SCIENCE-DRIVEN DATA MANAGEMENT FORMULTI-TIERED STORAGE

December 3, 2015

CCP 2015, Guwahati India

S. Klasky	ORNL, GT U. Tenn., NCSU
H. Abbasi, Q. Liu, F. Wang	ORNL
J. Lofstead, M. Curry, L. Ward	Sandia
M. Parashar	Rutgers
C. Maltzahn	UCSC
M. Ainsworth	Brown, ORNL

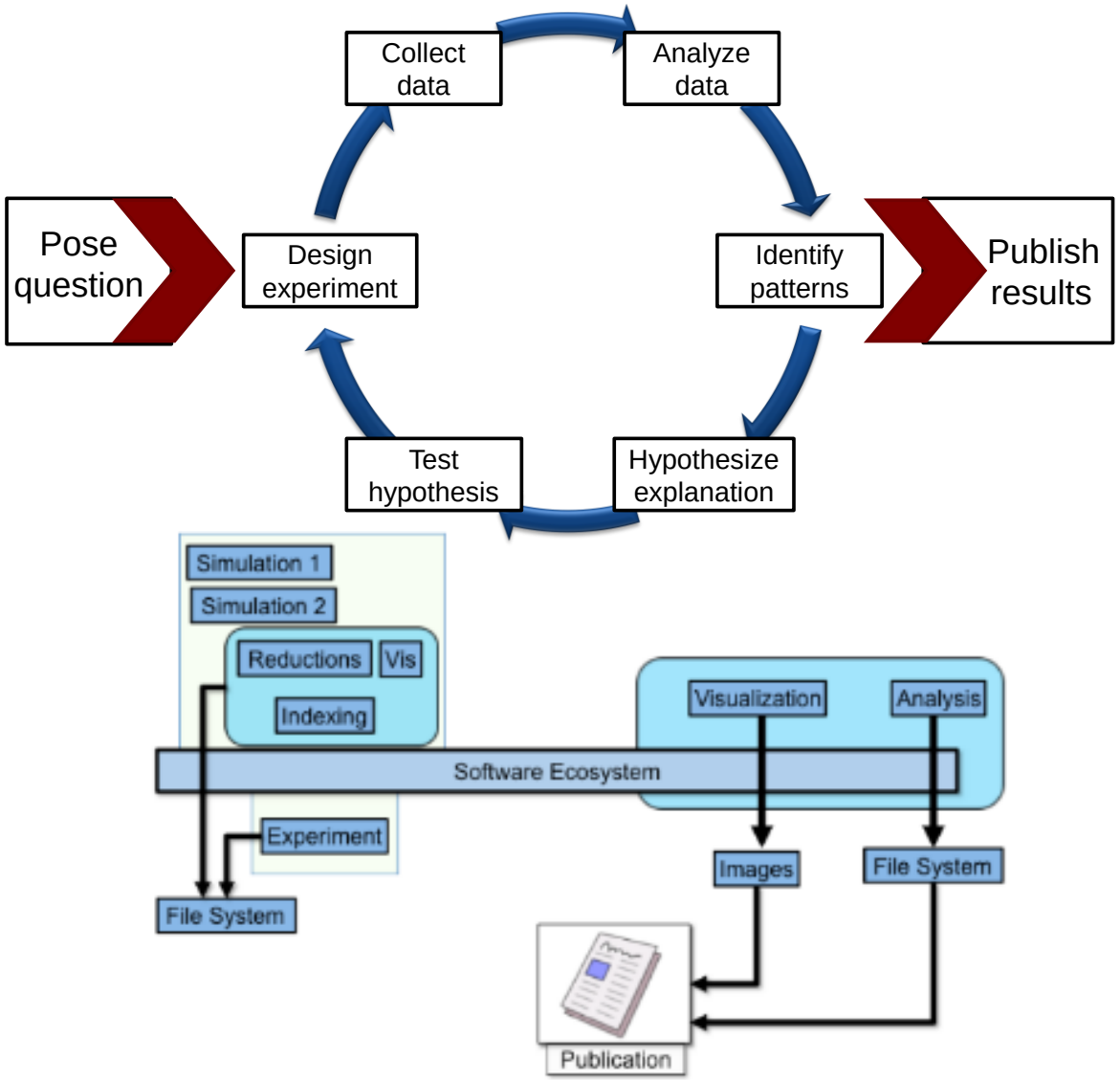
klasky@ornl.gov

Outline

- Motivation
- SIRIUS Building blocks
- Data Description
- Auditing
- Data Refactoring
- Metadata searching
- Fuzzy predictable performance

Where do we spend our time in science

- Goals
  - Accelerate this process
  - Make the process **predictable**
  - Make the process **adaptable**
  - Make the process **scalable** as the complexity increases
  - Make the software **easy-to-use**
- Observation
  - Most of the time is spent in managing, moving, storing, retrieving , and turning the science data into **knowledge**

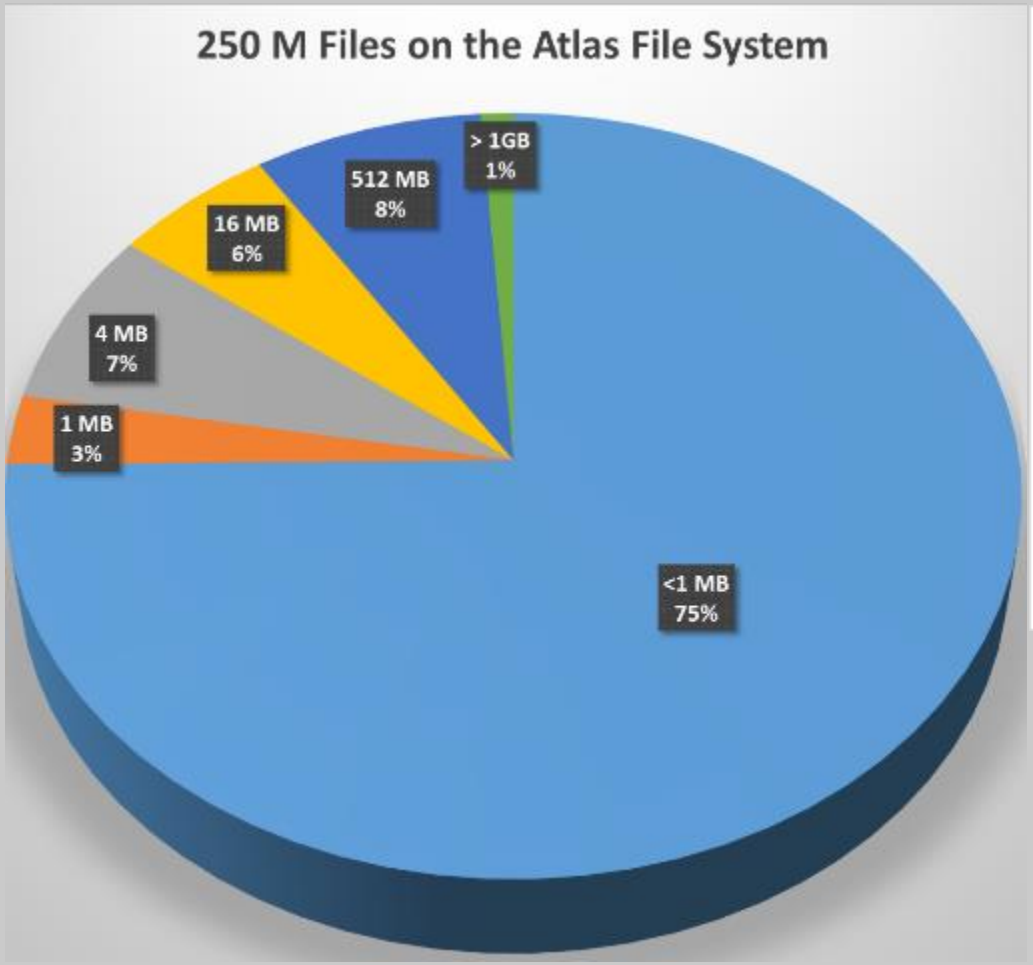


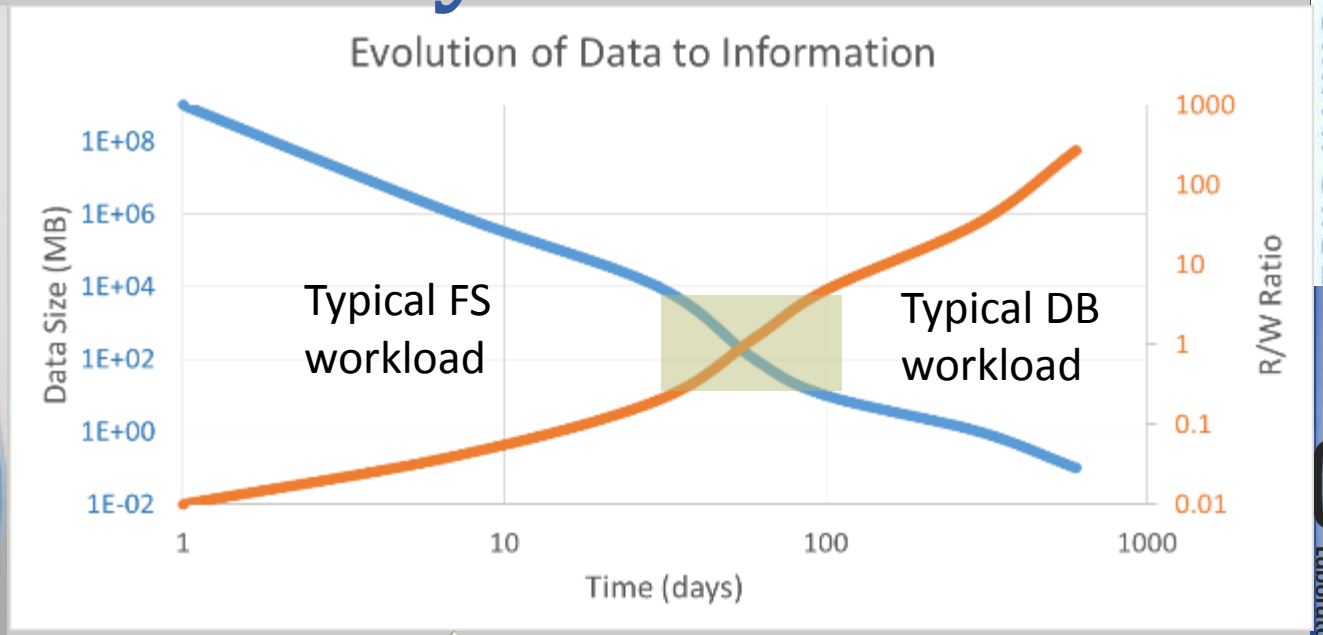
Next Generation DOE computing

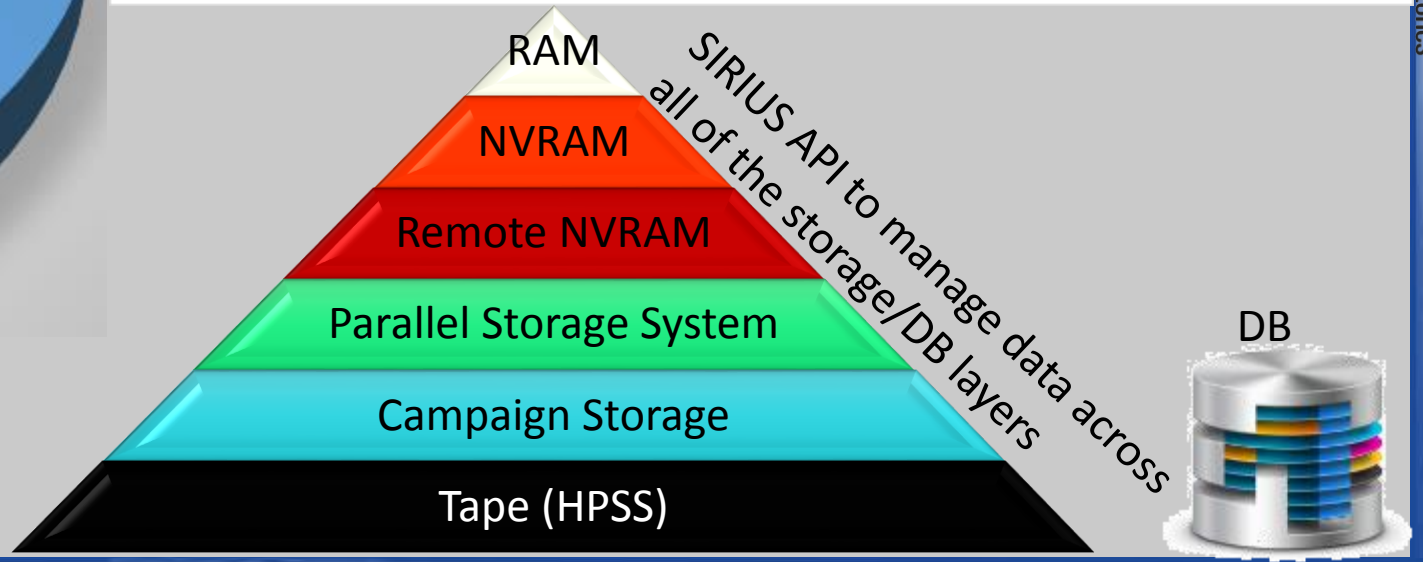
System attributes	NERSC Now	OLCF Now	ALCF Now	NERSC Upgrade	OLCF Upgrade	ALCF Upgrades	
Name Planned Installation	Edison	TITAN	MIRA	Cori 2016	Summit 2017-2018	Theta 2016	Aurora 2018-2019
System peak (PF)	2.6	27	10	> 30	150	>8.5	180
Peak Power (MW)	2	9	4.8	< 3.7	10	1.7	13
Total system memory	357 TB	710TB	768TB	~1 PB DDR4 + High Bandwidth Memory (HBM)+1.5PB persistent memory	> 1.74 PB DDR4 + HBM + 2.8 PB persistent memory	>480 TB DDR4 + High Bandwidth Memory (HBM)	> 7 PB High Bandwidth On-Package Memory Local Memory and Persistent Memory
Node performance (TF)	0.460	1.452	0.204	> 3	> 40	> 3	> 17 times Mira
Node processors	Intel Ivy Bridge	AMD Opteron Nvidia Kepler	64-bit PowerPC A2	Intel Knights Landing many core CPUs Intel Haswell CPU in data partition	Multiple IBM Power9 CPUs & multiple Nvidia Voltas GPUS	Intel Knights Landing Xeon Phi many core CPUs	Knights Hill Xeon Phi many core CPUs
System size (nodes)	5,600 nodes	18,688 nodes	49,152	9,300 nodes 1,900 nodes in data partition	~3,500 nodes	>2,500 nodes	>50,000 nodes
System Interconnect	Aries	Gemini	5D Torus	Aries	Dual Rail EDR-IB	Aries	2nd Generation Intel Omni-Path Architecture
File System	7.6 PB 168 GB/s,	32 PB 1 TB/s, Lustre®	26 PB 300 GB/s GPFS™	28 PB 744 GB/s Lustre®	120 PB 1 TB/s GPFS™	10PB, 210 GB/s Lustre initial	150 PB 1 TB/s Lustre®

klasky@ornl.gov

Abstractions across File System to DB







klasky@ornl.gov

RSVP SIRIUS

Two Basic principles of Sirius

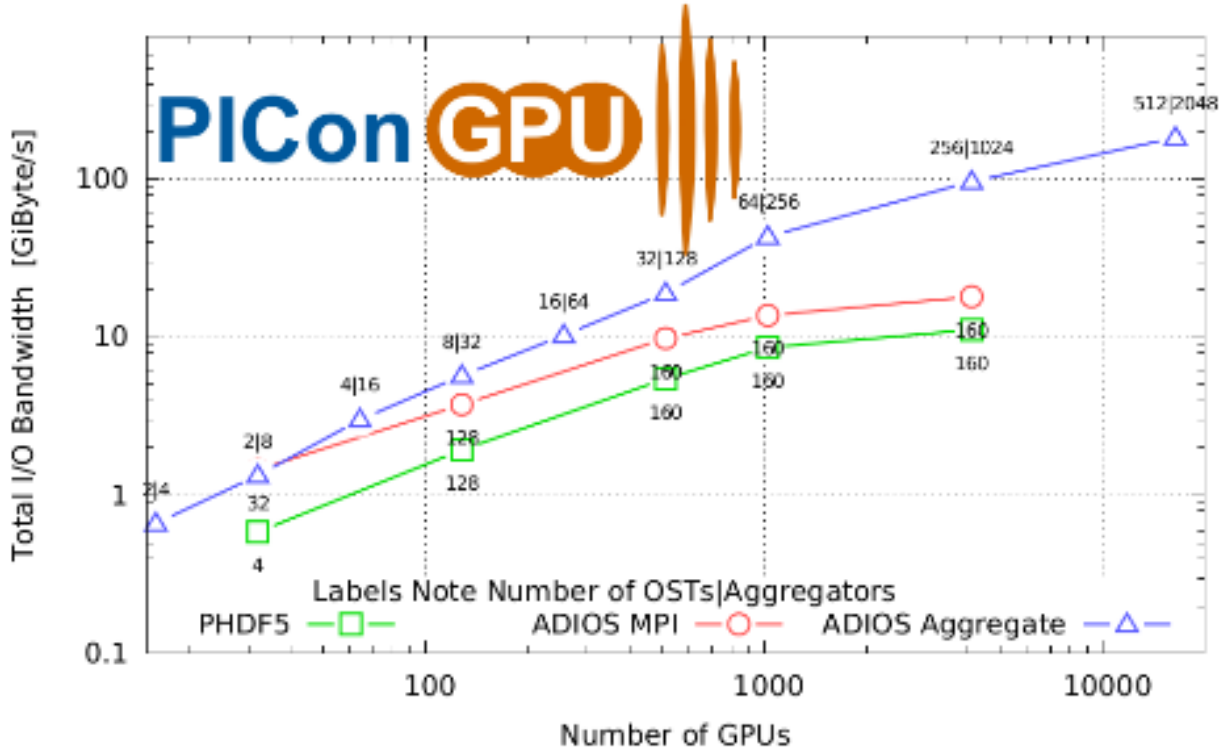
- **Principle 1**: A knowledge-centric system design that allows user knowledge to define data policies
  - Today SSIO layers are written in a stove-pipe fashion, and quite often do not allow optimizations to take place
  - Re-design the layers in a highly integrated fashion where users place their intentions into the system and actions will statically and dynamically take place to optimize for the system and for individual requests
- **Principle 2:** Predictable performance and quality of data in the SSIO layers needs to be established so science can be done on the exascale systems in a more efficient manner
  - Without predictable performance, not only can the runs be slowed down because of shared resource contention but also it affects key science decisions

Outline

- Motivation
- SIRIUS Building blocks
- Data Description
- Auditing
- Data Refactoring
- Metadata searching
- Fuzzy predictable performance

ADIOS

- An I/O abstraction framework
- Provides portable, fast, scalable, easy-to-use, metadata rich output
- Choose the I/O method at runtime
- Abstracts the API from the method
- Need to provide solutions for “90% of the applications”



- **Astrophysics**
- **Climate**
- **Combustion**
- **CFD**
- **Environmental Science**
- **Fusion**
- **Geoscience**
- **Materials Science**
- **Medical: Pathology**
- **Neutron Science**
- **Nuclear Science**
- **Quantum Turbulence**
- **Relativity**
- **Seismology**
- **Sub-surface modeling**
- **Weather**

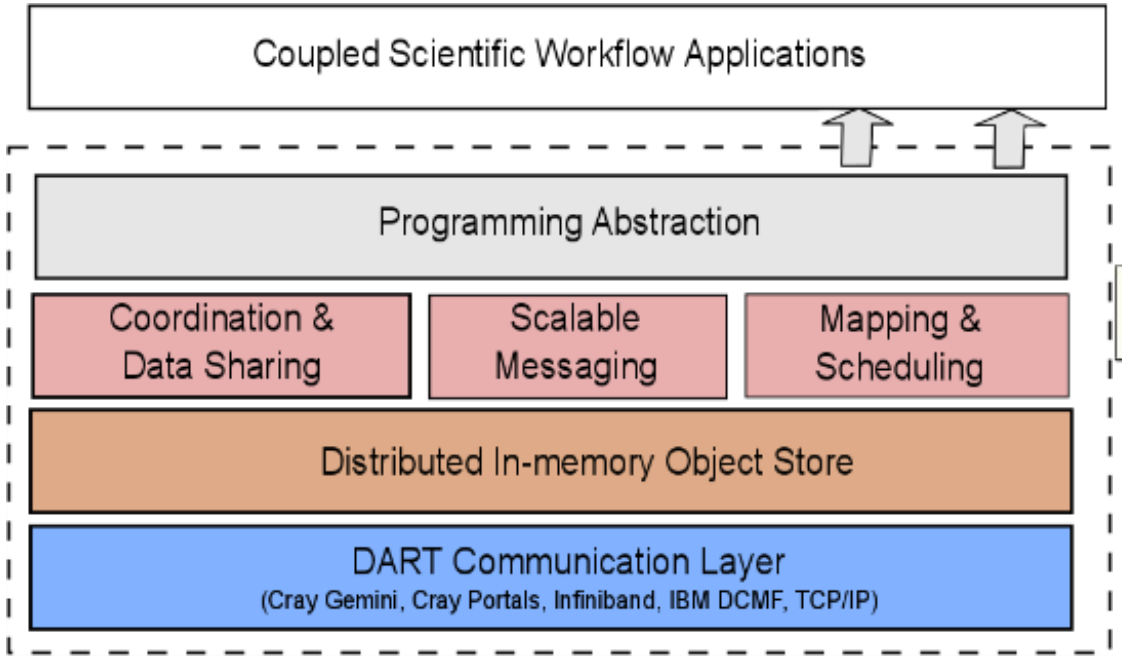
- Klasky, S. Ethier, Z. Lin, K. Martins, D. McCune, R. Samtaney, *Grid-based parallel data streaming implemented for the gyrokinetic toroidal code* in *Proceedings of the 2003 ACM/IEEE conference on Supercomputing*, ACM, p. 24.
- J. F. Lofstead, S. Klasky, K. Schwan, N. Podhorszki, C. Jin, *Flexible io and integration for scientific codes through the adaptable io system (adios)* in *Proceedings of the 6th international workshop on Challenges of large applications in distributed environments*, ACM, pp. 15–24.
- J. Lofstead, F. Zheng, S. Klasky, K. Schwan, *Input/output apis and data organization for high performance scientific computing* in *Petascale Data Storage Workshop, 2008. PDSW'08. 3rd*, IEEE, pp. 1–6.
- J. Lofstead, F. Zheng, S. Klasky, K. Schwan, *Adaptable, metadata rich IO methods for portable high performance IO* in *Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, IEEE, pp. 1–10.
- H. Abbasi, J. Lofstead, F. Zheng, K. Schwan, M. Wolf, S. Klasky, *Extending i/o through high performance data services* in *Cluster Computing and Workshops, 2009. CLUSTER'09. IEEE International Conference on*, IEEE, pp. 1–10.
- Q. Liu, J. Logan, Y. Tian, H. Abbasi, N. Podhorszki, J. Y. Choi, S. Klasky, R. Tchoua, J. Lofstead, R. Oldfield, et al.. Hello ADIOS: the challenges and lessons of developing leadership class I/O frameworks. *Concurrency and Computation: Practice and Experience* **2014**, *26*, 1453–1473.

http://www.nccs.gov/user-support/center-projects/adios/

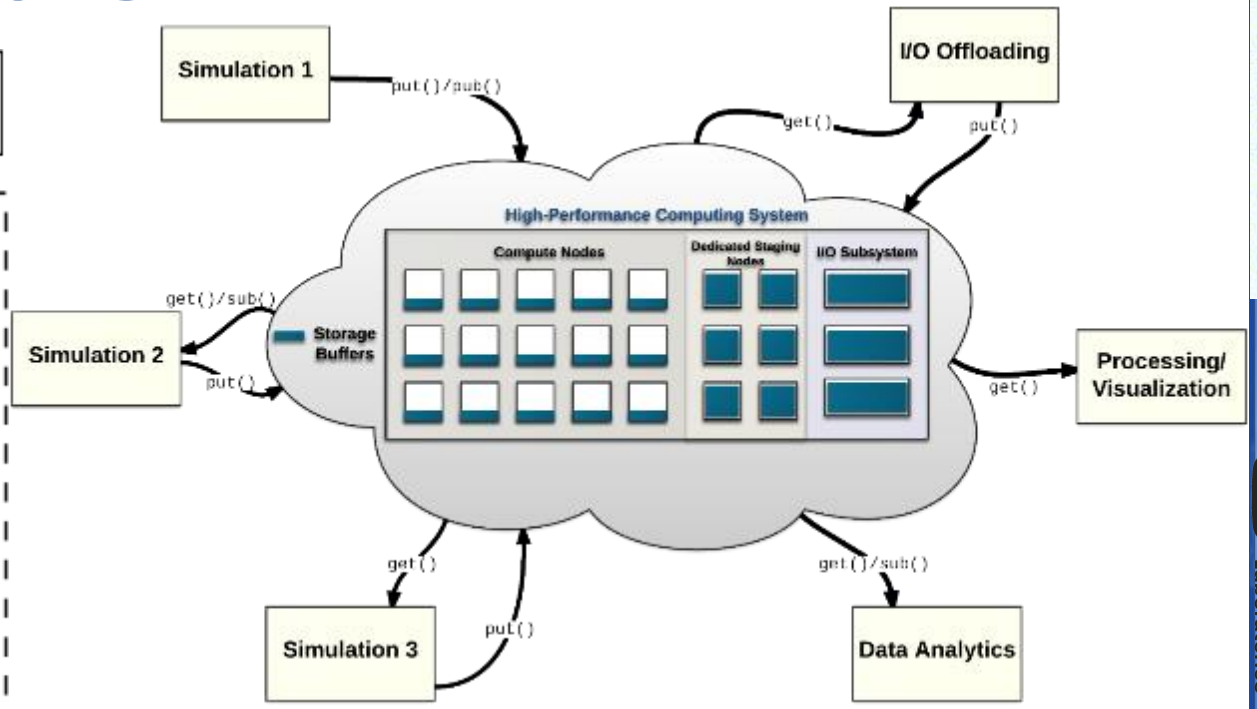
klasky@ornl.gov

RSVP SIRIUS

DataSpaces – Rutgers



The DataSpaces Abstraction



- Virtual shared-space programming abstraction
  - Simple API for coordination, interaction and messaging
- Distributed, associative, in-memory object store
  - Online data indexing, flexible querying
- Adaptive cross-layer runtime management
  - Hybrid in-situ/in-transit execution
- Efficient, high-throughput/low-latency asynchronous

The Sirocco Object Store – SOS - Sandia

- A two-part system:
  - A low-level, hierarchical fixed-depth object storage system
  - Smart clients that expose user APIs
- Light Weight File System-inspired philosophy
  - Clients bring/opt-in to services they require
  - Naming, locking, distributed transactions
- Peer-to-peer inspired design
  - Ephemeral servers and clients
  - Data and location(s) are decoupled
  - Greedy optimization of QoS (network, storage, reliability)
  - Popularity drives copy creation

http://www.cs.sandia.gov/Scalable_IO/sirocco/

Outline

- Motivation
- SIRIUS Building blocks
- Data Description
- Auditing
- Data Refactoring
- Metadata searching
- Fuzzy predictable performance

Data Descriptions

How do we describe/annotate data generated from applications?

- To capture application knowledge and communicate them to middleware/storage
  - **Data utility**
  - **Relationships between datasets**
  - Semantics
- To specify user requirements
  - QoS :E.g., bandwidth
  - Policy: E.g., where and how long should my data stay on a storage layer

Conceptual APIs

1. **sirius_init**
2. **sirius_finalize**
3. **sirius_open** -intent
4. **sirus_close** -intent
5. **sirius_set_refactor**
   -refactoring_fn
6. **sirius_get_refactor.**
   -refactoring_fn
7. **sirius_sched_write**
   -utility
8. **sirius_sched_read**
   -utility
9. **sirius_set_deadline**
   -deadline, fuzziness
10. **sirius_est_write**
    -time_est, confidence
11. **sirius_est_read**
    -time_est, confidence
12. **sirius_perform_write**
13. **sirius_perform_read**
    -deadline, fuzziness

How does the system reassemble the pieces after the data has been refactored?
Re-factoring might mean combing different data chunks together, and this takes time!

Outline

- Motivation
- SIRIUS Building blocks
- Data Description
- Auditing
- Data Refactoring
- Metadata searching
- Fuzzy predictable performance

New techniques for "Data Intensive Science"

AUDITOR: An additional "simulation" whose purpose is to monitor the fine scale simulation and initiate appropriate actions when anomalies are detected

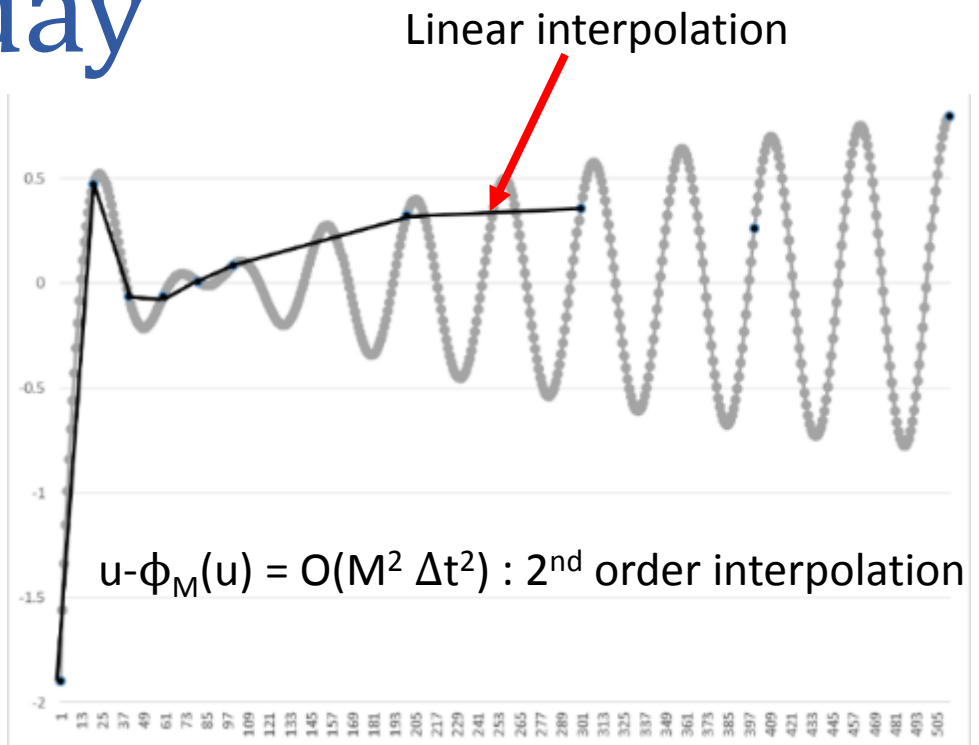
Examples

- Trigger a: checkpoint, roll-back, local change in a function, …
- Not confined to stability issues because it will always reset
- Can allow data regeneration cheaply

Basic quantities in Information Theory

- Data stream S and for $x \in S$ let $P_r(X=x) = p_x \in [0,1]$
- Shannon Information Content: $h(x) = - \log_2 p_x$
- Entropy $H(S) = - \Sigma\, p_x \log_2 p_x$
- **Noisy/random data has HIGH ENTROPY**

Current practices of today

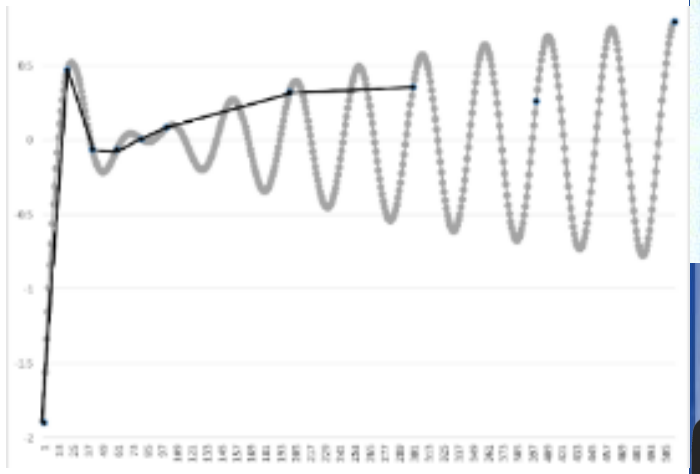


- Want to write data every n^{th} timestep
  - Because of the Storage and I/O requirements users are forced to writing less
- Common practice is to write data at every m^{th} timestep, stride = M
- If the users reconstruct their data, u(t), at the n^{th} timestep, they need to interpolate between the neighboring timesteps
  - $\Phi_M(u)$ = interpolant on coarser grid (stride M), reduce storage my 1/M
- Assume (C=constant depending on the complexity of the data)
  - Original storage cost = 32*N bits (floats)
  - New storage cost = 32*N/M bits + $\{ 23 - \log_2 (C\ M^2\ \Delta t^2)\}N$
  - Ratio = $(1/M - 1/16\ \log_2 M) - 1/16\ \log_2 \Delta t$ + constant

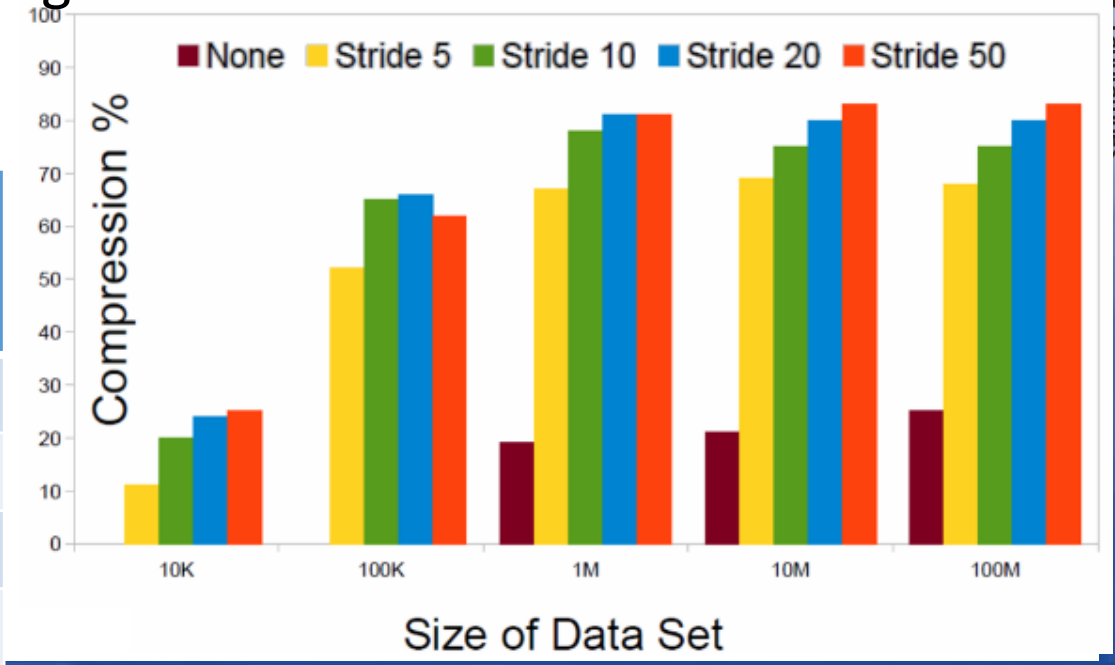
Cost to store ϕ_M +
Cost to store mantissa of u- $\phi_M(u)$

Compression with an interpolation auditor

- Linear interpolation (LA) is the auditor
- If we look at 10MB output, with a stride of 5
  - Total output = 50MB for 5 steps
  - 10 MB, if we output 1 step, 43MB “typical lossless compression”, 18MB, using linear auditing but lossless
- Investigating adaptive techniques



Stride	1 step (MB)	lossless compression (MB)	Linear Audit (MB)	Total Data in 50 steps, typical compression	**Total data in 50 steps in LA**
5	10	43	18	430	**180**
10	10	85	25	850	**125**
20	10	170	40	1700	**100**
50	10	**425**	100	4250	**100**



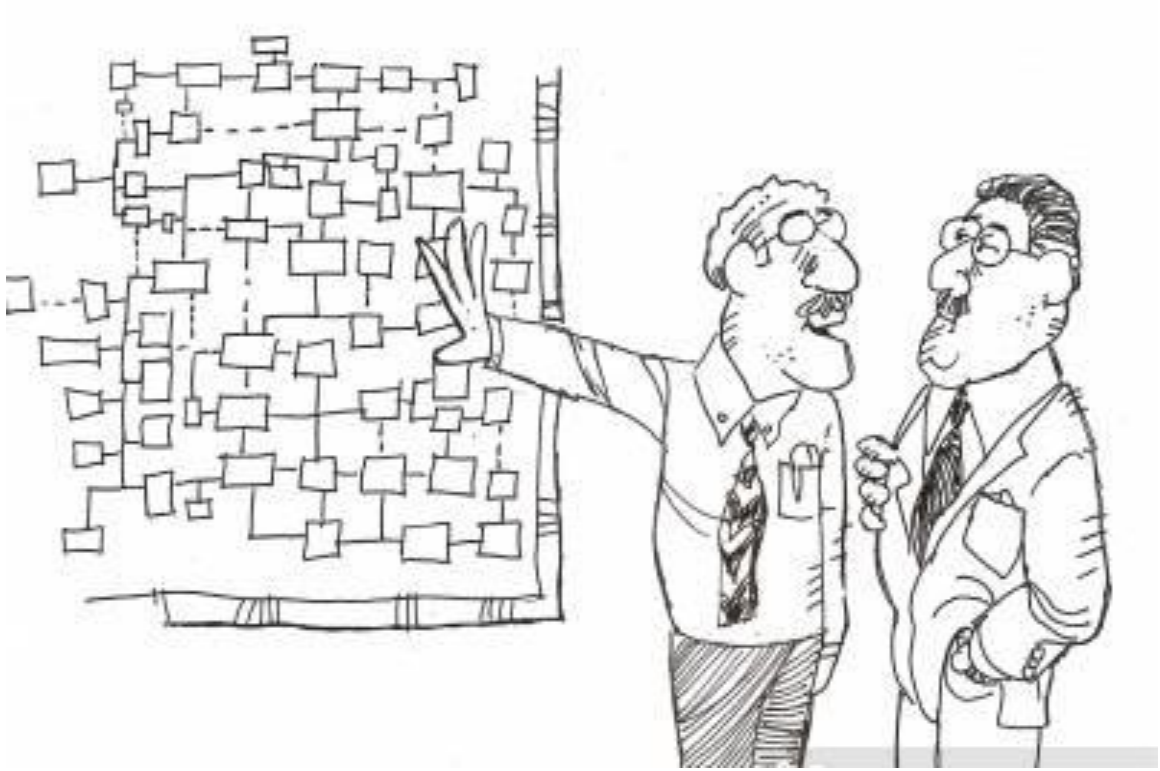
klasky@ornl.gov

Other types of auditors

- The key to better auditors is to understand what you are simulating/observing
  - Use a reduce model
  - Use less resolution
  - Use a linear equation for short spatial/temporal regions
- And other ways to refactor
  - Precision based re-organization
  - Frequency based re-organization - Wavelets
  - More knowledgeable auditors
    - Cost of data re-generation vs. data storage/ retrieval
- Storage becomes more than a stream of bytes
  - Data + Code + workflow

Outline

- Motivation
- SIRIUS Building blocks
- Data Description
- Auditing
- Data Refactoring
- Metadata searching
- Fuzzy predictable performance



Computer science challenges in Data Refactoring

- To understand the cost associated with refactoring data, e.g., the CPU cycles, extra memory consumed, and communication.
  - For an auditor-like approach
    - Where should the auditor run, taking advantage of locality as much as possible?
    - How much resources to allocate?
    - How will the auditor and the application communicate?
    - Most importantly, how do we take both error bound and cost into consideration.
- Does the refactoring affect the fidelity of the applications? If so, how much?
- After data is refactored, how do we map it to the storage hierarchy? How do we enforce policy?

Utility-driven Data Placement

Goal: Determine placement of data objects **vertically** across different levels of the memory hierarchy, e.g., SSD or DRAM, and **horizontally** at different staging nodes

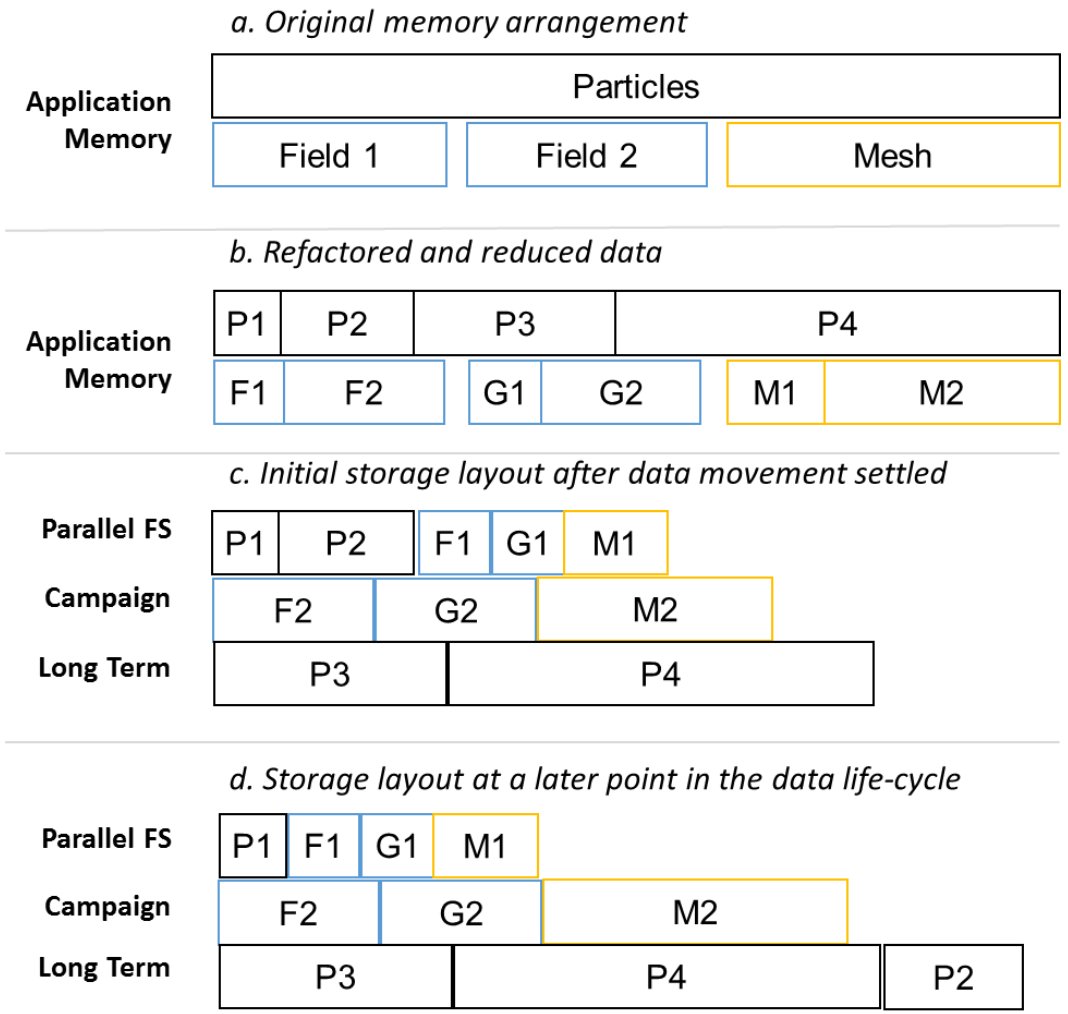
- Utility quantifies the relative value of data objects in the staging area based on anticipated data read patterns
  - Utility based on data access patterns (monitored and learnt at runtime) and the location of the application and staging nodes within the system network topology
  - For example, data objects with ***higher data utility*** are placed ***closer*** to the computation nodes accessing it

1. *Exploring Data Staging Across Deep Memory Hierarchies for Coupled Data Intensive Simulation Workflows.* T. Jin, F. Zhang, Q. Sun, H. Bui, M. Romanus, N. Podhorszki, S. Klasky, H. Kolla, J. Chen, R. Hager, C. Chang, M. Parashar. *IEEE* IPDPS'15, May 2015

2. *Adaptive Data Placement For Staging-Based Coupled Scientific Workflows.* Q. Sun, T. Jin, M. Romanus, H. Bui, F. Zhang, H. Yu, H. Kolla, S. Klasky, J. Chen, M. Parashar. ACM/IEEE SC'15, Nov. 2015.

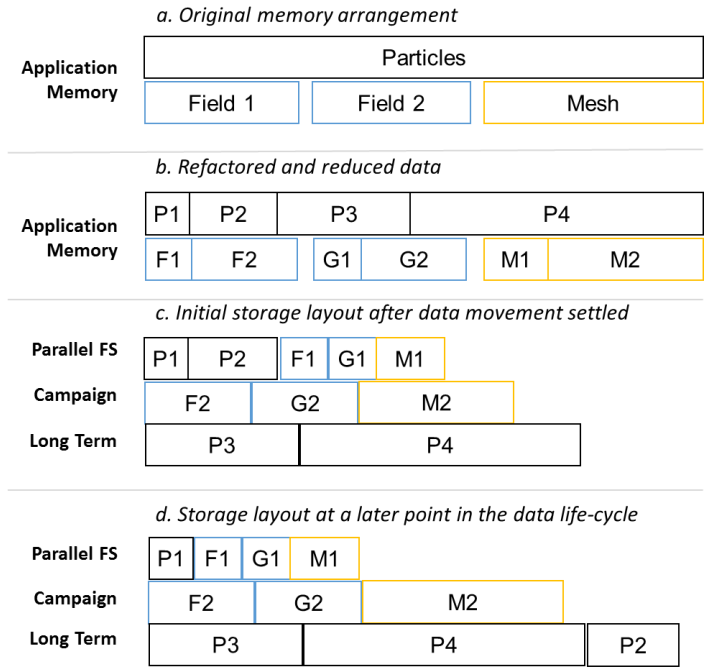
Data Refactoring and Utility Functions

- Need to explore what and where the computation will occur
  - Flexibility in location based on past work
  - Flexibility in which operation to perform on which chunk of data
- Code generation or code containers are potential study targets
- Maintaining relationship between data chunks
- Carry attributes from generation to consumption and feedback into a utility computation



Utility and the User

- Describes how long a data chunk will live at a level of the storage hierarchy
- Utility is a broad description
  - Spatial or temporal utility of data
  - Utility based on in-data features
  - Utility based on statistical features
- Utility has a large component from the user and the use case
  - Experimental design factors in here
  - Solving a specific scientific problem => specific data utility function
- API for ingesting user preferences and combining with historical provenance
- Dynamic utility for online analysis/visualization use cases



e.g. The utility of F1 may be defined more explicitly as, for example, (priority=1, (time-NVRAM=8 hours, time-PFS=30 days, time-CAMPAIGN=100 days, time-TAPE=1000 days), (priority=2, (time-NVRAM=1 hours, time-PFS=4 days, time-CAMPAIGN=100 days, time-TAPE=300 days).

Outline

- Motivation
- SIRIUS Building blocks
- Data Description
- Auditing
- Data Refactoring
- Metadata searching
- Fuzzy predictable performance

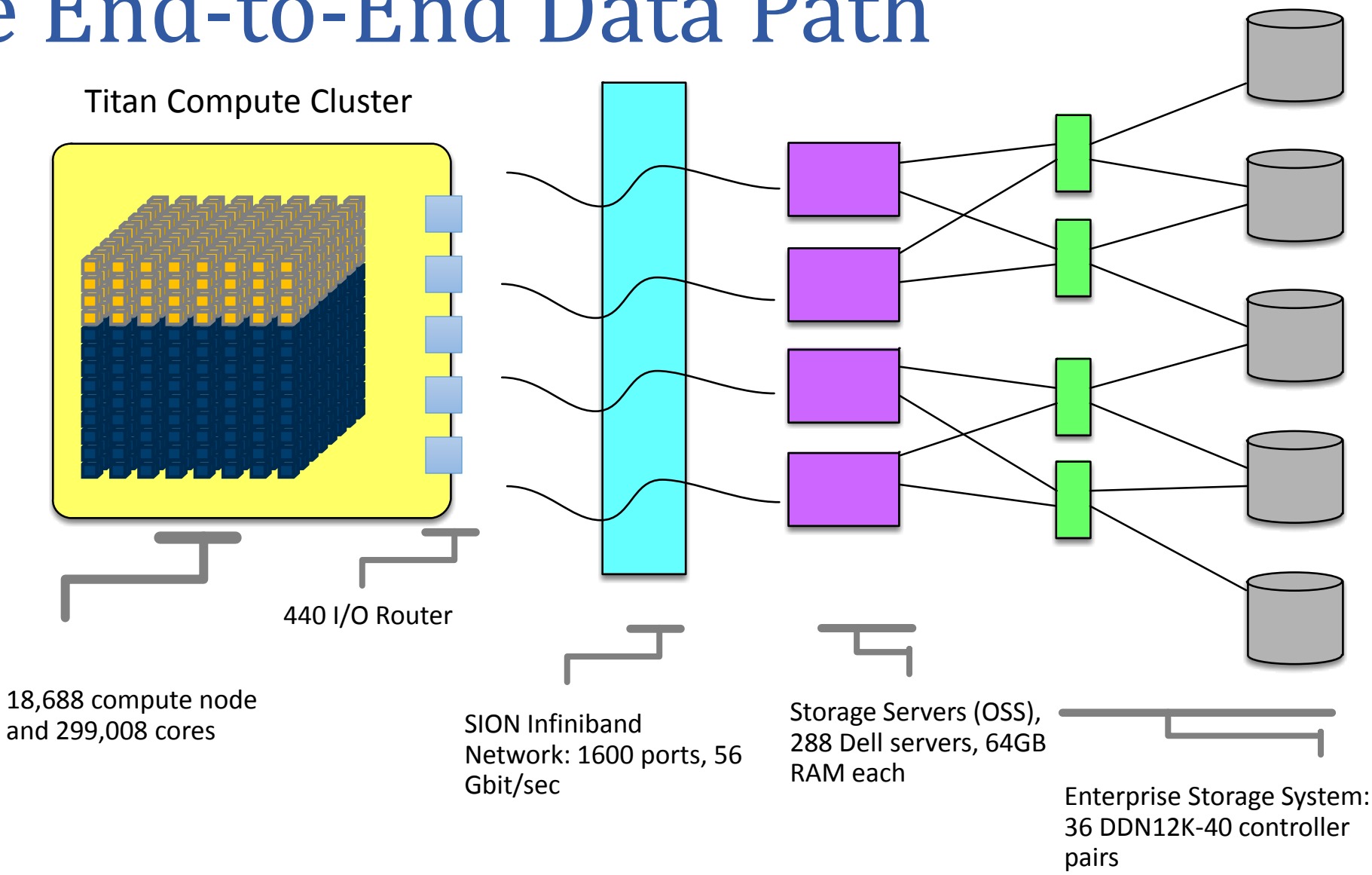
Challenges in metadata searching

- Storage devices rather than a file system: no built-in metadata operations as part of IO
- Distributed pieces EVERYWHERE
- Resilience copies
- Storage devices come and go
- Performance characteristics can vary considerably
- Pick a variety of storage targets based on data “importance”
- Different data “compression” on different data pieces
- Try to “guarantee” performance
  - Need to consider decompression/regeneration time if multiple versions exist
- Enhance placement decision based on predicted future use
  - Based on tracking previous use (which needs to be tracked somehow)

Outline

- Motivation
- SIRIUS Building blocks
- Data Description
- Auditing
- Data Refactoring
- Metadata searching
- Fuzzy predictable performance

The End-to-End Data Path



Autonomic Runtime Optimization

Autonomic Objective (AO):

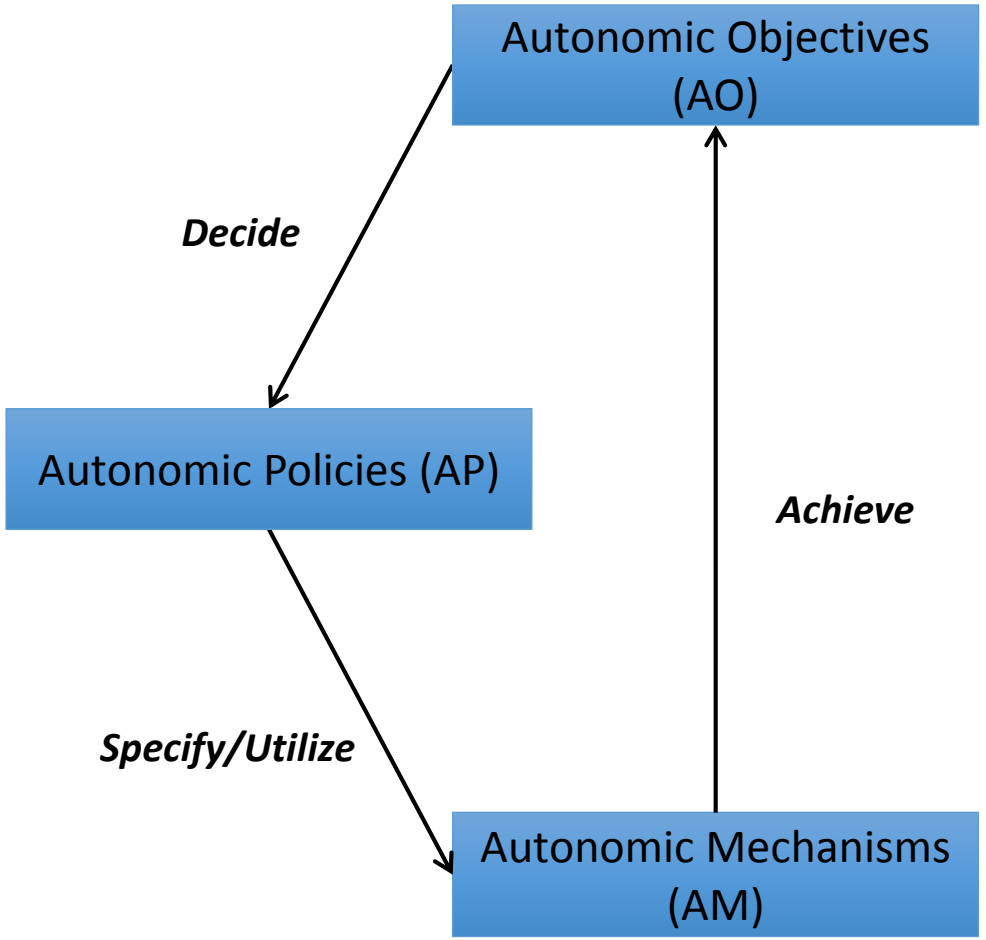
- A requirement/objective/goal defined by the user
- *E.g. minimize data movement, optimize throughput, etc.*

Autonomic Policy (AP)

- A rule that defines how the objectives should be achieved, i.e., which mechanisms should be used
- *E.g. use a specific data placement adaptation to minimize data movement , etc.*

Autonomic Mechanism (AM)

- An action that can be used to achieve an AO
- *E.g. use topology-aware and access-pattern driven data placement to minimize data movement, etc.*

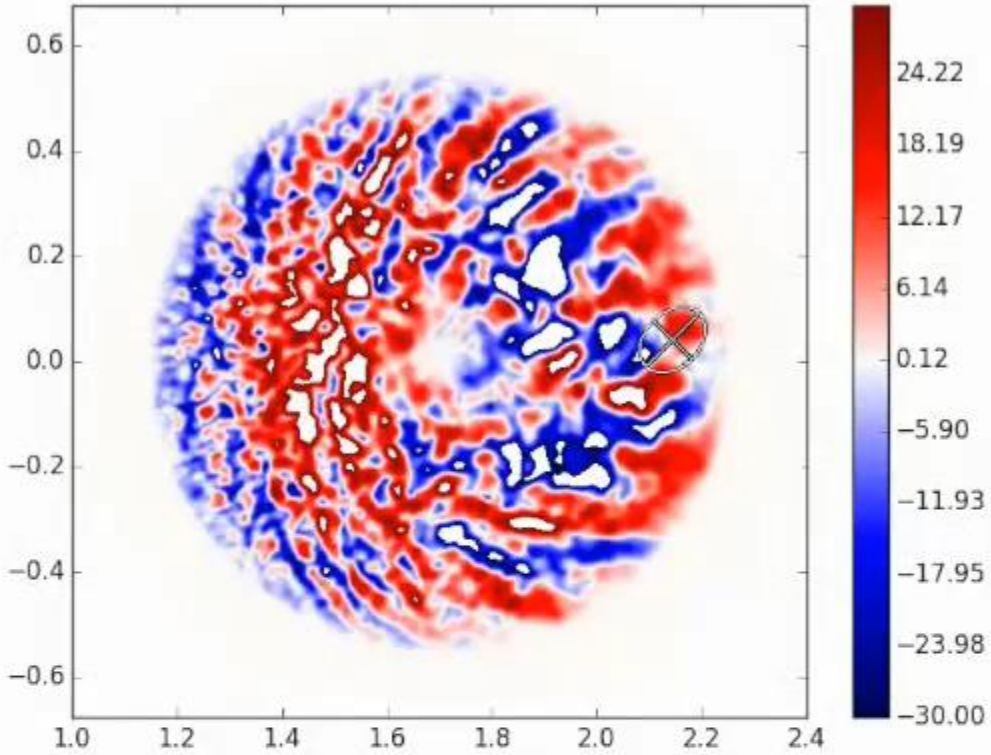


Challenges

- Scalable admission control
  - Need scalable control plane (leverage existing work on causal metadata propagation and online sampling for latency/error trade-offs)
- Resource-specific schedulers to make performance of resource predictable
  - Example: read/write separation at flash devices
- Online sampling to provide quick latency/resolution trade-offs
  - Without significantly interfering with ongoing workload

Summary

- Sirius is attempting to redefine I/O based on key findings
  - I/O abstractions (Posix) do not give the system enough information to fully optimize
  - Data is too big to keep in one place and current systems purge data without user intervention
  - Variability is too large, and users are not in control of their data
- Scientific Data is not random data
  - There is content to the data
- Auditing calculations to prioritize, reduce data sizes but keep the information is critical to reduce the time of understanding



R1 R2 Angle X Y REV TRACKING
0.0717066031626 0.0517881022841 45.0 2.14366197183 0.0422535211268 1.0 0

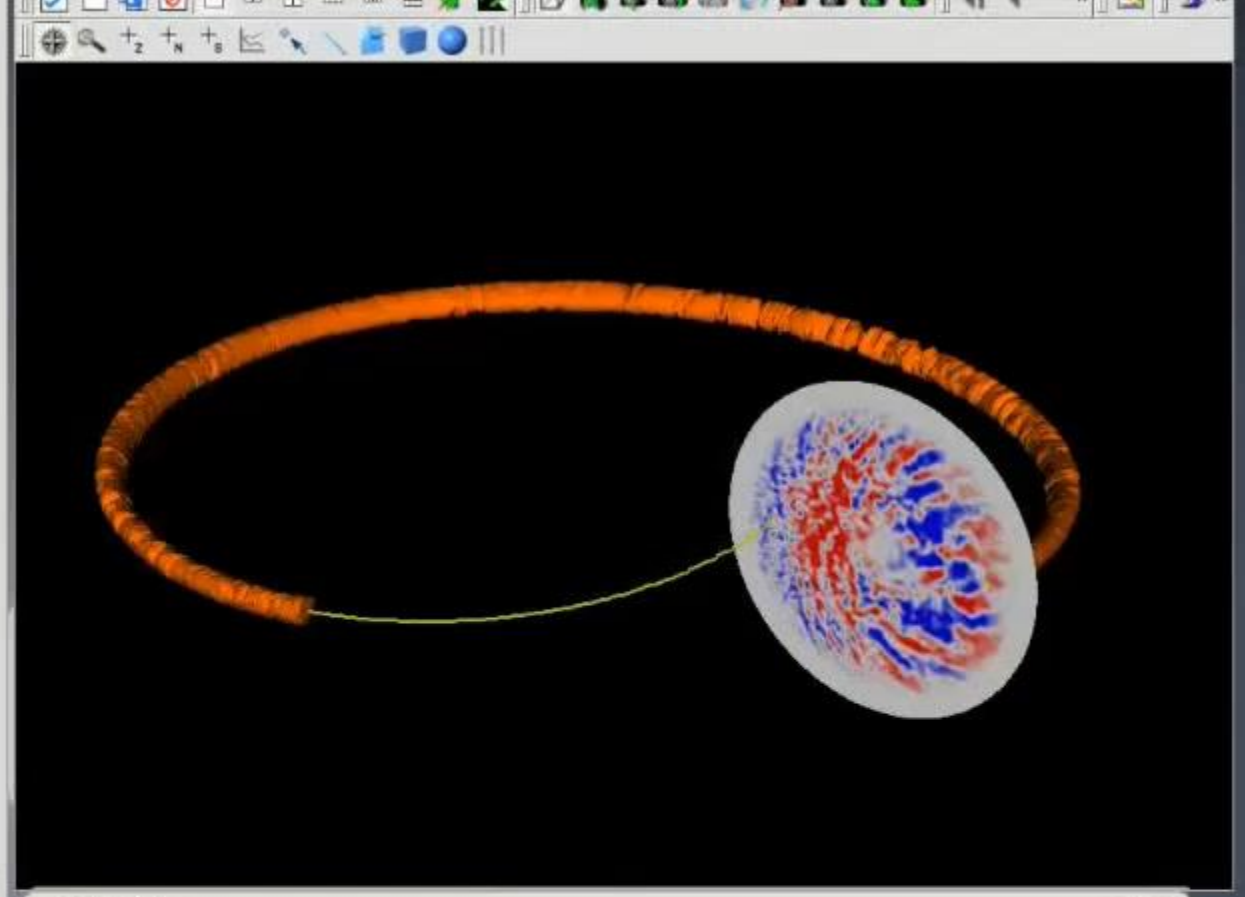
MultiPick

Accept Multi Select

Clear Selection

Number of Revolutions

ID Tracking: OFF



Sith-ADIOS

File Edit View Search Terminal Tabs Help

Sith-ADIOS × | sith-Visit × | sith × | focus × | rhea-visit-build ×

```
>>> Sleeping ... 5 (seconds)
>>> Reading ... totalf_itg_tiny/xgc.3d.08010.bp
>>> Writing ... xgc.3d.bp
>>> Written 84,764,672 bytes, Elapsed 0.066 seconds (Throughput: 1,216.485 MB/sec)
>>> Reading ... xgc-bbox.bp
>>> Reading ... totalf_itg_tiny/restart_dir/xgc.restart.08010.bp
>>> Reading ... totalf_itg_tiny/restart_dir/xgc.restart.08010.bp.info
>>> Read particle efficient ...
>>> Reading particles elapsed 2.191 seconds
>>> Writing ... xgc.particle.bp
>>> Written 7,251,168 bytes, Elapsed 0.050 seconds (Throughput: 139.048 MB/sec)
>>> Sleeping ... 5 (seconds)
>>> Reading ... totalf_itg_tiny/xgc.3d.08020.bp
>>> Writing ... xgc.3d.bp
>>> Written 84,764,672 bytes, Elapsed 0.049 seconds (Throughput: 1,637.854 MB/sec)
>>> Reading ... xgc-bbox.bp
>>> Waiting a newer version: xgc-bbox.bp
>>> Waiting a newer version: xgc-bbox.bp
>>> Reading ... totalf_itg_tiny/restart_dir/xgc.restart.08020.bp
>>> Reading ... totalf_itg_tiny/restart_dir/xgc.restart.08020.bp.info
```

Questions

klasky@ornl.gov