs, over the past year Cray has developed a module to expose metrics aggregated from HSN performance counters to user space at a node level via their `gpcdr` module [10].

A userspace init script on each node at boot time configures the `gpcdr` module to report the HSN metrics described in Section II. A configuration file provides a definition of each metric. The script combines these definitions with runtime routing data to determine the combination of performance counters to use to generate each metric on that node. It configures the `gpcdr` module with those combinations. The `gpcdr` module then provides those metrics via files in the `/sys` filesystem.

The difficulties in implementation of the LDMS RDMA transport were due to the lack of readily available documentation and that early versions of the Cray Linux Environment (CLE) did not support the functionality we required from user space. The latter issue was resolved with the release of CLE4.1UP01 in late 2012. Direct Cray support resolved documentation issues.

IV. LDMS

LDMS is Sandia's solution to meet the needs and requirements just described. It was initially developed and deployed in Sandia's production HPC environment. NCSA also had needs and requirements that, due to the scale and proprietary nature of their Cray XE/XK platform *Blue Waters*, could not be met using commodity monitoring tools such as Ganglia (see Section IV-E). Collaboratively, Sandia, NCSA, and Cray extended and deployed LDMS on the 27,648 node *Blue Waters* system. This section describes the basic LDMS infrastructure, its functional components, configuration details that give it flexibility, details of both memory and CPU footprint on both compute nodes and aggregators, and finally a brief comparison with other monitoring tools in which we highlight differentiating features which address our particular needs.

A. Overview

LDMS is a distributed data collection, transport, and storage tool that supports a wide variety of configuration options. A high level diagram of the data flow is shown in Figure 1. The three major functional components are described below. The host daemon is the same base code in all cases; differentiation is based on configuration of plugins for sampling or storage and on configuring aggregation of data from other host daemons.

Samplers run one or more sampling plugins that periodically sample data of interest on monitored nodes. The sampling frequency is user defined and can be changed on the fly. Sampling plugins are written in C. Each plugin defines a collection of metrics called a *metric set*. Multiple plugins can be simultaneously active. By default each sampling plugin operates independently and asynchronously with respect to all others. Memory allocated for a particular metric set is overwritten by each successive sampling and no sample history is retained within a plugin or the host daemon.

Aggregators collect data in a pull fashion from samplers and/or other aggregators. As with the sampler, the frequency of collection is user defined and operates independently of other collection operations and sampling operations. Distinct metric sets can be collected and aggregated at different frequencies. Unlike the samplers, the aggregation schedule cannot be altered once set without restarting the aggregator. The number of hosts collected from by a single aggregator is referred to as the *fan-in*. The maximum fan-in varies by transport but is roughly 9,000:1 for the socket transport in general and for the RDMA transport over Infiniband. It is $> 15,000 : 1$ for RDMA over Cray's Gemini transport. Daisy chaining is not limited to two levels and multiple aggregators may aggregate from the same sampler or aggregator *ldmsd*. Fan-in at higher levels is limited by the aggregator host capabilities (CPU, memory, network bandwidth, and storage bandwidth).

Storage plugins write in a variety of formats. Currently these include MySQL, flat file, and a proprietary structured file format called Scalable Object Store (SOS). The flat file storage is available in either a file per metric name (e.g. "Active" and "Cached" memory are stored in 2 separate files), or a Comma Separated Value (CSV) file per "metric set" (e.g., $\{$Active, Cached$\}$ memory information set is stored in a single file). The frequency of storage is dependent on the frequency with which valid updated metric set data is collected by an aggregator that has been configured to write that data to storage. Collection of a metric set whose data has not been updated or is incomplete does not result in a write to storage in any format.

B. LDMS Components

This section describes, in greater detail, the components that support the functionality just described. It also describes typical configuration options used in deployment of these components.

ldmsd: The base LDMS component is the multi-threaded *ldmsd* daemon which is run in either sampler or aggregator mode and supports the store functionality when run in aggregator mode. The *ldmsd* loads sampling, transport, and storage plugin components dynamically in response to process-owner issued configuration commands. Access is controlled via

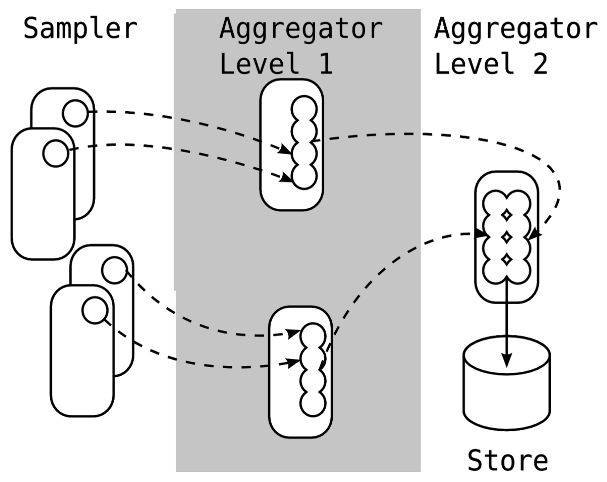


Fig. 1. In the LDMS use case depicted, rounded rectangles represent *ldmsd*s while circles depict metric sets. The shaded region separates levels in the hierarchy (samplers, first level aggregators, second level aggregators left to right respectively). Arrows depict the direction of data flow. The second level aggregator writes to local disk.

permissions on a UNIX Domain Socket. At run time *ldmsd* is also configured to listen for incoming connection requests on a socket. While the request and configuration information use the UNIX socket transport, a stable connection for data transfer is established using a transport protocol specified by the user. The same transport plug-in is used to manage all connections to a *ldmsd* underlying type. Currently TCP sockets (*sock* transport plugin), Infiniband/iWARP RDMA (*rdma* transport plugin), and Gemini RDMA (*ugni* transport plugin) interconnect types are supported.

Samplers: Each sampling plugin combines a specific set of data into a single *metric set*. A few of the Lustre filesystem data entries from a metric set are shown below:

```
U64 17588842          dirty_pages_hits#stats.snx11024
U64 27547858          dirty_pages_misses#stats.snx11024
U64 1551040415605     read_bytes#stats.snx11024
U64 111681033094      write_bytes#stats.snx11024
U64 33185713          open#stats.snx11024
U64 33459578          close#stats.snx11024
```

An *ldmsd* instance can support many such sampling plugins all of which run from a common worker thread pool whose size is user defined at *ldmsd* run time. Libevent (2.0 or newer) is used to schedule sampling activities on user-defined time intervals. Sampling plugins have options for wall-time driven (synchronous) or interval driven (asynchronous) operation. More than one sampling plugin hosted by a particular *ldmsd* may sample a given metric value; avoiding such inefficiencies is up to the user. Finally, *ldmsd*s hosting one or more sampling plugins are generally referred to as *samplers* in this document as that is their main function. There are two chunks of contiguous memory associated with each metric set. First is the metadata describing the elements of the data chunk (name, user-defined component ID, data type, offset of the element from the beginning of the data chunk) and a metadata generation number (MGN) which is modified when the metadata is modified. Second is the chunk of sampled data values, which includes the MGN, the current data, a data generation number (DGN) incremented as each element is updated and a *consistent* flag. The MGN enables a consumer to determine if the metadata it has stored matches that associated with the data. The DGN enables a consumer to discriminate between new and stale data. The *consistent* status flag tells a consumer if the data in the metric set all came from the same sampling event.

Aggregators: LDMS daemons configured to collect metric sets from sampler and/or other aggregator *ldmsd*s (See Figure 1) are called *aggregators*. The smallest unit of collection is the metric set. In order to collect from either a sampler or another aggregator, a connection is established from the collecting aggregator to the collection target. LDMS incorporates mechanisms to enable initiation of a connection from either side in order to support asymmetric network access. One or more metric sets are defined for each connection along with associated collection interval and transport information. Multiple connections may be established between an aggregator and a single collection target. This supports different metric sets having different sampling frequencies. After connection setup, only the data portion of a metric set is pulled from a target in order to minimize network bandwidth. The data portion is roughly 10% of the total set size.

Each data collection is performed by a worker thread from a common worker thread pool. Typically the worker thread pool is no larger than the number of CPU cores available on the host machine. A separate thread pool is configured to perform connection setup. The connection thread pool was incorporated to mitigate collector thread starvation that could occur on large scale systems such as *Blue Waters* while trying to set up a large number of connections that might get hung in timeout on problem nodes.

Aggregators also have the facility to have connections defined as *standby*. This enables an aggregator to maintain connections to a set of samplers that it will not actually pull data from unless it is notified that the aggregator that was supposed to handle those samplers (their primary) is down. This is desirable for large scale systems that would lose a lot of data between a primary aggregator going down and another starting up. Note that there is currently no internal mechanism for a standby aggregator to detect a primary has gone down automatically. This is accomplished either manually or by an external watchdog program that provides notification.

Storage: Storage plug-ins are run on aggregators and handle the task of writing data from metric sets to stable storage with the defined storage format type. There is a dedicated thread pool to flush data to stable storage. The flush frequency depends on the number of metric sets collected by the aggregator. In all formats a time stamp and user configured component identifier (associated with each metric) is also written. Because all *ldmsd*s in a system operate asynchronously, it is possible that from one aggregator collection to the next a metric set has not been updated or, though highly unlikely, that a collection occurs during the time a sample is being written on the sampler of interest. If either or both of these cases occur, the old or partially modified metric set is not written to storage and collection is scheduled for the next collection interval. The DGN and *consistent* status flag are used to detect these conditions.

Figure 2 provides a data flow diagram, annotated with the associated API's, for each of the components just described.

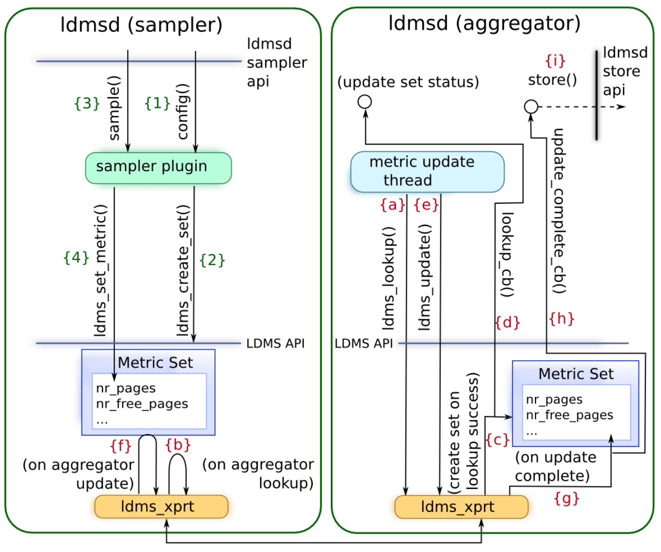


Fig. 2. Diagram of LDMS data flow with internal calls and API for the pull model. {1} - {4} are sampler flows, which are independent from aggregator flows {a} - {i}. Sampler flow starts from *ldmsd* load and configure a sampler plugin {1}, and the plugin consequently creates an LDMS set {2}. Afterward, *ldmsd* periodically tells the plugin to sample metric values from the data source {3}. On each sample call, the plugin reads metric values from data sources and updates the values in the metric set accordingly {4}. Aggregator flow starts with an update thread in *ldmsd* aggregator performing a set lookup {a}. The LDMS library on the sampler end will reply to the lookup request with the metric set information or error, if the set was not found {b}. If the lookup returns an error, the metric update thread will keep performing lookup in the next update loop (back to {a}). After lookup completes, a local metric set is created as a mirror of the remote metric set {c}, and a `lookup_cb()` function is called to notify *ldmsd* {d}. In the next update loop, the metric update thread will do the data update {e}. If the transport is RDMA over IB or UGNI, the data fetching {f} will not consume CPU cycles. On data update complete, the metric values in the aggregator metric set are updated {g}, and `update_complete_cb()` is called to notify *ldmsd* {h}. *ldmsd* optionally stores the updated data if a store is configured {i}.

C. Common Configuration Options

While a complete description of all configuration options is beyond the scope of this paper, we present those used in a typical deployment.

A sampler is created by running a *ldmsd* and configuring sampler plugins. Configuration options for *ldmsd* samplers are: transport type, port number, Unix Domain Socket path and name, debugging log file path, and plug-in name. Common control flags for an individual sampler are: component ID, metric set name, sampling interval, and optionally synchronous and offset. Some samplers have additional control options.

An aggregator is created by running a *ldmsd* and configuring target connections, metric sets to collect, and collection frequencies. Note that many metric sets can be collected on a single connection. The configuration options in this case for the base *ldmsd* are: transport, port number, Unix Domain Socket path and name, log file path and name, amount of memory to allocate for metric sets, and number of worker and connection threads to create. Configuration options for adding target metric sets to collect are: target host, connection type, transport, port, metric set(s), and period of collection. Note that transport and port number must match those of the target but not collection and sampling periods. Optionally redundant collection connections can be defined for fast failover purposes. Collection can be defined to be *synchronous*. Note that *synchronous* operation refers to an attempt to collect (or sample) relative to particular times as opposed to relative to an arbitrary start time.

Storage is configured by creating and configuring an aggregator and then configuring a storage instance. Each store type has specific configuration parameters. The following parameters apply to the CSV storage type: store type, store path and file name, metric set to store, optionally write header to separate file. Storage may be specified at a {producer, metric name} granularity, though the typical use case is to specify just the metric set. In the typical case, the corresponding data from all producers is stored.

D. Resource Footprint

In this section we provide a general description of storage, memory, CPU, and network footprints for both samplers and aggregators on a production capacity Linux cluster at SNL and the Cray XE/XK *Blue Waters* system at NCSA. The actual resource usage for any deployment will depend on sampler plugin mechanisms, sampling and collection frequencies, metric set size, and storage configuration.

LDMS requires less than four megabytes of file system space to install and less than two megabytes of memory per node for samplers to run in typical configurations. LDMS is run per node, not per core but can be bound to a core using a variety of platform specific mechanisms (e.g., numactl).

Memory registration of a few kilobytes is needed for RDMA-based transport of locally collected data. Aggregation nodes require a similar amount of registered memory per connection. Additionally, a sampler requires storage for both data and metadata for each metric set, and an aggregator requires this for each metric set it will be collecting plus some additional memory for managing this data. On *Chama* the metric sets consume 44kB and on *Blue Waters* 24kB. A custom memory manager is employed to manage memory allocation.

Data storage requirements are modest for the CSV storage plug-in. A day's worth of data for 467 metrics per node on SNL's 1296 node *Chama* system with 20 second sampling/collection intervals is about 27GB (10TB raw or 2.5TB compressed for a year). On *Blue Waters* a day's worth of raw data stored to CSV is about 43GB (16TB raw or 4TB compressed for a year).

CPU utilization of samplers on compute nodes is typically a few hundredths of a percent of a core in both deployments at a sampling period of 1 second. First level aggregators on *Chama*, each hosting 7 metric sets (467 metrics) from 156 samplers with collection intervals of 20 seconds, utilize about 0.1 percent of a core and 33MB of memory. The second level aggregator, aggregating from 8 first level aggregators consumes about 2 percent of a core and 150MB of memory. On *Blue Waters* each aggregator actively collects 1 metric set with 194 metrics from 6912 nodes and maintains standby connections and state to another 6912. Here each aggregator is allocated 900MB of RAM and consumes about 100% of a core.

On *Chama* the total data size for all 7 metric sets being collected over the IB network from each node is 4kB. This translates into an additional 5MB transiting the IB fabric every 20 seconds (note that this traffic is spread across the whole system not on a single link). On *Blue Waters* it is 44MB.

E. Related Work

1) Monitoring systems: Numerous tools exist for system monitoring with Ganglia [11], in particular, in widespread use. Ganglia has a number of features that make it unsuitable even for general monitoring of large scale HPC systems. The project info page [11] only claims scalability to 2000 nodes. It has a considerable number (7) of installation dependencies, most of which are not typically installed on large scale capability platforms such as the Cray XE/XK platforms.

In addition, Ganglia and Nagios [12] typically target larger collection intervals (10's of seconds to 10's of minutes) and many fewer metric variables than LDMS with data used for general monitoring (Ganglia) and failure alerts (Nagios) rather than for system and application resource utilization analysis. Neither supports subsecond collection frequencies. Frequent collection using Ganglia can have significant impact. On *Chama* we found the collection time per metric for Ganglia vs. LDMS from `/proc/stat` and `/proc/meminfo` to be about two orders of magnitude greater (i.e. 126 usec per metric for Ganglia vs. 1.3 usec per metric for LDMS). Ganglia includes both data and its description (metadata) at each transmission but user-defined thresholds are typically set to reduce the amount of data sent. This thresholding can reduce behavioral understanding if set too high. Ganglia stores to RRDTool [13] which ages out data and thus requires a separate data move if long term storage is desired.

Vendor specific implementations (e.g., Cray's SEDC [14]) for system monitoring have the potential for better performance than general purpose tools. However, these may not be open source and may not support user-addition of samplers and/or general output formats.

The open-source tool most similar to LDMS is Performance Co-Pilot (PCP) [15], with some overlap to LDMS in design philosophies. PCP supports a pull-only model, a single-hop-only transport, a single archive format used by its display and analysis tools, and a large variety of data acquisition plugins. LDMS supports C language performance-oriented plugin storage and transport APIs, but currently provides no GUI tools.

2) Profiling systems: Collectl [16] and `sar` [17] are single host tools for collecting and reporting monitoring values. Neither include transport and aggregation infrastructure. Both can continuously write to a file or display; collectl can also write to a socket. While both have command line interfaces as well, use of these by applications would require an `exec` call and parsing of output data by the application, as opposed to a library interface. Only collectl supports subsecond collection intervals.

LDMS is not intended as a substitute for profiling tools, such as OProfile [18] or CrayPat [19]. LDMS takes a complimentary approach by collecting system and application resource utilization information as a continuous baseline service. Additionally, since LDMS samplers can be configured on-the-fly, independent multi-user support can be configured at run-time to provide higher fidelity insight on a per user/job basis.

3) Performance data transport: Few HPC-oriented libraries exist to provide scalable, fault tolerant, transport and aggregation of frequent small, fixed size, binary messages, although numerous libraries support Map-Reduce [20] based on the TCP transport. MRNet [21] is a tree-based overlay network which can be used as the transport for tools built upon it. Its intended use is data reduction at the aggregation points as opposed to transport of all the data. Currently there is no support for RDMA. The tree setup does not support multiple connection types or directions. In contrast, LDMS uses an interval driven polling and aggregation process (above the level of the network transport plugins) which eliminates duplicate messages, bypasses and later retries non-reporting hosts, and allows fail-over support for non-reporting aggregators.

F. Blue Waters - Deployment

Figure 3 depicts the high level configuration on *Blue Waters*. The sampler daemons are installed in the boot image for compute nodes and are started automatically at boot. Cray service nodes launch aggregator *ldmsd*s from the shared root partition. Four service nodes were selected to serve as aggregators in a manner that evenly distributes traffic across the slowest dimension of the high speed network. Their data store files are linked to a named pipe that is used by syslog-ng to forward the data to NCSA's Integrated System Console (ISC) database. ISC is a single point of integration of nearly all system log, event, and state data that allows us to take actions based on information that spans subsystems. ISC both archives the data for future investigations as well as stores the most recent 24 hours of node metrics for live queries.

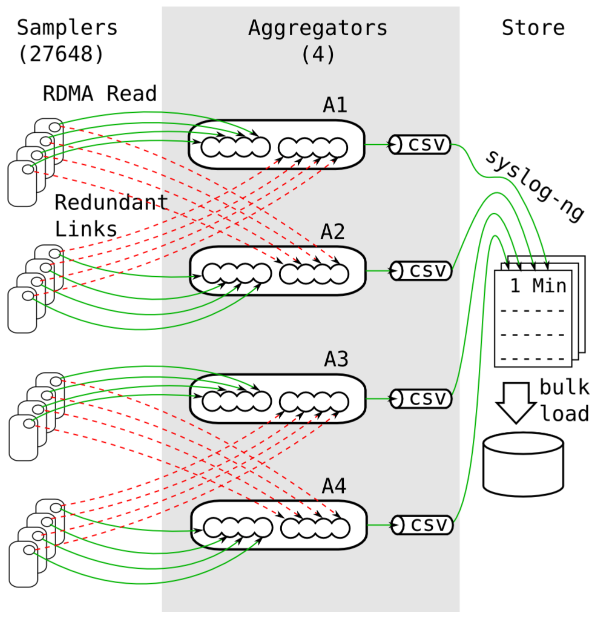


Fig. 3. LDMS configuration on *Blue Waters*. This includes redundant connections (dashed arrows) to each sampler *ldmsd* for fast failover capability. Rather than writing directly to local stable storage, the aggregators each write a CSV file to a local named pipe. Data from this pipe is forwarded by syslog-ng to the ISC where it is bulk loaded into a database.

On *Blue Waters*, a sampler collects one custom dataset whose data comes from a variety of independent sources, including HSN information from the `gpcdr` module, lustre information, LNET traffic counters, network counters, and cpu load averages. In addition we derive information over the sample period, including percent of time stalled and percent bandwidth used. The latter uses estimated theoretical maximum bandwidth figures based on link type. In production, we currently sample at 1 minute intervals.

G. SNL Capacity Systems - Deployment

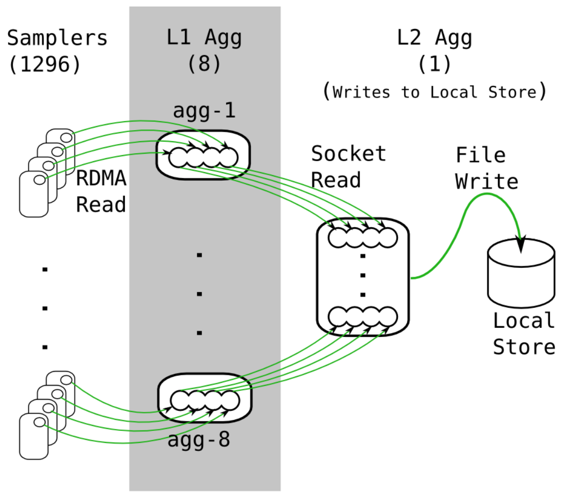


Fig. 4. LDMS configuration on *Chama*. RDMA is used for data reads between the samplers and first level aggregators while socket connections are used between first and second level aggregators. The second level aggregator aggregator writes data in CSV format to local disk.

Figure 4 depicts SNL's LDMS deployment on *Chama*. We package LDMS software in RPMs which supplement the TOSS2 [22] distribution. These RPMs provide libevent 2.0 [23], LDMS, and our default set of storage and sampler plug-ins in relocatable library form to simplify installation in the image directories served to diskless nodes. At boot time, initialization scripts launch the hierarchy of samplers and aggregators. We tailor machine-specific configuration files to ensure that the daemons connect efficiently; LDMS does not support autodiscovery-based configuration.

We deploy LDMS samplers on the compute nodes and aggregators on service nodes of *Chama*, using the RDMA transport to minimize interference with computations. A disk-full server runs a second level aggregator configured with the CSV storage plug-in and the socket transport, pulling data from the service nodes.

Users seeking additional data on these systems may run another LDMS instance configured to use their specified samplers and a different network port as part of their batch jobs. The owner of an LDMS instance controls it through a local UNIX Domain socket.

On *Chama*, a sampler collects 7 independent metrics sets from sources in `/proc` and `/sys` including memory, cpu utilization, lustre information, nfs information, ethernet and IB traffic. In production, we sample at 20 second intervals.

V. Impact Testing and Analysis

We ran both actual applications and test codes concurrently with LDMS to assess the impact of additional operating system (OS) noise and network contention on applications' execution times. The effect of OS noise on application execution times on Linux clusters has been well studied (e.g., [24]–[26]). Ferreira et. al. [26] have shown that various OS noise patterns for a consistent and representative HPC "noise amount" can result in slowdowns of up to several orders of magnitude for representative HPC applications at large scale (up to 10K nodes). The LDMS experiments for this study were conducted using several realistic sampling/collection periods (i.e., 1, 20, and 60 seconds) to check for OS noise impact on application run times.

A. Blue Waters

We ran a variety of short benchmarks across the machine. PSNAP is discussed in Section V-A1 and Figure 5; all others are described in subsequent subsections and summarized in Figure 6.

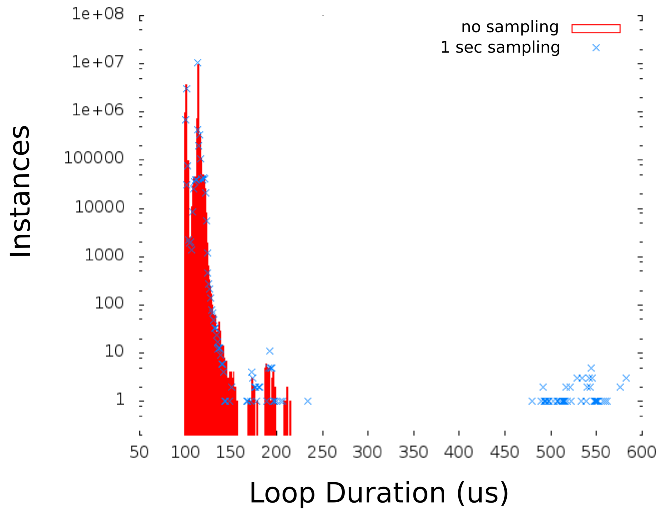


Fig. 5. PSNAP results: Histogram of occurrences vs. loop time (μs) with 1 second sampling data (Xs') compared to none (red boxes).

1) PSNAP: PSNAP [27] is an OS and network noise profiling tool which performs multiple iterations of a loop calibrated to run for a given amount of time. On an unloaded system, variation from the ideal amount of time can be attributed to system noise. We ran PSNAP with and without monitoring in order to determine the additional impact of the monitoring. PSNAP was run without its barrier mode, making the effects on each node independent. 32 tasks per node were executed with a 100 μs loop.

Figure 5 compares monitored and unmonitored results. The one second sampling interval shows an additional $50-60$ events, out of 16 million total, out in the tail with an additional delay of $375-475$ μs. This is in line with the expected delay caused by the known sampling execution time of order 400 μs and the expected number of occurrences given the execution time of around a minute and the sampling period of 1 second.

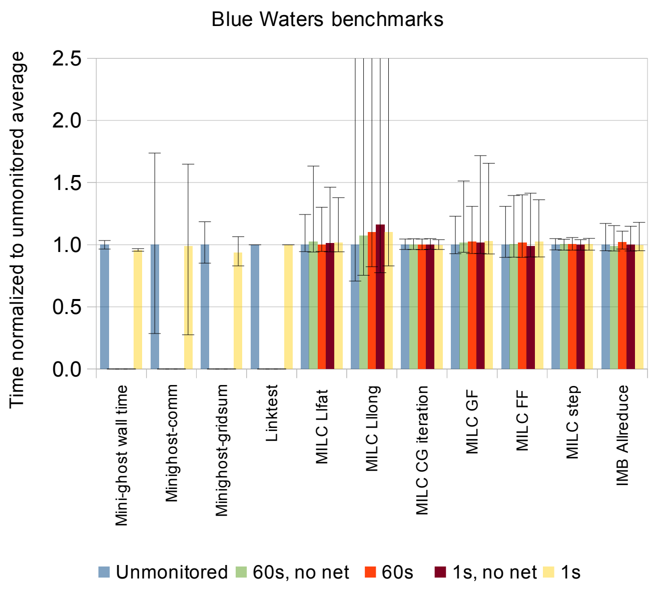


Fig. 6. Benchmark Variation Under LDMS variations. The 'no net' variations disable aggregation and storage to differentiate impact due to changed network behavior. Error bars show the range of observations.

While an application process running on the node can only be impacted during the sampling time, an MPI application might wait upon processes on other nodes and therefore random sampling across nodes might result in greater impact. The synchronized sampling feature (Section IV-B) has the additional benefit that all sampling across the nodes can be coordinated in time, thereby bounding the number application iterations affected.

2) MILC: MILC [28], [29] is a large scale numerical simulation used to study quantum chromodynamics on a wide variety of platforms. It is sensitive to interconnect performance variation. The test problem application was run on 2744 XE nodes with a topology aware job submission to minimize congestion. It uses a 64B Allreduce payload in the Conjugate Gradient (CG) phase with a local lattice size of 6^4. Overall performance is a combined function of all phases, with overall performance most dependent on the CG phase which has many iterations per step. Within phases, performance variations of the average time do not consistently increase with expected increasing impact of each LDMS configuration. Even when variation of the average is measurable, the variation is within the wide range of observed values. No statistically significant impact was observed.

3) LinkTest: Cray has developed an MPI program that measures the individual link performance within a job. For this test we measure the extreme cases of unmonitored and monitoring at one second intervals. We used 10,000 iterations of 8kB messages. This gives us multiple collections of data per link test. The unmonitored result is 1.74278 milliseconds per packet and the monitored time is 20 nanoseconds shorter. The difference is not statistically significant.

4) MiniGhost: MiniGhost [30] is used for studying only the communications section of similar codes. Our instrumented version [31] of MiniGhost reports total run time, time spent in communication, and time spent in a phase which includes waiting at the barrier (GRIDSUM). We chose input that yields 90 second run time on $8,192$ nodes in order to determine effects on communication. Because of the short runtime, three repetitions of the code were made at the extremes: unmonitored and sampling at one second intervals. Each repetition was launched on the same nodes and an internally computed ordering based on the known communication pattern of the application was used. There was no negative impact in any measure when using LDMS at the 1 second collection interval.

5) IMB: We tested the Intel MPI benchmark (IMB) for MPI_AllReduce on a set of 2744 nodes. This node set was topology optimized for maximum network performance. This test used a 64B payload and 24 tasks per node. Overall, there is not a correlating impact with the LDMS variants.

B. *SNL Capacity System*

We assessed the impact of LDMS on different applications. Three conditions were considered: no LDMS (NM - unmonitored) sampling on the node at 20 second intervals (LM - low monitoring) and sampling on the nodes at one second intervals (HM - high monitoring). We ran the applications as a consistent ensemble of simulations to nearly fully utilize *Chama*. Two Sierra Low Mach Module: Nalu (Nalu, Section V-B1) simulations utilizing 1,536 and 8,192 PE, two CTH (Section V-B3) simulations utilizing 1,024 and 7,200 PE, and two Sierra/SolidMechanics implicit (Adagio, Section V-B2) simulations utilizing 512 and 1,024 PE. Each one of these simulations was forced to use a consistent set of nodes to assess variability and each ensemble was simulated three times. In addition PSNAP was run on 19200 processing elements under NM, HM, and a special case of HM with only half the samplers.

A dedicated application time (DAT) was used to perform this study. During this time, the only simulations running on the machine were ours. However, *Chama* shares its Lustre file system with another cluster, which may have caused contention with our applications for these resources.

Our application performance results, other than those of PSNAP, are summarized in Figure 7 and explained below.

1) Sierra Low Mach Module: Nalu: Nalu is an adaptive mesh, variable-density, acoustically incompressible, unstructured fluid dynamics code. Nalu is fairly representative of implicit codes using Message Passing Interface (MPI) and demonstrates good-to-ideal weak scaling trends [32]. The simulation performed is a jet in crossflow [33]. Preliminary traces collected for 5 time steps of the test configuration show that 47.5% of its time is spent in computation, 44% of its time on MPI "sync" operations, and the last 8.5% on other MPI calls. We expect Nalu to be sensitive to both node and network slowdown.

Nalu has built-in timers of specific parts. The `mpiexec` invocation was wrapped by `time` to provide a global timer of the simulation. Although all runs used the same set of nodes, the cost of major internal phases varied widely for the 8,192 PE simulations, particularly for the continuity equation a 200 second spread is seen in the unmonitored runs. This variation has not been observed with identical simulations at this scale

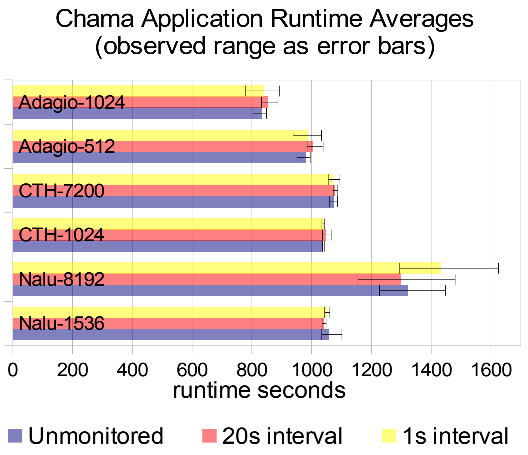


Fig. 7. Application Performance Variation. All three applications with core counts indicated were run as a job ensemble multiple times for each LDMS configuration. In all runs the rank to host/core mappings were identical to eliminate variation due to configuration differences.

on Cielo and is likely caused by operating system noise; a root cause analysis is currently being performed. Variations in the cost of numerous shorter phases only sometimes cancel those large variations. The variation present within these 8,192 PE simulations dwarfs any speedup or slowdown caused by the LDMS monitoring. Ergo, LDMS monitoring appears to have *no practical impact* on the run time of these Nalu simulations.

2) Adagio: Adagio is a Lagrangian, three-dimensional code for finite element analysis of solids and structures built on the Sierra Framework [34], [35]. The model used studies the high velocity impact of a conical war-head meshed with about half a million elements. We use Adagio to investigate the impact of LDMS monitoring on the run time because Sierra applications consume a large fraction of the node-hours on Sandia HPC clusters. Restart files are dumped to the high speed Lustre I/O subsystem. The combination of mesh files, results files, history files, restart files, and a log file pose storage requirements approaching terabytes and typically require the system to maintain throughputs in hundreds of gigabytes/sec. A large fraction of the computation time is in the contact mechanics which stresses the communications fabric. The combination of the computations, communications and I/O characteristics make this a good application to investigate the impact of LDMS. As can be seen from Figure 7, there is no appreciable impact from LDMS compared to the noise in this data.

3) CTH: CTH is a multi-material, large deformation, strong shock wave, solid mechanics code that uses MPI for communications [36]–[40]. The CTH (version 10.3p) simulation we benchmarked is a 3D shock physics problem with adaptive mesh refinement (AMR) typical of those run at Sandia: the three-dimensional evolution of a nuclear fireball [41]. In general, processors exchange large messages (several MB in size) with up to six other processors in the domain, with a few small message MPI_Allreduce operations. CTH is sensitive to both node and network slowdown [42], [43]. We studied CTH interaction with LDMS due to its heavy use in SNL HPC environments and its sensitivity to perturbation on both nodes and network. The 1,024 core simulation is set to execute 600 time steps while the 7,200 core simulation is set to execute 1,200 time steps, targeting 18 minutes of sustained run time. LDMS monitoring appears to have *no effect* on the run time of these CTH jobs.

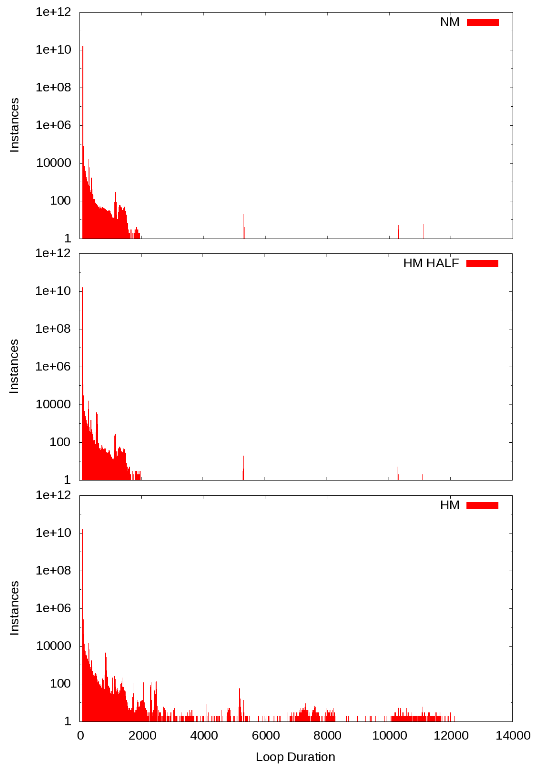


Fig. 8. Histogram of instances vs loop duration (us) for no monitoring (NM)(top), 1 sec sampling with samplers contributing about half the metrics (HM HALF); and and 1 sec sampling interval for all samplers (HM) (bottom).

4) PSNAP: PSNAP was run on *Chama* under the conditions of: no monitoring (NM) , LDMS sampling on the nodes at 1 sec intervals with samplers contributing about half the metrics (HM HALF), and all samplers at 1 sec intervals (HM). $1M$ iterations of a 100 μs loop on 1200 nodes were used in both cases. PSNAP was not run in barrier mode. Histograms of loop instances vs loop duration (us) for each case are presented in Figure 8.

While NM and HM HALF are comparable, there are substantially more elements in the tail in HM case. Sampling impact is expected to be subject to the number of samplers and the time a sampler spends in sampling.

VI. Characterizations

While detailed data analysis is not a part of this work, in this section we present some initial views of the data of interest that motivate its collection. Over a 24 hour period for *Blue Waters* the dataset has 40 million data points per metric or 7.7 billion data points total. For *Chama* the data set is 5.6 million data points per metric or 2.6 billion data points total. We show that for data of this size, features of interest can be discerned even in simple representations.

A. *Blue Waters*

For *Blue Waters*, we specifically target presentation and features of data that must be considered system-wide in order to understand the impact on an application's performance due to the distributed network topology. This is the first insight into this type of system information at this scale and fidelity.

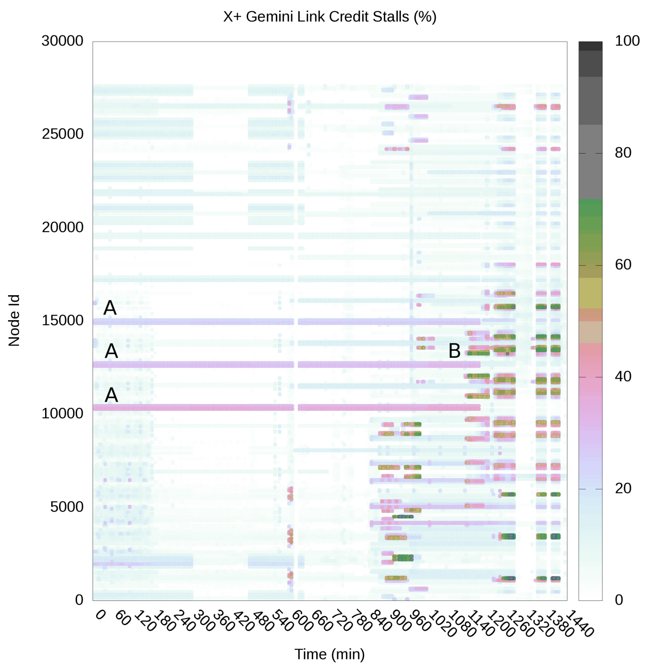
As mentioned in Section II, an application's performance may be degraded by an application's communication pattern resulting in traffic routed through significantly congested links. The routing algorithm between any 2 Gemini is well-defined; thus the links that are involved in an application's communication paths can be statically determined. Viewing values in both the node and Gemini spaces can thus give insight into which applications may be impacted by network congestion.

Information is thus presented a) in 2D layout of values per node over a 24 hr period and b) for a single system snapshot in time shown in the 3D Gemini mesh coordinate space. In order that high value features may stand out better: a) in the 2D plots, plotted points are larger than the natural point size (i.e., there are not 27K pixels plotted in the y dimension) and thus may overlap, and b) in the 3D plots data point size is proportional to data values. In both cases quantities under a threshold value of 1 have been eliminated from the plots.

1) HSN Link Stalls: One of the metrics used to identify network congestion is time spent in *stalls*. The Gemini network utilizes credit-based flow control. When a source has data to send but runs out of credits for its next hop destination, it must pause (*stall*) until it receives credits back from the destination. Stalls when sending from one Gemini to another are referred to as *credit stalls*. We collect stall counts in each of the link directions and also derive the percent of time a link spent stalled on output link resources.

Over the 24 hour period considered here, the maximum value of percent time spent stalled over each 1 minute sample interval obtained was 85 percent in the $X+$ direction. Stalls for each node in this direction over time are presented in Figure 9 (top). 2 nodes share a Gemini and thus have the same value, however, since application analysis may be in terms of nodes, we report this information for each node separately. It can be seen that significant congestion can persist for many hours: for example, significant durations of data values in the $30-40+\%$ range for up to 20 hours (label A), and of data values in the $60+\%$ range for up to 1.5 (e.g., label B) hours.

A system snapshot at the time of the maximum value of percent time stalled is shown in terms of the X, Y, Z network mesh coordinates in Figure 9 (bottom). The mesh topology is a 3D torus with dimensions 24x24x24. Thus the entire network is shown within the bounding box. Each data point represents a Gemini. The maximum is in the group in the rear of the figure. Because of the toroidal connectivity, this group wraps in X and connects with the group in the left at the same value of Z (circled, labeled C). Another high value region is part of a group in the XY plane that extends into Z (labeled D). For this quantity, features naturally have extent in the X direction. Characterization of such features through time can give insight into how a congested link or region could have cascading effects. Although there can potentially be 13824 data points in this figure, the areas of high congestion are easily discerned.



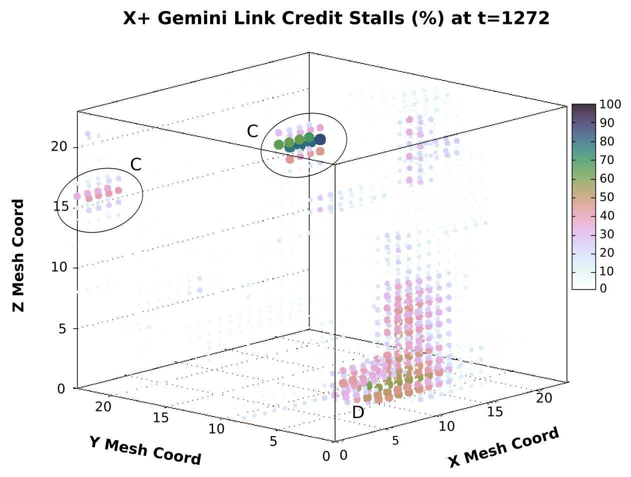


Fig. 9. Time (%) spent in credit stalls in the X+ direction (per node Gemini) at 1 minute intervals over a 24 hours period (top) for *Blue Waters*. A system snapshot showing regions of congestion in the 24 x 24 x 24 Gemini 3D torus coordinate space (bottom). Torus wrap-around not shown. Due to the distributed Gemini network topology, long durations of high stall levels can degrade the performance of multiple applications sharing those links. In the node view (top) some Gemini are seen to spend 30-40 percent of time in stalls for up to 20 hours (label A). A maximum (85%) is shown in the snapshot (bottom, label C rear). Labeled regions further described in Section VI-A.

2) Bandwidth Used: A related but different quantity reflective of network congestion is percent of theoretical maximum bandwidth used (Section IV-F). The theoretical maximum is dependent on the link media type. The highest value over the course of the same day is in the $Y+$ direction at 63 percent. This is shown in Figure 10. Note the value is significantly higher than typically observed values in the system over this time and is readily apparent in the figure.

3) Lustre Opens: Figure 11 illustrates how observing system wide information can provide a simple means to determine

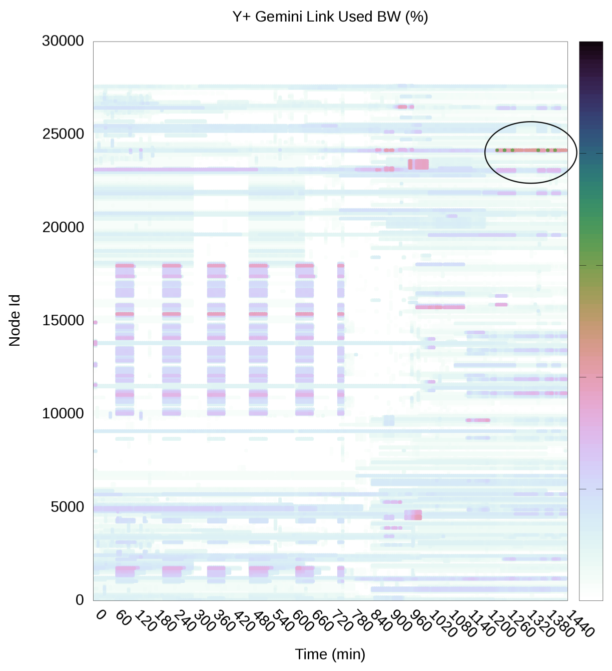


Fig. 10. Percent bandwidth used in the Y+ direction per node for each 1 minute interval over an entire day. Maximum bandwidth used (63 percent) is apparent (circled).

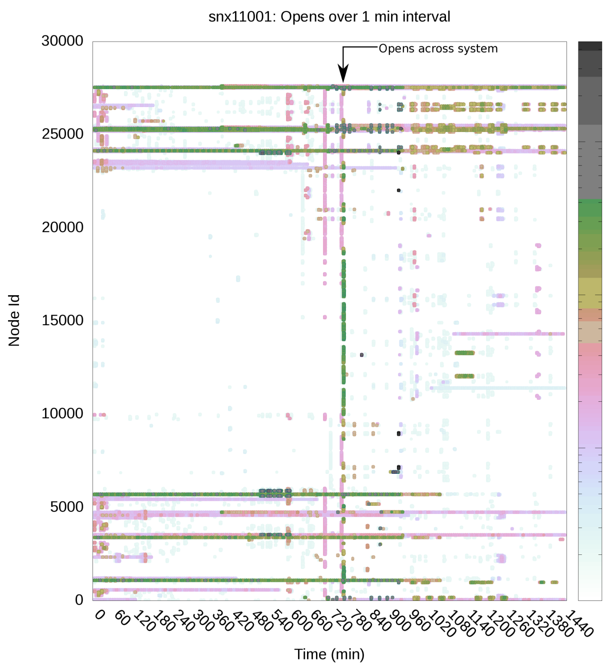


Fig. 11. Lustre opens. Significant opens across system indicated by arrow. Horizontal lines are indicative of significant and sustained level of opens from a few nodes.

what system components over what times are consuming particular resources. In this figure it can be seen from the horizontal lines that certain hosts are performing a significant and sustained level of Lustre "opens". These can be easily correlated with user and job. The vertical lines show times when Lustre "opens" occur across most nodes of the system. Whether this is indicative of a large user job or a system wide problem can be determined by seeing if the nodes involved are from a single job or span many.

B. *SNL Capacity System*

On *Chama*, in addition to creating system views we combine the system information with scheduler data to build application profiles. A profile for a 64 node job terminated by the OOM killer is shown in Figure 12. Total per node memory available is 64G. Memory imbalance and change in resource demands with time are readily apparent. This type of information can give the user insight into the application's demands under production conditions and guide better application-to-resource mapping where there is imbalance.

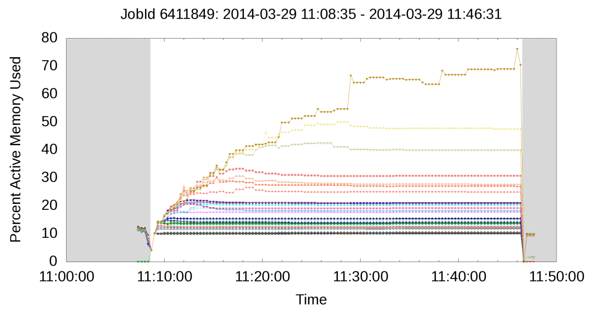


Fig. 12. Application profiles are built from LDMS and scheduler data. Active memory for a 64 node job terminated by the OOM killer is shown. Each colored line is a different node's values (Node legend is suppressed). Grey shaded ares are limited pre and post job times in order to verify the state of the nodes upon entering and exiting the job. Imbalance and change in resource demands with time are apparent.

VII. Conclusion

We have described our motivations for continuous whole system monitoring of network, file system and other resource related metrics. We have described our Lightweight Distributed Metric Service framework for performing this task and demonstrated its suitability for large-scale HPC environments, particularly with regard to design and deployment features, scalability, resource footprint and system and application impact. We have presented representations of system and application data from both NCSA's *Blue Waters* Cray XE/XK platform and SNL's *Chama* cluster in which features of interest can be discerned. In particular, the unique suitability of LDMS for large-scale HPC environments has enabled the first insight at this size and fidelity into continuous HSN performance counter data on *Blue Waters*.

VIII. Future Work

Future work involves deriving insight from the data obtained here, particularly with regard to performance issues that require consideration of the full system data. The large dataset size due to the necessary fidelity of the data presents challenges in visual representations and analysis. On the *Blue Waters* system we seek to discover, and perhaps enable mitigation of, performance issues due to the shared network.

References

[1] N. Wolter, M. O. McCracken, A. Snavely, L. Hochstein, T. Nakamura, and V. Basili, "Whats Working in HPC: Investigating HPC User Behavior and Productivity," *CTWatch Quarterly*, vol. 2, no. 4A, pp. 9–17, 2006.

[2] A. Bhatele and L. Kale, "Benefits of Topology-aware Mapping for Mesh Topologies," *Parallel Processing Letters*, vol. 18, no. 4, pp. 549–566, 2008.

[3] A. Verma, P. Ahuja, and A. Neogi, "Power-aware Dynamic Placement of HPC Applications," in *Proc. 22nd Annual International Conference on Supercomputing*, ser. ICS '08. New York, NY, USA: ACM, 2008, pp. 175–184. [Online]. Available: http://doi.acm.org/10.1145/1375527.1375555

[4] W. Denzel, J. Li, P. Walker, and Y. Jin, "A Framework for End-to-End Simulation of High-performance Computing Systems," *SIMULATION*, vol. 86, no. 5-6, pp. 331–350, 2010. [Online]. Available: http://sim.sagepub.com/content/86/5-6/331.abstract

[5] A. Peña, R. Carvalhorea, J. Dinan, P. Balaji, R. Thakur, and W. Gropp, "Analysis of Topology-dependent MPI Performance on Gemini Networks," in *Proc. 20th European MPI Users' Group Meeting*, ser. EuroMPI '13. New York, NY, USA: ACM, 2013, pp. 61–66. [Online]. Available: http://doi.acm.org/10.1145/2488551.2488564

[6] "The Gemini Network," Aug 2010. [Online]. Available: http://wiki.ci.uchicago.edu/pub/Beagle/SystemSpecs/Gemini_whitepaper.pdf

[7] R. Alverson, D. Roweth, and L. Kaplan, "The Gemini System Interconnect," in *Proc. 2010 IEEE 18th Annual Symposium on High Performance Interconnects*, 2010.

[8] A. Bhatele, K. Mohror, S. Langer, and K. Isaacs, "There Goes the Neighborhood: Performance Degradation due to Nearby Jobs," in *Proc. Int'l Conf. on High Performance Computing, Networking, Storage and Analysis*, 2013.

[9] "Blue Waters." [Online]. Available: https://bluewaters.ncsa.illinois.edu

[10] "Managing System Software for the Cray Linux Environment," Cray Inc. S-2393-4202, Tech. Rep., 2013.

[11] "Ganglia." [Online]. Available: http://ganglia.info

[12] "Nagios." [Online]. Available: http://nagios.org

[13] "RRDTool." [Online]. Available: http:/rrdtool.org

[14] Cray Inc., "Using and Configuring System Environment Data Collections," Cray Doc S-2491-7001, 2012.

[15] "Performance Co-Pilot Programmer's Guide," March 2014. [Online]. Available: http://oss.sgi.com/projects/pcp/doc/pcp-programmers-guide.pdf

[16] "Collectl." [Online]. Available: http://collectl.sourceforge.net

[17] S. Godard, "sar(1) - linux man page." [Online]. Available: http://linux.die.net/man/1/sar

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

[19] Cray Inc., "Using Cray Performance Analysis Tools," Cray Doc S-2376-52, 2011.

[20] "MapReduce," 2014. [Online]. Available: http://en.wikipedia.org/wiki/MapReduce

[21] "MRNet: A Multicast/Reduction Network." [Online]. Available: http://www.paradyn.org/mrnet/

[22] "Tripod Operating System Software 2.0," March 2014. [Online]. Available: http://code.google.com/p/chaos-release/wiki/CHAOS_Description

[23] "Libevent - an event notification library," March 2014. [Online]. Available: http://libevent.org

[24] K. J. Barker, K. Davis, A. Hoisie, D. J. Kerbyson, M. Lang, S. Pakin, and J. C. Sancho, "Using Performance Modeling to Design Large-Scale Systems," *Computer*, vol. 42, no. 11, pp. 42–49, 2009.

[25] F. Petrini, D. Kerbyson, and S. Pakin, "The Case of the Missing Supercomputer Performance: Achieving Optimal Performance on the 8,192 processors of ASCI Q," in *Proc. 2003 ACM/IEEE Conference on Supercomputing*, 2003.

[26] K. Ferreira, R. Brightwell, and P. Bridges, "Characterizing Application Sensitivity to OS Interference Using Kernel-Level Noise Injection," in *Proc. Int'l Conf. on High Performance Computing, Networking, Storage and Analysis*, 2008.

[27] "PAL System Noise Activity Program," Los Alamos National Laboratory Performance and Architecture Laboratory (PAL), March 2014. [Online]. Available: http://www.c3.lanl.gov/pal/

[28] G. Bauer, S. Gottlieb, and T. Hoefler, "Performance Modeling and Comparative Analysis of the MILC Lattice QCD Application su3_rmd," in *Proc. 12th Int'l. IEEE/ACM Symp. on Cluster, Cloud, and Grid Computing*, 2012.

[29] R. Fiedler and S. Whalen, "Improving Task Placement for Applications with 2D, 3D, and 4D Virtual Cartesian Topologies on 3D Torus Networks with Service Nodes," in *Proc. Cray User's Group*, 2013.

[30] R. F. Barrett, C. T. Vaughan, and M. A. Heroux, "MiniGhost: A Miniapp for Exploring Boundary Exchange Strategies Using Stencil Computations in Scientific Parallel Computing," Sandia National Laboratories, Albuquerque, NM, Tech. Rep. SAND2012-10431, 2012.

[31] M. Deveci, S. Rajamanickam, V. Leung, K. Pedretti, S. Olivier, D. Bunde, U. V. Catalyurek, and K. Devine, "Exploiting Geometric Partitioning in Task Mapping for Parallel Computers," in *Proc. 28th Int'l IEEE Parallel and Distributed Processing Symposium*, 2014.

[32] A. M. Agelastos and P. T. Lin, "Simulation Information Regarding Sandia National Laboratories' Trinity Capability Improvement Metric," Sandia National Laboratories, Albuquerque, New Mexico 87185 and Livermore, California 94550, Technical report SAND2013-8748, October 2013.

[33] S. Muppidi and K. Mahesh, "Passive Scalar Mixing in Jets in Crossflow," in *44th AIAA Aerospace Sciences Meeting and Exhibit*. Reno, Nevada: American Institute of Aeronautics and Astronautics, Inc., January 2006.

[34] H. C. Edwards, "SIERRA Framework Version 3: Core Services Theory and Design," Sandia National Laboratories, Albuquerque, New Mexico 87185 and Livermore, California 94550, Technical report SAND2002-3616, November 2002.

[35] H. C. Edwards, T. S. Coffey, D. Sunderland, and A. B. Williams, "Sierra Toolkit Tutorial," Sandia National Laboratories, Albuquerque, New Mexico 87185 and Livermore, California 94550, Technical report SAND2011-0814C, February 2011.

[36] 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.

[37] E. S. Hertel, Jr., R. L. Bell, M. G. Elrick, A. V. Farnsworth, G. I. Kerley, J. M. McGlaun, S. V. Petney, S. A. Silling, P. A. Taylor, and L. Yarrington, "CTH: A Software Family for Multi-Dimensional Shock Physics Analysis," in *Proc. 19th Int'l Symposium on Shock Waves*, 1993, pp. 377–382.

[38] "CTH Shock Physics," http://www.sandia.gov/CTH/index.html, Albuquerque, New Mexico 87185 and Livermore, California 94550, April 2014.

[39] S. J. Schraml and T. M. Kendall, "Scalability of the CTH Shock Physics Code on the Cray XT," in *A reprint from the 2009 DOD High Performance Computing Users Group Conference*. Army Research Laboratory, Aberdeen Proving Ground, 2009, pp. 377–382, ARL-RP-281.

[40] B. Schmitt, "CTH Strong Scaling Results for a One Trillion Zone Problem to 1.56 Million Cores," Sandia National Laboratories, Albuquerque, New Mexico 87185 and Livermore, California 94550, Technical report SAND2013-8753P, November 2013.

[41] B. Jennings, R. Ballance, L. Sowko, and J. Noe, "2013 HPC Annual Report: Page 7, Modeling Optical Signals from Urban Nuclear Explosions For Yield Determination," Sandia National Laboratories, Albuquerque, New Mexico 87185 and Livermore, California 94550, Technical report SAND2014-0803P, 2014.

[42] C. Vaughan and D. Doerfler, "Analyzing Multicore Characteristics for a Suite of Applications on an XT5 System," Sandia National Laboratories, Albuquerque, New Mexico 87185 and Livermore, California 94550, Technical report SAND2010-2998C, 2010.

[43] J. Jayaraj, C. Vaughan, and R. Barrett, "Exploring Workloads of Adaptive Mesh Refinement," Sandia National Laboratories, Albuquerque, New Mexico 87185 and Livermore, California 94550, Technical report SAND2014-1395C, February 2014.