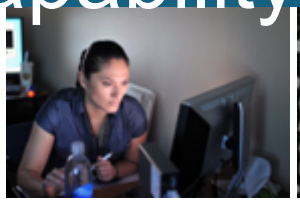
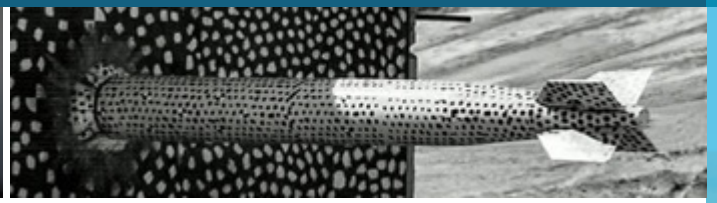
SAND2021-10918PE

Sandia National Laboratories

Integrated System and Application Continuous Performance Monitoring and Analysis Capability





08/24/2021

Final L2 Milestone Review

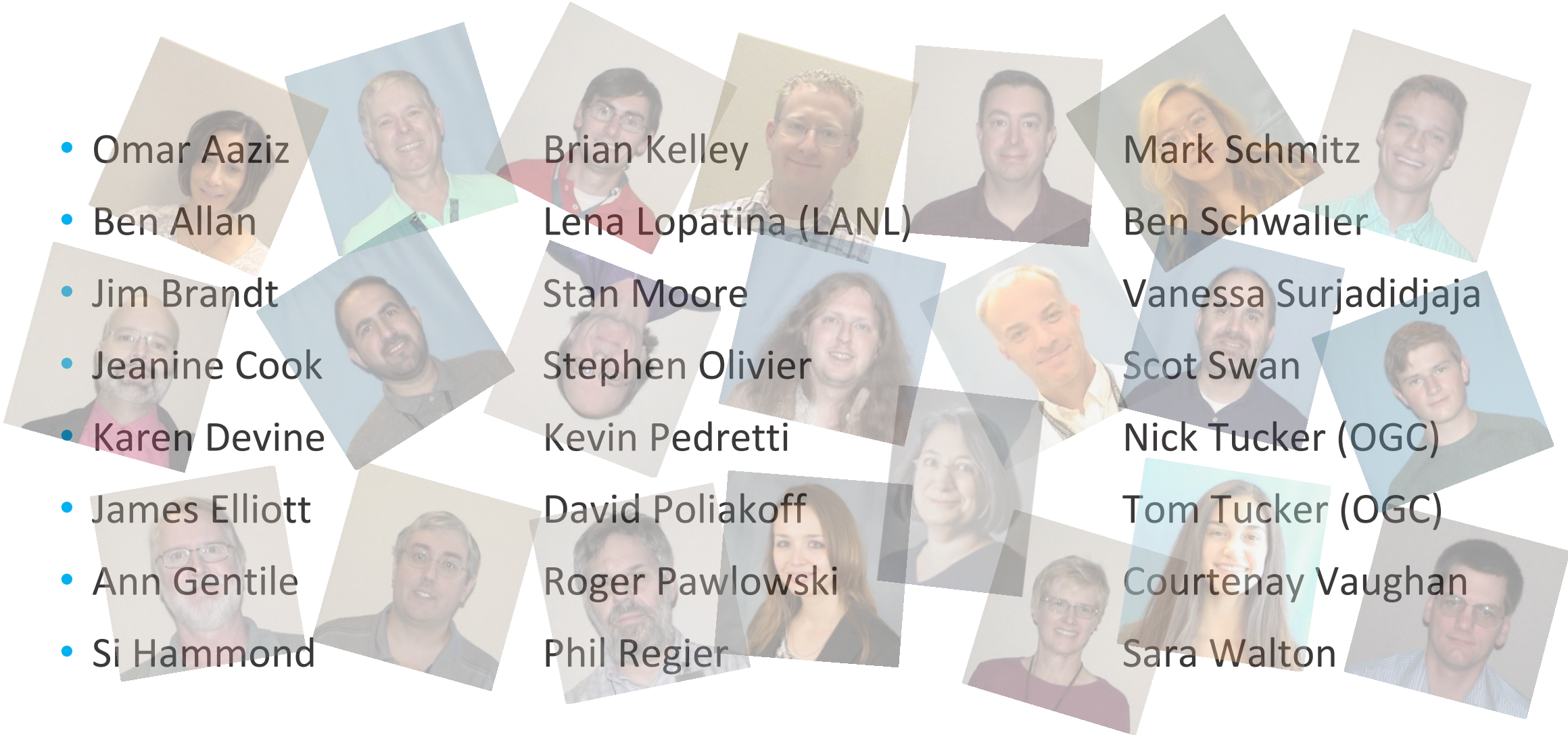
U.S. DEPARTMENT OF ENERGY | NNSA

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.

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



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

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.

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

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

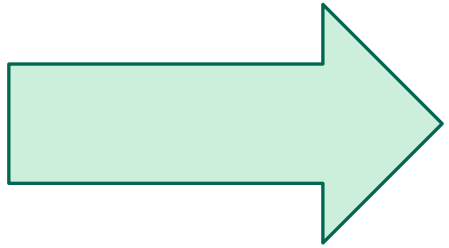
Kokkos Provides Required Instrumentation

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

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

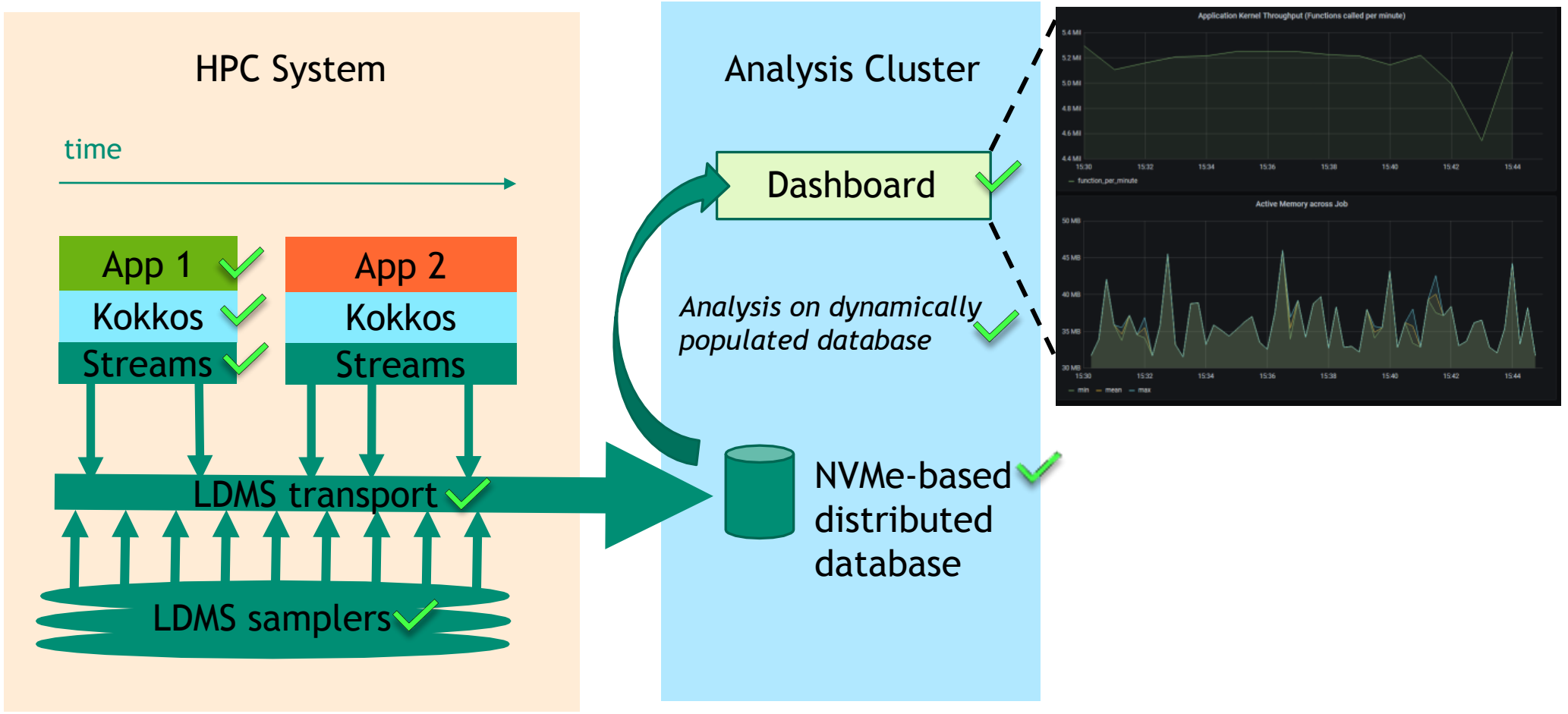
Just need to publish application data to the LDMS Streams API, add store functionality in order to store application performance metrics to the same database as the system data, convert raw data to a progress/performance metric, and present to users…

Integrated System and Application Continuous Performance Monitoring and Analysis Capability

Applications dynamically and irregularly inject data into the LDMS transport

LDMS continuously and regularly collects and transports full system data

Data Flow Diagram



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

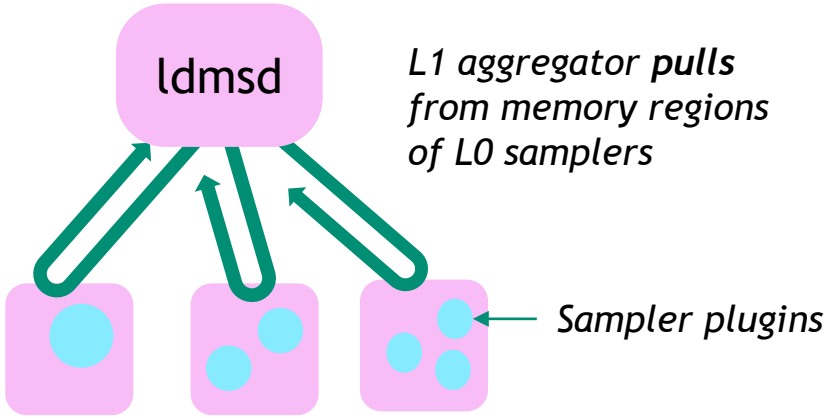
Enabling Application Data Injection via LDMS

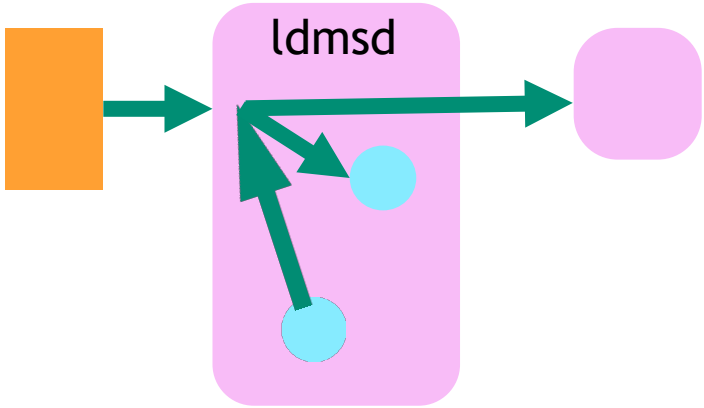
LDMS - **low-overhead (<1% application)** data collection, transport, and storage capability designed for **continuous monitoring** supporting **run time analytics** and feedback.

- 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

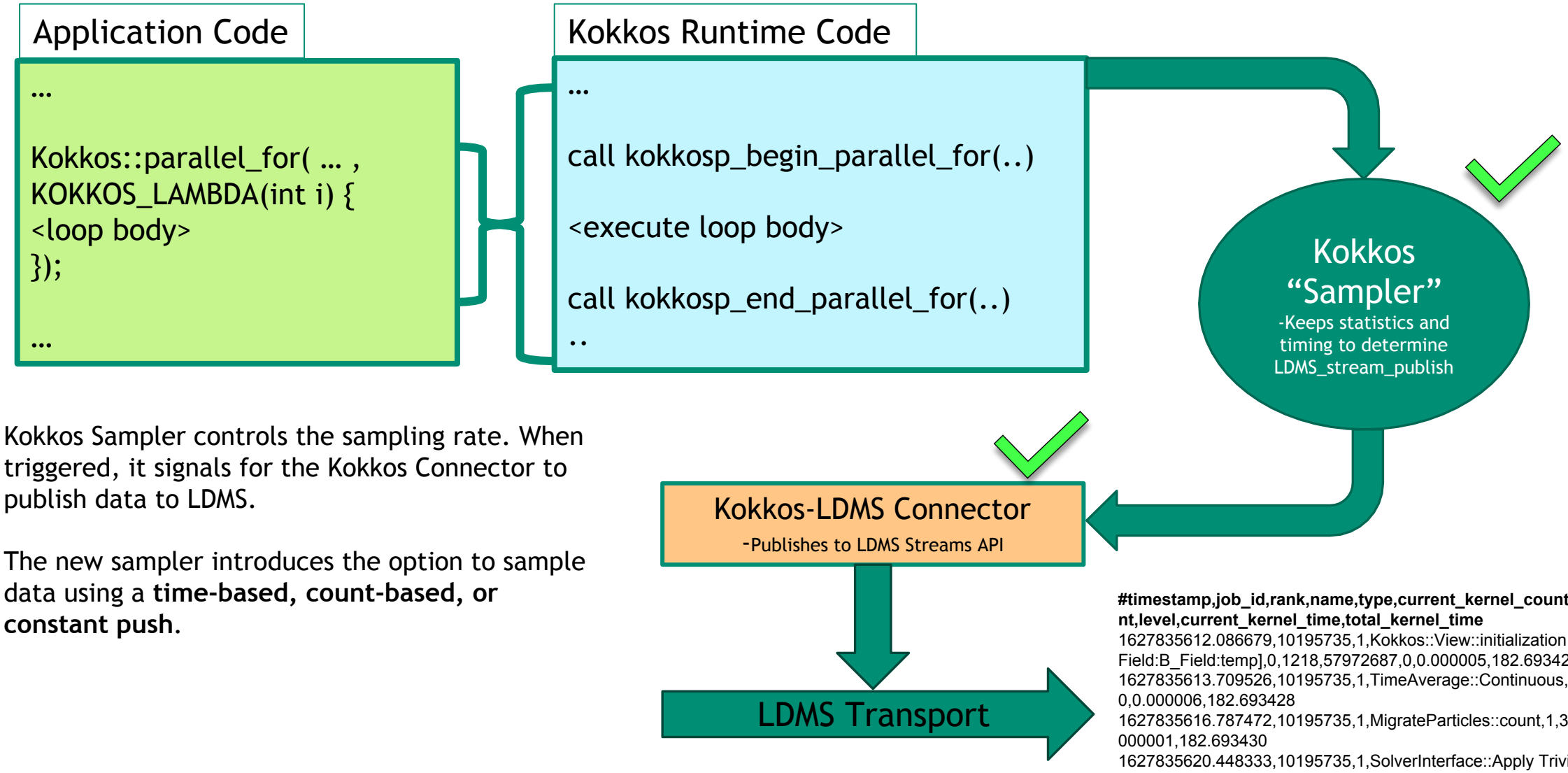
- Transport is **push** based and supports **asynchronous** event data (e.g. scheduler and log data)
- **Unstructured** data





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

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**.

Logical Subgroup Descriptions

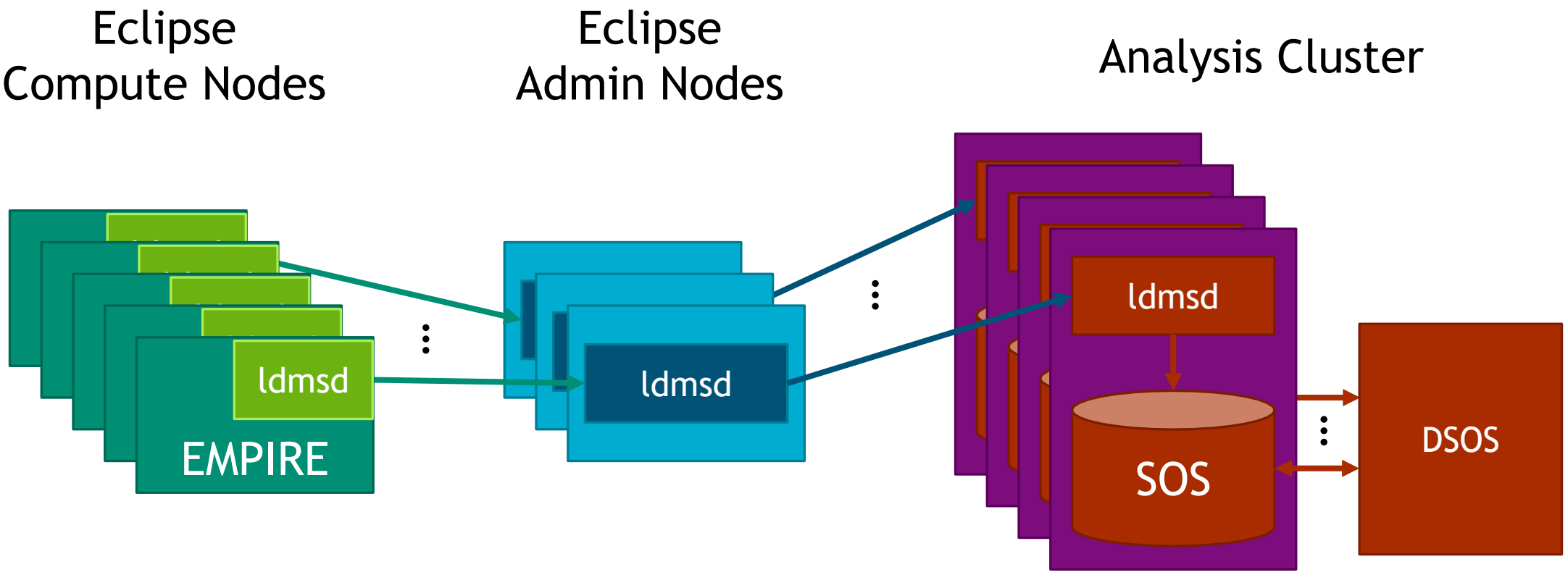
Application-Streams Architecture

- 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

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

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

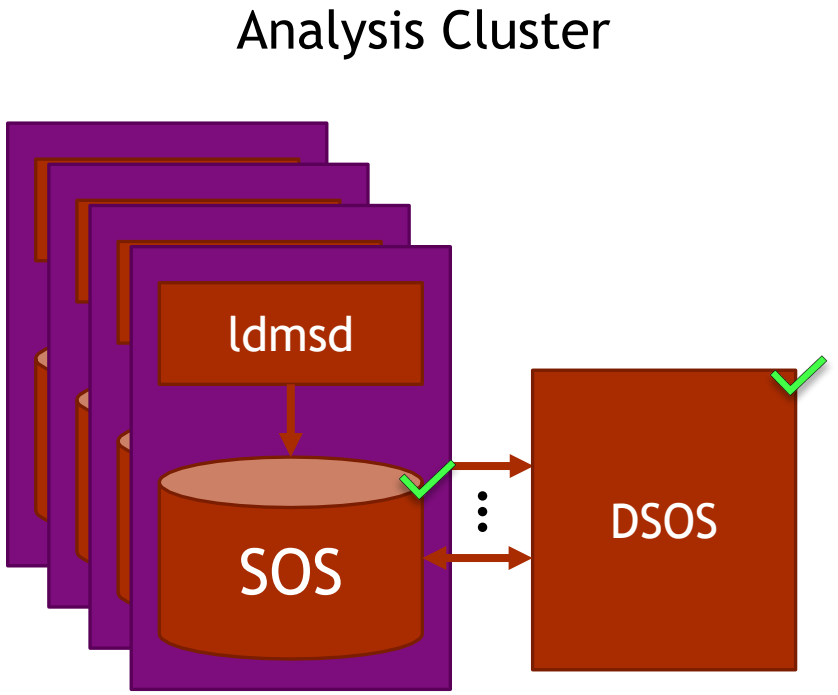
Analysis and Visualization

- **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

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**

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

Analysis and Visualization Presentation Overview

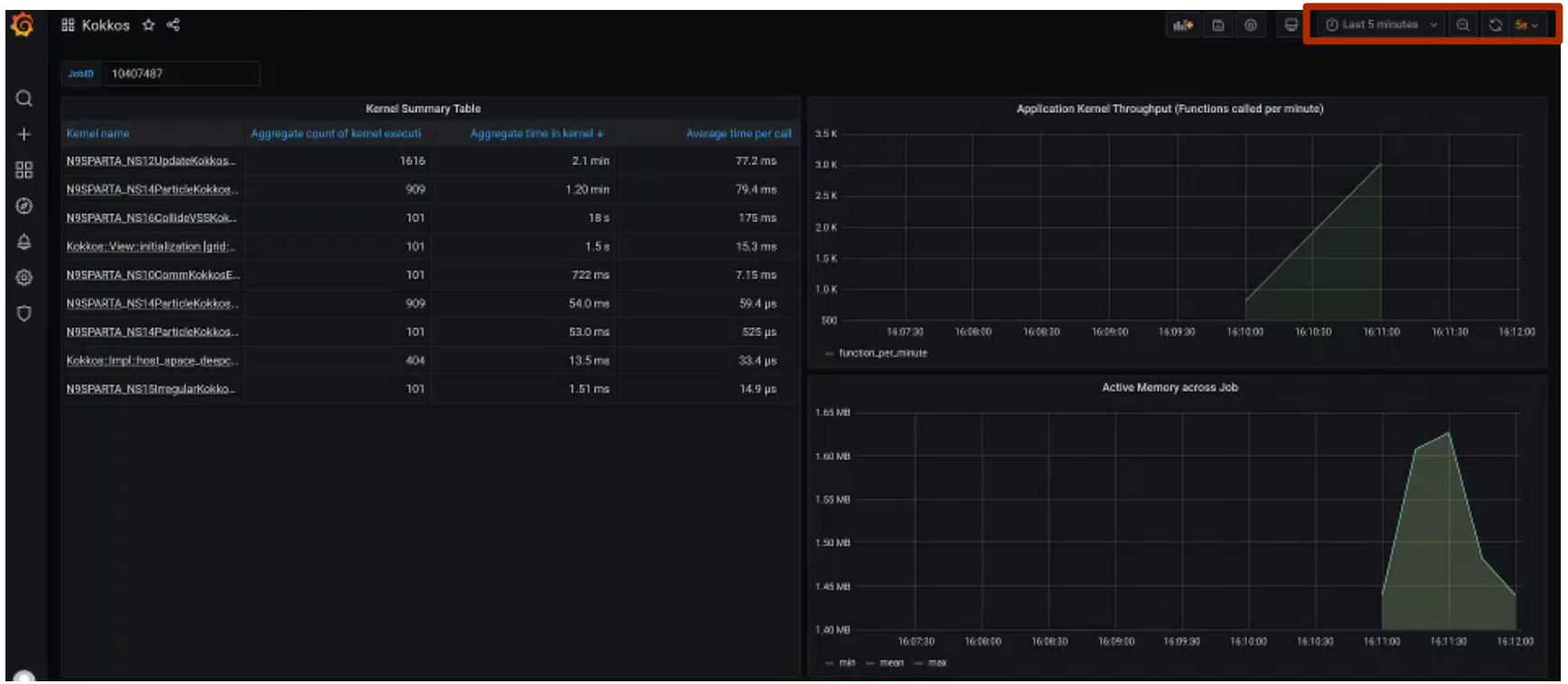
Created two Grafana dashboards to visualize an application's Kokkos data

- 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**

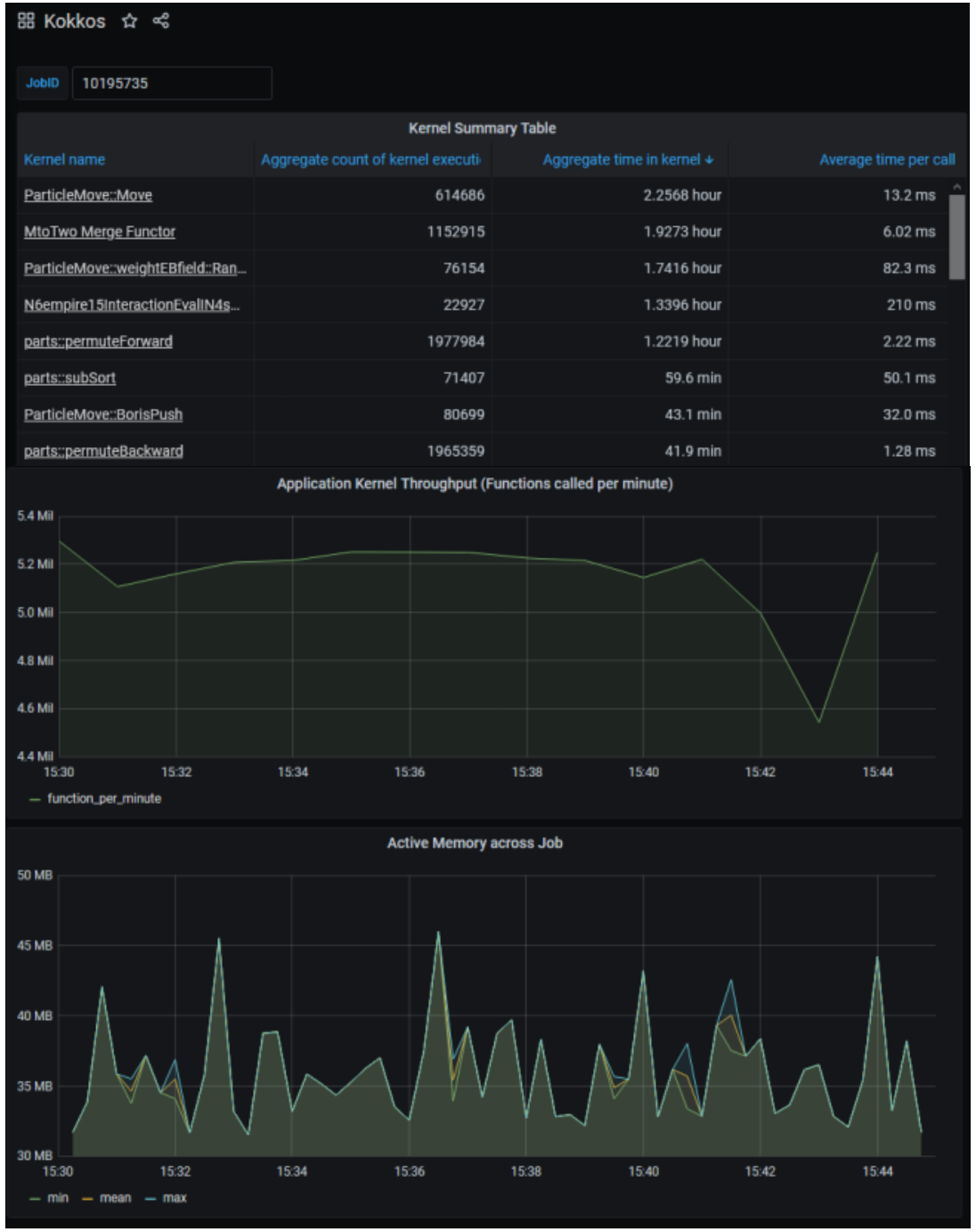
Video of live feed of job's data
(5 second update intervals)



Analysis and Visualization

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

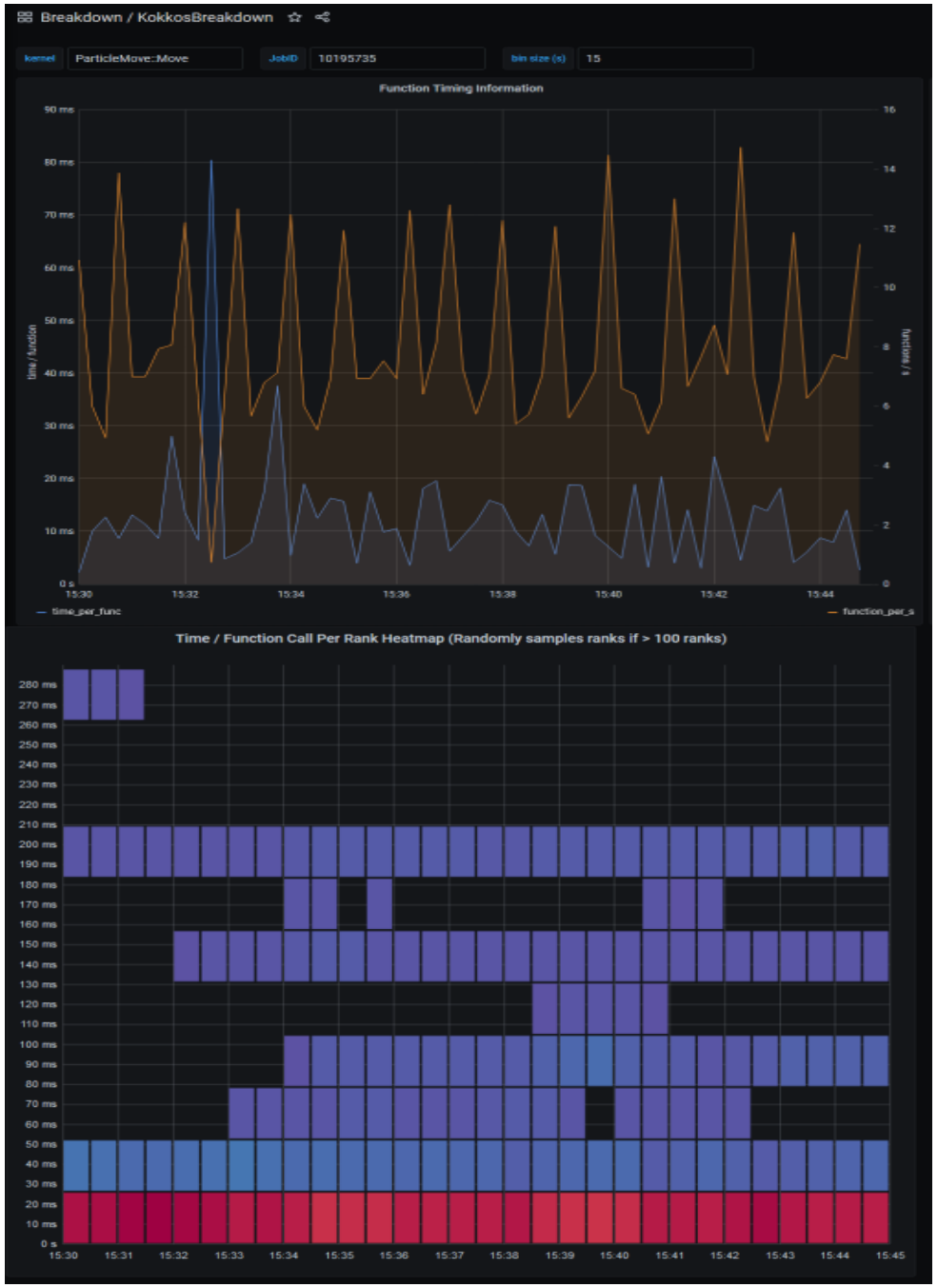
- Kernel Summary Table which shows all kernels, their times called, time spent in kernel, and average kernel execution time, in the time range specified
  - Each kernel has a link to drilldown to the next dashboard
- Application Kernel **Throughput** which is a time series graph of how many kernels have 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



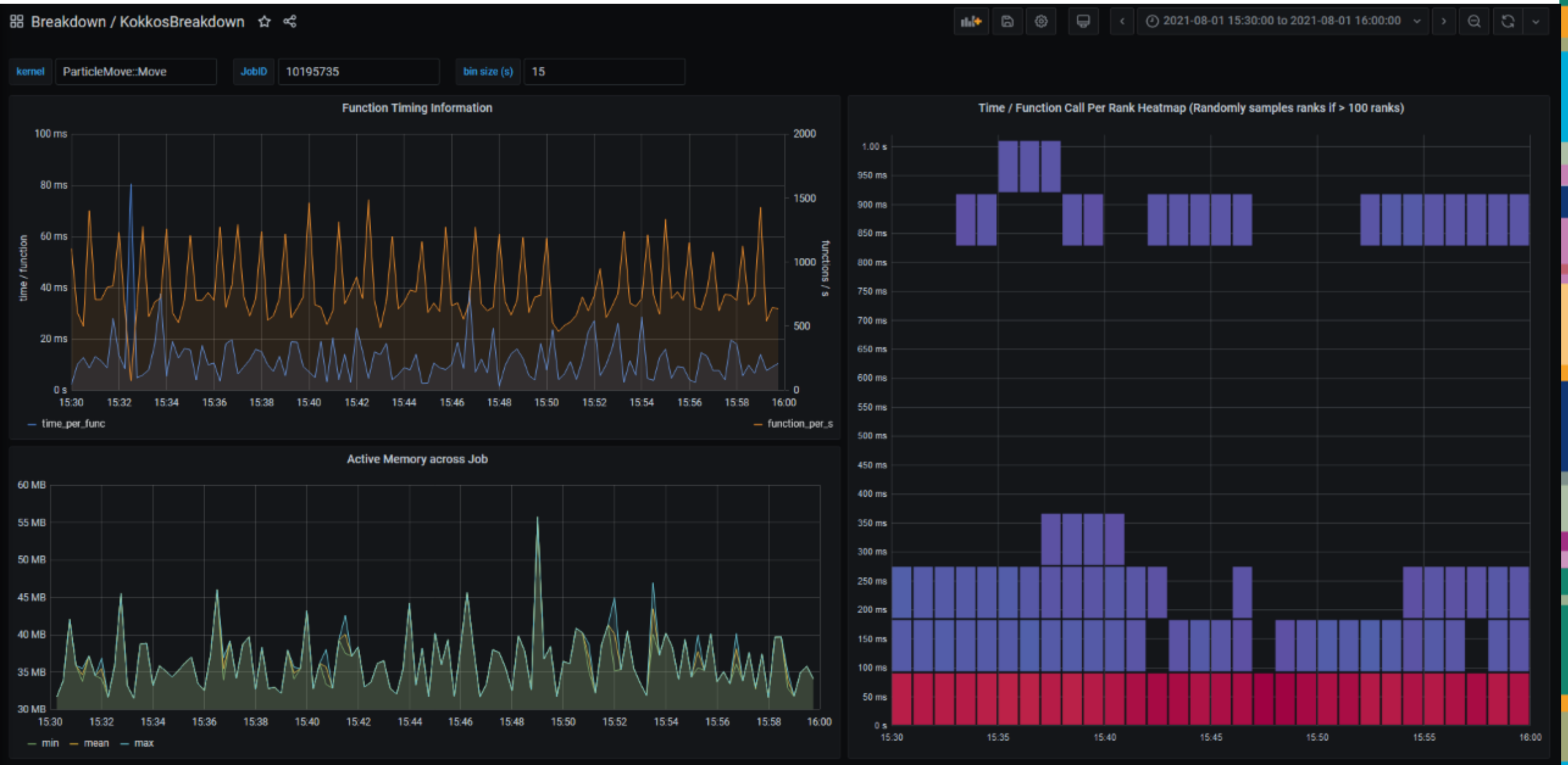
Analysis and Visualization

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



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

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