SANDIA REPORT
SAND2021-11954
Printed September 2021

Sandia National Laboratories

Integrated System and Application Continuous Performance Monitoring and Analysis Capability (Final)

Omar Aaziz, Ben Allan, Jim Brandt, Jeanine Cook, Karen Devine, James Elliott, Ann Gentile, Si Hammond, Brian Kelley, Lena Lopatina (LANL), Stan Moore, Stephen Olivier, Kevin Pedretti, David Poliakoff, Roger Pawlowski, Phil Regier, Mark Schmitz, Ben Schwaller, Vanessa Surjadidjaja, Scot Swan, Nick Tucker (OGC), Tom Tucker (OGC), Courtenay Vaughan, Sara Walton

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185
Livermore, California 94550

Issued by Sandia National Laboratories, operated for the United States Department of Energy by National Technology & Engineering Solutions of Sandia, LLC.

Available to DOE and DOE contractors from

U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865) 576-8401
Facsimile: (865) 576-5728
E-Mail: reports@osti.gov
Online ordering: http://www.osti.gov/scitech

Available to the public from

U.S. Department of Commerce
National Technical Information Service
5301 Shawnee Road
Alexandria, VA 22312

Telephone: (800) 553-6847
Facsimile: (703) 605-6900
E-Mail: orders@ntis.gov
Online order: https://classic.ntis.gov/help/order-methods

ABSTRACT

Scientific applications run on high-performance computing (HPC) systems are critical for many national security missions within Sandia and the NNSA complex. However, these applications often face performance degradation and even failures that are challenging to diagnose. To provide unprecedented insight into these issues, the HPC Development, HPC Systems, Computational Science, and Plasma Theory & Simulation departments at Sandia crafted and completed their FY21 ASC Level 2 milestone entitled "Integrated System and Application Continuous Performance Monitoring and Analysis Capability." The milestone created a novel integrated HPC system and application monitoring and analysis capability by extending Sandia's Kokkos application portability framework, Lightweight Distributed Metric Service (LDMS) monitoring tool, and scalable storage, analysis, and visualization pipeline. The extensions to Kokkos and LDMS enable collection and storage of application data during run time, as it is generated, with negligible overhead. This data is combined with HPC system data within the extended analysis pipeline to present relevant visualizations of derived system and application metrics that can be viewed at run time or post run. This new capability was evaluated using several week-long, 290-node runs of Sandia's ElectroMagnetic Plasma In Realistic Environments (*EMPIRE*) modeling and design tool and resulted in 1TB of application data and 50TB of system data. *EMPIRE* developers remarked this capability was incredibly helpful for quickly assessing application health and performance alongside system state. In short, this milestone work built the foundation for expansive HPC system and application data collection, storage, analysis, visualization, and feedback framework that will increase total scientific output of Sandia's HPC users.

ACKNOWLEDGMENT

We would like to give special thanks to Anthony Agelastos, Douglas Pase, Joel Stevenson, and Gary Lawson for their contribution to a Dedicated Application Time (see Section 4.1.1) on *Eclipse* for the purpose of evaluating LDMS overhead impact on applications and an initial testing of Kokkos publishing application data to *LDMS Streams*. Their contribution consisted of development of an application work package, which they ran over a 24-hour time period, and their post-run analysis validating low overhead ($\sim< 1.0\%$) for the deployment configuration of samplers and associated collection frequencies and targeted application data injection sizes and frequencies

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Sandia National Laboratories

U.S. DEPARTMENT OF ENERGY

SAND2021-11939 O

FY21 ASC FOUS/CSSE/ATDM L2 Milestone 7842: Integrated System and Application Continuous Performance Monitoring and Analysis Capability

Executive Summary

Author: James Brandt, 09/05/2021

Introduction

The overall goal of this work was to develop and deploy a unique capability for the run time and postprocessing examination of application progress and performance data in conjunction with High Performance Computing time-varying system data. Examination of this combined data is necessary to give insight into the causes of performance variation, degradation, and some failure cases. The infrastructure developed in this work leveraged Sandia's Lightweight Distributed Metric Service (LDMS) and Kokkos tools to provide the data and developed interoperability between them. The infrastructure also leveraged Sandia's Distributed Scalable Object Store (DSOS) database to support storage performance requirements. Dashboards and analytics for the combined data were developed as part of this work. The infrastructure was deployed on a Sandia CTS-1 system. A roadmap for enhancements and deployments was developed to extend this capability to other Sandia and, potentially, other DOE systems.

Milestone Description and Completion Criteria

As displayed in the ASC Implementation Plan (IP) and the Milestone Reporting Tool (MRT), the milestone description and completion criteria state:

"This L2 milestone will demonstrate the use of SNL data collection, analysis, and visualization framework/tools, deployed on a Sandia production SRN platform, to provide both system and application relevant run-time and post-run information for a rolling 2-week interval. We will demonstrate a capability for continuous collection of system data, an application progress metric(s), and an application throughput metric for an ASC-relevant code. We will provide a capability to store this data and a visualization interface that will enable a user to look at application progress in conjunction with system conditions, both at run time and post-run. We are targeting LDMS for the transport and aggregation of Trilinos-enabled application progress data and of system data. We are targeting the ATDM Application EMPIRE for deployment and its Proxy, MiniEM, for capability development. CSSE's Application Performance Team will be supporting development and testing.

Completion Criteria: Successful deployment of infrastructure on CTS-1 system. Demonstration of capability on target application run(s) on CTS-1 system. Lessons learned and feedback from stakeholders for future capability augmentation priorities will be documented."

Impact Statement

This work represents a significant milestone along SNLs march towards understanding and mitigating causes of the significant performance variation commonly experienced by large-scale tightly coupled simulation applications run on HPC systems.

Specifically, this work enables production high-fidelity monitoring of application performance and throughput characteristics in conjunction with system operational state and behavioral characteristics. Analysis of this data can provide insights into causes of performance degradation and high application performance variability on HPC systems. Improved visibility into, and understanding of, application and system resource interactions over lifetimes of application executions will, in turn, enable improved science and engineering throughput and higher overall HPC system efficiency.

Note that our approach is well suited to production use as it does not require code changes or recompilation on the part of the user to collect this valuable information and adds negligible application performance-impacting overhead.

Summary of Work Done

All the completion criteria have been met as follows:

- Successfully deployed our monitoring infrastructure on Sandia's CTS-1 system, Eclipse, which has been running continuously, in production, since January 28, 2021. Data from this deployment is being stored to Sandia's DSOS database on our monitoring and analysis cluster, *Shirley*, for at least a 2-week rolling time window.
- Identified, and presented, total number of kokkos kernel executions per-minute as a throughput metric and the rate of a particular kernel (ParticleMove::Move) call as a problem specific progress metric for the ASC-relevant code, *EMPIRE*.
- Demonstrated gathering and presenting, both post-run and at run time, these progress and performance metrics in conjunction with the *Active_memory* system metric for the application *EMPIRE* running on *Eclipse*, at scales of 32 to 290 nodes, via our Grafana-based visualization interface using Sandia's DSOS as the scalable data store.
- Documented the feedback we received from users who utilized the tools during the milestone period. We also identified and documented future work to take advantage of these new capabilities.

Path Forward

As mentioned above, the successful completion of this effort has given Sandia an end-to-end deployment of the capability to gather, store, analyze, and visualize application performance and progress data in the context of time varying system state metrics. We plan to develop analytics to gain insight into root causes of application performance variation and degradation. In addition to the application monitoring collaborations at Sandia, we have started several collaborations with other DOE laboratories to integrate their data collection tools into the infrastructure. We will investigate run-time response to conditions of interest in the data. We will be expanding the features of this capability and deploying it on additional Sandia systems as indicated by the roadmap presented in this work.

In Support of Stewardship Capability Delivery Schedule (SCDS)?

N/A

NOMENCLATURE

Aggregator LDMS daemon which aggregates (push or pull) data from other LDMS daemons

ATDM Advanced Technology Development and Mitigation, an element of the NNSA Advanced Simulation and Computing program

CSSE Computational Systems and Software Environments, an element of the NNSA Advanced Simulation and Computing program

CTS Commodity Technology System (production HPC systems)

DSOS Distributed Scalable Object Store

Eclipse Production CTS-1 system outfitted with data collection infrastructure to support this milestone

EMPIRE The "ElectroMagnetic Plasma In Realistic Environments" is a modeling and design tool for plasma environments

FOUS Facilities, Operations, and User Support, an element of the NNSA Advanced Simulation and Computing program

IC Integrated Codes, an element of the NNSA Advanced Simulation and Computing program

LDMS Lightweight Distributed Metric Service

LDMS Streams LDMS's Publish/Subscribe service

Kokkos Application portability layer leveraged in this milestone for publishing application telemetry to *LDMS Streams*

Kokkos-LDMS Connector Code responsible for interfacing between *Kokkos Sampler* and *LDMS Streams*

Kokkos Sampler Code responsible for gathering information about Kokkos kernel executions and passing it to the *Kokkos Connector*

metric set LDMS construct for fixed schema sets of structured system data

Sampler LDMS plugin or daemon which collects data

SOS Scalable Object Store - Object store database used in this work

1. INTRODUCTION

The overall goal of this work is to develop and deploy a unique capability for the run time and postprocessing examination of application progress and performance data in conjunction with High Performance Computing time-varying system data. Examination of this combined data is necessary to give insight into the causes of performance variation, degradation, and some failure cases. The infrastructure developed in this work leveraged Sandia's Lightweight Distributed Metric Service (LDMS) and Kokkos tools to provide the data and developed interoperability between them. The infrastructure also leveraged Sandia's Distributed Scalable Object Store (*DSOS*) database to support storage performance requirements. Dashboards and analyses for the combined data were developed as part of this work. The infrastructure was deployed on a Sandia CTS-1 system. A roadmap for enhancements and deployments was developed to extend this capability to other Sandia and, potentially, other DOE systems.

The rest of this report is organized as follows: We first provide the raw milestone text along with scope statements in Section 2. We provide both an overview and detailed background discussion of the architectural components of the data collection, transport, storage, and visualization components in Section 3. We next discuss relevant deployment details in Section 4. Details about application and system metrics being derived and displayed are described in Section 5. A description of the main components utilized to support analysis and visualization are presented in Section 6. Feedback from users and stakeholders is documented in Section 7. In Section 8 we provide a short tabulation of lessons learned in the execution of this milestone. We have documented, in Section 9, the relevant work leading up to this milestone as well as our planned path forward. In Section 10 we provide a checkbox version of the completion criteria along with a short description of the evidence of completion. Finally, references can be found at the end of the document.

2. MILESTONE

The text of the milestone is as follows:

> *Description:* This L2 milestone will demonstrate the use of Sandia data collection, analysis, and visualization framework/tools, deployed on a Sandia production SRN platform, to provide both system and application relevant run-time and post-run information for a rolling 2-week interval. We will demonstrate a capability for continuous collection of system data, an application progress metric(s), and an application throughput metric for an ASC-relevant code. We will provide a capability to store this data and a visualization interface that will enable a user to look at application progress in conjunction with system conditions, both at run time and post-run.
>
> We are targeting LDMS for the transport and aggregation of Trilinos-enabled application progress data and of system data. We are targeting the ATDM Application EMPIRE for deployment and its Proxy, MiniEM, for capability development. CSSE's Application Performance Team will be supporting development and testing.

Completion Criteria:

- Successful deployment of infrastructure on CTS-1 system.
- Demonstration of capability on target application run(s) on CTS-1 system.
- Lessons learned and feedback from stakeholders for future capability augmentation priorities will be documented.

Section 10 provides a detailed compilation, in checklist form, of the completion criteria along with short overview statements of how they were satisfied. The body of this report provides the details to support those statements.

While the intent of the capability is vast, for the FY20 milestone, we explicitly scoped the work as described below:

In scope

- Developing and deploying an integrated architecture for application and system information
- Collecting application and system state metrics from a CTS-1 system at run time
- Providing a visual interface for derived application performance and throughput metrics alongside system metrics
- Demonstrating this interface on run time CTS-1 data with the ability to do historical investigation up to two weeks
- Providing information on instrumentation overhead and application performance impact

Not in scope

- Tuning system parameters to avoid application performance variation
- Deriving causality of application performance variation
- Correlating system state with application performance
- Determining best system or application data to collect
- Production-hardened deployment of collection infrastructure / analysis

All *not in scope* work items are intended to be Future Capability Augmentation Priorities (see Section 7).

3. ARCHITECTURE

In this section we first provide a motivational overview of the architecture that we developed and deployed in this work. We then provide deeper background information on the functional components and sub-systems deployed including motivation for the choices made.

3.1. Overview

We leverage the Lightweight Distributed Metric Service (LDMS [2]) that is currently being used for the collection of system data. System data can be obtained using both generic and system-specific data collection plugins (typically referred to as *sampler* plugins). LDMS daemons running these plugins (referred to as *samplers*) are typically run on the components for which data is being collected (e.g., compute nodes for node-local file I/O, networking, CPU, memory information and Lustre file system components for global Lustre information). For the aggregation/transport of data, LDMS is typically run in a mode that *pulls* system data from the sampler daemons, particularly those running on compute nodes, at regular intervals in order to minimize CPU overhead and hence performance impact on applications.

We have developed a new message bus capability for LDMS in order to enable injection of information on-demand, in a *push* mode, into its data stream. This new publish/subscribe capability is called *LDMS Streams*. The *LDMS Streams* capability can be used to inject application progress and performance information, during run time, into the LDMS data stream concurrent with system resource state data injection and enable them to be stored into the same data store.

While data can be injected, via the *LDMS Streams* API, from any source, we have implemented this capability within the Kokkos [7] portability abstraction layer. This has the advantage of encapsulating the injection capability and thus requiring no application changes or recompilation. Further, we can leverage the performance and progress instrumentation already implemented within Kokkos and not burden the user or application programmer with specifying what data to collect.

Our architecture, depicted in Figure 3-1, is used to collect system data at regular intervals with those intervals being defined based on the time scales of conditions of interest. For this milestone work, the production CTS-1 system that we have deployed this capability on is *Eclipse*. All data is transported from *Eclipse* to a *monitoring* cluster where it is written into our distributed NVMe-based high performance database, *DSOS*. *DSOS* was developed, and extended as part of this milestone work, in order to support the demands of high data ingest rates in conjunction with data browsing and run time analyses.

Long-term data, older than two weeks, is moved to mass storage and can be loaded back into the database if needed for performing analyses over longer time periods. Raw data and the results of analyses are presented during run time on Grafana-based dashboards. Our goal is to enable insight, *during run time*, into applications' performance characteristics in conjunction with system performance resource utilization and performance characteristics. In particular we are interested in insights into strong correlations between the two and in discovery of causality where application performance degradation is identified.

3.2. LDMS

In this subsection we provide a background on LDMS, its use for system data collection, and a high level description of how we enhanced LDMS to support the injection of application progress

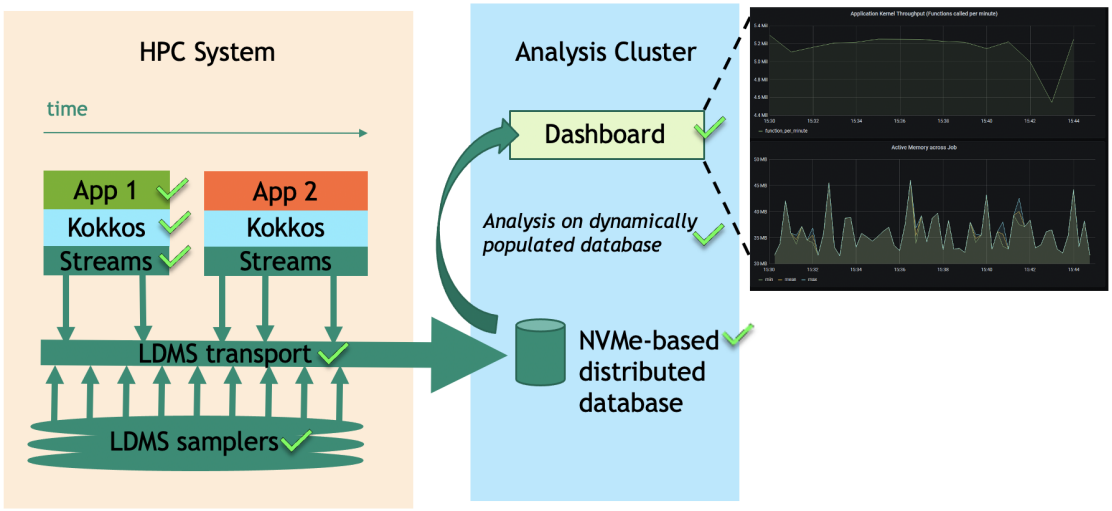


Figure 3-1 Data Flow Diagram of Integrated System and Application Performance Data Analysis Capability. Application progress and performance data is injected into the LDMS data stream which regularly transports data collected from system data sources. The combined information is treated in a standardized way, easing development of analyses and visualizations for application performance in combination with system conditions. Green check marks indicate capabilities developed as part of this work.

and performance data. Since application and system data are inherently different, we highlight the differences and the design decisions made as a result.

LDMS was designed for lightweight extreme-scale data collection, transport, and storage. It is intended to be run as a continuous system service, periodically and synchronously collecting data from instrumented components across the entire system and efficiently transporting it, as it is collected, to one or many endpoints in order to support run time analyses and visualizations that can be used to provide insights and feedback on actionable timescales.

3.2.1. System Data Collection

System data is typically intended to be used to provide insight into the state of the system at any given point in time. Thus data must be obtained *synchronously* across all instrumented components across the whole system. Collecting this data on regular intervals enables not only system state *snapshots* but also provides trajectory information for the state data collected. In order to minimize performance impact on applications, the on-node operations associated with monitoring data collection and transport have been kept to a minimum.

Because the system data is of the same format each time it is collected, the data lends itself to the use of a *structured* data format, which we call a *metric set*. An example is shown in Figure 3-3. The structure enables more efficient gathering of the data while minimizing data movement. That is, the well known layout is leveraged to define the layout of the memory regions. *Meta-data* which describes the layout, names, and types of the data fields (e.g., MemTotal u64) is only transported at the start or upon change, reducing data movement.

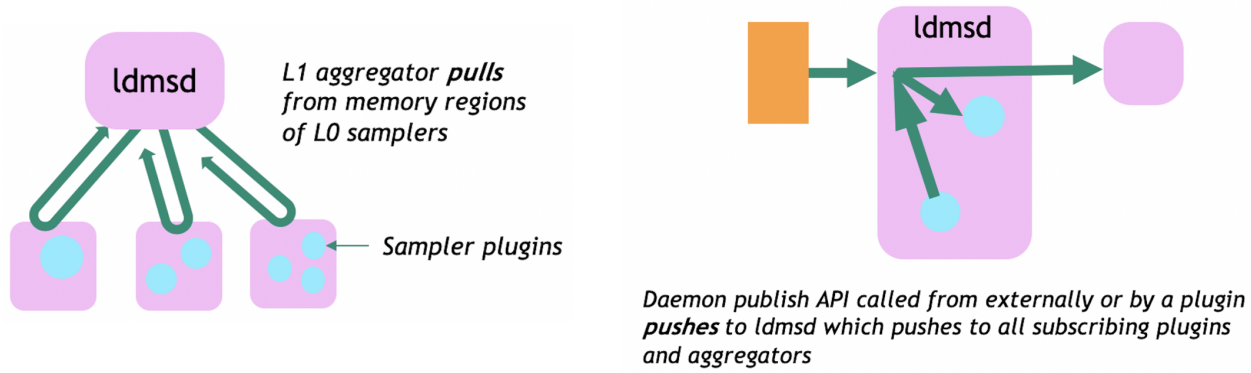


Figure 3-2 LDMS data collection and transport modes utilized in this work. Blue circles indicate plugins into the pink LDMS daemons. Green arrows indicate communications and data flows. (left) System data is typically *pulled* at regular intervals from other *aggregator* daemons in order to minimize the on-node capabilities required and thus application impact. (right) Application data is *pushed* on demand into the LDMS daemon which then *publishes* the data to subscribers which can be both local plugins or remote daemons (figures from [4]).

```
voltrino:/opt/ovis/etc/ldms # ldms_ls -h nid00006 -x sock -p 412 -a munge -lv nid00063/meminfo
Schema         Instance                Flags  Msize  Dsize  UID    GID    Perm       Update            Duration          Info
-------------- ----------------------- ------ ------ ------ ------ ------ ---------- ----------------- ----------------- --------
meminfo        nid00063/meminfo            CR   1976    416      0  44476 -rwxrwx--- 1570023154.002980          0.000031 "updt_hint_us"="1000000:300000"
-------------- ----------------------- ------ ------ ------ ------ ------ ---------- ----------------- ----------------- --------
Total Sets: 1, Meta Data (kB): 1.98, Data (kB) 0.42, Memory (kB): 2.39

========================================================================

nid00063/meminfo: consistent, last update: Wed Oct 02 07:32:34 2019 -0600 [2980us]
M u64        component_id                       63
D u64        job_id                             0
D u64        app_id                             0
D u64        MemTotal                           131899768
D u64        MemFree                            129865232
D u64        MemAvailable                       129416140
D u64        Buffers                            23380
D u64        Cached                             508992
D u64        SwapCached                         0
D u64        Active                             277760
D u64        Inactive                           307892
D u64        Active(anon)                       193908
D u64        Inactive(anon)                     184852
D u64        Active(file)                       83852
```

Figure 3-3 LDMS metric set queries. System data is represented in a structured data format designed to minimize data movement. A given LDMS Sampler (e.g, meminfo shown) collects one or more data field names, types, and values (e.g., MemTotal, u64, 131899768). Meta-data consisting of information about the layout, names, and types of data, is only pushed once or upon change, reducing data movement. Set permissions (e.g., `rwxrwx--`) can be used for access control.

Data is typically collected on-node by *sampler plugins* of an LDMS *sampler* daemon at regular intervals and inserted into the data regions of metric set data structures. An off-node LDMS *aggregator* daemon then *pulls* the data at regular intervals from the data memory regions of the metric set data structures. In this way, all of the logic and capability for transport is kept off-node. Figure 3-2 (left) illustrates this mode of collection and aggregation.

LDMS also provides security through authentication, association of a user and group ID with each metric set, and associated permission bits to enable access control, as shown in Figure 3-3. (More detail on the metric set can be found in [2].)

Both socket-based and RDMA transports are supported, with the latter minimizing CPU interference. While in Figure 3-1 we show the transport directing all data to a single off-platform database, the data can be directed to an arbitrary number of consumers. More information can be found at the OVIS project's LDMS github site at `https://github.com/ovis-hpc/ovis` [1].

3.2.2. Application Data Collection

Application related data is inherently different than system data. Application data is event-based data that occurs *asynchronously* across the system and hence requires a *push-based* methodology in order to minimize on-node memory consumption, data loss, and latency between occurrence and remote recording of events. Because events may vary in their content, the data format must be *variably structured*.

In order to support on-demand publication of variably formatted information, we have enhanced LDMS to include a *publish/subscribe bus* capability. This capability and the associated API for publishing and subscribing is called *LDMS Streams*. LDMS daemons and plugins subscribed to a *LDMS Streams* tag receive any data events that arrive with that tag. This is illustrated in Figure 3-2 (right). When being published, data is specified as either `string` or `JSON` format. The *LDMS Streams* feature leverages the efficient and secure LDMS transport to additionally support efficient application related data collection and run time transport and storage to the same database in which system data is being stored.

3.3. Kokkos

Kokkos is a parallel programming ecosystem for performance portability across multi-core, many-core, and GPU node architectures [7][6]. At its heart is a library-based API that uses modern C++ metaprogramming to abstract away architecture-specific execution and data management details so that programmers can focus on exposing parallelism in their applications. The companion Kokkos Kernels library provides tuned implementations of common computational kernels hereafter referred to simply as *kernels* for performance-critical use cases. Finally, and of primary interest to this milestone, the Kokkos Profiling interface [8] provides application timers and utilities to connect with third-party tools from vendors (e.g., Intel VTune) and open source efforts (e.g., HPCToolkit). Since Kokkos is Sandia's chosen vehicle for performance portability of ASC IC and ATDM applications and also in widespread use among other DOE laboratories, it is an ideal candidate for our work.

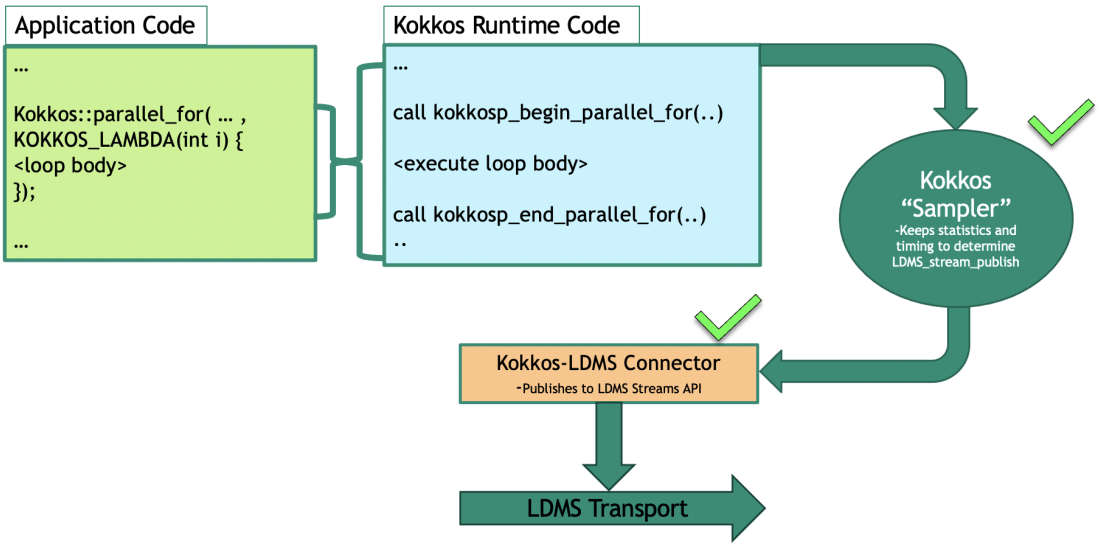


Figure 3-4 This figure depicts an application utilizing the Kokkos platform. As part of this milestone we have modified the *Kokkos Profiling* interface to include a *Kokkos Sampler* facility to sample information about a subset of *kernel* executions. The *Kokkos Sampler* takes user input on which *kernel* executions are to be sampled. As an example: if the user enters 100, every 100th *kernel* execution will be sampled. For a sampled *kernel* the timestamp, duration, total number of that *kernel*'s executions sampled by the reporting process, and name of *kernel* sampled will be returned and published via the *Kokkos-LDMS Connector* to the LDMS daemon running on the node on which the sampled *kernel* was executed. Note that the green check marks show components developed, and completed, as part of this milestone.

```
#timestamp,job_id,rank,name,type,current_kernel_count,total_kernel_count,level,current_kernel_time,total_kernel_time
1627835612.086679,10195735,1,Kokkos::View::initialization [diagnostic:Solver Field:B_Field:temp],0,1218,57972687,0,0.000005,182.693422
1627835613.709526,10195735,1,TimeAverage::Continuous,0,24758,57972788,0,0.000006,182.693428
1627835616.787472,10195735,1,MigrateParticles::count,1,3540,57972889,0,0.000001,182.693430
1627835620.448333,10195735,1,SolverInterface::Apply Trivial BC,0,7512,57972990,0,0.000002,182.693432
```

Figure 3-5 CSV formatted output of the application data injected in JSON format. Timestamp is global and can be used to associate the occurrence of a sampled *kernel* and time-windowed statistics with the system data at that time. Kernels are reported by name. The count of a particular *kernel* reflects that downselection of event reporting occurs.

In this work we leverage the *Kokkos Profiling* interface as a vehicle to publish timestamped *kernel* events and timings to the *LDMS Streams* interface. This sub-component and associated data flow are depicted in Figure 3-4. This information is then transported via the LDMS transport and stored to the same database in which the system data is being stored. This enables simple time alignment with system data and visualization of both together. Some example application data output converted into CSV format for display is shown in Figure 3-5.

To support low-overhead monitoring of applications, a *Kokkos Sampler* (fig 3-4) was developed to control the sampling rate of Kokkos *kernel* executions, along with the rate of messages being delivered to the *LDMS Streams* pipeline. The *Kokkos Sampler* sampling rate is set at the beginning of a run (in the batch script) as an environmental variable. For this milestone work the *Kokkos Sampler* was configured to collect 1% of messages from all kernels for each rank. Additionally, we developed the *Kokkos Connector* (fig 3-4) which establishes, and validates the existence of, a connection to the local LDMS sampler daemon. Assuming a valid connection, the *Kokkos Connector* takes the output of the *Kokkos Sampler*, demangles the *kernel* names, and publishes the message in JSON format to the *LDMS Streams* interface.

3.4. Storage

Each of the 16 nodes of our monitoring and analysis cluster *Shirley* has 56TB of NVMe storage. This NVMe storage was used exclusively for storage, and querying, of system and application data using our Distributed Scalable Object Store (*DSOS*) database.

DSOS is a coordination layer sitting on top of our Scalable Object Store *SOS* and is depicted in Figure 3-6. *DSOS* presents a scalable distributed database with a variety of features that enable the simultaneous large-scale data ingest and queries required for this work. *DSOS* presents a single, unified database to the end user. All *SOS* databases within a *DSOS* domain can be queried in parallel as the result of single call to *DSOS* and a single set of aggregate data will be returned as a result. *SOS* and *DSOS* are designed specifically to address the domain-specific needs of large-scale HPC monitoring with respect to low latency ingest and query of large volumes of data. They are collaboratively developed by Sandia and Open Grid Computing (OGC). While not specifically developed for this milestone, it is a testament to the flexibility of this database that we were able to make significant changes, including performance improvements, to how information is being stored, indexed, and retrieved over this milestone window. This would not have been possible using commercial databases.

Besides its scalable data ingest and query performance, *DSOS*'s flexible indices were especially critical for execution of this milestone. *SOS/DSOS* indices can be based on any metric, or combination of metrics, within an LDMS schema. Additionally, indices can be created and removed as needed without reloading data. These features were critical in identifying the optimal index patterns for the different query use cases we had within our analyses. *DSOS* also has Python, C, and C++ APIs and a command line interface for interacting with the data. The Python API is used within all the analyses and the command line interface is a fast method to test queries and examine data. Query optimization was necessary because of the amount of data we ingest and store. Roughly one month of Eclipse system data stored in a *DSOS* database is over 50TB,

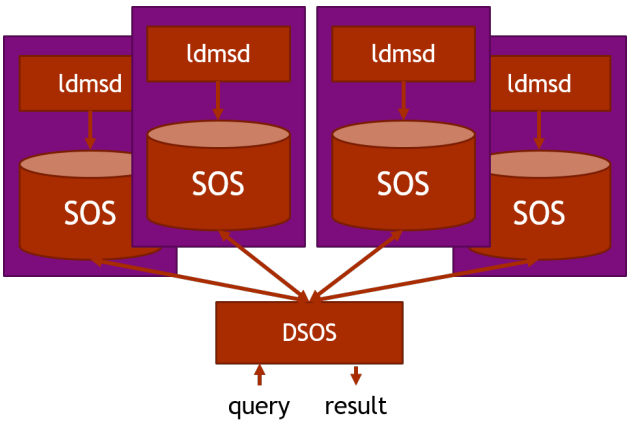


Figure 3-6 *SOS* database coordination on Shirley cluster using *DSOS*. A single DSOS query is parallelized across all SOS databases. The data from each of those databases is collected and sorted to present a single result for the user.

Figure 3-7 Analysis and Visualization Pipeline Block Diagram. Queries from a Grafana web browser are sent through an Apache server to a Django application. The queries can specify a Python analysis module to call, which queries the *DSOS* database and manipulates the returned DataFrame. This DataFrame is returned to the Grafana browser after being correctly formatted to create meaningful visualizations.

including indices. Two separate week-long 290-node EMPIRE runs added 900GB to the distributed database.

3.5. Analysis and Visualization

Prior to this milestone, we had created an analysis and visualization (A&V) pipeline to derive metrics of interest from LDMS system data and to show the results on a Grafana dashboard [10]. A high level diagram of our analysis and visualization (A&V) pipeline is shown in Figure 3-7.

The A&V pipeline starts with a user query from a Grafana interface. The time range, type of analysis, and any input parameters are sent through an Apache server to a custom Django Python application. This Django application interprets the query sent from Grafana. Some queries merely request data directly from the *DSOS* database, which is then queried through the *DSOS* Python API, and returned as a Pandas DataFrame, JSON formatted, and returned to Grafana. However, most of the dashboards used have an analysis call to transform raw data to something more interesting. In this case, the Django application calls the requested Python analysis to interact with the database. The analysis modules allow for endless possibilities limited only by Python coding ability and time that the user must wait for the result.

A key feature of this pipeline is that analysis is done when, and only when, a user requests a visualization in Grafana. Although this increases query time, it saves significant computation and storage resources as compared to traditional methods which require running all analyses across all data and saving to a results database. Additionally, analysis modules and resultant data formatting can be easily changed without needing to repopulate a results database. This pipeline does support an "always analyze" usage model, which we have found applicable for select analyses, but these were not a part of this milestone. Significant work was done, as part of this milestone, to optimize database queries and Python analyses for fast Grafana query times.

For this milestone, we also made substantial changes to the internals of the pipeline to improve scalability and performance. The previous pipeline had longer query times because it used a single-node *SOS* database rather than *DSOS*. To incorporate *DSOS*, major changes were made in the Django application to conform to the DSOS Python API and in the Python modules to make better use of the *DSOS* SQL-like query structure.

4. DEPLOYMENT

LDMS has been deployed on the majority of our production HPC clusters since 2015, with stable release version updates being part of our maintenance workflow. As part of this milestone work we scale-tested and deployed our latest LDMS version, which includes the *LDMS Streams* capability described in Section 3.2.2, on our CTS-1 *Eclipse* system (described below in Section 4.1). As part of the upgrade we also moved from collecting system data at once per minute to once per second.

4.1. Scalable high-frequency data collection

The overall scheme of hardware and software deployed is illustrated in Figure 4-1. In detail, an LDMS data collection service (S0) instance (otherwise known as a LDMS sampler daemon) is deployed on all of our *Eclipse*[9] cluster nodes including 1488 computes, 12 logins, 24 Lustre gateways, and 8 administrative nodes. Application or job launch processes can publish tagged event data to the collecting service on the compute or login node where they are running. We divide the cluster nodes into 16 groups and deploy one LDMS first level aggregation service (S1) instance per group. Each S1 instance collects *LDMS Streams* and system data from its assigned S0 instances. The data sampling plugins deployed on *Eclipse* and their collection intervals (in seconds) are listed in Table 4-1.

The sixteen S1 instances are spread across just three administrative nodes. On our analysis cluster *Shirley*, (further described in Section 4.3) a second level aggregation service (S2) instance runs one-per-node, capturing and locally storing the data flow from its paired *Eclipse* S1 service instance. Within *Eclipse*, all LDMS data is transported over the fast Omnipath network. *Shirley* S2 instances connect to *Eclipse* S1 instances via a 10Gb/s Ethernet network. In addition to the S2 data storage to the *SOS* databases, the S1 instances store system metric data to CSV files on administrative node private 2TB scratch disks. These CSV files allow for data recovery in the

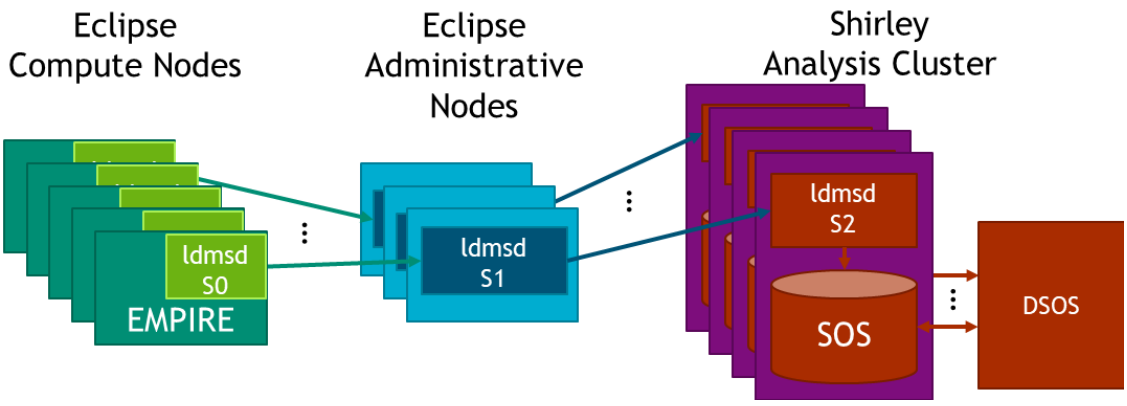


Figure 4-1 Deployment Architecture

event of a temporary outage of *Shirley* or the the network between *Shirley* and *Eclipse*. These files are moved daily from the local scratch disk to network storage.

plug-in	interval	data
filesingle	60	power and temperatures
meminfo	1	/proc/meminfo for free, active, and other memory usages
vmstat	1	/proc/vmstat for numa and other metrics
procstat	1	/proc/stat for cpu tick, interrupt, and process counts
loadavg	1	/proc/loadavg for 1, 5, 15 minute loads
procnet	1	Ethernet device traffic and errors
opa2	1	Omnipath device traffic and errors
procnfs	1	NFS v3 events and traffic
lnet_stats	1	Lustre network traffic and errors
lustre_client	1	Per-mount-point Lustre events and traffic
dstat	1	LDMS daemon I/O, memory, CPU and file descriptor usages

Table 4-1 Describes plugin names, types of data gathered, and collection periods (in seconds)

4.1.1. Data collection overheads

As part of the milestone, in Jan 2021 we held a 30-hour DAT on Eclipse for LDMS Version 4 overhead testing and to validate the interoperability of our initial application, Kokkos sampler, and Streams functionality. This involved substantial work up front in determining applicable workload and metrics to collect as well as all of the infrastructure and analysis/visualization configuration. The statistical results of this DAT will be documented in detail in a separate publication. The DAT compared runs with and without LDMS present on the compute nodes for run sizes from 1 to 1024 nodes for a variety of applications that stressed different hardware subsystems. In summary performance overheads were observed as follows:

- The typical total wall-clock overhead observed was less than 1% while the typical run-to-run variation observed without LDMS was 1.3%.

- There was no discernible run length difference between publishing 1% of all Kokkos events and publishing no Kokkos events.
- The aggregate LDMS network traffic leaving Eclipse is approximately 3% of a 10Gb/s link in the administrative network.
- There is $\sim$ 6MB of memory used for LDMS on the compute nodes where we are collecting $\sim>$ 5000 metrics per node. This is not expressed as a percent of memory because it doesn't scale with node memory size.

4.2. Administrative controls

Unprivileged *Eclipse* users connect to LDMS daemons on a port designated for this purpose. The munge service is used for authentication. The aggregation hierarchy inside *Eclipse* is interconnected using a different port and a shared secret, restricted to administrative processes, for authentication. In this way, application data injection to LDMS can be easily disabled if needed. The S1 instances (up to six of which run on each administrative node) listen for connections on other designated ports; the *Shirley* S2 instances authenticate to the S1s with the administrators' shared secret. The current implementation has no software limits on the local or aggregate rate of user application data injection; the 10Gb/s Ethernet network does pose a hardware limit.

4.3. The Shirley Monitoring and Analysis System

As part of this milestone work we designed and deployed our *Shirley* monitoring and analysis cluster. *Shirley* consists of 16 storage and analysis nodes each comprising dual 2.40GHz Xeon 6240R processors (48 cores total) with 1.35B of RAM and 56TB of NVMe storage (spread across 8 drives). To best support the recently released CPUs, high-speed network features, NVME storage, and analysis and visualization software, we deployed the Redhat Enterprise Linux 8.4 operating system across the whole *Shirley* system.

5. APPLICATION AND SYSTEM METRICS

We collect a wide range of metrics. Some are directly observable, while others are derived from other metrics. Some reflect shared resource usage, such as file system and network utilization, while others are specific to the user's allocation, such as CPU and memory utilization. To determine metrics of interest in the context of this milestone we decided to go through a requirements gathering process involving both application and system monitoring experts.

This milestone required that we demonstrate a capability for continuous collection of system data, application progress metric(s), and an application throughput metric. The deployed installation of LDMS included several system samplers that enable collection of system data such as jobid, load average, CPU and memory utilization, various Ethernet, Omnipath, NFS, Lustre file system and Lustre networking events, motherboard temperatures and power, and aggregator daemon performance metrics (see Table 4-1). The goal of this requirement is to display system metric data

along side application throughput metric data, enabling a user to visually identify periods of execution during which system behavior affects application throughput. Since memory can often adversely affect application performance, we chose **Active memory**, for the nodes hosting the job/application of interest, as the system metric for the dashboard display.

In scientific engineering applications, there is typically some sort of iterative solve process that reflects conditions of the physical system being simulated. For the *EMPIRE* application, a method called *ParticleMove::Move* is called on each iteration of the solve loop. Computing *ParticleMove::Move* invocations per unit time (sec) would give a good indication of application progress. The dashboard provides an application profile in which the user can select the function/method for which to compute invocations per time. Therefore, this is robust across applications and for *EMPIRE* we chose **ParticleMove::Move invocations/sec** in a specified time range to reflect application progress.

Defining an application throughput metric is challenging because it's tempting to choose to measure behavior that is application-specific such as particle updates per second. Although users can define custom metrics for display, we chose to determine a first-order application throughput metric that is robust and applicable to all applications, which is **total functions/methods called per minute** within a specified time range. Application throughput reduction is indicated when this metric decreases in value.

The visualization dashboard displays these metrics through various plots which will be discussed in the next section.

6. SYSTEM AND APPLICATION DATA VISUALIZATION

For this milestone, we created two Grafana dashboards to visualize an application's Kokkos instrumentation data, each with several analysis-driven panels on them. The dashboards will be referred to as the job-level dashboard and the kernel-level dashboard. Grafana was chosen because it is an open-source tool gaining traction across the tri-lab HPC monitoring community for HPC visualization and is tailored towards visualizing time-series data. A Grafana interface consists of dashboards populated by panels. Each panel visualizes data obtained from a query to the A&V backend. The dashboard has a time-range picker for users to query data in a chosen time frame. This time-range is part of the query used in each of the panels on the dashboard. Panels in our infrastructure are always aligned in time.

The job-level dashboard in Figure 6-1 shows high level information about the application across all kernels in the job alongside a system metric. There are three panels in this dashboard. The first panel is called the Kernel Summary Table that shows all kernels, the number of times each kernel was called, the time spent in each kernel, and the average kernel execution time in the time-range specified. This table is useful for quickly understanding the behavior of kernels called in the application in this time. Each kernel name in the table is also a hyperlink to the kernel-level dashboard for that kernel.

The second panel on this dashboard presents the application throughput metric and is a time-series plot of the total kernel calls per minute across all ranks in the application. This panel

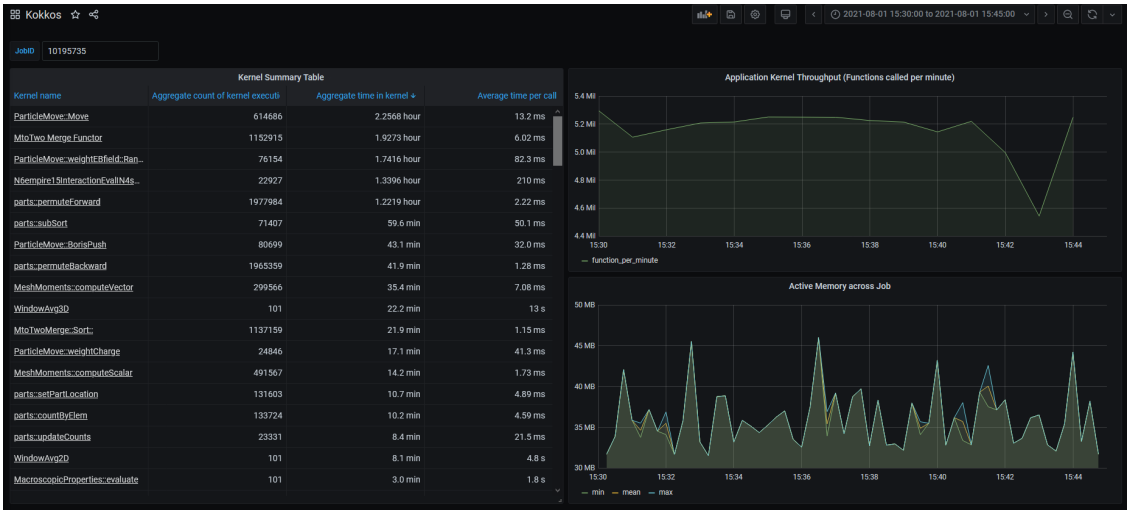


Figure 6-1 Job-level Grafana dashboard with a kernel summary table, an application throughput time-series plot, and a memory usage time-series plot

provides the trend of kernel executions in the application. Dips in the time-series plot might indicate the application stalled or is shifting to a new phase of the application.

The final panel is a time-series plot of the active memory usage of nodes within the job. For each time point, the graph shows the average memory usage across all nodes and the maximum and minimum memory usage on a single node. The memory usage panel displays the system metric of choice for this milestone and helps users track memory usage and balancing across the job.

The kernel-level dashboard in Figure 6-2 displays information pertinent about a single kernel within the application. The dashboard has a text box for users to fill in which kernel is of interest to them. Additionally, they can navigate to this dashboard using the job-level dashboards Kernel Summary Table that will automatically fill in this text box. There is also a text box for users to control the bin size used in the underlying analyses. This bin size determines the level of granularity for the returned results. For example, a bin size of 10 will return plots where each data point is an average across 10 seconds. This allows users to see general trends or fine details as needed.

There are two panels on the kernel-level dashboard. The first panel is a time-series plot called Function Timing Information that has two lines on it. The blue one displays the average time per kernel execution over time. The average here is taken across all ranks in the application. This can indicate if there are times when a kernel takes very long to execute, which might be indicative of a problem. When this dashboard is populated using a kernel indicative of scientific performance, such as *ParticleMove::Move* as in the case of *EMPIRE*, this line is an application progress metric. The orange line in this panel shows the number of times that kernel is being called per second. This line indicates if a kernel is being called more or less than usual which could indicate stalls or phase changes.

The second panel on this dashboard shows a heatmap of the execution time across ranks in the application. Blue colored bins mean a few ranks are in this execution time range, red colored bins mean many ranks are in that range. This panel helps users understand the distribution of execution

Figure 6-2 Kernel-level Grafana dashboard with a function timing information time-series plot and an kernel execution heatmap

times and identify ranks acting as outliers. In the *EMPIRE* milestone runs, we identified ranks consistently acting as outliers that were previously unknown to the *EMPIRE* developers. A limitation of this panel is that Grafana heatmap visualizations are computationally expensive and hence the latency from query to results can be significant for a large number of ranks (note this has no impact on the application as these queries and computations are performed on *Shirley*). To get around this, we randomly sample ranks when the application uses more than 100 ranks. Rank zero is always kept in the sampled ranks as it is often important for developers to understand.

Both dashboards were shown to display information about a 290-node *EMPIRE* milestone run. Users were able to use the time-range picker to view the different derived metrics across any part of the job. We also showed, using a two-node run of the application SPARTA for simplicity, that the dashboard could present a live view of a currently running application.

7. STAKEHOLDER FEEDBACK AND FUTURE CAPABILITY AUGMENTATION PRIORITIES

In this section we present Stakeholder Feedback and Capability Augmentation Priorities. Our timeline for addressing these and for rolling out the capability on our other HPC systems is presented in our Roadmap (Section 9).

7.1. Feedback

Stakeholders in this work are primarily users of our HPC systems who would be interested in the performance and progress understanding that live data and analyses can provide. For the user feedback, we targeted the *EMPIRE* developers and analysts who could then see the capability as it

pertains to their code. There were additional requests from system administrators who were part of the team and can also benefit from the capability as it would show conditions on the system and insights into underperforming applications.

The feedback was overwhelmingly positive about the work's utility and potential. Stakeholders also made feature requests as described below.

EMPIRE developer and analyst comments:

- "There was no noticeable impact on performance on small or large simulations when LDMS was enabled"
- "I fully expect enabling LDMS to become the default *EMPIRE* behavior on supported platforms"
- "Being able to see the dashboard's real-time updating of simulation performance is so much better than manually finding that information in simulation log files"
- "Quickly plotting simulation metrics helps us quickly assess job health and progress, saving time and decreasing cognitive load"
- "Clear, clean layout without presenting too much information"

Requested enhancements:

- The number one request was that they would like to be able to have a subset of *kernels* always collected (e.g., the *ParticleMove::Move kernel*)
- More information about filesystem I/O alongside application event data
- Rename data labels to improve understanding

  Will be adding in-depth descriptions about the data and underlying analyses
- Add bit-rate to application throughput panel to show how much data is being ingested by the backend

  Will be useful for adjusting sampling rate in the future

7.2. Capability Augmentation Priorities

The work was deliberately scoped (Section 1) to address the development of this capability and its deployment on one system and targeting a single ASC relevant application. All of the *not in scope* work items are intended to be Future Capability Augmentation Priorities. Those items are repeated below:

Not in Milestone Scope: Future Capability Augmentation Priorities

- Tuning system parameters to avoid application performance variation
- Deriving causality of application performance variation
- Correlating system state with application performance

- Determining best system or application data to collect
- Production-hardened deployment of collection infrastructure / analysis

Particular priorities to improve the architecture and to enable better metric selection and associated analyses identified by the team are listed below. These are broken down by the various conceptual elements of the work.

Architecture:

- Explore additional lightweight methods for sampling of Kokkos kernel execution information
  - Self adjusting data volume production
  - User-controlled variable sampling rate and always sampling specified kernels

Metric Selection:

- Add PAPI events/metrics to analyses and dashboards
- Define metrics for, and implement, performance bottleneck detection

Visualization and Analysis:

- Analyses with both application and system data to automatically identify correlations
- Advanced analyses, such as rank clustering or historical variance investigation, of application data

8. LESSONS LEARNED

Throughout the course of this L2 milestone, we encountered some unexpected primary challenges and several lessons learned. The unexpected challenges we faced were not technical in nature but pertained primarily to team dynamics, specifically communication and engagement.

Communication in the all-virtual environment is difficult by default, but we faced team communication challenges that were also related to the size and organizational diversity of the team and the complexity and interdependences of the project. Specifically,

- Because of the breadth of the technologies involved, not all team members were intimately familiar with the technical issues of various sub-components of the project.

  Mitigation: Initiating monthly technical deep-dive presentations on various aspects of application and system monitoring.

- More communication among the sub-groups is needed to increase engagement and overall project knowledge.

  Mitigation: Regular synchronous and asynchronous status updates. We need large project distributed tools for organization, communication, and asynchronous status. Potential tools are currently being investigated.

Several lessons were learned during project execution that we aim to improve upon in future development and deployment of the monitoring system. Specifically:

- Large projects should have a single point of contact for final design decisions and these decisions should be presented and explained to the team in a timely manner.
- It is vitally important to document, with detailed diagrams, sub-component descriptions, functionality descriptions even seemingly simple project details (e.g., network diagrams, expected functionalities of sub-components and their interfaces and interactions with others).
- Staff training on data analysis would speed problem diagnosis and mitigation and tool/infrastructure validation. Note that we will address this in a system monitoring deep-dive presentation.

These issues have been discussed among the milestone management team and will be part of a whole-team discussion in the future to ensure that we define adequate solutions to mitigate problems within the project as we move into the future.

9. CAPABILITY PRODUCTION ROADMAP

In this section, we present our multi-year roadmap for development of new features and production deployment on systems at Sandia. Features include the priority items from Section 7 and from the *not in scope* items in Section 1. These roadmap diagrams highlight key events in the timelines of:

- System Data Collection - includes production installations
- Application Data Collection - includes application information via system sources (e.g., PAPI) and application information via Kokkos and non-Kokkos enabled applications
- Monitoring Features - includes features of all aspects of the architecture (e.g., data injection, dashboards etc.)

Figure 9-1 covers years FY20-22. FY20 is included to enable identification of the new capabilities developed as part of this milestone. FY22 shows the intended enhancements of the demonstrated capabilities, given the success of this milestone. Details are presented in the figures, but highlights of these years are as follows:

- *FY20* - LDMS version without *LDMS Streams* in production on SRN production systems. Application data restricted to that available from the system. System data pipeline to non-distributed *SOS* database with production dashboards for system data.
- *FY21 (milestone capabilities indicated by check marks)* - LDMS version *with LDMS Streams* in production on CTS-1 system with higher frequency data collection than in previous year. Application data injection from Kokkos via *Kokkos-LDMS Connector*. *Distributed DSOS* database increases scalability and performance. Dashboards include application and system data displayed concurrently and time-aligned.

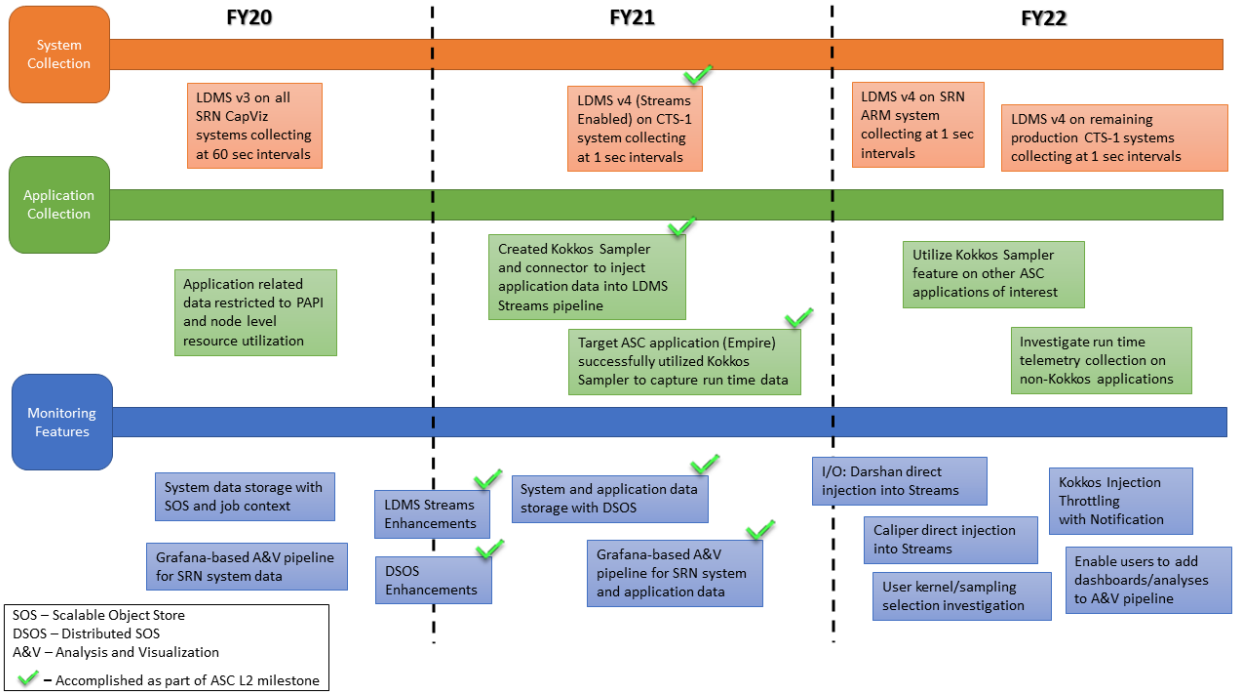


Figure 9-1 Roadmap - Part1: FY20-22

Initiated in this year, but not part of the milestone, is a collaboration with Argonne National Laboratory (ANL) to inject I/O data from their Darshan [5] tool into the data stream via the *LDMS Streams* capability. This will provide Sandia with additional IO data and will provide Darshan the ability to provide run time output while bounding its memory footprint to do so.

- *FY22* - Expansion of deployment to other CTS-1 systems and Sandia's Restricted Network (SRN) ARM system. The application base will be expanded beyond EMPIRE. Collecting application data via non-Kokkos applications will be investigated, most notably by a collaboration with Lawrence Livermore National Laboratory (LLNL) to inject data from their Caliper [3] tool directly into the *LDMS Streams*. Additional features for the architecture and dashboards are shown in the figure.

  Expansions of the capabilities are also listed as Planned Activities for FY22 in the ASC Implementation Plan. Of particular note are training and documentation resources about the use of these capabilities.

Figure 9-2 covers years FY23 and beyond. Details are presented in the figures, but highlights of these years are as follows:

- *FY23* - Deployment on CTS-2 and roadmap for Sandia's Secure Network (SCN) machines. Particular high-profile applications are targeted in this year. Increased analytic capabilities including Machine Learning (ML) and statistical models, based on our active research, are also planned.

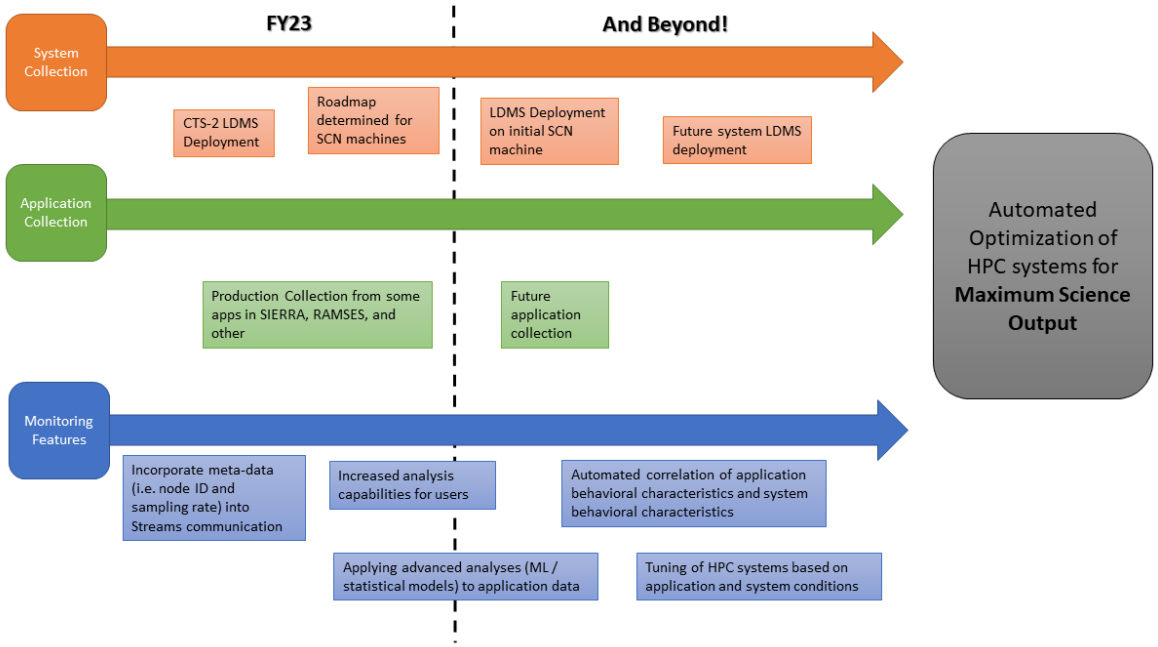


Figure 9-2 Roadmap - Part2: FY23 and beyond

- *Beyond* - While we have enabled unprecedented insight into application progress in conjunction with system conditions, the ultimate goal of this work is **Automated Optimization of HPC Systems for Maximum Science Output driven by run time monitoring and analysis**. We will be continuing work toward automated analysis and dynamic feedback in future years.

It should be noted that while the capabilities documented here were performed in execution of an ASC level 2 milestone, as part of this work we have also production hardened these capabilities and they will continue to be operated and offered to users and system administrators as a production capability. Note also that FOUS is committed to the deployment and support of the infrastructure on Sandia's systems. This includes both HPC platform support and supporting subsystems, such as the analysis and visualization clusters, which are maintained by FOUS. An eventual goal is the interactivity of code teams' dashboards, often currently maintained by CSSE, with the dashboards in this work. Long-term architectural and platform support will be explored as the functionality is expanded to multiple clusters.

10. COMPLETION CRITERIA

The Milestone text and Completion Criteria are listed in Section 1. In this section, we provide a checklist breaking out the key features of the milestone capabilities, as provided in the description in Section 2, and a summary of the evidence of its deployment and demonstration. Associated Figures and Sections are also referenced in the checklist.

☑ *Successful deployment of infrastructure on CTS-1 system*

Target version of LDMS (i.e., Streams enabled) has been in continuous deployment on the 1500 node CTS-1 System, ***Eclipse*** since Jan 28, 2021

☑ *Demonstration of capability for continuous* ***collection and storage*** *of system data over a 2-week rolling window*

We have demonstrated continuous collection and storage over a 30-day (2 x 2 weeks) window of system data on *Eclipse*

- A 30-day window produced ∼60TB (including indexing overhead) of data stored in NVMe-based Scalable Object Store (SOS) databases distributed across 14 nodes of the Shirley Monitoring and Analysis cluster. This is < 10% of the NVMe storage capability of Shirley.
- Rolling window previously demonstrated on a single SOS database on our Bitzer system

☑ *Identification of an* ***application throughput*** *metric(s) for an ASC-relevant code*

Throughput indicated by the total number of Kokkos kernel executions per-second over a defined time window while running the ***EMPIRE*** application. Shown in Figure 6-2

☑ *Identification of an* ***application progress*** *metric for an ASC-relevant code*

Number of kernel calls per unit time over a defined time window (15 sec. default) for a kernel indicative of science work accomplished (*ParticleMove::Move*)

☑ *Demonstration of capability on target application (*EMPIRE*) run(s) on CTS-1 system (*Eclipse*)*

Demonstrated 32- to 290-node *EMPIRE* application runs on *Eclipse*

☑ *Demonstrate a visualization interface that will enable a user to look at post-run application progress in conjunction with system conditions*

Shown statically in Figure 6-1

☑ *Demonstrate a visualization interface that will enable a user to look at run time application progress in conjunction with system conditions*

Shown statically in Figure 6-1

Run time dashboard capabilities were demonstrated in a video in the final review of milestone

☑ *Document feedback and future work*

Documented in Section 7 and Section 9

REFERENCES

[1] Ovis/LDMS. http://github.com/ovis-hpc/ovis.

[2] 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 *SC14: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 154–165, 2014.

[3] David Boehme, Todd Gamblin, David Beckingsale, Peer-Timo Bremer, Alfredo Gimenez, Matthew LeGendre, Olga Pearce, and Martin Schulz. Caliper: Performance introspection for hpc software stacks. In *SC '16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 550–560, 2016.

[4] J. Brandt, J. Cook, A. Gentile, et al. Enabling application and system data fusion. Sandia National Laboratories Report SAND2021-4475 C, 2021.

[5] P. Carns, R. Latham, R. Ross, K. Iskra, S. Lang, and K. Riley. 24/7 characterization of petascale I/O workloads. In *Proc. 2009 Workshop on Interfaces and Architectures for Scientific Data Storage*, 2009.

[6] C.R. Trott, et al. Kokkos 3: Programming model extensions for the exascale era. *IEEE Transactions on Parallel and Distributed Systems*, to be published. https://doi.ieeecomputersociety.org/10.1109/TPDS.2021.3097283.

[7] H. Carter Edwards, Christian R. Trott, and Daniel Sunderland. Kokkos: Enabling manycore performance portability through polymorphic memory access patterns. *Journal of Parallel and Distributed Computing*, 74(12):3202 – 3216, 2014. Domain-Specific Languages and High-Level Frameworks for High-Performance Computing.

[8] Simon D. Hammond, Christian R. Trott, Daniel Ibanez, and Daniel Sunderland. Profiling and debugging support for the kokkos programming model. In Rio Yokota, Michèle Weiland, John Shalf, and Sadaf Alam, editors, *High Performance Computing*, pages 743–754, Cham, 2018. Springer International Publishing.

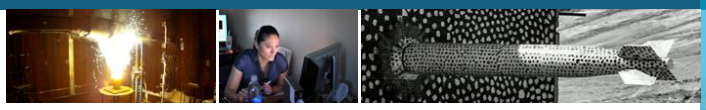
[9] Martin Meuer. SNL/NNSA CTS-1 Eclipse. https://www.top500.org/system/179422/.

[10] Benjamin Schwaller, Nick Tucker, Tom Tucker, Benjamin Allan, and Jim Brandt. HPC System Data Pipeline to Enable Meaningful Insights through Analytic-Driven Visualizations. In *Proc. 2020 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 433–441. IEEE, 2020.

INITIAL COMMITTEE REVIEW

Sandia National Laboratories

Integrated System and Application Continuous Performance Monitoring and Analysis Capability



12/9/2020

Initial L2 Milestone Review

U.S. DEPARTMENT OF ENERGY NNSA

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

2 L2 Milestone (Enabling Capability)

Milestone Description

- Demonstrate the use of SNL data collection, analysis, and visualization framework/tools, to provide both system and application relevant run-time and post-run information for a rolling two-week interval
- Deploy on a Sandia production SRN platform
- Demonstrate a capability for continuous collection of system data, an application progress metric(s), and an application throughput metric for an ASC-relevant code
- Provide a capability to store this data and a visualization interface that will enable a user to look at application progress in conjunction with system conditions, both at run time and post-run

Milestone Targets

- SNL's Lightweight Distributed Metric Service (LDMS) for the transport and aggregation of Trilinos-enabled application progress data and of system data
- ATDM Application EMPIRE for deployment and its Proxy, MiniEM, for capability development
  - CSSE's Application Performance Team will be supporting development and testing

3 Motivation / Impact

Answering key performance questions

- Is my performance variation due to system conditions or code changes (users)?
- How can I know if the system is having problems (system managers and users)?
- What are the architectural requirements given the site's workloads (acquisitions teams)?
- How can the system provide more effective and efficient services (architects, system managers, and support staff)?

Current issues affecting the ASC program

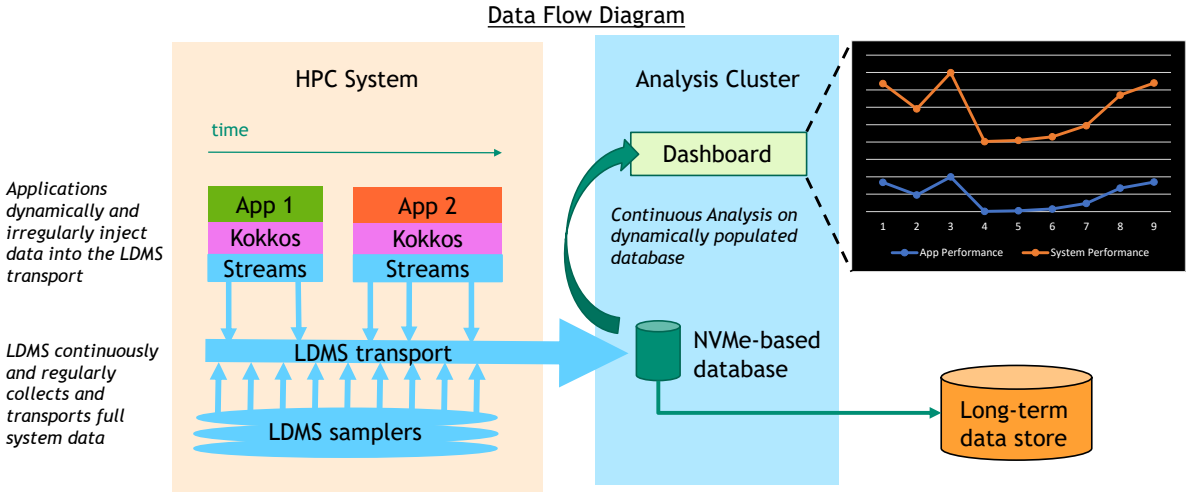
- Unknown sources of code performance variation, degradation, and runtime failure lead to longer development cycles and high machine and human resource costs.
- More insight into workload needs and causes of performance variation and degradation is required to optimize new machine design

Impact of this work on application teams and system administration at Sandia

- Application teams can better diagnose causes of performance variation or failure and determine whether they are application or system related.
- System administrators can better understand application behaviors and system resource needs as well as system issues and how they are affecting application performance.

4

Integrated System and Application Continuous Performance Monitoring and Analysis Capability



5 Completion Criteria

Stated L2 Completion Criteria

- Successful deployment of infrastructure on CTS-1 (Eclipse) system
- Demonstration of capability on target application run(s) on CTS-1 system
- Lessons learned and feedback from stakeholders for future capability augmentation and priorities will be documented

Scope Definition

- In scope
  - Developing and deploying an integrated architecture for application and system information
  - Collecting application and system state metrics from a CTS-1 system at runtime
  - Providing a useful visual interface for derived application performance and throughput metrics alongside system metrics
  - Demonstrating this interface on runtime CTS-1 data with the ability to do historical investigation up to two weeks
  - Providing information on instrumentation overhead and application performance impact
- Not in scope (future capability augmentation)
  - Tuning system parameters to avoid application performance variation
  - Deriving causality of application performance variation
  - Correlating system state with application performance
  - Determining best system or application data to collect
  - Production-hardened deployment of collection infrastructure / analysis

6

Interacting Milestone Sub-Groups

Application and System Metrics:

- Determine meaningful information regarding performance, progress, and throughput information of interest to the stakeholders.
- Determine application and architecture specific hardware counters and software data sources to transform into information.

Application-Streams Architecture:

- Utilize and enhance LDMS to support efficient and scalable runtime transport of both application and system data.
- Efficient application timing and performance data will be injected into the LDMS transport by Kokkos/Trilinos.

Analysis and Visualization:

- Convert raw application timing and performance data plus system data into meaningful visualizations through the development of Python analyses and Grafana dashboards using existing system-data visualization pipelines.

Deployment:

- Partnership with Eclipse admins and Advanced Architecture Testbeds team to identify and address all requirements for deployment of the full L2 architecture.
- Partner with LDMS developers to ensure necessary features. Deploy architecture on all platforms and infrastructure.

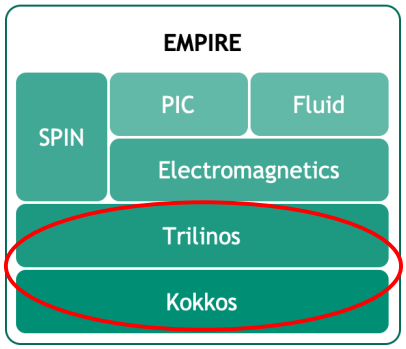
20 people across: LDMS, Application Performance, Kokkos, Trilinos, CapViz, and EMPIRE Teams

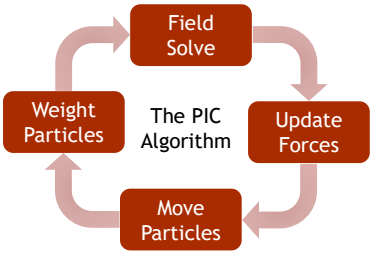

Application and System Metrics
Lead: Jeanine Cook
Stan Moore - SNL

8

Application and System Metrics: EMPIRE Background

PIC (particle-in-cell) is EMPIRE's most mature capability





Particle-In-Cell

Code developed from ground-up under ATDM

Field Solve

Built on Trilinos components for rapid development progress

Diagnostics are cross cutting

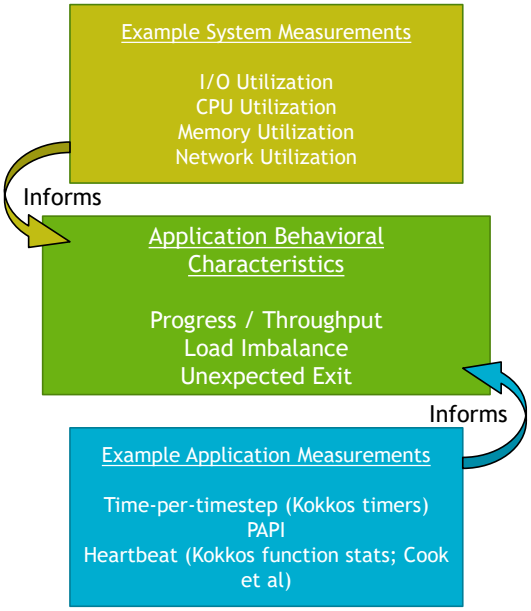
Kokkos is foundational to all parts of the code

Application and System Metrics: First Order Questions

The fusion of system and application metrics on the same display can provide deeper understanding of application behavioral characteristics

Example questions that would benefit from application-system data fusion

- Is a change in application progress observed with a simultaneous change in I/O, memory, or network utilization?
- Can load imbalance among ranks in the application be seen by inconsistent CPU and/or memory utilization across nodes?
- Can application heartbeat help narrow location of unexpected exit or behavior of interest?
  - Application heartbeat is a regular application feature that marks an interval of note



10 Application and System Metrics: Plan of Action

Identify which system and application metrics should be analyzed and integrated on the display to aid in understanding problems such as unexpected application termination and application performance variation

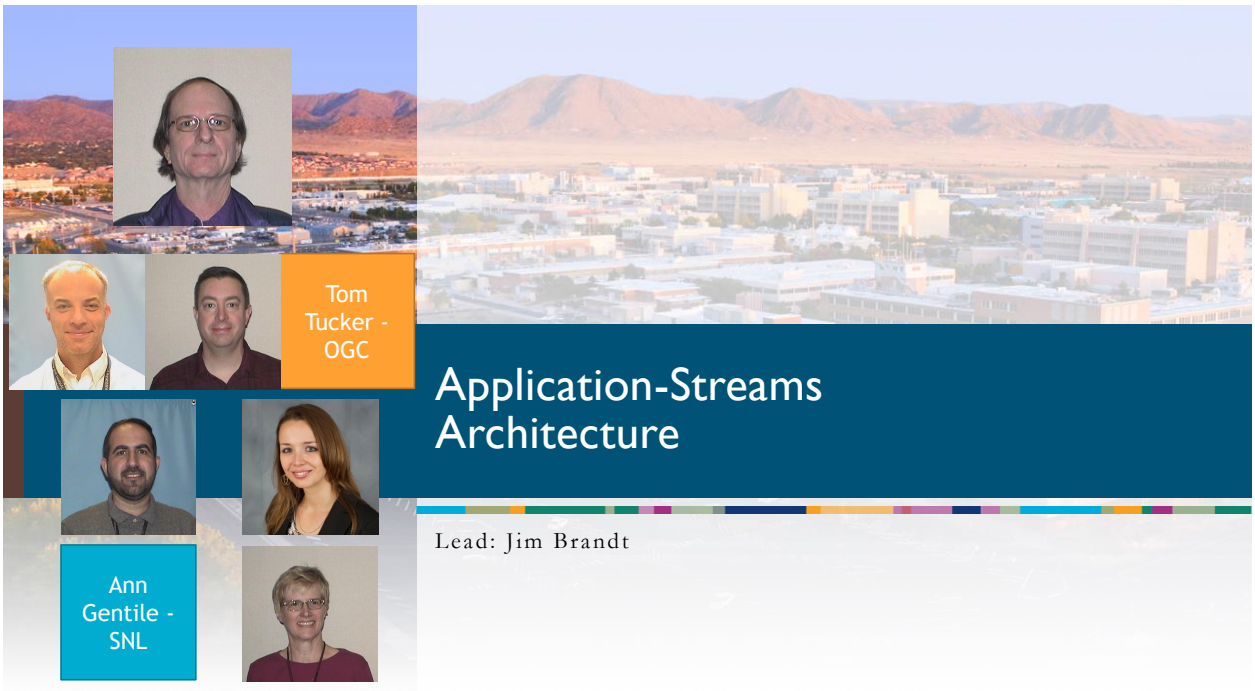
- Determine architecture-specific hardware performance counters and appropriate metrics that should be extracted
- Determine how to aggregate per rank data

Determine application "heartbeat" and progress indicators

- Utilize Teuchos and Kokkos timing and other application information

Collaborate with Deployment and Application-Streams Architecture subgroups to determine fidelity of collection, both necessary and possible, to produce meaningful results

Collaborate with Analysis and Visualization subgroup to determine ideal ways to present collected data

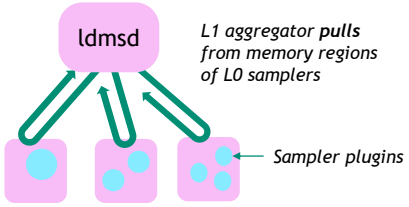

Tom Tucker - OGC
Ann Gentile - SNL
Application-Streams Architecture
Lead: Jim Brandt

12

Application-Streams Architecture: LDMS Background

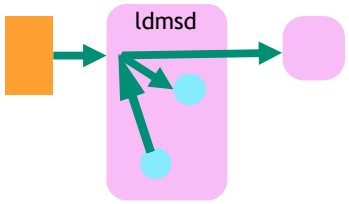
LDMS - low-overhead data collection, transport, and storage capability designed for continuous monitoring supporting runtime analytics and feedback.

- Typically pull based to minimize CPU interference



LDMS Streams – push-based publication of loosely formatted information to subscribers

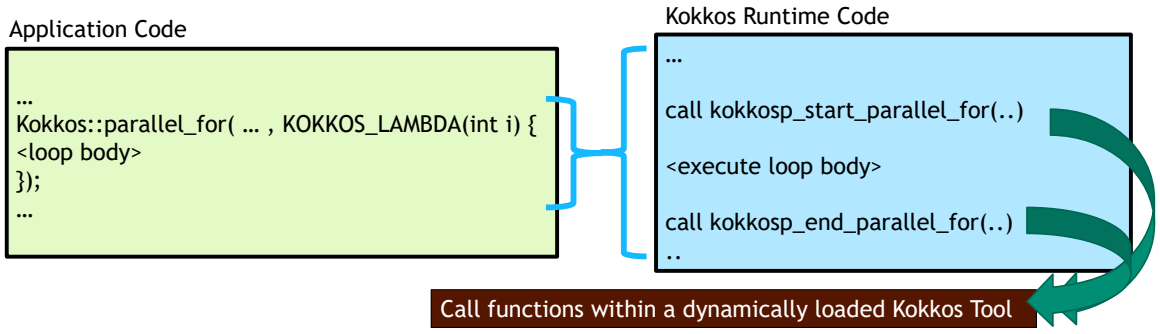
- LDMS Streams push is asynchronous



*Daemon publish API called from externally or by a plugin **pushes** to ldmsd which pushes to all subscribing plugins and aggregators*

13

Application-Streams Architecture: Kokkos Background



Kokkos Tool External Connectivity - kernels and Teuchos timers within Trilinos are configured to dynamically load a Kokkos supplied "connector". This requires no recompilation for profile enabled code and can be used for any Kokkos application (not just Trilinos, EMPIRE etc)

Kokkos Tool Internal Connectivity – hook points already exist for kernels (parallel-for, reduce, scan), "regions" (arbitrary points in code which can stack) and "sections" (arbitrary points in code which may overlap)

Application-Streams Architecture: Plan of Action

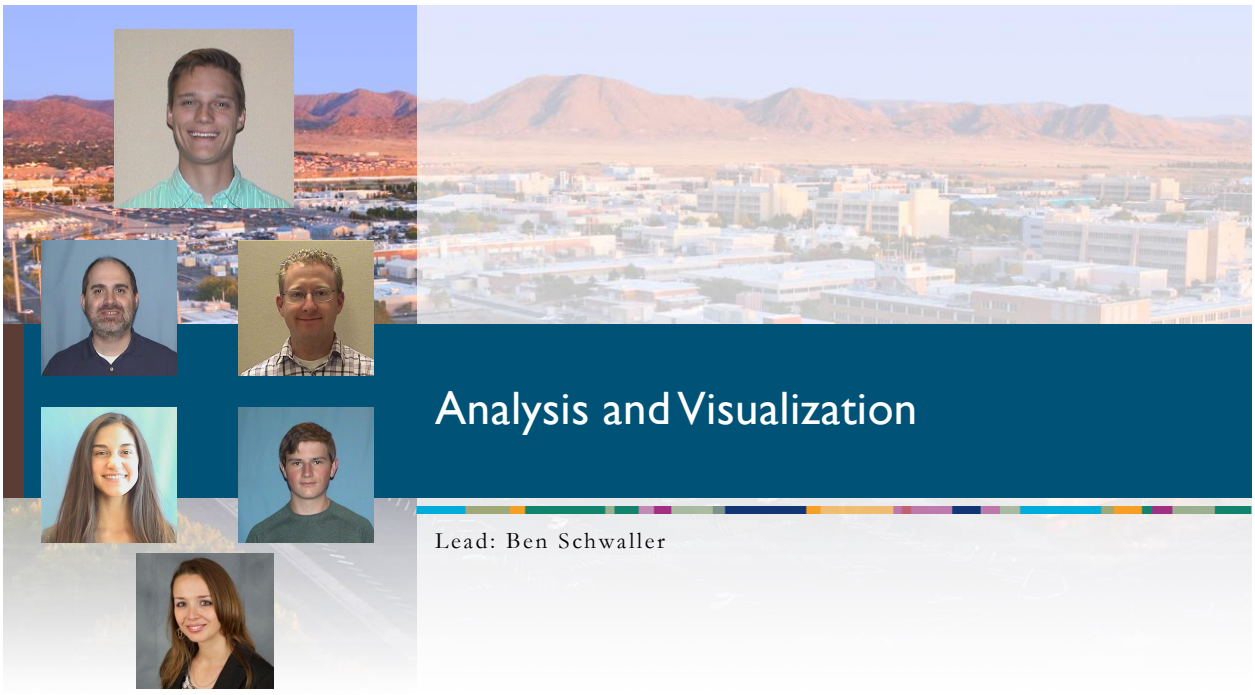
Create a Kokkos Connector to LDMS Streams
- Initial prototype is working on Sandia Voltrino Cray XC50 open ART system

Enable transport of the Kokkos data targeted from the Application and System Metrics Subgroup
- *Not* targeting full application timer traces

Enable dynamic discovery and adjustment of target information and fidelity of collection
- Significant diagnostic data and call stack may not be known a-priori

Scale LDMS Streams to meet application output requirements
- Enable N ranks to be writing individually
- Investigate rate-limiting feedback in the Streams architecture

Measure overhead of instrumentation and assess performance impact on application
- Report out worst-case scenarios and associated configurations



Analysis and Visualization

Lead: Ben Schwaller

16

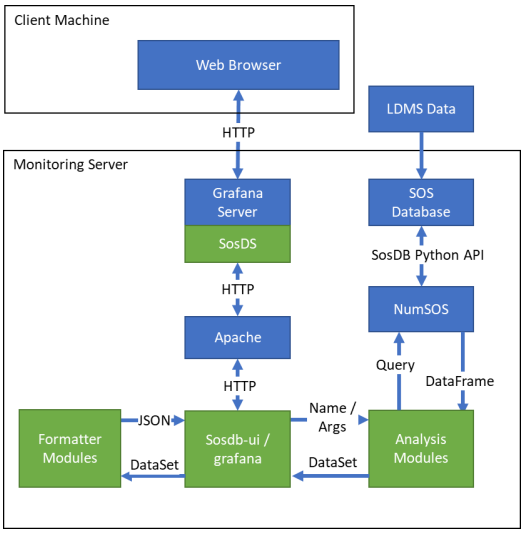
Analysis and Visualization: Background

9328 FY20 work created a Grafana interface for analyzing and visualizing SRN CTS system data collected by LDMS

Data can be directly queried from a database or have a python module perform analysis alongside the query

- Allows for flexible development of visualizations as analysis only happens during a query rather than over all data
- Any user with appropriate permissions can add and change analytics and create their own queries with analysis
- The Scalable Algorithms (1465) group has also performance analysis scripts which could be translated to python analysis modules

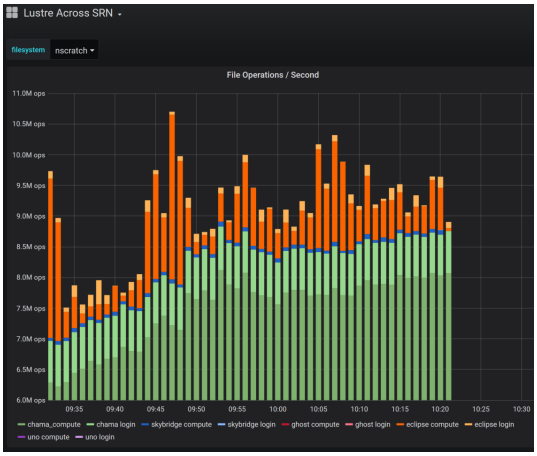
```
def _get_max(self,metrics):
    where_ = [ ['timestamp' , Sos.COND_GE, self.start ] ]
    if self.end > 0:
        where_.append(['timestamp', Sos.COND_LE, self.end ])
    self.src.select(['timestamp'] + metrics,
                    from_ = [ self.schema ],
                    where = where_,
                    order_by = 'time_job_comp'
        )

    self.xfrm = Transform(self.src, None, limit=self.mdp)
    resp = self.xfrm.begin()
    while resp is not None:
            resp = self.xfrm.next()
            if resp is not None:
                self.xfrm.concat()
    self.xfrm.mean(metrics)
    data = self.xfrm.pop()
    df = pd.DataFrame(data.tolist(), columns=metrics)
    res = DataSet()
    res.append_array(len(df.columns), 'metric', df.columns)
    res.append_array(len(df.iloc[0]), 'max', df.iloc[0])
    return res
```



17

Analysis and Visualization: Background

Current interface gives us a variety of insights into system status, from center-wide filesystem performance to high memory usage jobs on a system



Top 10 High Memory Jobs

Job ID	Node Name	Mem Used Ratio	Job Start	Job End
6996725	ec486	88.53%	2020-10-08 07:09:00	2020-10-08 07:56:00
6996690	ec1052	86.68%	2020-10-08 07:09:00	2020-10-08 07:56:00
6996722	ec428	85.58%	2020-10-08 07:09:00	2020-10-08 07:56:00
6996719	ec213	83.35%	2020-10-08 07:09:00	2020-10-08 07:56:00
6996755	ec931	81.61%	2020-10-08 07:09:00	2020-10-08 07:56:00
6996718	ec740	78.56%	2020-10-08 07:09:00	2020-10-08 07:56:00
6996754	ec635	78.45%	2020-10-08 07:09:00	2020-10-08 07:56:00
6996751	ec509	72.94%	2020-10-08 07:09:00	2020-10-08 07:56:00

Top 10 High Memory Idle Nodes

Node Name	Mem_Used_Ratio	Job Start	Job End
ec486	0.89	2020-10-08 07:09:00	2020-10-08 07:56:00
ec1052	0.87	2020-10-08 07:09:00	2020-10-08 07:56:00
ec428	0.86	2020-10-08 07:09:00	2020-10-08 07:56:00
ec213	0.83	2020-10-08 07:09:00	2020-10-08 07:56:00
ec931	0.82	2020-10-08 07:09:00	2020-10-08 07:56:00
ec740	0.79	2020-10-08 07:09:00	2020-10-08 07:56:00
ec635	0.78	2020-10-08 07:09:00	2020-10-08 07:56:00

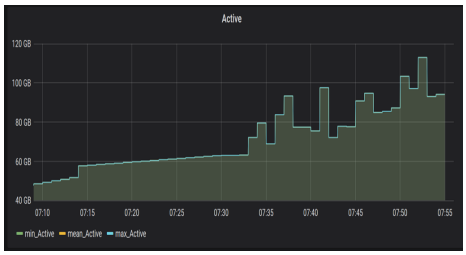
18

Analysis and Visualization: Plan of Action

Plot derived application performance metrics alongside relevant system performance metrics to produce dashboards that can provide a fused view of system performance and application performance

- Create Python analyses to parse application performance data and derive throughput and performance metrics
  - Analysis modules will be hosted and executed on an analysis cluster
  - Initial data retention policy is two weeks
- Create queries and dashboards that make intuitive sense to visualize the derived application metrics
  - Iterate with system administrators, analysts, and other subgroups to polish visualizations and dashboard navigations

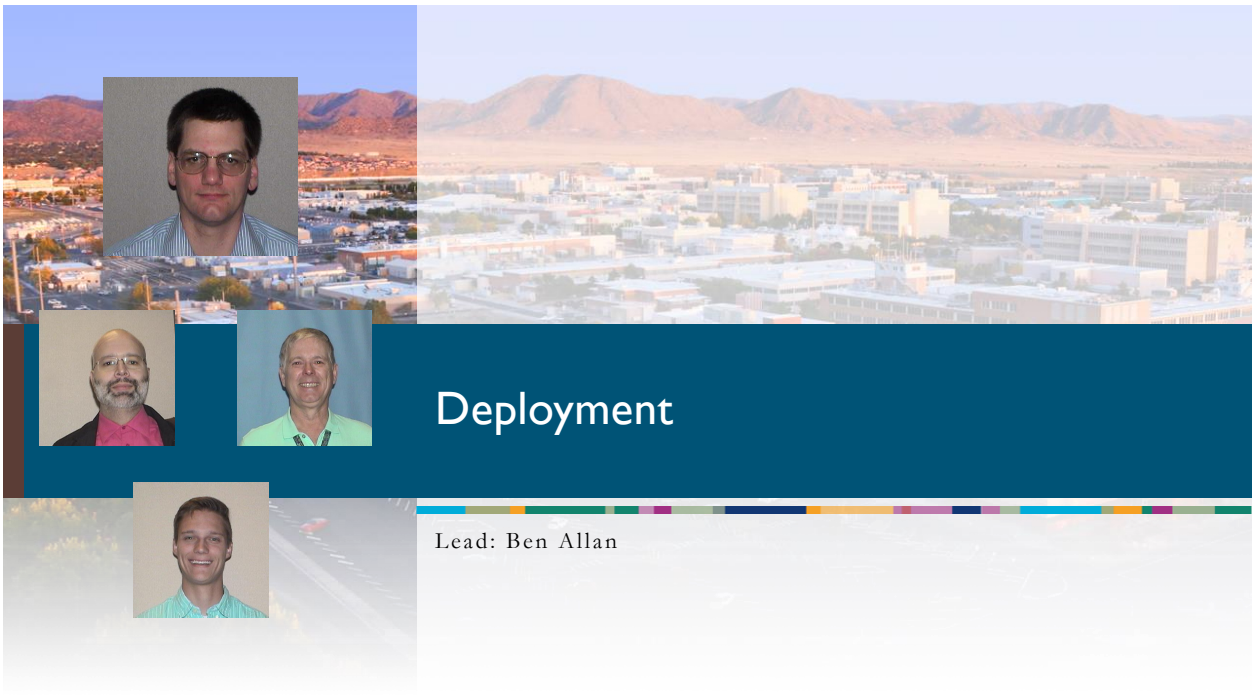
System Visualization



Timeseries plot of active memory of an application

Application Visualization

Timeseries plot of function/second for specific miniEM function by rank


Deployment
Lead: Ben Allan

20 Deployment: Background

CAPVIZ has previously deployed LDMS in production (2015-2020)

- Production deployment of the latest LDMS software to collect ***system metrics*** is part of our regular work for FY21 (orthogonal to the L2 milestone, but includes new Streams functionality).

Analyzing and visualizing system data

- Analysis cluster currently collects system data via LDMS once per minute from SRN clusters for interactive analysis & visualization development

21

Deployment: Plan of Action

Test deployment of latest version of LDMS on CTS-1 testbed

- Implement application and system data pipeline from CTS-1 testbed to analysis cluster

Deploy latest version of LDMS, with a functional Streams interface, on Eclipse

- Integrate LDMS in the production image for Eclipse
- Allow Streams interface to be written to by an application with data transported to storage
- Allow administrators to control what users / applications can write to the Streams interface

Ingest Eclipse data on analysis cluster and have the capacity to store and analyze data over a two-week window

- Deploy latest visualization framework on analysis cluster

Completion Criteria Discussion

23

Completion Criteria

Stated L2 Completion Criteria

- Successful deployment of infrastructure on CTS-1 (Eclipse) system
- Demonstration of capability on target application run(s) on CTS-1 system
- Lessons learned and feedback from stakeholders for future capability augmentation and priorities will be documented

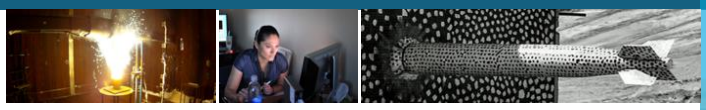
Scope Definition

- In scope
  - Developing and deploying an integrated architecture for application and system information
  - Collecting application and system state metrics from a CTS-1 system at runtime
  - Providing a useful visual interface for derived application performance and throughput metrics alongside system metrics
  - Demonstrating this interface on runtime CTS-1 data with the ability to do historical investigation up to two weeks
  - Providing information on instrumentation overhead and application performance impact
- Not in scope (future capability augmentation)
  - Tuning system parameters to avoid application performance variation
  - Deriving causality of application performance variation
  - Correlating system state with application performance
  - Determining best system or application data to collect
  - Production-hardened deployment of collection infrastructure / analysis

MIDYEAR COMMITTEE REVIEW

Sandia National Laboratories

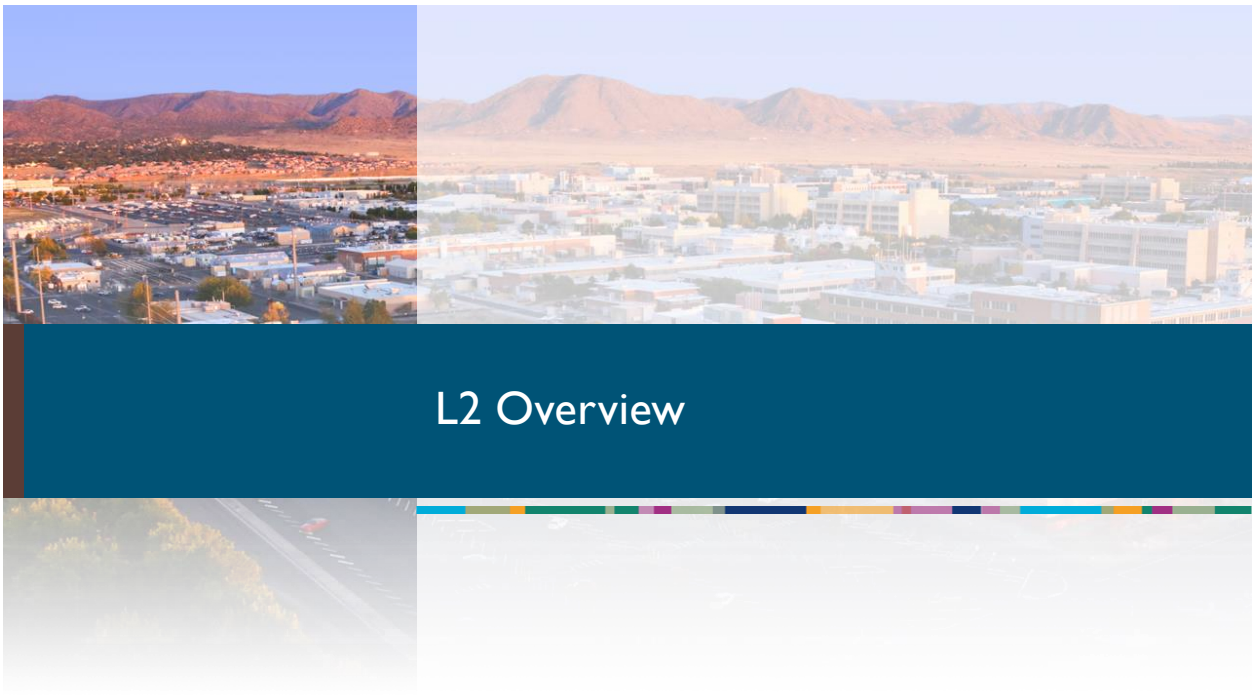
Integrated System and Application Continuous Performance Monitoring and Analysis Capability



4/6/2021

Midyear L2 Milestone Review

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.


L2 Overview

3

L2 Milestone Overview

Milestone Description

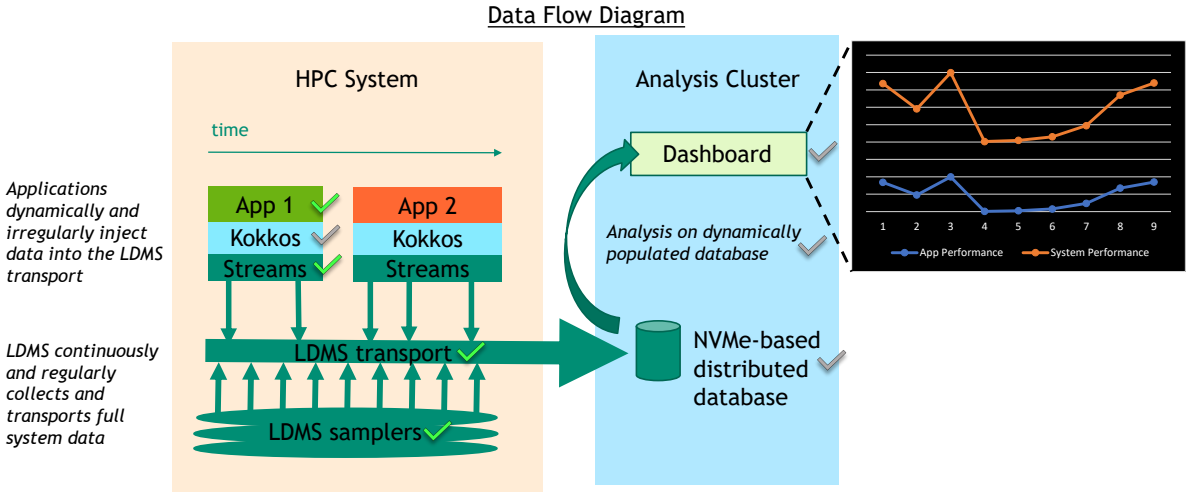
- Demonstrate the use of **SNL data collection, analysis, and visualization framework/tools**, to provide both **system and application relevant** run-time and post-run information for a rolling **two-week interval**
  - **Note: This does not imply a 2-week continuous application run**
- Deploy on a **Sandia production SRN** platform
- Demonstrate a capability for continuous collection of **system data**, an **application progress metric**(s), and an **application throughput metric for an ASC-relevant code**
- Provide a capability to **store this data and a visualization interface** that will enable a user to look at **application progress in conjunction with system conditions**, both at run time and post-run

Milestone Targets

- SNL's Lightweight Distributed Metric Service (LDMS) for the transport and aggregation of Trilinos-enabled application progress data and of system data
- ATDM Application EMPIRE for deployment and its Proxy, MiniEM, for capability development
  - CSSE's Application Performance Team will be supporting development and testing

4

Integrated System and Application Continuous Performance Monitoring and Analysis Capability




High-level Midyear Achievements
and Remaining Work

6

Eclipse DAT (part 1) – LDMS (Streams version) Overhead Testing

GOAL: validate low-overhead at 1Hz sampling intervals with real-world application mixes to enable running LDMS (Streams version) continuously on Eclipse in production

Took a DAT on CTS-1 system Eclipse on 1/29-1/30/2021 and conducted a series of tests over the course of 28 hours

Installed the LDMS version which enables the Streams application data collection across Eclipse

- Collected CPU, network, filesystem, hardware performance counters, and memory metrics at 1 Hz (~1700 metrics)
- Using analysis and visualization pipeline, displayed administrative network traffic to understand infrastructure impact of LDMS collection and storage

In close collaboration with 9326 HPCPRO team, ran a suite of applications across Eclipse to simulate typical HPC application workloads

- Observed acceptable overhead of < ~1% across the application suite with a full set of configured samplers running at a sampling interval of 1Hz

LDMS (Streams version) has remained running on Eclipse in production since the DAT

- Storage is performed on our production analytics cluster Bitzer

7 Eclipse DAT (part 2) – Streams Testing

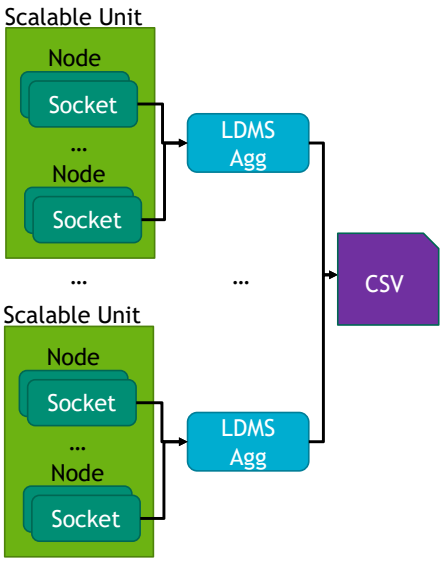
GOAL: Functionality Testing of Kokkos-to-LDMS Streams capability to inject application data into the LDMS Data Stream

Conducted test using ~1000 nodes and two applications: EMPIRE and SPARTA

- Goal was to test functionality first rather than performance tuning
- Demonstrated successful end-to-end pipeline of injecting application performance data via the Kokkos connector into the LDMS Stream on Eclipse and storing results to our analytics cluster

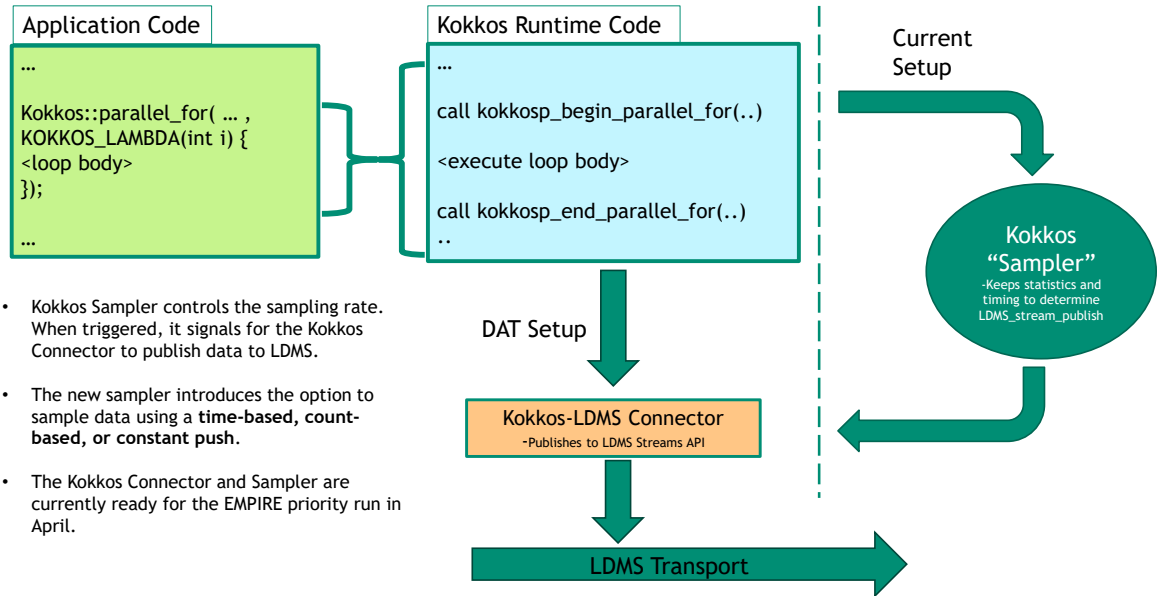
CSV format:

```
#rank,timestamp,job-id,kokkos-perf-data:time,kokkos-perf-data:type,kokkos-perf-data:name,kokkos-perf-data:count
0,100907.012310,8290750,0.000003,0,"Kokkos::View::initialization [Kokkos::Random_XorShift64::state]",2
0,100907.012360,8290750,0.000008,0,"Kokkos::View::initialization [DualView::modified_flags]",5
0,100907.012400,8290750,0.000014,0,"Kokkos::View::initialization [SurfCollide:nsingle]",4
```



8

Kokkos to LDMS publish



- Kokkos Sampler controls the sampling rate. When triggered, it signals for the Kokkos Connector to publish data to LDMS.
- The new sampler introduces the option to sample data using a **time-based, count-based, or constant push**.
- The Kokkos Connector and Sampler are currently ready for the EMPIRE priority run in April.

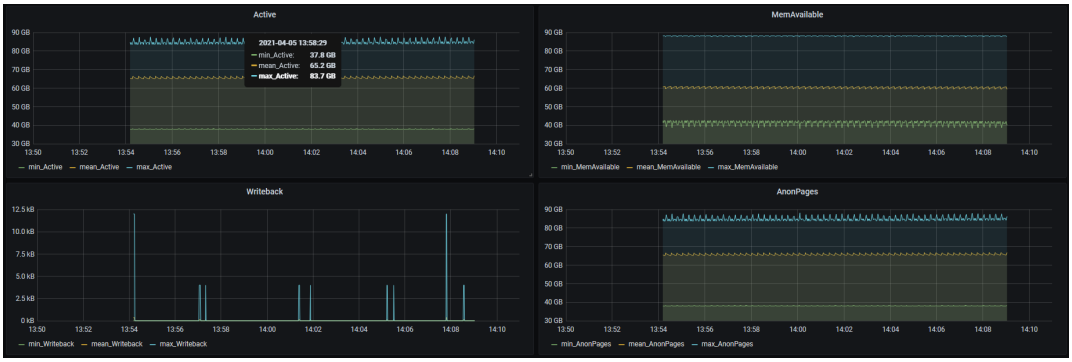
9

Dashboards Already Having an Impact!

Initial EMPIRE runs before milestone were trying to understand memory usage

- Lower node count runs were not working for a given problem size

Memory usage dashboard revealed that a minimum of 24 nodes were needed for the desired problem to fit into compute node memory



10

Remaining Work

On track to complete the end-to-end pipeline of application and system data collection, storage, analysis, and visualization

Remaining items:

- Improve storage side performance (e.g., format processing, ingest, write to store (media dependent))
- Extend back-end storage to include high-speed distributed database
  - Currently configured for CSV-based storage format
- Finalize Kokkos connector functionality via iterative test and refactor process
- Craft analyses and dashboards to present application and system data together during run time
- Demonstration of 2 week continuous ingest and display of system data along with EMPIRE progress/performance data during EMPIRE runs over the same period

Stretch Goal: Support application-data collection for milestone EMPIRE run evaluation

- Priority approved for 6 EMPIRE runs with 290 nodes (20% of Eclipse) of 7 days (in excess of the 96 hours of the long QoS). This also supports the gathering of high-resolution base lines including late-onset physics for EMPIRE milestone.
  - The first of the 6 runs will be performed the week of April 12, 2021


Stan Moore - SNL
Application and System Metrics
Lead: Jeanine Cook

Midyear Achievements

Initial specification of application data to be displayed in conjunction with performance counter and system data

- Application-sourced data supplied via **Kokkos** and injected into LDMS Stream: Time per timestep (EMPIRE)
- Determined relevant metrics to application progress and performance supplied via **LDMS samplers**:
  - Core-level hardware performance counter data: Instructions per cycle (IPC), L2/L3 cache misses per 1000 instructions (MPKI), % cycles throttled
  - Node-level data: Memory and CPU counters
  - Determination of data and rates tested at scale during EMPIRE DAT

Working with Analysis and Viz team to determine meaningful presentation of HW performance counter data and node-level data

Working with Architecture team to test and understand options for application-sourced data including timing, frequency, and events of interest

13

Remaining Work Items

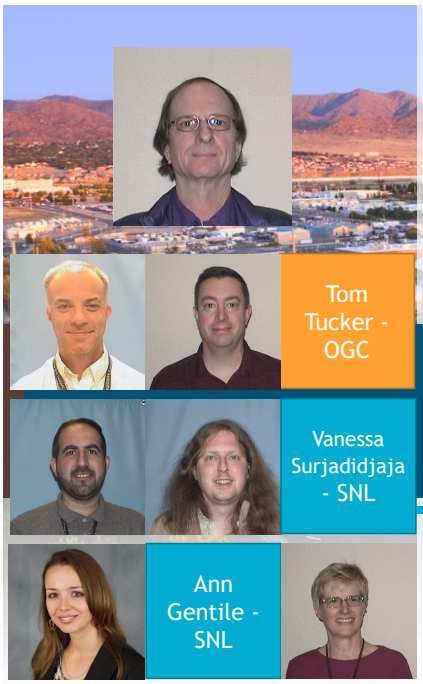
Finalize system data to be initially displayed with application and hardware performance counter data

REAL WORK IS COMING:

- Once Kokkos/Streams testing is done, we will define precisely which timers/metrics we will track/extract from application timing data
  - Work with analysis and visualization subgroup to finish initial analysis and visualization backend based on application and system data definition (above)
- Once the initial analysis and visualization backend is complete, this subgroup will be instrumental in using the backend on Eclipse with EMPIRE and giving feedback with respect to:
  - Actual data/metrics being collected and visualized
  - Implementation of the visualization backend

Post L2 follow-on work:

- Start the process of implementing performance bottleneck detection into run time analyses
- Expand application progress/performance metric collection and analysis to additional applications (e.g., Sierra)



Application-Streams Architecture

Lead: Jim Brandt

Midyear Achievements

Kokkos Streams interface has been refactored and is functional

- Incorporated a new **Kokkos sampler** that controls the flow of data between Kokkos and LDMS
  - Timer-based: will publish data for last kernel executing in defined time window (e.g. every 30 seconds or 1 min)
  - Count-based: will publish data for every Nth occurrence of a kernel execution
  - Constant: will publish data on all kernel executions
- Testing if Kokkos infrastructure can handle the maximum amount of data generated by a production application (SPARTA)

LDMS streams enhancements for JSON

- Only parse JSON when access to data is needed

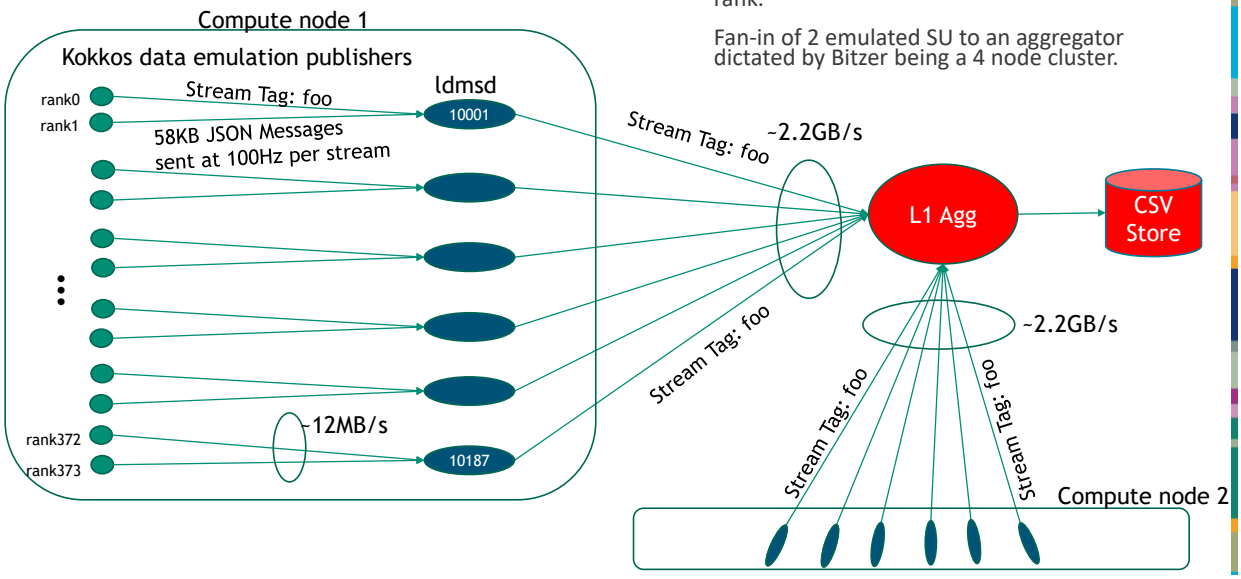
LDMS streams scale testing (see next slide)

Developed and deployed a LDMS store plugin that writes Kokkos kernel data to CSV store

Streams Scale Testing

Tested Streams on a testbed system to simulate Eclipse scale system without failure. 1000 kernel executions per 10ms per rank.

Fan-in of 2 emulated SU to an aggregator dictated by Bitzer being a 4 node cluster.



17 Remaining Work Items

The Kokkos-connector continues to evolve as we converge on the optimal set of information and volume of data

Development on efficient parsing of Kokkos kernel data for storage to both CSV and Database

Test performance and bottlenecks of LDMS / Kokkos pipeline at extreme application scale

- Tuning last level aggregator scale-out to ensure data storage can keep up with data being published at scale
- We will provide considerations for application injection performance in the final report

- **Post L2** follow-on work:
  - Other formats of stream data to improve extreme-scale performance?
  - Explore additional lightweight methods for sampling of Kokkos kernel execution information
    - Self adjusting data volume production

Analysis and Visualization

Lead: Ben Schwaller

19

Midyear Achievements

System Diagnostic Information:

Created dashboards and analyses to visualize ethernet load on administrative nodes on Eclipse during DAT

- Verified that the aggregate network load from LDMS was small when running continuously
- Could have high instantaneous load if all daemons restart simultaneously

Application-System Information:

Created dashboards and supporting analyses to visualize time-series of different derived metrics from PAPI counter data such as L2 misses per instruction

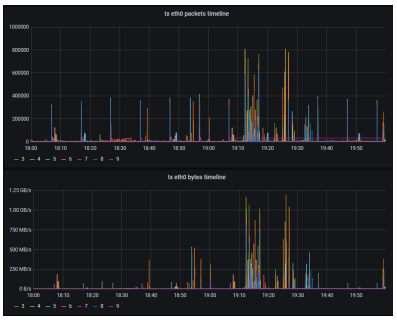
- PAPI data is one of the system metrics identified as useful to visualize alongside application data

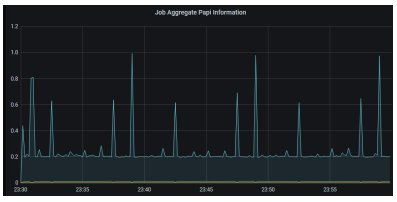
In progress of creating a **"scorecard"** report for every job run on Eclipse

- The scorecard will calculate a variety of summarization statistics about the job including avg CPU usage, memory usage, and filesystem bytes read and written
- Desired by code teams to better track their jobs at a glance
- This workflow, the first done in this infrastructure, can also be used for any analysis that is desired to be run across all new information

Wrote skeleton code for analyzing application data output and creating time / timestep time-series plots

- Waiting on Kokkos Streams code to be finalized





20 Remaining Work Items

Create analyses and dashboards to plot application and system data together

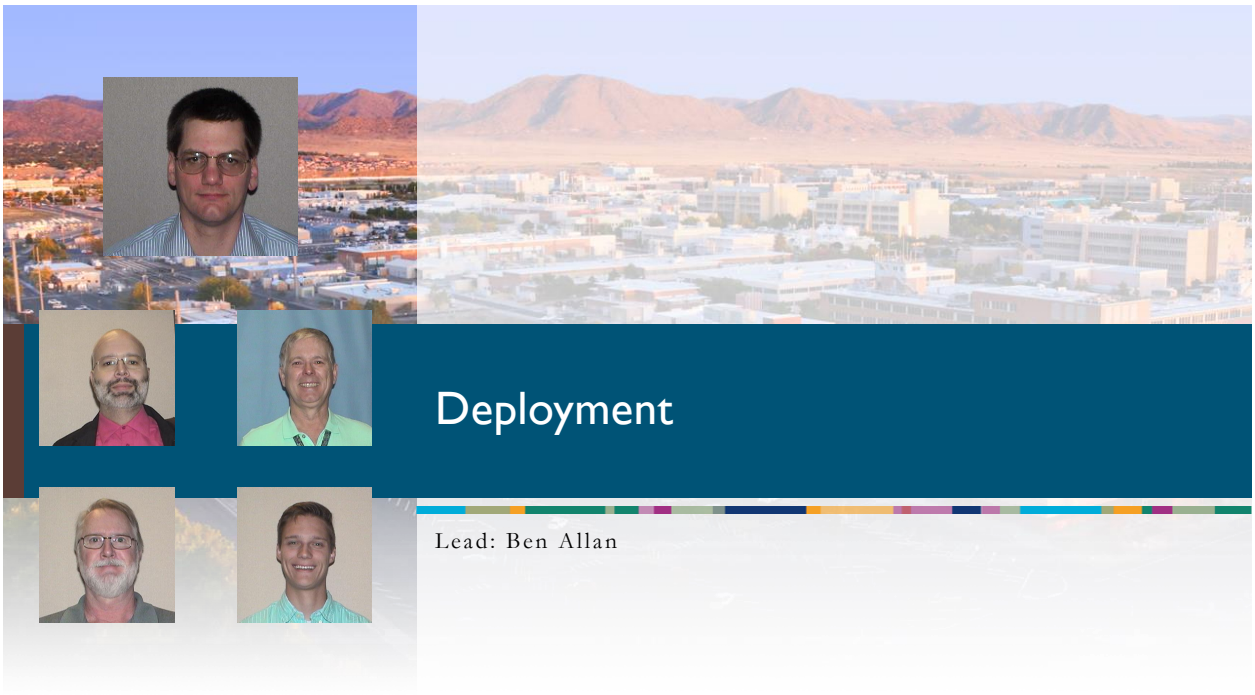
- Application visualization is awaiting finalized data from the Kokkos Streams infrastructure
  - Skeleton code and prototypes using old data have already been created
  - Desired derived metrics have already been discussed with the application team but further iteration will be necessary
- System data analysis and visualization is complete
  - Need to combine system and application analyses to create single panes with both app and sys data
- Iterate with code analysts / developers on the optimal layout of dashboards and their drill-down information to support workflows

Finish user scorecard infrastructure

- Iterate with users on desired dashboard layouts

Post-L2 work:

- Initial analysis with both application and system data to search for correlations
- Advanced analysis, such as rank clustering or historical variance investigation, of application data


Deployment
Lead: Ben Allan

Midyear Achievements

Production deployment of LDMS v4 using Omnipath RDMA transport

- In continuous use on Eclipse (L2 target platform) since Jan 28, 2021
  - Established network path >= 10Gb/s from CAPVIZ systems to monitoring systems.
  - Quantified the impact of 1Hz system metrics traffic on the administrative top-of-cluster fabric: ~3MB/sec/SU
- Enables use of **LDMS Streams interface** and hardware performance counters (syspapi)
- Provides a **broad set of data of interest** to system admins: Jobid, load average, CPU, memory, Ethernet, Omnipath, NFS, Lustre FS and Lustre networking, motherboard temperatures & power, aggregator daemon performance metrics

Demonstrated user selection of named LDMS compute node configurations via Slurm

- Provides users with the ability to select desired **admin-defined** LDMS configuration alongside their application run

Brought online the analysis cluster Bitzer which hosts the analysis and visualization engines and provides two-weeks of data storage

23 Remaining Work Items

Move user selection of compute node LDMS configuration into production

Work with OGC to deploy distributed database across the analytics cluster, Bitzer, to enable two-weeks worth of data to be stored in a single distributed database

Update the default group of per-core hardware counters to be collected when the syspapi sampler is active, based on needs determined by applications team

Switch to new Lustre FS data collection plugin from LLNL

(Post – L2): Demonstrate an administrative method for managing the LDMS Streams data flow from user applications to ensure minimum quality of service for other clients

- Critical administrative network clients: NFS, slurm, GPFS
- Investigate kernel-based controls and other approaches


Completion Criteria

25

Completion Criteria

L2 Completion Criteria

- Successful deployment of infrastructure on CTS-1 (Eclipse) system
- Demonstration of capability on target application run(s) on CTS-1 system
- Lessons learned and feedback from stakeholders for future capability augmentation and priorities will be documented

Scope Definition

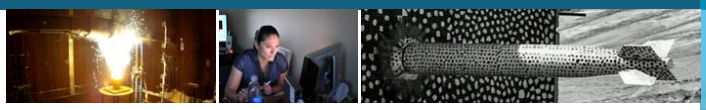
- In scope
  - Developing and deploying an integrated architecture for application and system information
  - Collecting application and system state metrics from a CTS-1 system at runtime
  - Providing a useful visual interface for derived application performance and throughput metrics alongside system metrics
  - Demonstrating this interface on runtime CTS-1 data with the ability to do historical investigation up to two weeks
  - Providing information on instrumentation overhead and application performance impact
- Not in scope (future capability augmentation)
  - Tuning system parameters to avoid application performance variation
  - Deriving causality of application performance variation
  - Correlating system state with application performance
  - Determining best system or application data to collect
  - Production-hardened deployment of collection infrastructure / analysis


Q&A / Discussion / Feedback

FINAL COMMITTEE REVIEW

Sandia National Laboratories

Integrated System and Application Continuous Performance Monitoring and Analysis Capability

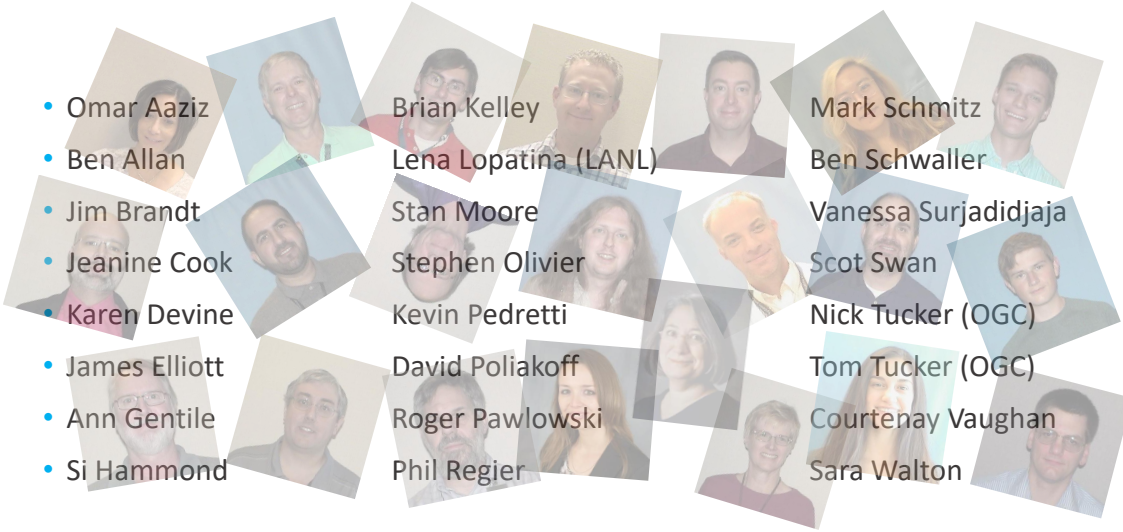


08/24/2021

Final L2 Milestone Review

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Multi-Center, Multi-Department, and Multi-Lab Effort with 24 Participants



- Omar Aaziz
- Ben Allan
- Jim Brandt
- Jeanine Cook
- Karen Devine
- James Elliott
- Ann Gentile
- Si Hammond
- Brian Kelley
- Lena Lopatina (LANL)
- Stan Moore
- Stephen Olivier
- Kevin Pedretti
- David Poliakoff
- Roger Pawlowski
- Phil Regier
- Mark Schmitz
- Ben Schwaller
- Vanessa Surjadidjaja
- Scot Swan
- Nick Tucker (OGC)
- Tom Tucker (OGC)
- Courtenay Vaughan
- Sara Walton

3

Outline

L2 Text and Completion Criteria

Overview: Motivation and Architecture

Detail
- Architecture
- Deployment
- Application and System Metrics
- Analysis and Visualization

Feedback & Future Work

Completion Criteria Checklist

Acknowledgements


L2 Overview & Completion Criteria

5

ASC FY21 IP Text

Description: This L2 milestone will demonstrate the use of SNL data collection, analysis, and visualization framework/tools, deployed on a Sandia production SRN platform, to provide both system and application relevant run-time and post-run information for a rolling 2-week interval. We will demonstrate a capability for continuous collection of system data, an application progress metric(s), and an application throughput metric for an ASC-relevant code. We will provide a capability to store this data and a visualization interface that will enable a user to look at application progress in conjunction with system conditions, both at run time and post-run.

We are targeting LDMS for the transport and aggregation of Trilinos-enabled application progress data and of system data. We are targeting the ATDM Application EMPIRE for deployment and its Proxy, MiniEM, for capability development. CSSE's Application Performance Team will be supporting development and testing.

Completion Criteria:

- Successful deployment of infrastructure on CTS-1 system.
- Demonstration of capability on target application run(s) on CTS-1 system.
- Lessons learned and feedback from stakeholders for future capability augmentation priorities will be documented.

6 L2 Milestone Overview

Milestone Description

- Demonstrate the use of **SNL data collection, analysis, and visualization framework/tools**, to provide both **system and application relevant** run-time and post-run information for a rolling **two-week interval**
  - **Note: This does not imply a 2-week continuous application run**
- Deploy on a **Sandia production SRN CTS-1** platform
- Demonstrate a capability for continuous collection of **system data**, an **application progress metric**(s), and an **application throughput metric for an ASC-relevant code**
- Provide a capability to **store this data and a visualization interface** that will enable a user to look at **application progress in conjunction with system conditions**, both at run time and post-run

Completion Criteria Checklist

1. Successful deployment of infrastructure on CTS-1 system
2. Demonstration of capability for continuous **collection and storage** of system data over a 2-week rolling window
3. Identification of an application performance metric(s) for an ASC-relevant code
4. Identification of an application throughput metric for an ASC-relevant code
5. Demonstration of capability on target application run(s) on CTS-1 system
6. Demonstrate a visualization interface that will enable a user to look at **post-run** application progress in conjunction with system conditions
7. Demonstrate a visualization interface that will enable a user to look at **run time** application progress in conjunction with system conditions
8. Document feedback and future work

8

Completion Criteria Scope Information

Scope Definition

- In scope
  - Developing and deploying an integrated architecture for application and system information
  - Collecting application and system state metrics from a CTS-1 system at runtime
  - Providing a useful visual interface for derived application performance and throughput metrics alongside system metrics
  - Demonstrating this interface on runtime CTS-1 data with the ability to do historical investigation up to two weeks
  - Providing information on instrumentation overhead and application performance impact
- Not in scope (future capability augmentation)
  - Tuning system parameters to avoid application performance variation
  - Deriving causality of application performance variation
  - Correlating system state with application performance
  - Determining best system or application data to collect
  - Production-hardened deployment of collection infrastructure / analysis


Overview: Motivation &
Architecture

Motivation for this Work

Urgent problem: Critical science results are being delayed due to inability to diagnose critical issues

- Currently, large-scale application runs (SNL production, Trinity) can have high performance variability or suffer failure for reasons often unknown
- Costly HPC resources are being wasted by applications that do not complete or exceed their estimated runtime

Solution provided by this milestone:

- Gain continuous insight into application performance in system context:
  - During run time via several pre-defined, intuitive, and user customizable visualizations
  - Post-run via visualization interface and access to complete application and system data storage
- Does not require code change or recompilation on the part of the user to collect this information

The Devil is in the Implementation Details

Tracking application progress/performance at scale is difficult at best but impossible in most cases using existing performance/profiling tools – significant disruption of application performance profile and/or application/tool crashes

Utilize low overhead accounting currently being performed in applications and periodically write timestamped results to system monitoring data store using the already installed LDMS monitoring framework for transport!

- Need to inject per-rank information into local LDMS daemon for scalability
  - What information will convey performance/progress and variation?
- Need to collect a subset of total information to minimize application overhead
- Need simple well defined information format for packing on application side and parsing on far end
- Need to defer parsing information to storage cluster

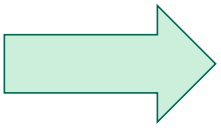
Kokkos Provides Required Instrumentation

Developers have already included instrumentation

Two different types of instrumentation are exposed:

- Kokkos native instrumentation (e.g., track kernel executions and timings)
- Teuchos timers

This telemetry can already be provided to the user in **files** as **periodic dumps or Post-Run**



LDMS Streams Provides Needed Transport Capability

LDMS Streams is a publish/subscribe push-based service provided as part of LDMS

- Support for both "string" and "JSON" data streams
- Originally developed to enable transport of SLURM job/step information to be bundled with traditional LDMS metric sets

Inject data as it is produced into the already deployed LDMS framework for **continuous access** by users and operations staff

13

Coupling Kokkos Instrumentation Capabilities With LDMS Scalable Transport and Storage

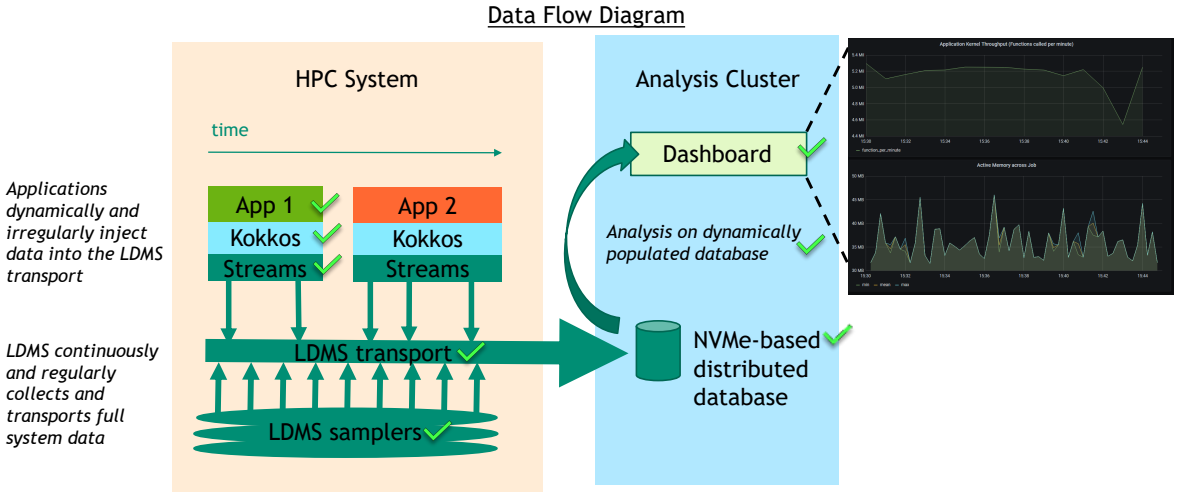
We chose to leverage existing Kokkos instrumentation capabilities and existing scalable LDMS publish/subscribe capability to enable:

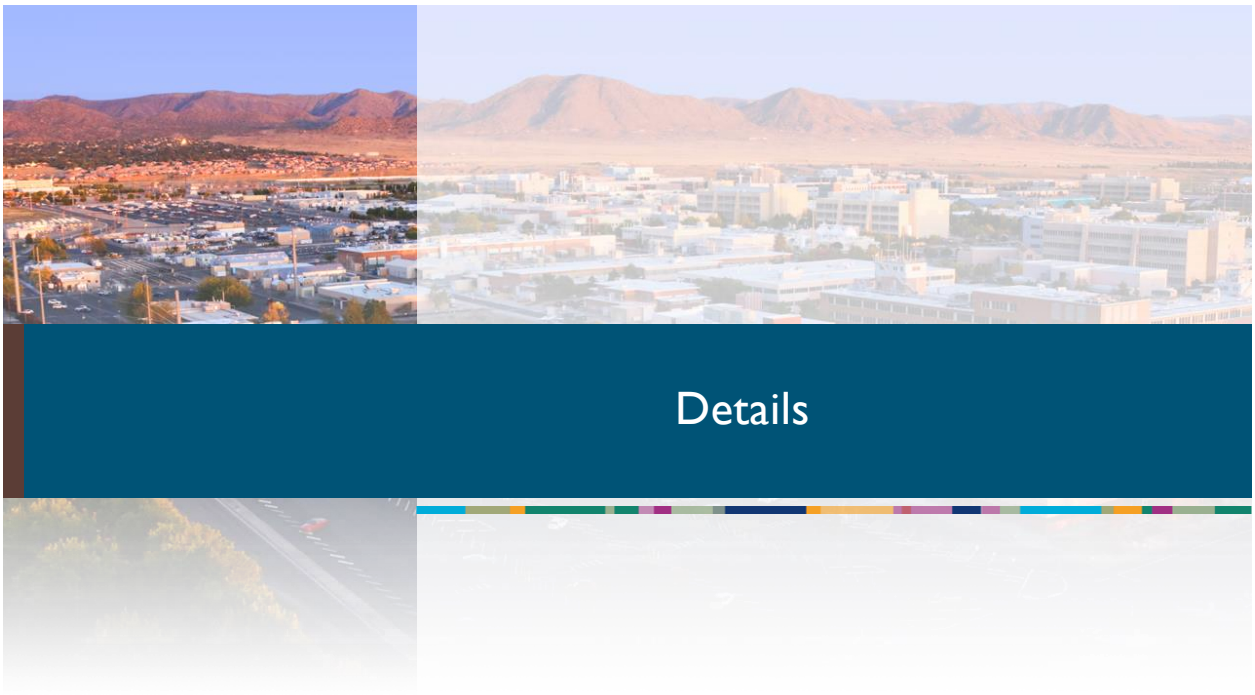
- Collecting performance event stream at system scale with low overhead
- Performing event data collection for long runs
- Publishing information to a scalable database to support analytics (run time and post processing)
- User interface for visualizing application data in a system context over long runs

Just need to publish application data to the LDMS Streams API, add store functionality in order to store application performance metrics to the same database as the system data, convert raw data to a progress/performance metric, and present to users…

14

Integrated System and Application Continuous Performance Monitoring and Analysis Capability





Details

Logical Subgroup Descriptions

Application-Streams Architecture

- Identify and implement mechanisms for per-rank publishing of Kokkos performance data, in JSON format, to the LDMS Streams API
- Ensure the LDMS Streams implementation, including aggregation and storage, is scalable and adds minimal and acceptable overhead to the running application

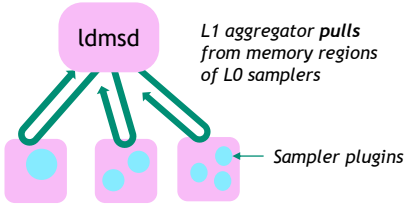
17 Enabling Application Data Injection via LDMS

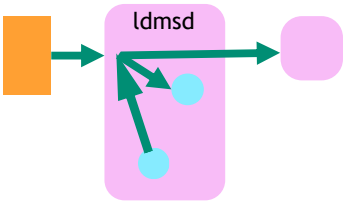
LDMS - **low-overhead (<1% application)** data collection, transport, and storage capability designed for **continuous monitoring** supporting **run time analytics** and feedback.

- System data collection is typically **synchronous** at regular (e.g., second or less) intervals
- **Structured** data format (i.e., metric set) designed to minimize data movement
- Transport is typically **pull** based to minimize CPU interference
- Transport to multiple arbitrary consumers over both RDMA and socket

LDMS Streams – **on demand publication** of loosely formatted information to subscribers

- Transport is **push** based and supports **asynchronous** event data (e.g. scheduler and log data)
- **Unstructured** data

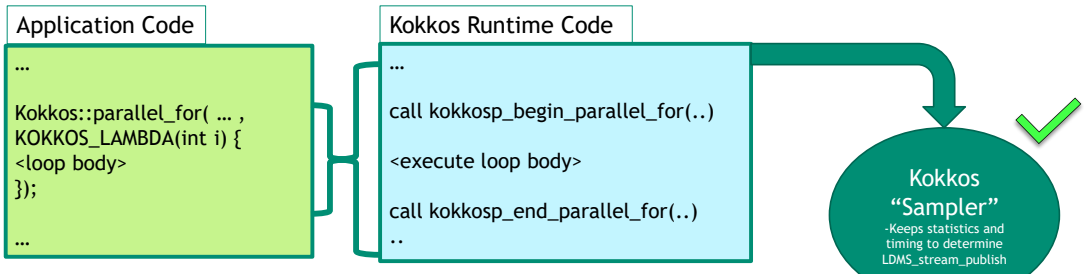




*Daemon publish API called from externally or by a plugin **pushes** to ldmsd which pushes to all subscribing plugins and aggregators*

18

Kokkos to LDMS publish



Kokkos Sampler controls the sampling rate. When triggered, it signals for the Kokkos Connector to publish data to LDMS.

The new sampler introduces the option to sample data using a **time-based, count-based, or constant push**.

#timestamp,job_id,rank,name,type,current_kernel_count,total_kernel_count,level, current_kernel_time,total_kernel_time
1627835612.086679,10195735,1,Kokkos::View::initialization [diagnostic:Solver Field:B_Field:temp],0,1218,57972687,0,0.000005,182.693422
1627835613.709526,10195735,1,TimeAverage::Continuous,0,24758,57972788,0,0.000006,182.693428
1627835616.787472,10195735,1,MigrateParticles::count,1,3540,57972889,0,0.000001,182.693430
1627835620.448333,10195735,1,SolverInterface::Apply Trivial BC,0,7512,57972990,0,0.000002,182.693432

19

Logical Subgroup Descriptions

Application-Streams Architecture

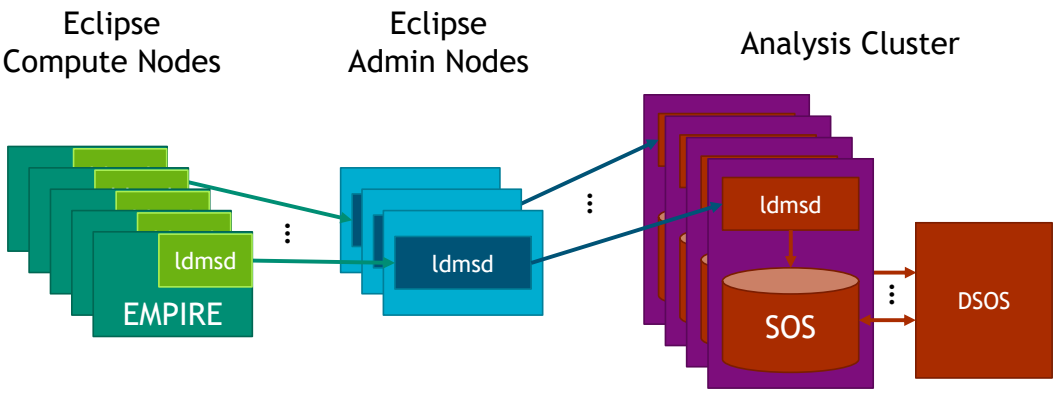
- Identify and implement mechanisms for per-rank publishing of Kokkos performance data, in JSON format, to the LDMS Streams API
- Ensure the LDMS Streams implementation, including aggregation and storage, is scalable and adds minimal and acceptable overhead to the running application

Deployment

- Continuous deployment of LDMS on Eclipse (CTS-1)
- Provisioning and stand-up of a monitoring and analytics cluster for continuous deployment of LDMS aggregators, storage of system and application data, and analysis and visualization of stored data

20

LDMS Eclipse Deployment Architecture



LDMS collects application and system data from Eclipse nodes and aggregates to our analysis cluster distributed database

Kokkos / LDMS Streams message sending was tested by sending a message every 10ms per rank across 2000 ranks without data loss

LDMS Eclipse Deployment Data

System data collected at 1 second intervals (~5,000 metrics per node or 650 billion data points per day)

- SLURM job, load average
- CPU & memory usage
- NFS & Lustre operations
- Ethernet & Omnipath traffic
- Lustre networking
- Motherboard temperatures & power
- Aggregator daemon performance
- Collector daemon memory use

Kokkos event stream data from each application MPI process

- Sampling ~1% of kernel executions (~20 events per rank per second or ~1 billion records per day)

Logical Subgroup Descriptions

Application-Streams Architecture

- Identify and implement mechanisms for per-rank publishing of Kokkos performance data, in JSON format, to the LDMS Streams API
- Ensure the LDMS Streams implementation, including aggregation and storage, is scalable and adds minimal and acceptable overhead to the running application

Deployment

- Continuous deployment of LDMS on Eclipse (CTS-1)
- Provisioning and stand-up of a monitoring and analytics cluster for continuous deployment of LDMS aggregators, storage of system and application data, and analysis and visualization of stored data

Application and System Metrics

- Determine metrics of interest for run-time and post run understanding of application progress and performance. These metrics need to be viewable in a system monitoring data context

Application and System Metrics of Interest

Progress metric: ParticleMove::Move - a kernel that represents science progress

- Number of kernel calls per second over a defined time window (15 sec. default)
  - This kernel gets called approximately once per second on each rank (**statistical approximation**)
- Time spent in the kernel from sample to sample provides insight into performance variation
- **Note** that since we are sampling, the data provides statistical estimates for both of these
- **Note** that this choice of kernel metric is the **users choice** and is not hard coded either for Kokkos or EMPIRE

Throughput metric: Number of kernel executions, across all application ranks, per minute over defined window (i.e., 60 seconds)

- This is approximately 5 million executions per minute for our 290 node runs

System metric: Active Memory is used as the system metric in these visualizations

- **Note** that our visualization engine provides the capability to choose any system metric over the full range of the ~5000 currently being collected

24

Logical Subgroup Descriptions

Application-Streams Architecture

- Identify and implement mechanisms for per-rank publishing of Kokkos performance data, in JSON format, to the LDMS Streams API
- Ensure the LDMS Streams implementation, including aggregation and storage, is scalable and adds minimal and acceptable overhead to the running application

Deployment

- Continuous deployment of LDMS on Eclipse (CTS-1)
- Provisioning and stand-up of a monitoring and analytics cluster for continuous deployment of LDMS aggregators, storage of system and application data, and analysis and visualization of stored data

Application and System Metrics

- Determine metrics of interest for run-time and post run understanding of application progress and performance. These metrics need to be viewable in a system monitoring data context

Analysis and Visualization

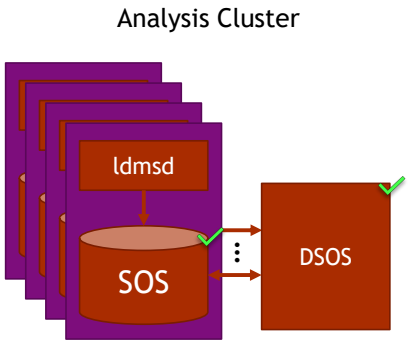
- **Identify and implement analyses** required to produce **appropriate application and system metrics** across the parallel store of system and application information
- **Implement a Grafana-based dashboard** to **enable user access** to application progress and performance metrics along with system monitoring metrics for both run time and post-run visualization

25

DSOS: Enabling Scalable Ingest and Queries for Analysis and Viz

Distributed Scalable Object Store (DSOS) is a scalable database with a variety of features which enable simultaneous large-scale data ingest and queries

- Designed specifically for large-scale HPC monitoring data ingest and query with flexibility to change and adapt as needs arise
- Coordinates databases across multiple devices and nodes to present a "single, unified" database to the end user
- High insert rate for continuous data collection
- Indices can be created or removed as needed for optimizing queries without reloading data
- Python, C, and C++ API and command line interface



Populated a DSOS database with ~1 month of system data and two week-long 290-node runs of EMPIRE for analysis and visualization

- Resulted in 50TB of system data and 900GB of application data
- EMPIRE got approval for 6 week-long 290-node runs (~20% of Eclipse)
  - This provided ample application data while also **supporting physics for EMPIRE milestones**

26 Analysis and Visualization Pipeline

User queries from Grafana dashboards are sent through a backend python application which can call python analyses to derive metrics from raw data

- In-query analyses save significant computation time/resources for creating analysis results
  - Only data of interest is analyzed and new analyses can be created without recreation of analysis results across the database

Python modules can query the database and return pandas DataFrames for analysis

- Significant work was done to optimize database queries and python analyses for fast Grafana query times

The backend application then takes DataFrames and formats them as JSON objects which Grafana can interpret

27 Analysis and Visualization Presentation Overview

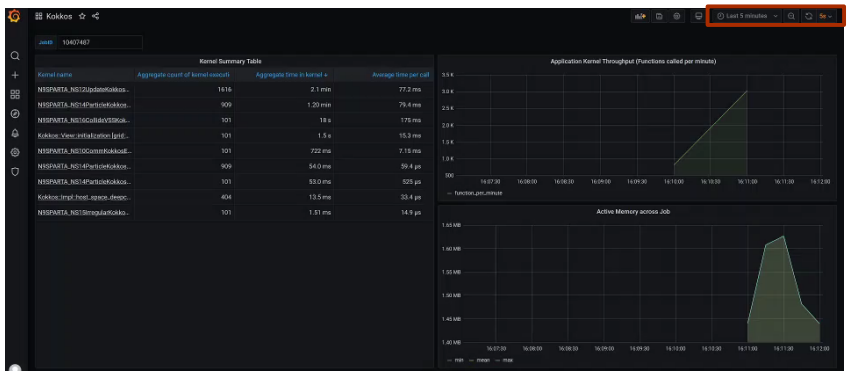
Created two Grafana dashboards to visualize an application's Kokkos data

- Job-level dashboard
- Kernel-level dashboard

Demonstrated analysis and visualization of both live and post-run data sets

- Video is of a simple 2-node SPARTA job at runtime
  - **Application kernel throughput**
  - **Active Memory**

Video of live feed of job's data
(5 second update intervals)

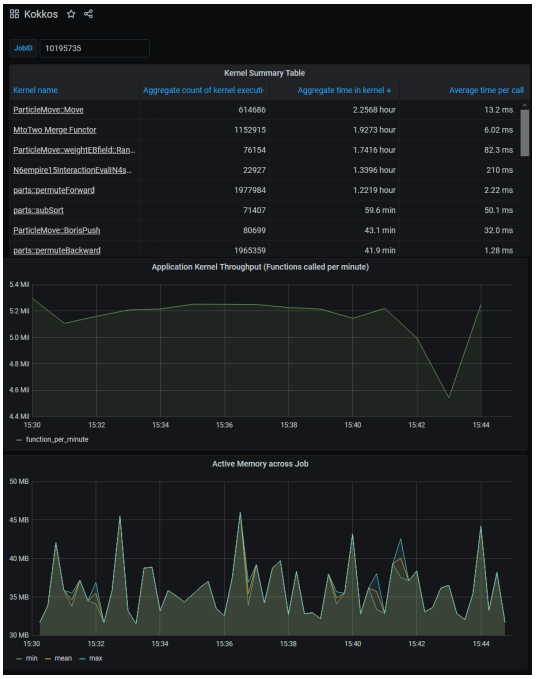


28

Analysis and Visualization

Job-level dashboard shows data from across the application and has 3 panels

- Kernel Summary Table which shows all kernels, their times called, time spent in kernel, and average kernel execution time, in the time range specified
  - Each kernel has a link to drilldown to the next dashboard
- Application Kernel **Throughput** which is a time series graph of how many kernels have executed per minute in the time range specified
- **Active Memory** across Job which shows the minimum, mean, and maximum memory usage of the nodes in the job over time

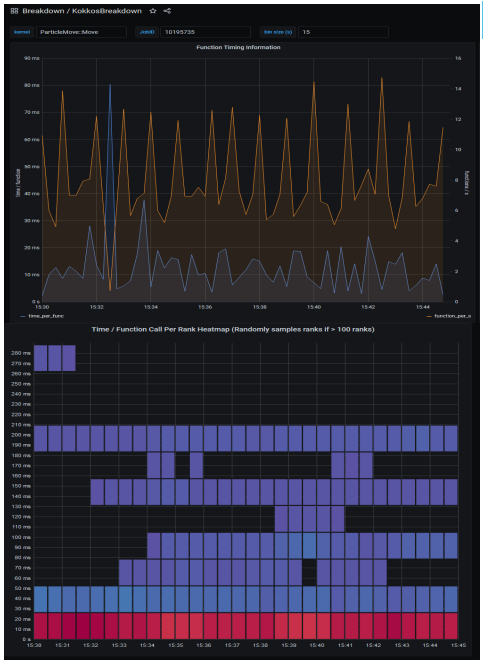


29

Analysis and Visualization

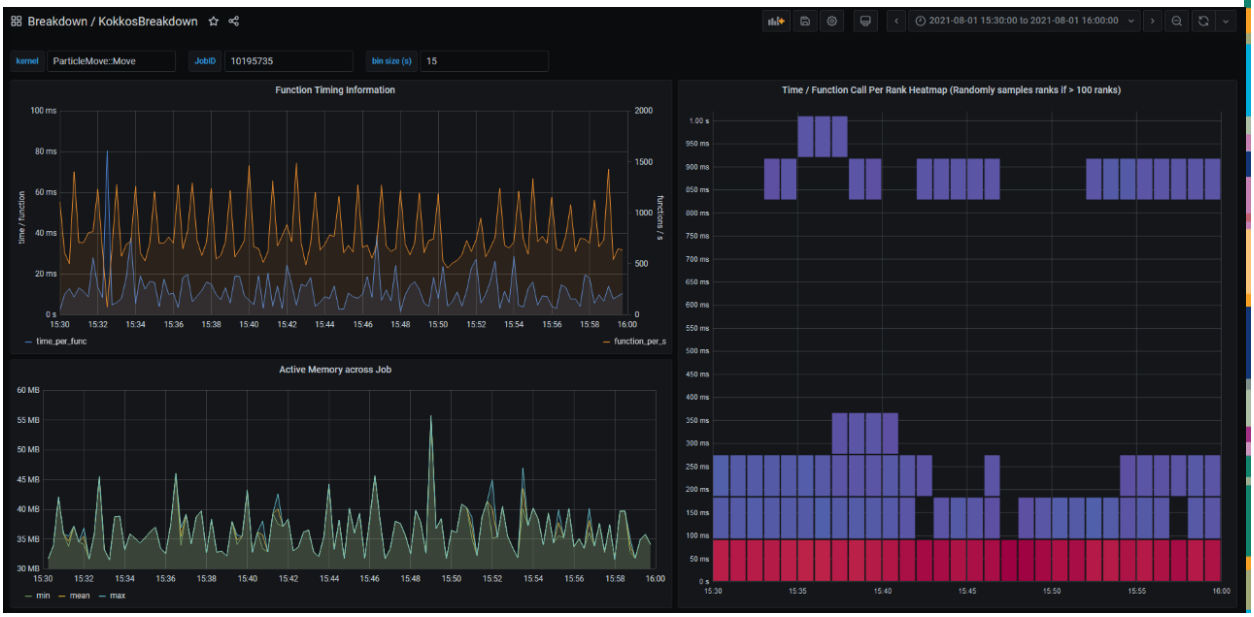
Kernel-level dashboard shows data specific to a chosen kernel across 2 panels **(progress metric)**

- Function Timing Information plot shows
  - **Average time per specific kernel execution across all ranks over time (Blue)**
  - **Number of specific kernel executions per second across all ranks over time (orange)**
- The bin size fillable box at the top of the dashboard enables users to bin the data to better understand the trends
  - I.e. for a 1 minute window, 1 second bins might reveal more relevant information and for a 2 hour window, 1 minute bins might be easier to understand
- The Time/Function Call Per Rank Heatmap shows how the execution time of functions across the ranks of the application
  - Red shows more ranks are in that execution range, blue shows less ranks
  - Showed that several EMPIRE kernels routinely had outlier ranks



30

Analysis and Visualization



Feedback from Stakeholders

Stakeholder Feedback

EMPIRE developer and analyst comments:

- "I fully expect enabling LDMS to become the default EMPIRE behavior on supported platforms"
- "There was no noticeable impact on performance on small or large simulations when LDMS was enabled"
- "Being able to see the dashboard's real-time updating of simulation performance is so much better than manually finding that information in simulation logfiles"
- "Quickly plotting simulation metrics helps us quickly assess job health and progress, saving time and decreasing cognitive load"
- "Clear, clean layout without presenting too much information"

Requested improvements:

- #1 request was that they would like to be able to have a subset of kernels always collected
  - I.e. Main time loop
- More info about filesystems and I/O alongside application data
- Rename labels of data to improve understanding
  - Will also be adding a panel with in-depth descriptions about the data and underlying analyses
- Add bit-rate to application throughput panel to show how much data is being ingested by the backend
  - Will be useful for adjusting sampling rate in the future

33 Future Capability Augmentation Priorities

Architecture:

- Explore additional lightweight methods for sampling of Kokkos kernel execution information
  - Self adjusting data volume production
  - User-controlled variable sampling rate and always sampling specified kernels

Metric Selection:

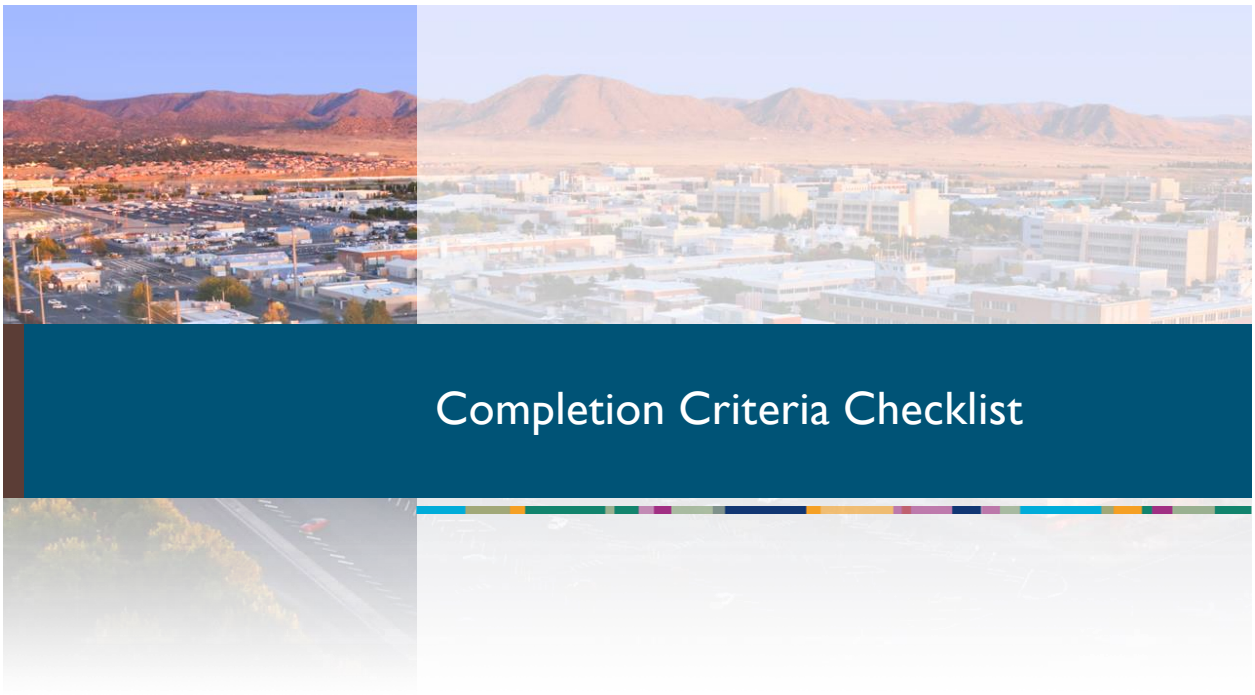
- Add PAPI events/metrics to analyses and dashboards
- Define metrics for, and implement, performance bottleneck detection

Visualization and Analysis:

- Analyses with both application and system data to automatically identify correlations
- Advanced analyses, such as rank clustering or historical variance investigation, of application data

General:

- Publication at a major conference

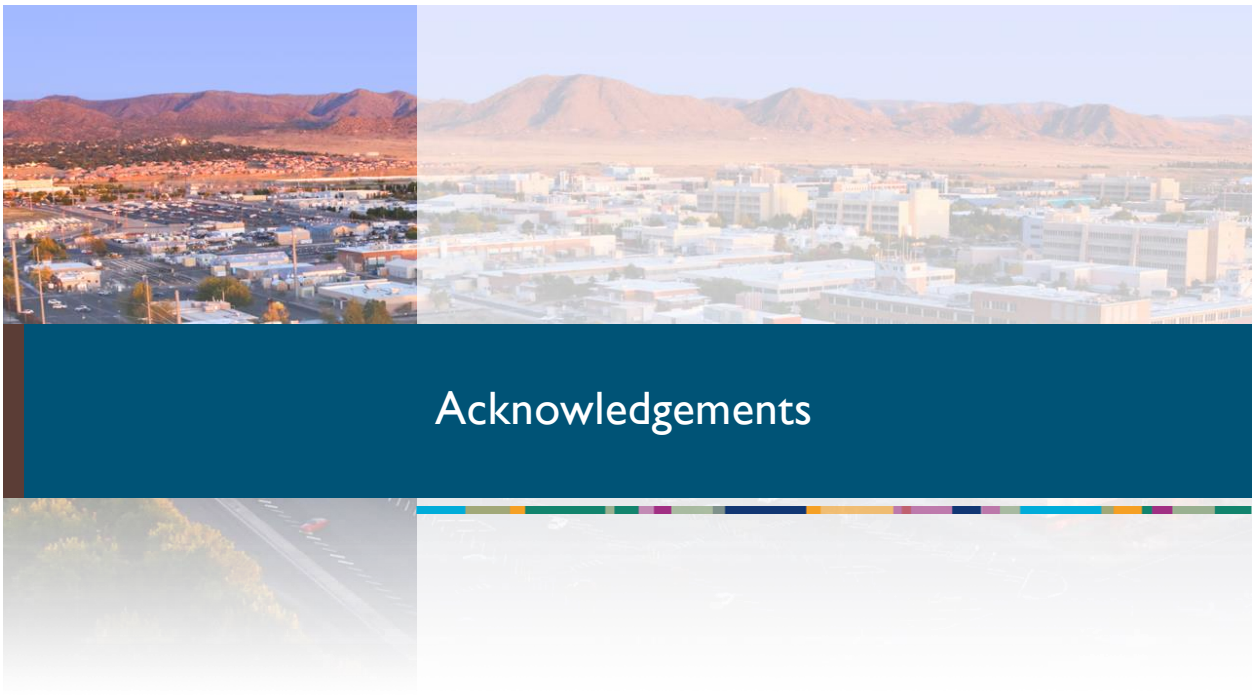

Completion Criteria Checklist

Completion Criteria Checklist

1. Successful deployment of infrastructure on CTS-1 system (Eclipse)
   - ✓ Target version of LDMS (i.e., Streams enabled) has been in continuous deployment on **Eclipse** since Jan 28, 2021
2. Demonstration of capability for continuous **collection and storage** of system data over a 2-week rolling window
   - ✓ We have demonstrated continuous collection and storage over a 30-day (2 x 2 weeks) window of system data on the 1500 node Eclipse cluster
     - A 30-day window produced ~60TB (including indexing overhead) of data stored in NVMe-based Scalable Object Store (SOS) databases distributed across 14 nodes of the Shirley Monitoring and Analysis cluster. This is < 10% of the NVMe storage capability of Shirley
     - Rolling window previously demonstrated on a single SOS database on our Bitzer system
3. Identification of an application **throughput** metric(s) for an ASC-relevant code (EMPIRE)
   - ✓ Throughput indicated by the total number of kokkos kernel executions per-minute over a defined time window while running the Empire application (see video)
4. Identification of an application **progress** metric for an ASC-relevant code (EMPIRE)
   - ✓ Number of kernel calls per second over a defined time window (15 sec. default) for a kernel indicative of science work accomplished (ParticleMove::Move)

Completion Criteria Checklist

5. Demonstration of capability on target application (EMPIRE) run(s) on CTS-1 system (Eclipse)
   - ✓ Demonstrated 32- to 290-node Empire application runs on Eclipse (1500 node CTS-1 production system)
6. Demonstrate a visualization interface that will enable a user to look at post-run application progress in conjunction with system conditions
   - ✓ Shown in slides 27-30
7. Demonstrate a visualization interface that will enable a user to look at run time application progress in conjunction with system conditions
   - ✓ Shown in slides 27-30
8. Document feedback and future work
   - ✓ Shown in slides 32-33



Acknowledgements

DAT Acknowledgements

As part of the L2 milestone, in Jan 2021 we held a 30-hour DAT on Eclipse for LDMS (v4) overhead testing and to validate the interoperability of our initial application + Kokkos Sampler + Streams functionality. This involved substantial work up front in determining applicable workload and metrics to collect as well as all of the infrastructure and analysis/visualization configuration.

Special thanks to:

- L2 members Mark Schmitz and Phil Regier for multiple days efforts in configuration and deployment of LDMS v4 on Eclipse ahead of the LDMS v4 TOSS Release as well as continuous support throughout the DAT
- 9327 for enabling the long-running DAT
- Anthony Agelastos, Douglas Pase, Joel Stevenson, and Gary Lawson of 9326 for their development of an application work package, which they ran over a 24-hour time period, and their post-run analysis validating low overhead (~<1.0%).

Fin

DISTRIBUTION

Email—External [redacted]

Name	Company Email Address	Company Name
Richard Gerber	ragerber@lbl.gov	Lawrence Berkeley National Laboratory
Lena Lopatina	lena@lanl.gov	Los Alamos National Laboratory
Cindy Martin	c_martin@lanl.gov	Los Alamos National Laboratory
Ben Santos	bsantos@lanl.gov	Los Alamos National Laboratory
Tom Tucker	tom@ogc.us	Open Grid Computing

Email—Internal [redacted]

Name	Org.	Sandia Email Address
Omar Aaziz	9328	oaaziz@sandia.gov
Benjamin Allan	9328	baallan@sandia.gov
Jim Brandt	9328	brandt@sandia.gov
Jeanine Cook	1422	jeacook@sandia.gov
Karen Devine	1465	kddevin@sandia.gov
James Elliott	1422	jjellio@sandia.gov
Ann Gentile	9328	gentile@sandia.gov
Mike Glass	1545	mwglass@sandia.gov
Simon Hammond	1422	sdhammo@sandia.gov
Rob Hoekstra	1420	rjhoeks@sandia.gov
Brian Kelley	1465	bmkelle@sandia.gov
Tom Klitsner	9320	tklitsn@sandia.gov
Steve Monk	9327	smonk@sandia.gov
Stan Moore	1444	stamoor@sandia.gov
Curt Ober	1446	ccober@sandia.gov
Stephen Olivier	1423	slolivi@sandia.gov

Name	Org.	Sandia Email Address
Roger Pawlowski	1446	rppawlo@sandia.gov
Kevin Pedretti	1423	ktpedre@sandia.gov
David Poliakoff	1422	dzpolia@sandia.gov
Phil Regier	9327	paregie@sandia.gov
Mark Schmitz	9327	mschmit@sandia.gov
Ben Schwaller	9328	bschwal@sandia.gov
James Stewart	1440	jrstewa@sandia.gov
Kevin Stroup	9328	kdstrou@sandia.gov
Matthew Scot Swan	1446	mswan@sandia.gov
Vanessa Surjadidjaja	9328	vjsurjad@sandia.gov
Courtenay Vaughan	1422	ctvaugh@sandia.gov
Sara Walton	9328	spwalto@sandia.gov
Technical Library	1911	sanddocs@sandia.gov

Sandia National Laboratories

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.