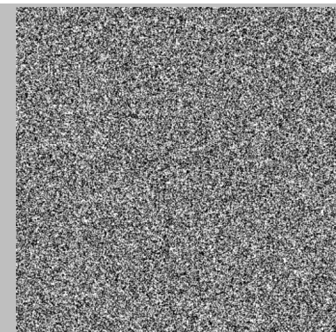
SAND2015-7830C

Exceptional service in the national interest

Sandia National Laboratories

Continuous Monitoring of HPC platforms and the applications run upon them



Chaotic, random applications, workflows, and dependencies

Toward Rapid Understanding of Production HPC Applications and Systems

Anthony Agelastos, Benjamin Allan, James Brandt, Ann Gentile, Sophia Lefantzi, Stephen Monk, Jeffry Ogden, Mahesh Rajan, Joel Stevenson

U.S. DEPARTMENT OF ENERGY

NNSA National Nuclear Security Administration

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000. SAND NO. 2015-7830 C

Executive Summary

- **Objective**: Gain rapid understanding of production high-performance computing (HPC) applications and platforms to enable timely troubleshooting and up-to-system-level studies to assist **developers**, **analysts**, **system administrators**, and **procurement** activities

- **Methodologies Used**
  - LDMS (Lightweight, Distributed Metric Service)
  - 5 production-relevant test cases
  - Scoring of LDMS results with test cases

- **Results**
  - LDMS has fidelity & capability to address production, high-level metrics of concern
  - Scoring provides a comfortable interface to the large quantities of data for all personas

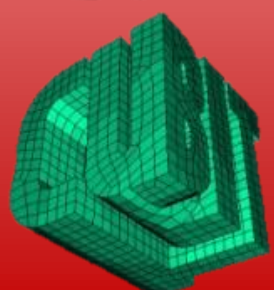
Production Simulation Overview

Parametric Analyses, Optimization

Automated Post-processing

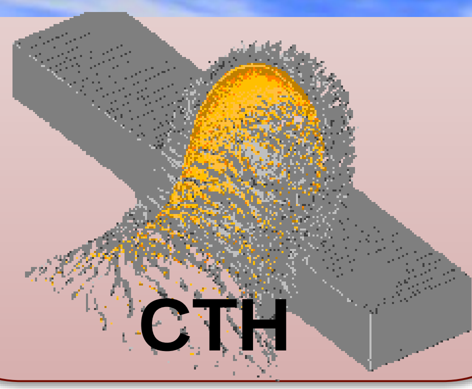
Pre-processing (Parallel & Serial)

DAKOTA



Simulation (Parallel & Serial)

Trilinos



Post-processing (Parallel & Serial)

ParaView

HDF The HDF Group

Verification & Validation, Uncertainty Quantification

Production workflows are complicated

Production Application Profiling

- Typical procedure for profiling single application
  1. Obtain executable with necessary characteristics, e.g., symbol table, minimal compiler optimizations, specific DWARF adherence, built with supported compiler
  2. Attach sample-based profiler to application and collect data
  3. Analyze data for problematic area(s) of interest
  4. Bracket area(s) of interest to a single iteration with minimum functions instrumented
  5. Instrument application with tracing and collect causal data

Hard

Harder

- Considerations
  - Each step above requires a level of effort from multiple people/orgs. that is typically overcome only by catastrophe or performance milestones
  - Oftentimes data provided by steps 2 & 5 contain more information than is needed to answer many high-level performance queries

Simple high-level overviews are welcome

Lightweight Monitoring: LDMS

- LDMS – Lightweight, Distributed Metric Service
  - Data collection, transport, and storage
- Features:
  - Data is “freely” available
    - Profiles are immediately available without any extra effort from analyst
  - Low CPU utilization, memory, network requirements
    - Does not impact the measured values
  - High-frequency collection (up to subsecond intervals)
    - Can resolve short duration and highly varying data features
  - Synchronized collection
    - Can compare values on different nodes since metrics are collected at the same time on each node
  - Whole-system views
    - Enables environmental insight

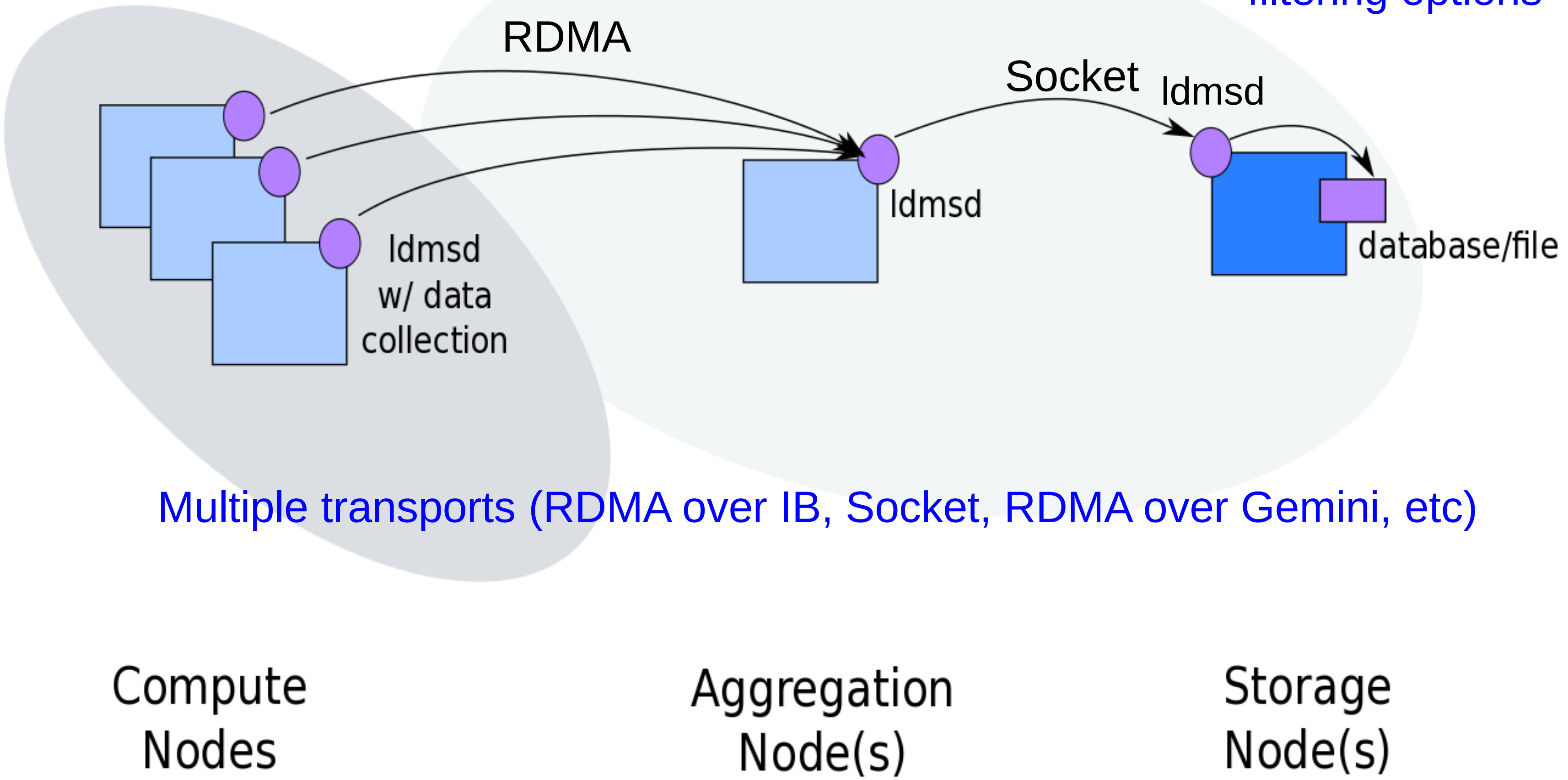
Lightweight Distributed Metric Service: A Scalable Infrastructure for Continuous Monitoring of Large Scale Computing Systems and Applications, Agelastos et al., SC14

Data Collection, Transport, and Storage

Plugins determine data to be collected

Arbitrary aggregation topologies

Plugins for storage: CSV, database, filtering options

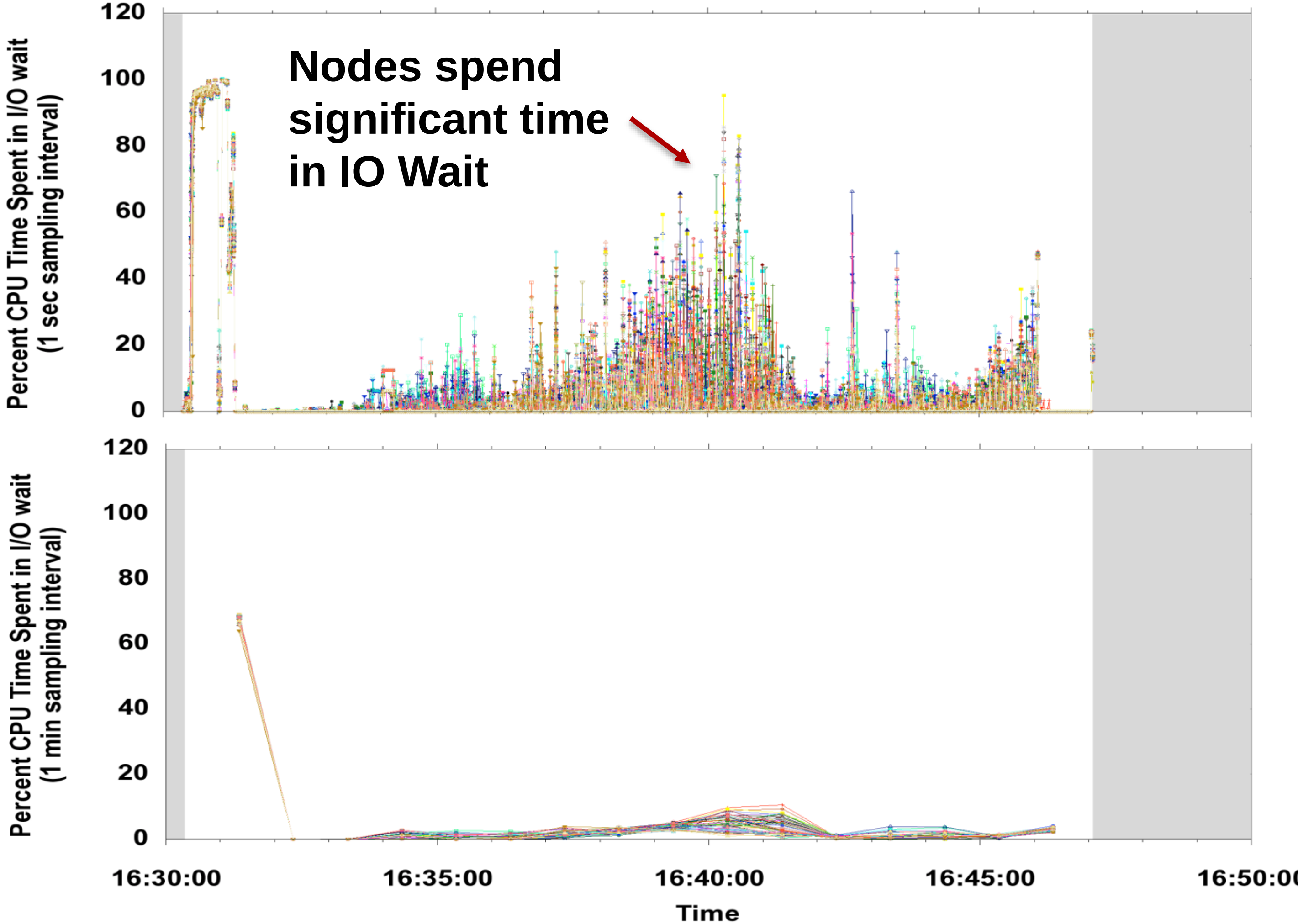


Multiple transports (RDMA over IB, Socket, RDMA over Gemini, etc)

Data Collection

- Data sampler plugins are pre-defined sets of data (e.g., "meminfo" plugin samples all values from `/proc/meminfo` at each sample time) that can be loaded by name and configured with respect to set name and sampling period; currently available plugins provide information about:
  - Lustre and NFS
  - Memory
  - IP Network
  - CPU
  - Infiniband and Cray (Gemini and Aries) performance
  - Machine Specific Registers (MSRs)
  - Interrupts
  - Disk I/O

High-frequency Sampling

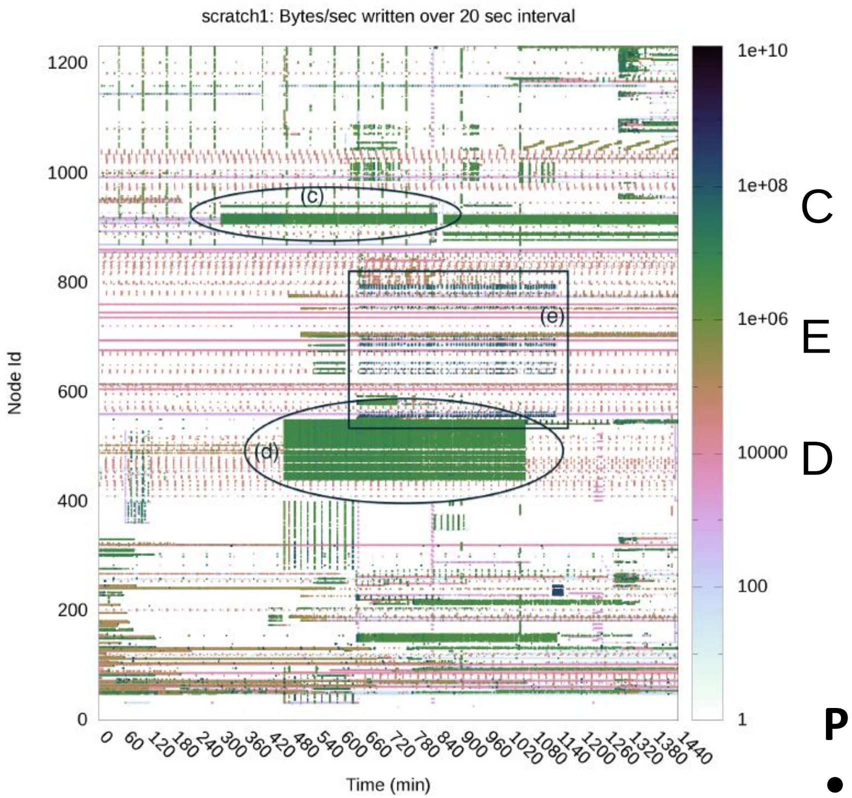


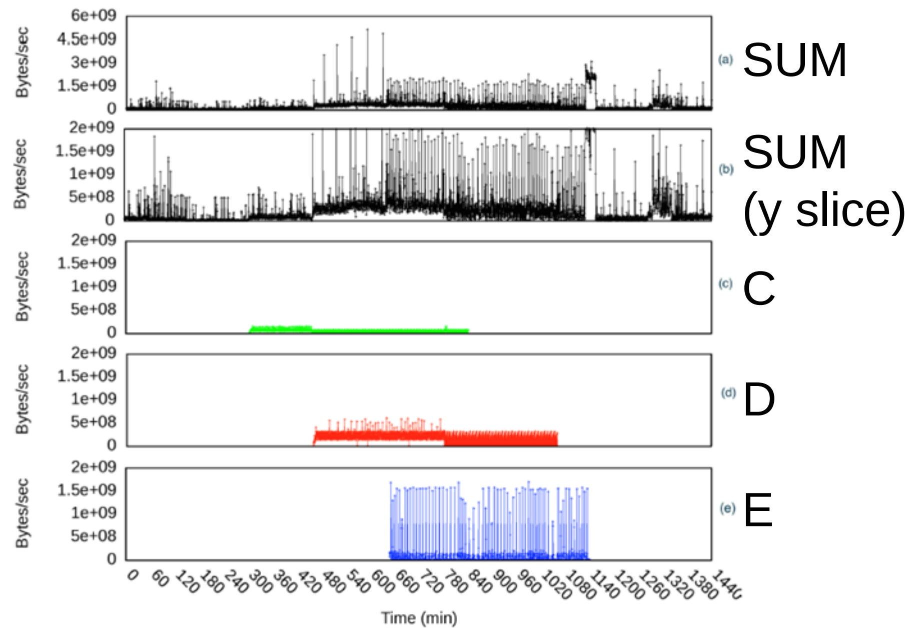
1 sec. intervals

60 sec. intervals

High-frequency sampling intervals are necessary

Whole-system Views





System heat map shows time and placement of jobs with high I/O and which jobs are in contention for same resource

Provides insight about: • system-wide utilization; • events correlated in time and space; • identify contention for shared resources; • understand varying production conditions that can explain performance variations

Whole-system view enables environmental insight

Relevant Production Cases

Nalu
… is an adaptive mesh, variable-density, acoustically incompressible, unstructured fluid dynamics code

CTH
… is a multi-material, large deformation, strong shock wave, solid mechanics code

Sierra/SM
… is a Lagrangian, three-dimensional code for finite element analysis of solids and structures (aka Adagio)

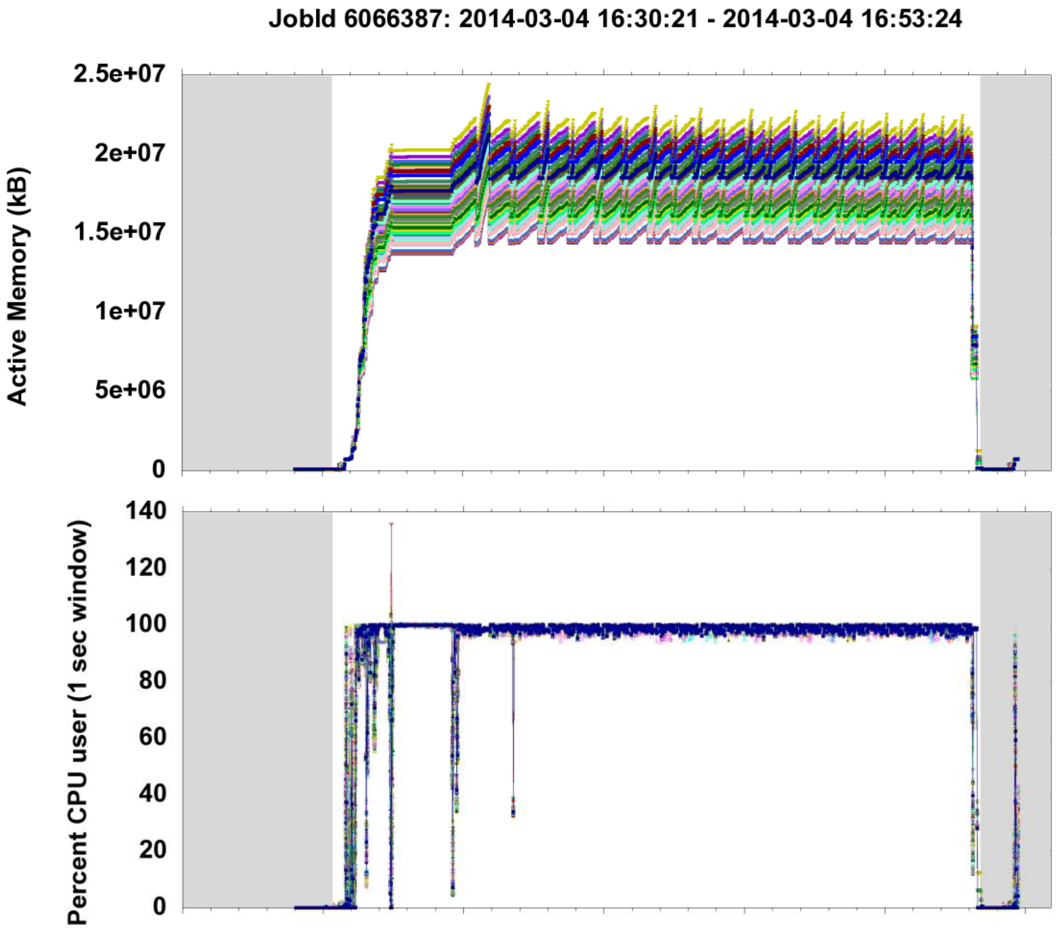
LAMMPS
… is a classical molecular dynamics code

Gaussian
… predicts the molecular properties, etc. of molecules and reactions in a variety of chemical environments

Nalu CPU & Memory Profiles

Observations

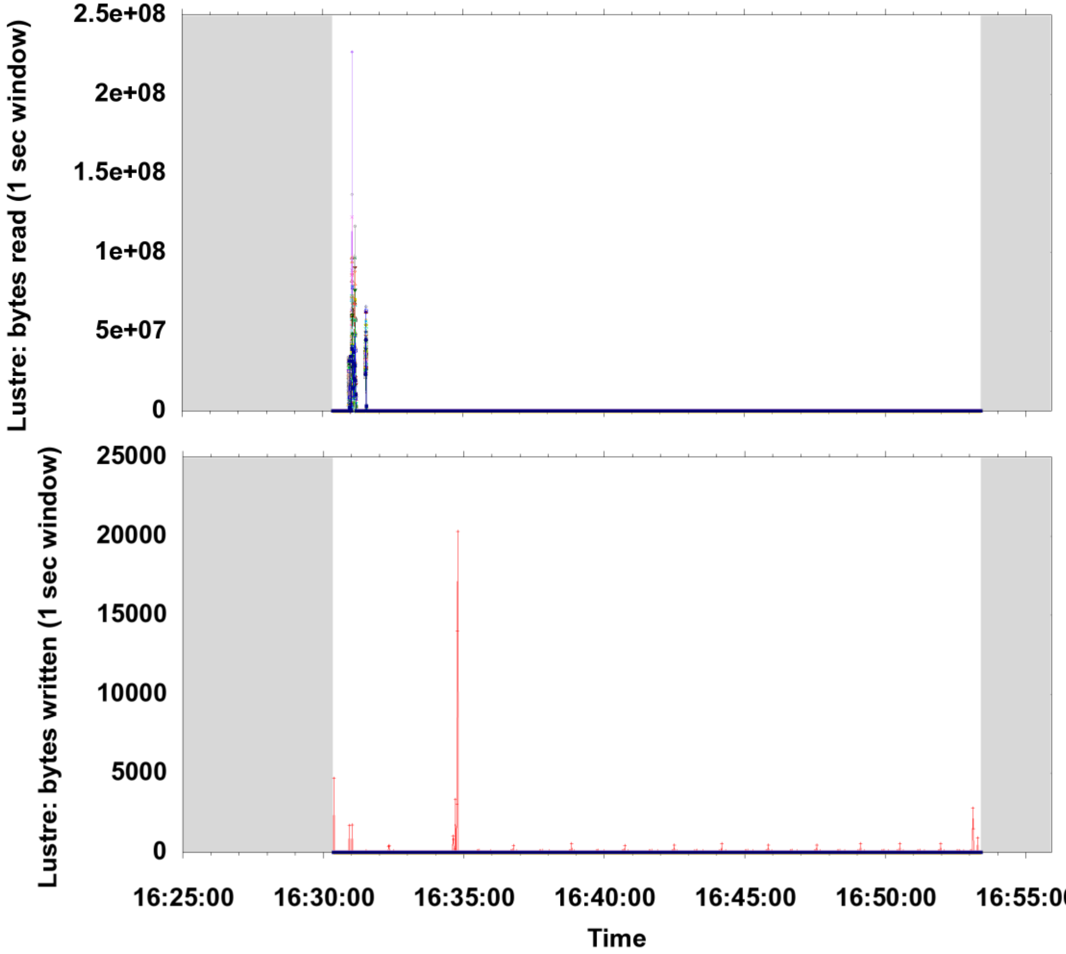
- The memory spread exhibited accounts for about 10% of the node's memory
  - The spread can likely be influenced by changing solver settings and decomposition methods
- The CPU utilization value greater than 100 is an artifact due to a missing data point



Nalu Lustre Read/Write Profiles

Observations

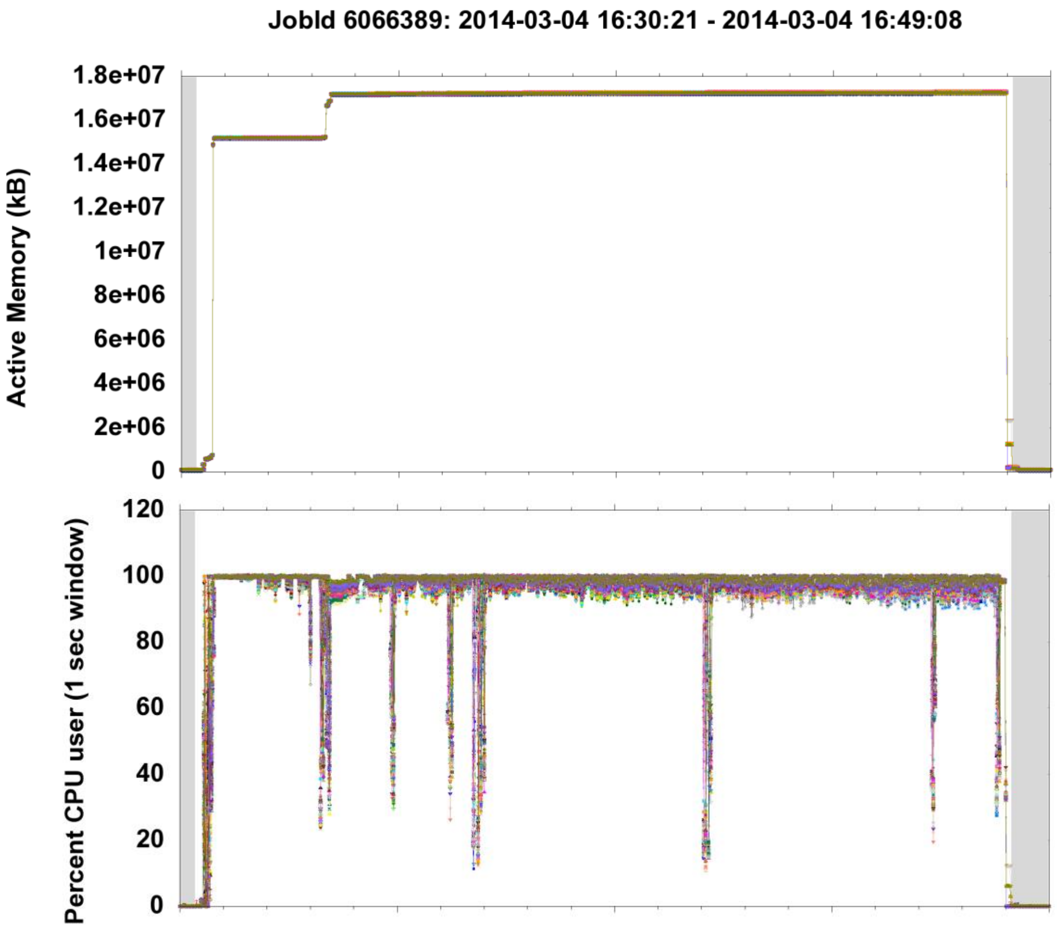
- This simulation was set up such that its only substantial I/O was the initial mesh read; the profiles echo this



CTH CPU & Memory Profiles

Observations

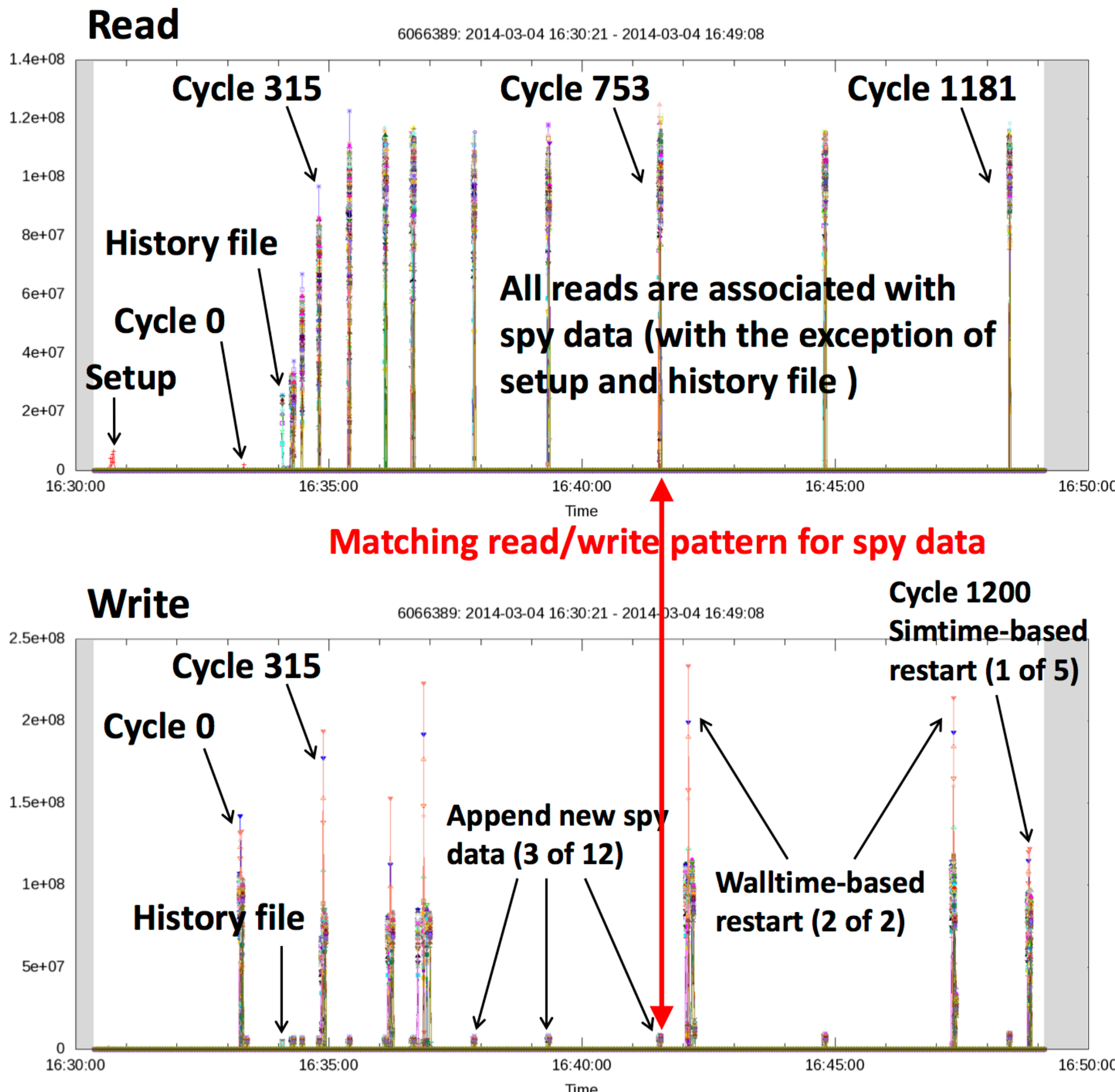
- This CTH simulation has well-balanced memory demands



CTH Lustre Read/Write Profiles

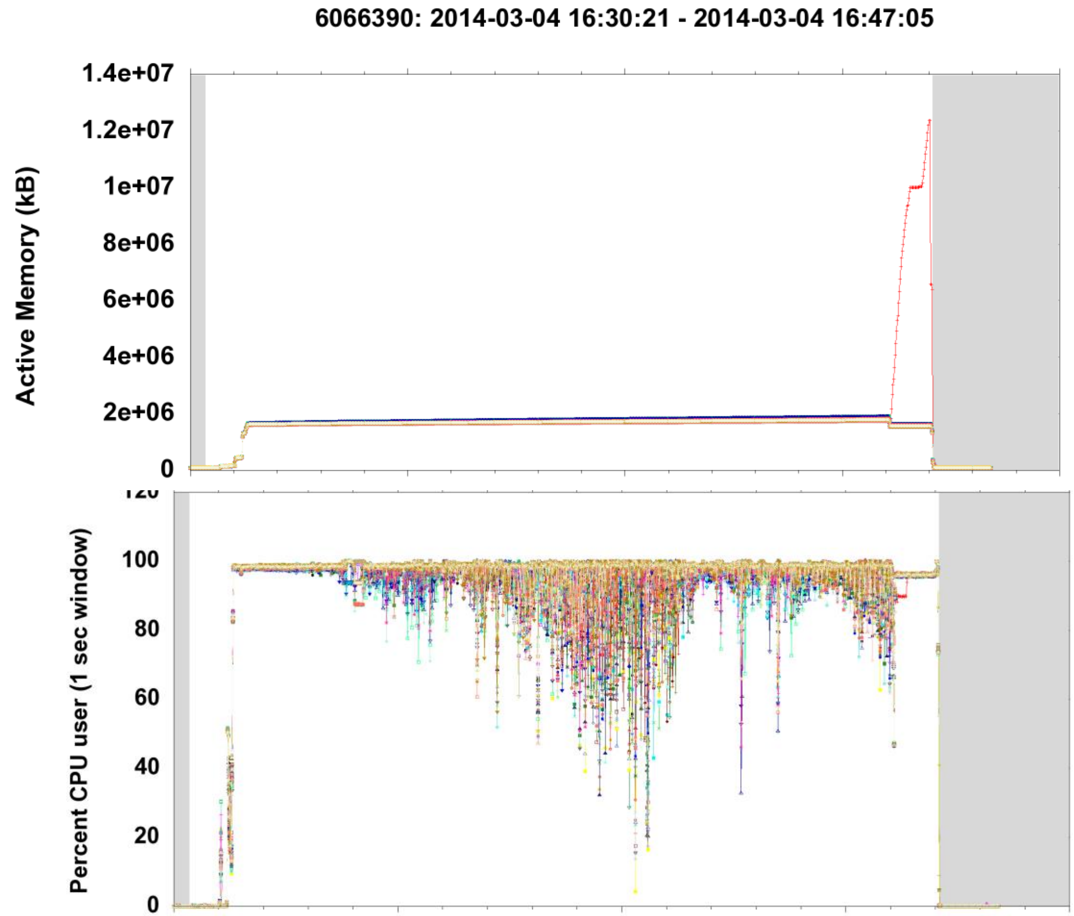
Observations

- CTH uses N-N I/O
- For this example 7,200 Spymaster data files are created only at beginning and then appended to throughout
- There are 12 writes during 19 minute simulation; each write requires a read to append
    - Reads exhibit increasing trend and then flatten



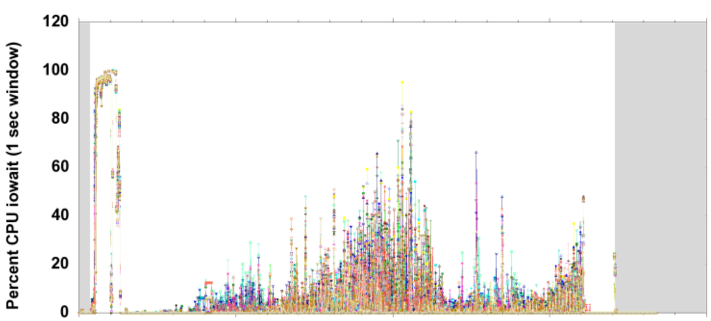
Sierra/SM CPU & Memory Profiles

Sandia National Laboratories



Observations

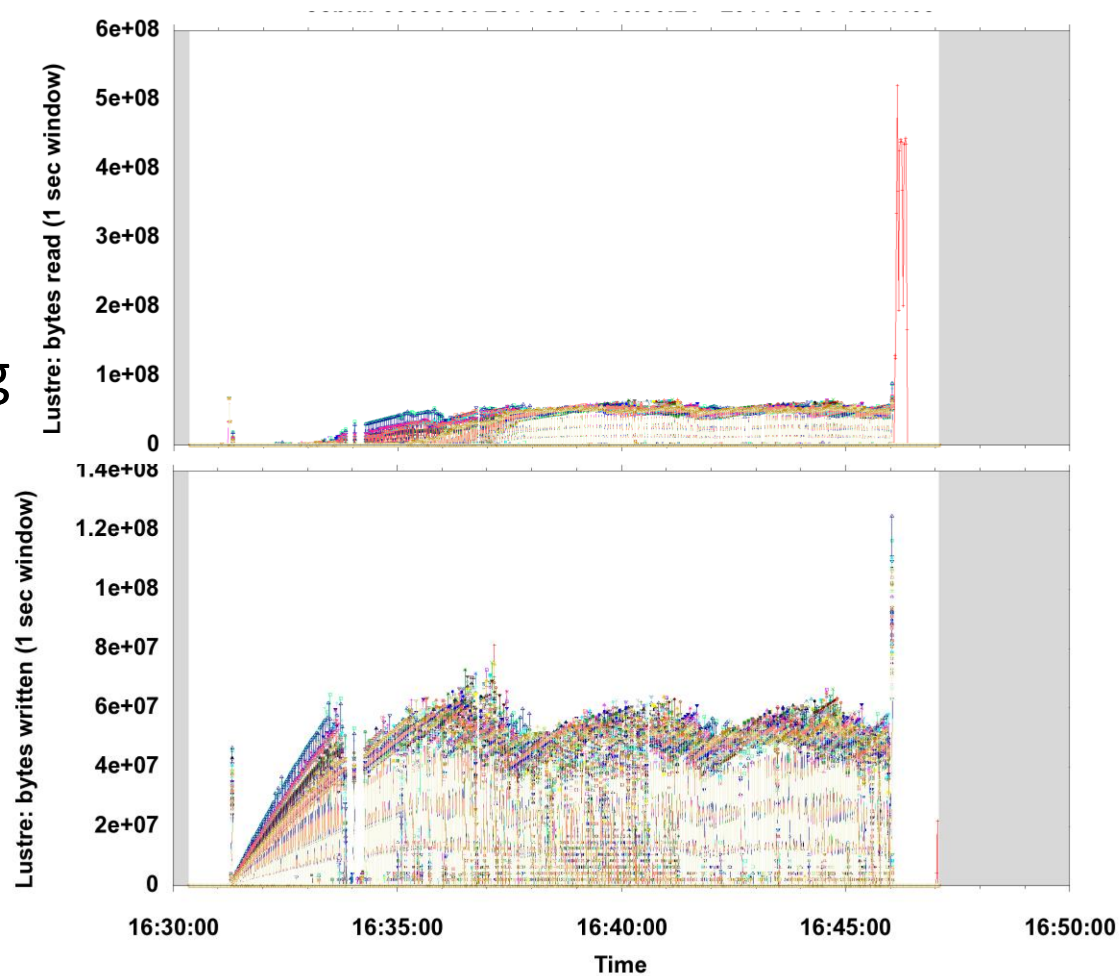
- This Sierra/SM simulation has well-balanced memory demands
  - The jump in memory is from attached mpiP
- The CPU dip halfway corresponds to I/O wait



Sierra/SM Lustre Read/Write Profiles

Observations

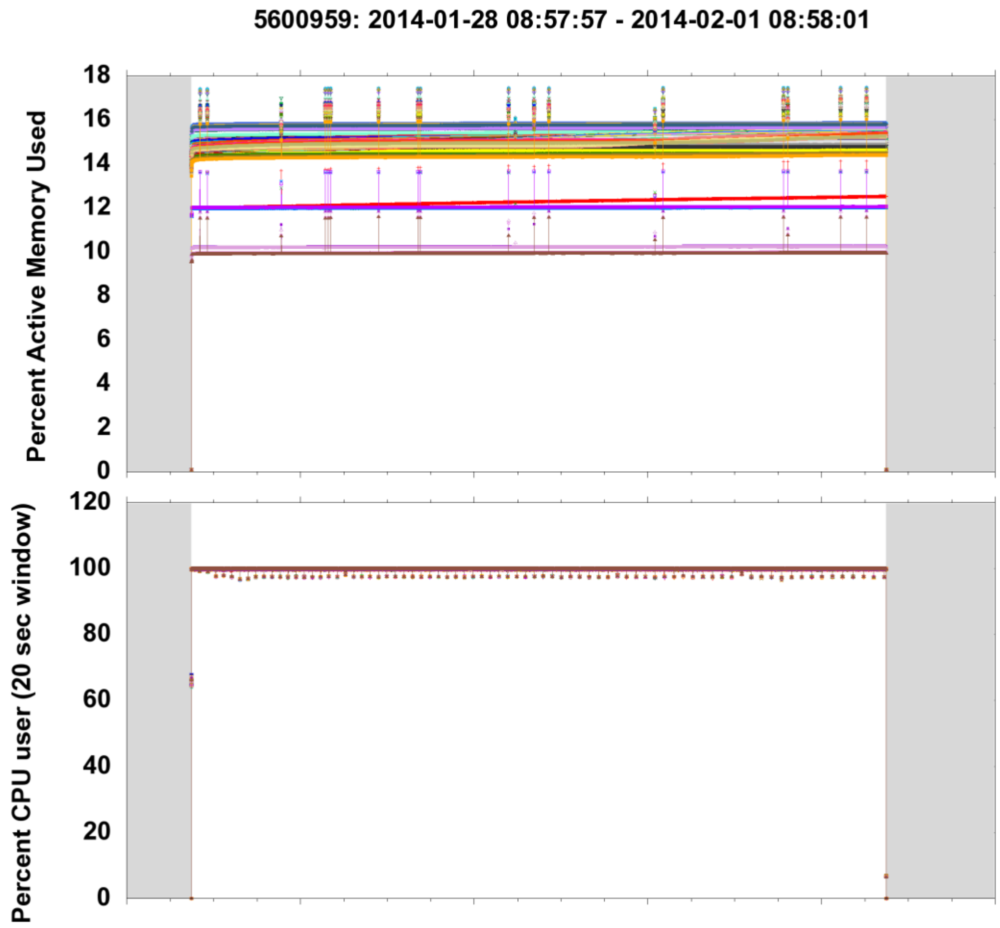
- Sierra/SM uses N-N I/O
- The Lustre writes of results data are impacting CPU utilization
  - The jump in bytes written at the end is due to this simulation also being profiled by mpiP



LAMMPS CPU & Memory Profiles

Observations

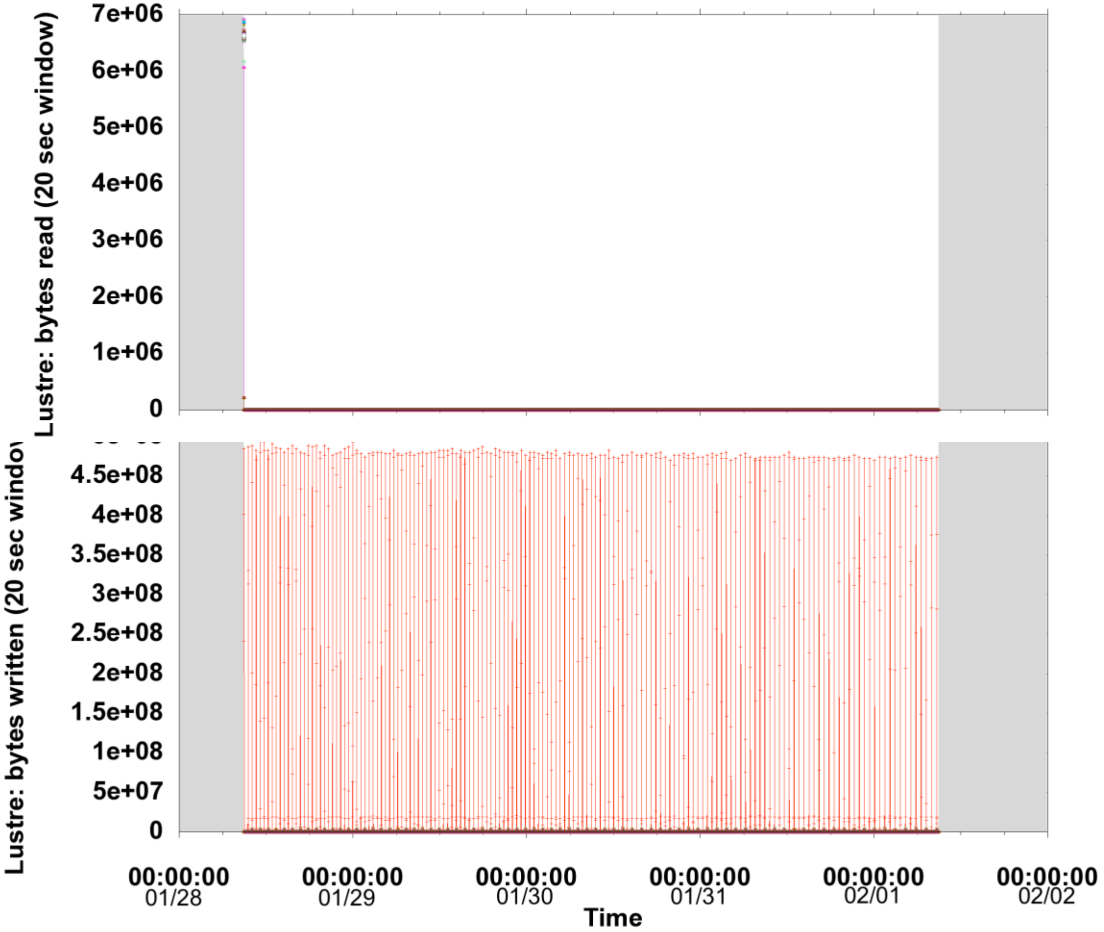
- The CPUs are fully utilized despite a low memory footprint which suggests it is CPU bound
- This simulation is a candidate to run on fewer nodes to increase ensemble throughput



LAMMPS Lustre Read/Write Profiles

Observations

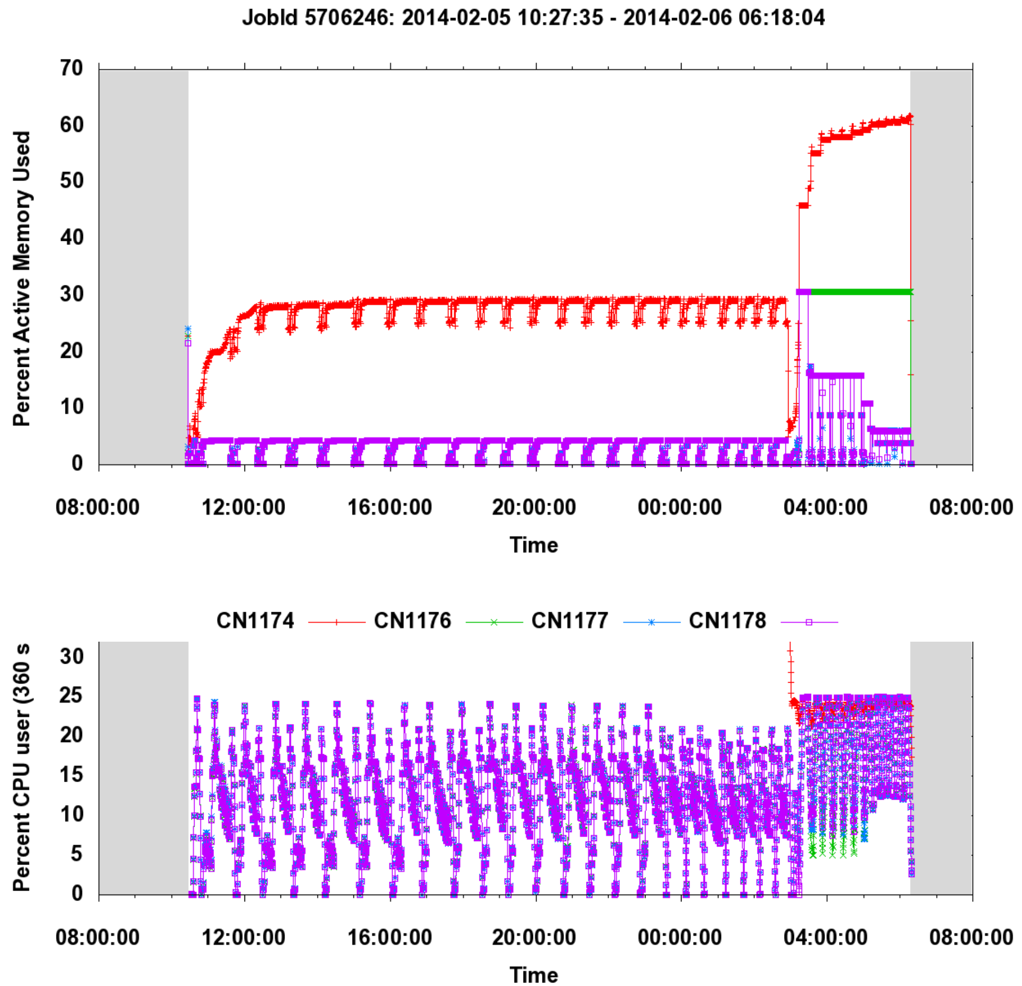
- This behavior is expected due to simulation parameters



Gaussian CPU & Memory Profiles

Observations

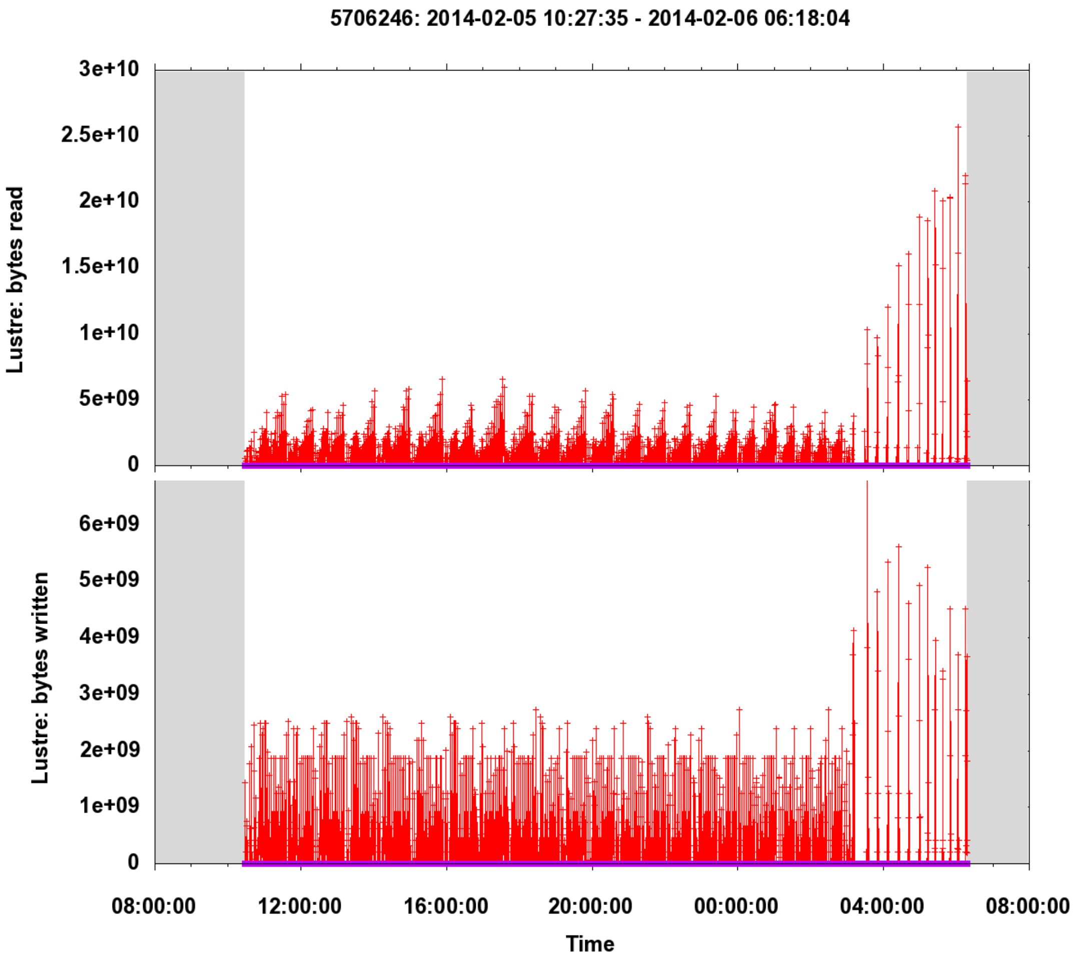
- This particular Gaussian simulation is highly imbalanced in memory and CPU utilization
- There are 2 distinct phases exhibited
  - The first phase requires the most time whilst the second requires the most memory
  - Phase 2's increase in memory is a common cause for OOM errors



Gaussian Lustre Read/Write Profiles

Observations

- This particular Gaussian simulation is highly imbalanced in I/O
  - Practically all I/O goes through the single head node



Scoring

Objective: Direct attention to imbalance, under-utilization, misconfiguration, and/or abnormal run-to-run variation

- High-imbalance scores point out single-node I/O and algorithmic-inefficient cases
- The contrast of low memory usage but high I/O and memory balance activity suggest possible memory bandwidth limitations
- Scores are not sorted or initially interpreted to provide consistent insight
  - For example, a high score is not always “good”
- Work in progress

Background Calculations & Functions

- Usage score = decile_lookup(μ%)
- Balance score = deviation_lookup(σ%)
- %Peak score = decile_lookup(%Peak)
- %Peak = max($M_{i,k}$); maximum of any node and any interval
- $\mu\% = 100 * \text{average}((M_{i,k} * \Delta t_{i,k}) \forall M_{i,k} \neq 0) / \text{LIMIT}$
- Activity% is $\text{count}(M_{i,k} \neq 0) / \text{count}(M_{i,k}) * 100$
- $\sigma\% = 100 * (\sigma / \mu)$
  - M_{ik} is the metric value of the *i*-th sample on the *k*-th node.
  - $\Delta t_{i,k}$ is the time between samples *i-1* and *i* on node *k*.
  - σ = std. deviation($\text{average}((M_i * \Delta t_i) \forall M_i \neq 0)_k$) over nodes in the time-weighted, per-node average
  - LIMIT is physical RAM (e.g., 64 GB) or fair share Lustre bandwidth: BW/MPI_Comm_size (e.g., 80 GB/s / N_p)

RAM Usage

	Single-node Peak (P.U.)		*Average Usage*		*Balance*	
App	**% Peak**	**Score**	**μ (%)**	**Score**	**σ (%)**	**Score**
Nalu	36	4	22	3	7.3	7
CTH	26	3	25	3	0.1	10
Adagio	18	2	2.4	1	3.6	9
Lammps256	17	2	15	2	0.1	8
Gaussian	61	7	12	2	108.4	1

Scoring Scale Legend

Category	Score	1	2	3	4	5	6	7	8	9	10	
Usage	μ <	10	20	30	40	50	60	70	80	90	100	← Decile
Balance	σ <	∞	35	30	25	20	15	10	7	4	1	← Deviation

Scoring Correlates With Memory Observations

Lustre Client Bandwidth R/W

	App	Single-node Peak (P.U.) % Peak	Score	Average Usage μ (%)	Score	Balance σ (%)	Score	% Activity
READ	Nalu	135	10	11	2	7.3	7	1.44
	CTH	65	7	42	5	9.7	7	1
	Adagio	39	4	3	1	10	6	22.14
	Lammps256	0.1	1	0.05	1	94	1	0.01
	Gaussian	30	3	0.5	1	200	1	11.92
WRITE	Nalu	0.01	1	0	1	2263	1	0.02
	CTH	122	10	15.1	2	14	6	2.2
	Adagio	9	1	2.6	1	5	8	37.7
	Lammps256	7	1	0.05	1	140	1	0.52
	Gaussian	8	1	0.22	1	200	1	10.1

Scoring Scale Legend

Category	Score	1	2	3	4	5	6	7	8	9	10	
Usage	μ <	10	20	30	40	50	60	70	80	90	100	← Decile
Balance	σ <	∞	35	30	25	20	15	10	7	4	1	← Deviation

Scoring Correlates With I/O Observations

Conclusions

- LDMS provides necessary capabilities for ***system-level*** profiling
  - **Subsecond sampling** enables accurate insight into behavior
  - **Efficient aggregation** enables minimal-to-no impact on application runtime
  - **System-wide-synchronized data collection** enables: • correlation of events in time and space; • identification of contention for shared resources; • understanding of varying production conditions that can explain performance variations ; • identification of hot spots ; • understanding of application resource demands vs. system provisioning ; • identification of application grouping (e.g. dense blocks of similar behavior implies tight spatial grouping which may imply less network contention and lower latency communication)
- Leveraging LDMS-derived data is an ideal initial profiling step
- Data compression via scoring facilitates broad understanding

Questions?

- Answers other than 42.