Exploiting Spark for HPC Simulation Data: Taming the Ephemeral Data Explosion

Salishan Conference on High Speed Computing

Ming Jiang
Computer Scientist

April 27, 2022

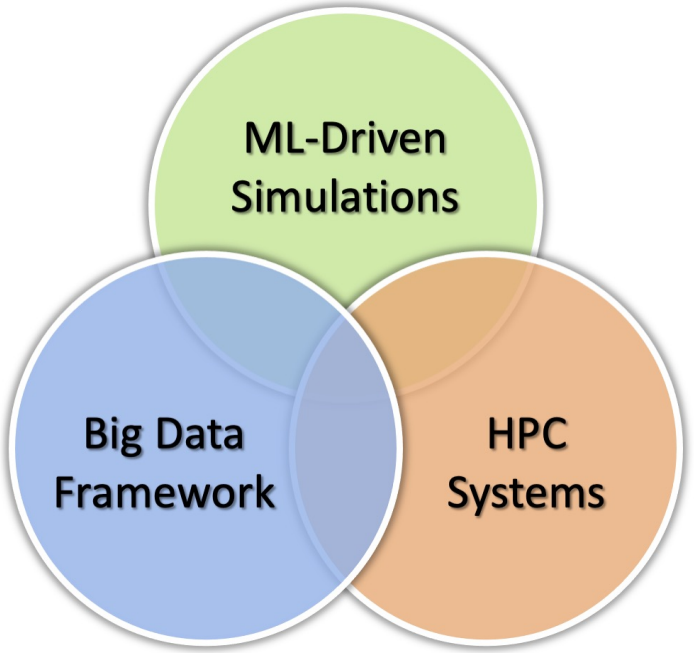
COMPUTING

LLNL-PRES-801486
This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC

Lawrence Livermore National Laboratory

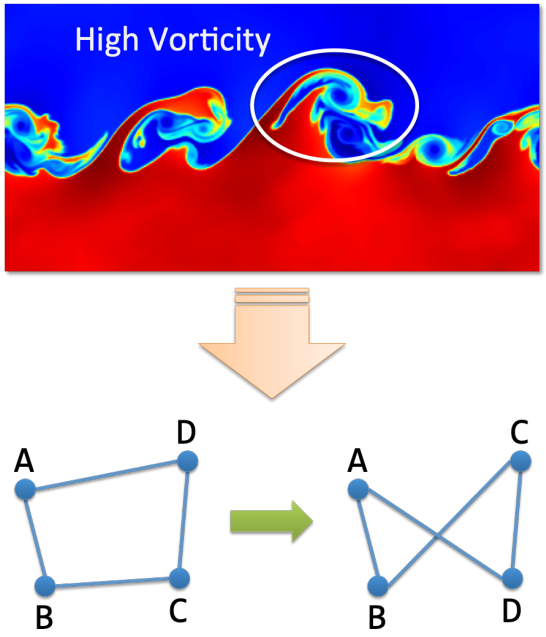
Applying machine learning to HPC simulation data is a challenge

- Growing interest in applying machine learning (ML) to multi-physics simulations on HPC systems
  - HPC simulation data can be massive!
  - Deploying ML algorithms at that scale is a challenge
- Big Data frameworks (Apache Spark) offer solution
  - Heavily utilized in industry
  - Demonstrated scalability … on Big Data clusters
- How practical is it for HPC users on HPC systems?
  - **Focus of this research:** identify *best practices* for exploiting Spark for massive HPC simulation data



ALE simulation codes are vital, but they're difficult to use!

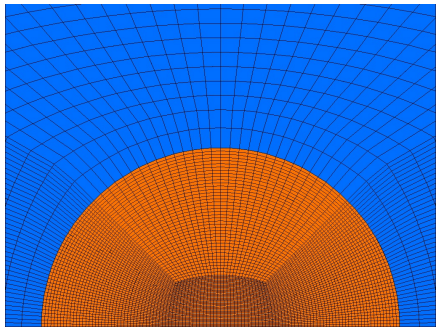
- Arbitrary Lagrangian-Eulerian (ALE) codes
  - Used in many scientific and engineering applications
  - Vital to DOE/NNSA Stockpile Stewardship Science
- **Problem:** ALE codes are difficult to use!
  - Suffer from code failures
  - Require user intervention
  - High time-to-solution
- **Objective:** exploit ML to automate ALE codes
  - Drastically reduce user time and effort needed to produce solutions



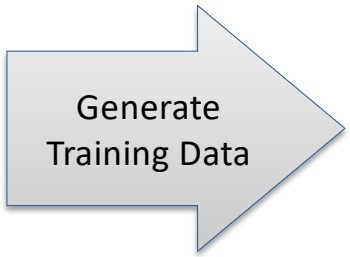
Mesh Tangling Failures

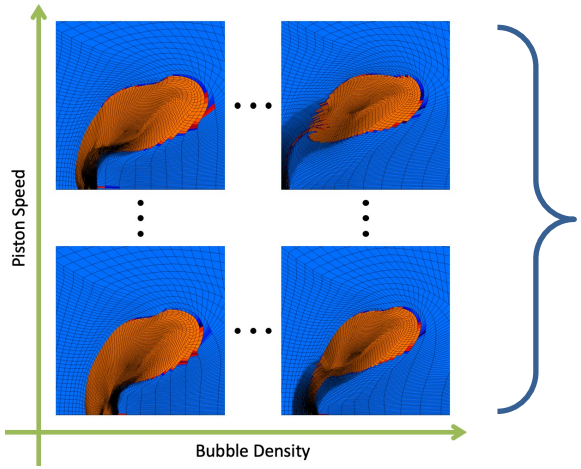
Use ML to avoid failures during ALE simulation runs

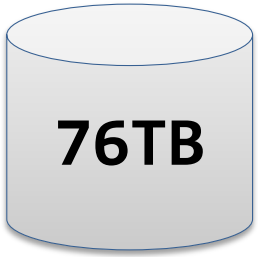
- We developed a supervised learning framework* to predict ALE simulation failures
  - We trained a Random Forest model to predict mesh tangling failures
  - For generating training/testing data, we varied initial conditions of simulation
  - Bubble Shock simulation training data is 76TB, with over **1 trillion learning examples!**



Bubble Shock Simulation





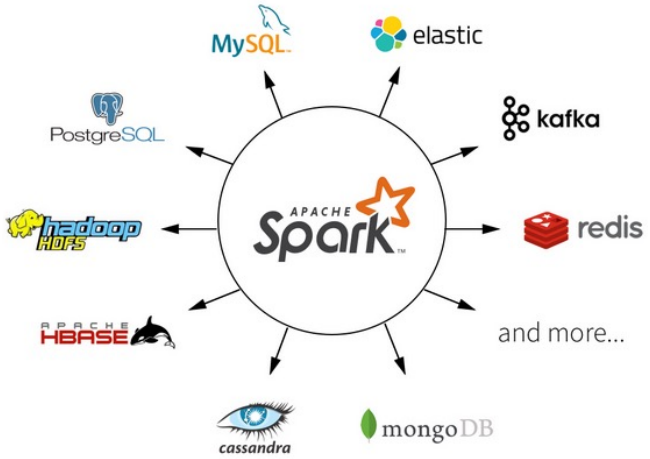


1,044,194,465,960
Learning Examples

*[ICMLA 2016]

Apache Spark supports distributed machine learning at scale

- Training ML on 76TB of HPC simulation data
  - — Custom solution
    - Write-your-own: parallel-distributed ML algorithm
    - One-off solution, good performance, R&D investment
  - — Off-the-shelf solution
    - Big Data framework: Apache Spark
    - General solution, good scalability, community supported



APACHE Spark™

Speed	Persist to memory and disk for fast computation
Simplicity	Rich set of APIs to exploit implicit data parallelism
Support	Multiple programming languages and database systems

Using Big Data frameworks in HPC systems is a major challenge!

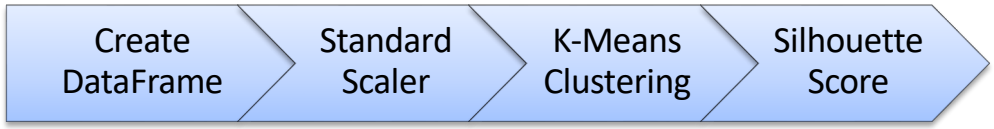
Big Data		HPC
• Abundant node-local storage • Hardware optimized for I/O • Yarn scheduler • Ethernet networking	**VS**	• Reliance on network storage • Hardware optimized for MPI • Slurm scheduler • Infiniband networking

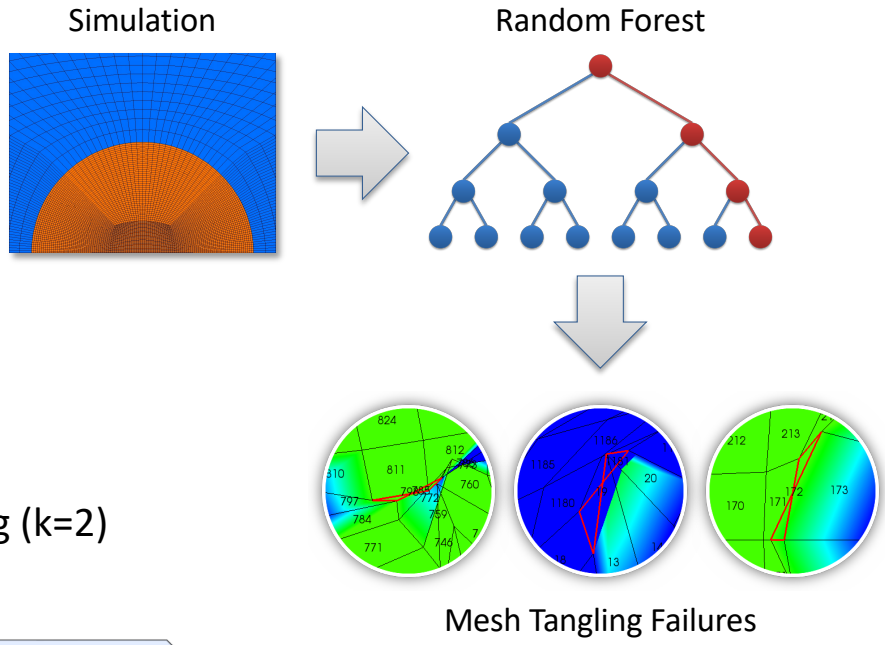
- Research in HPC+Spark
  - — Application integration, performance tuning, I/O frameworks, and scaling Spark
  - — [HPCAsia 2020]: we distilled a set of *best practices* for HPC users
    - **Key finding:** data scale matters! Our data is 2-3 orders of magnitude bigger than others!

Distributed machine learning for ALE+ML application

- ALE+ML: novel Spark application
  - **Random Forest**
    - Supervised learning for failure prediction
    - Calling a single, complex function

  - **Failure mode analysis**
    - Unsupervised learning using k-means clustering (k=2)
    - Composing a pipeline of lightweight functions





Our Initial Attempt at using Spark for ALE+ML

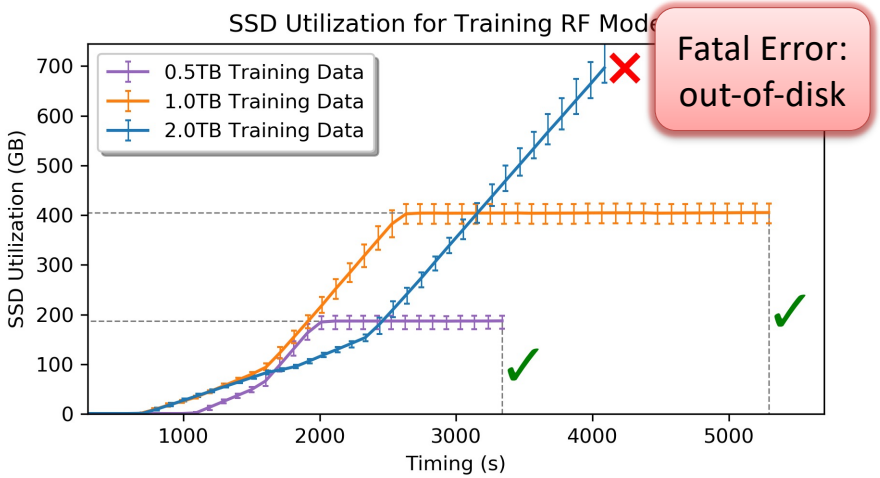
State-of-the-practice recommendations for HPC+Spark

- Our initial HPC+Spark experiment
  - We use **Magpie** to enable Spark on HPC
    - https://github.com/LLNL/magpie
  - Random Forest on 32 nodes for up to 2TB data
    - Experiments on LLNL Catalyst system
  - Original data resides on network storage (Lustre)
    - No HDFS or data replication
- **State-of-the-practice recommendations**
  - No compression
  - Persist to node-local storage (SSD)

LLNL Catalyst System

Per node: 128GB RAM + 745GB SSD

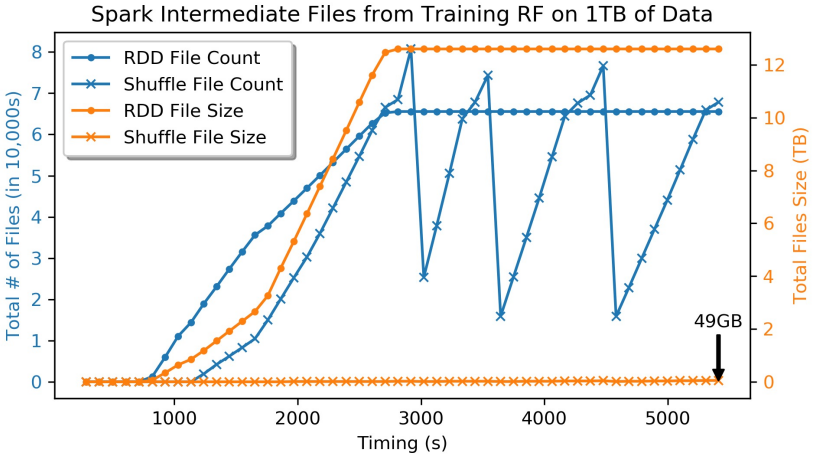
32 nodes: **27.3TB**



Data persistence in Spark can lead to *ephemeral data explosion*

- Resilient Distributed Dataset (RDD)
  - Fundamental data structure in Spark
  - Motivated by iterative (multi-pass) algorithms
  - Can be persisted (cached) to memory and/or disk

    - Save intermediate data for re-use
    - Speedup over Hadoop: 10x – 100x



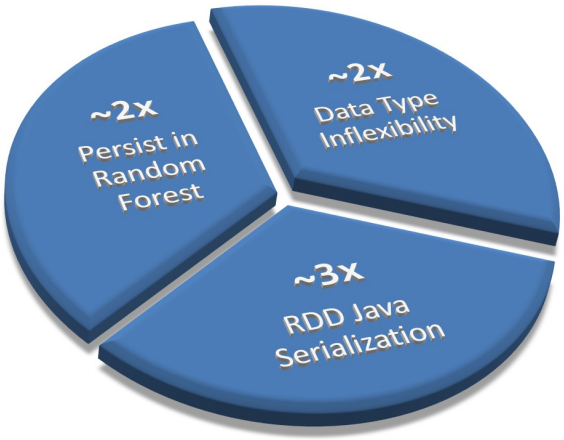
- Understanding RDD persistence
  - *"Blessing of persistence"*: eliminate redundant I/O
  - *"Curse of persistence"*: **ephemeral data explosion**
  - Fundamental tradeoff between space and time

1TB
Original Files

12.6TB
Intermediate Files

Where does the data explosion come from in Spark?

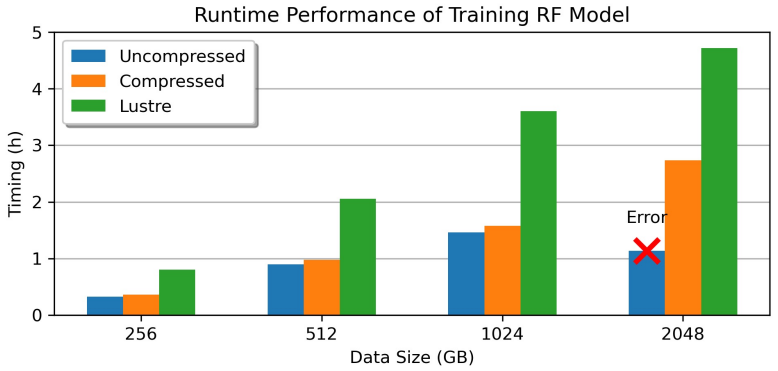
$$x\_factor = \frac{Spark\_disk\_utilization}{original\_data\_size} = \frac{12.6TB}{1TB} = 12.6x$$

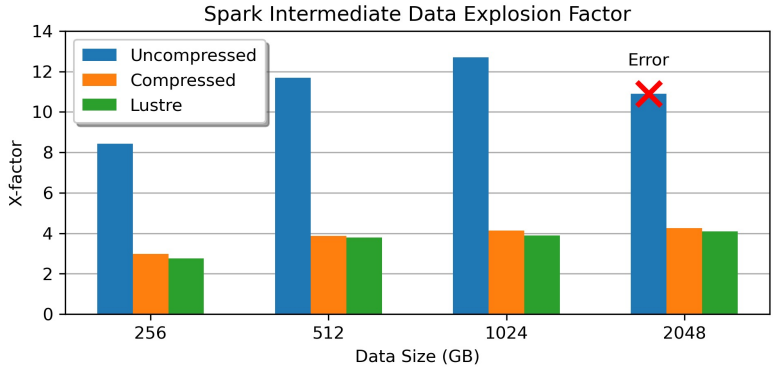


- **2x:** data type inflexibility
  - 4-byte floats in storage become 8-byte floats in Spark
- **~2x:** Random Forest implementation
  - Original and intermediate data are persisted to memory/disk
- **~3x:** Java object serialization overhead
  - Data is serialized when RDD files are written to disk
- *Caveat:* x-factor is both data and algorithm dependent

What can HPC users do about Spark data explosions?

- HPC environments are restrictive
  - Adding new hardware is not practical
  - Customizing Spark software is not practical
- What options are available to HPC users?
  - Reduce data by **compressing RDDs**
  - **Use network storage** to avoid out-of-disk errors
- Unfortunately, the state-of-the-practice recommends against using these options!
  - Spark documentation: https://spark.apache.org
  - Current literature in HPC+Spark



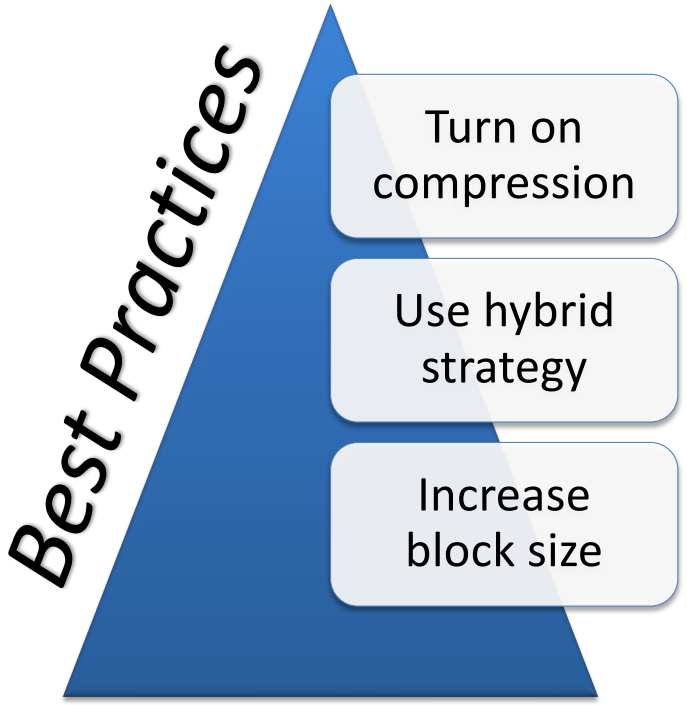


We present *best practices* to tame the ephemeral data explosion

- We distilled our results into a set of *best practices*
  - How to exploit Spark on massive HPC simulation data
  - **Novelty:** these *best practices* are in direct contrast to prevailing recommendations

State-of-the-practice ideal for data small enough to fit in memory or node-local storage

- We focus on user-configurable options in Spark
  - Rather than customizations to Spark itself
  - **Advantage:** general and applicable across different Spark versions/configurations



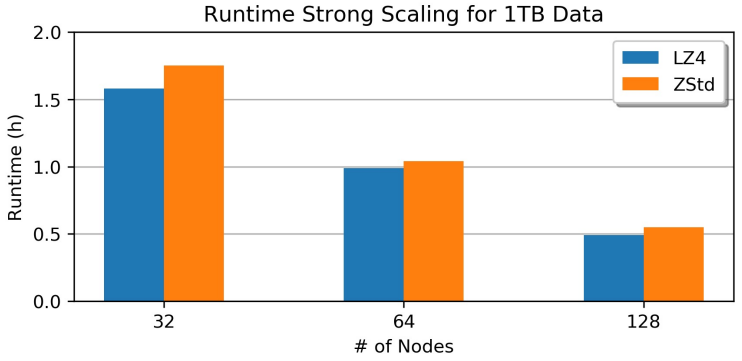
Our Evaluation Results of these *Best Practices*

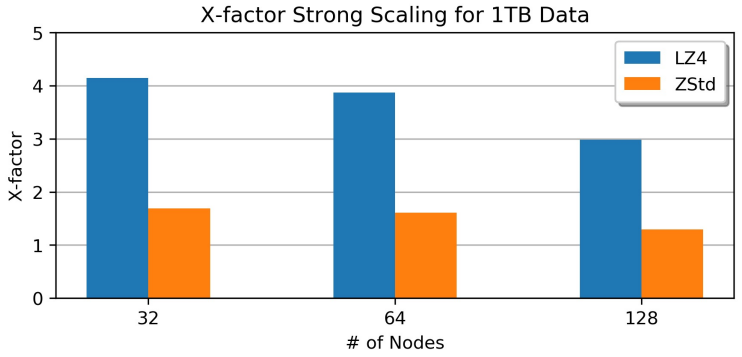
Config option #1: enable RDD compression with ZStd (level-1)

```
spark.rdd.compress = true
spark.io.compression.codec = zstd
```

- RDD compression is disabled by default
  - Prevailing view: noticeable runtime cost with little to no benefit
  - We enabled RDD compression and using different compressors (LZ4 vs ZStd)

	Default	LZ4	ZStd	% change
Runtime	1.5h	1.6h	1.8h	+20%
X-factor	12.6x	4.1x	1.7x	-87%

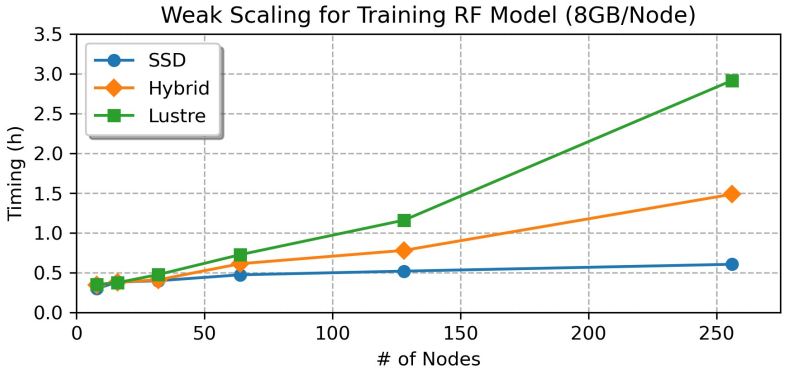




Config option #2: use hybrid strategy for Spark scratch space

```
spark.local.dir = /ssd/dir,/lustre/dir
```

- Spark scratch space for intermediate files
  - SSD: local and fast, but limited space
  - Lustre: shared and slow, but unlimited space
- We introduce **hybrid strategy**
  - Lesser-known feature: Spark accepts comma-separated list for scratch directories
  - We can combine SSD and Lustre in a round-robin fashion



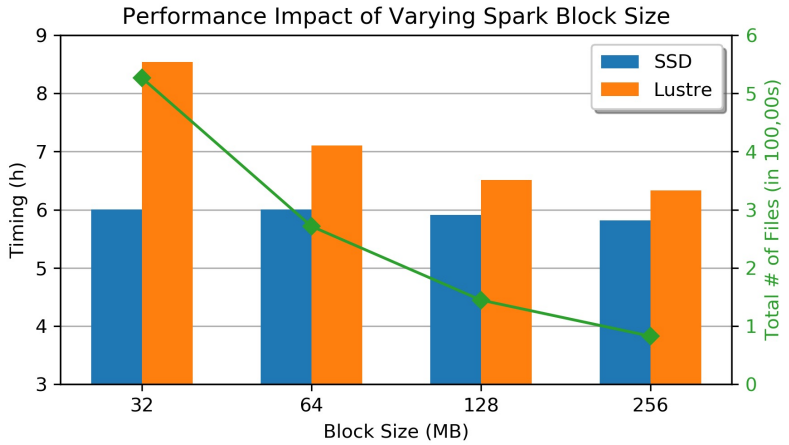
	SSD	SSD RDD-LZ4	Lustre	Lustre RDD-LZ4
Runtime	1.5h	1.6h	3.6h	2.2h
X-factor	12.6x	4.1x	3.9x	2.8x

* LLNL Lustre uses ZFS with transparent compression

Config option #3: increase block size for network storage persist

```
fs.local.block.size = 134217728
```

- Partitioning data into blocks has two effects
  - Number/size of Spark intermediate files
  - Number of Spark tasks
- File count/size dominates performance
  - Lustre I/O is a known bottleneck
  - Lustre performs better with fewer, larger files
- We confirmed that Spark controls Lustre block size through `fs.local.block.size`

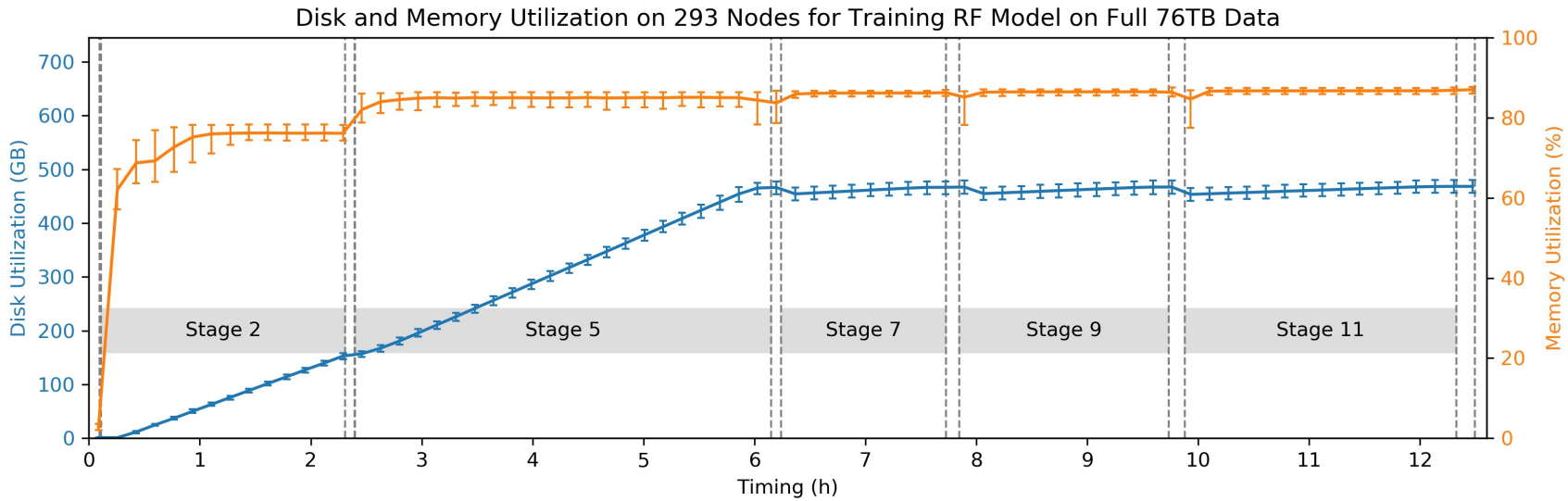


Lustre Performance

✓ Good for large file I/O

✗ Bad for many small file I/O

We successfully trained RF model on 76TB of data in 12.5 hours!

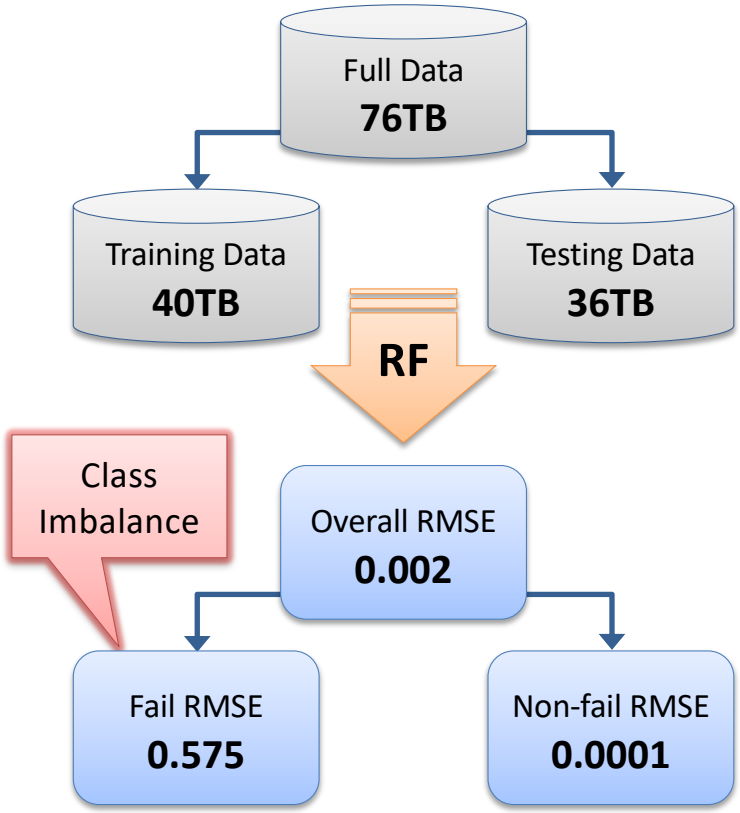


Memory fills as data is read from Lustre and persisted (stage 2)

Once memory fills up, SSD begins to fill as well (stage 5)

Random Forest regression with default depth=4 (stages 6-13)

Enable ML experiments with data sampling and class imbalance



Data Sampling Strategies

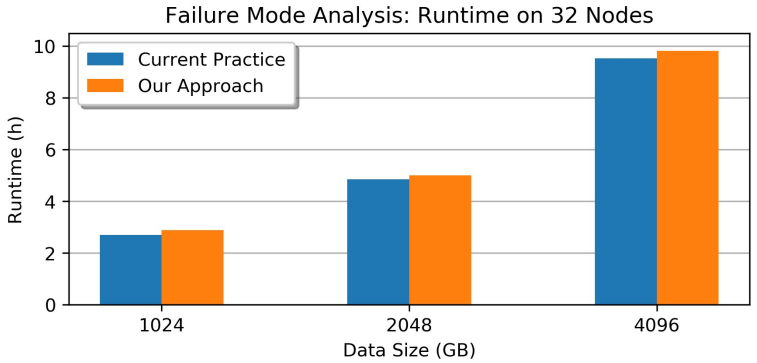
1. Sub-sample simulation variations
2. Sub-sample majority class (non-fails)

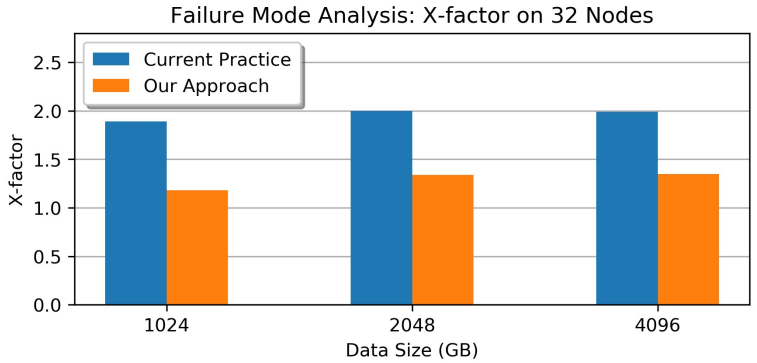
	Fail	Non-fail	Overall
1TB	0.266	0.0432	0.192
2TB	0.232	0.0368	0.166
4TB	0.213	0.0320	0.152
8TB	0.211	0.0294	0.152
16TB	0.209	0.0287	0.149
32TB	0.208	0.0288	0.149

Failure mode analysis using k-means clustering

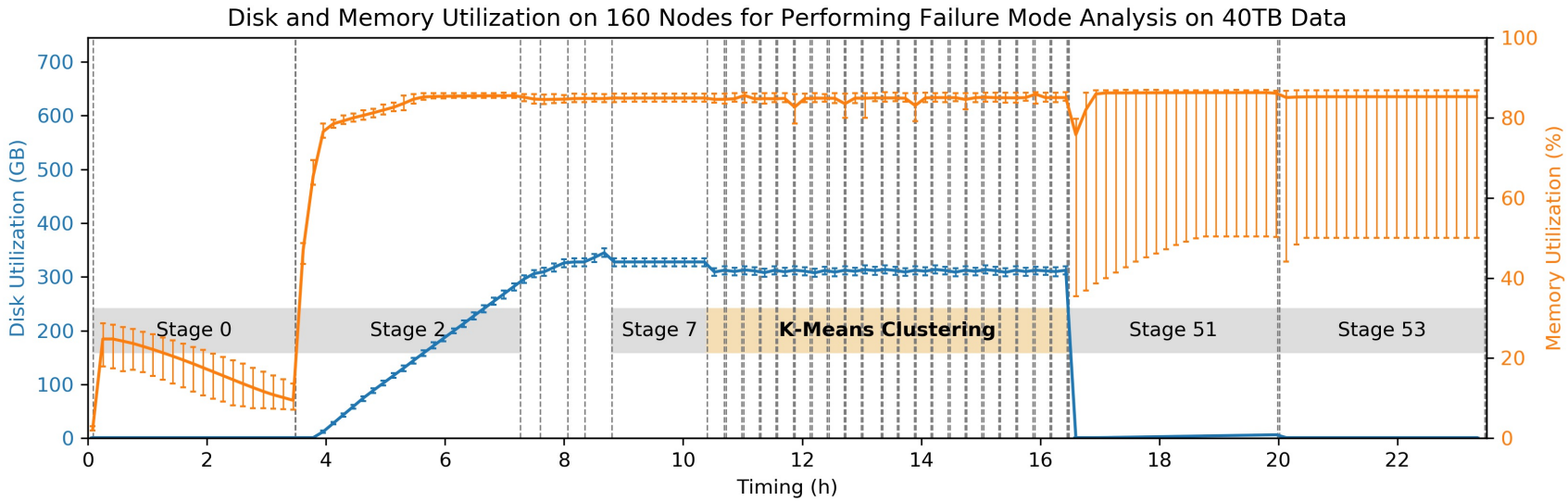
- Failure mode analysis pipeline
  - Did not include persists in-between function calls
  - Much smaller x-factor than Random Forest
- Comparing current practice with our approach
  - Negligible runtime difference
  - Reduce space requirements up to 30%

	10TB 40 nodes	20TB 80 nodes	40TB 160 nodes
Runtime	20.2h	21.7h	23.5h
X-factor	1.33x	1.34x	1.35x





We successfully performed failure analysis on 40TB of data



Memory fills as data is read from Lustre and persisted (stage 2)

Data standardization (stages 3-7)

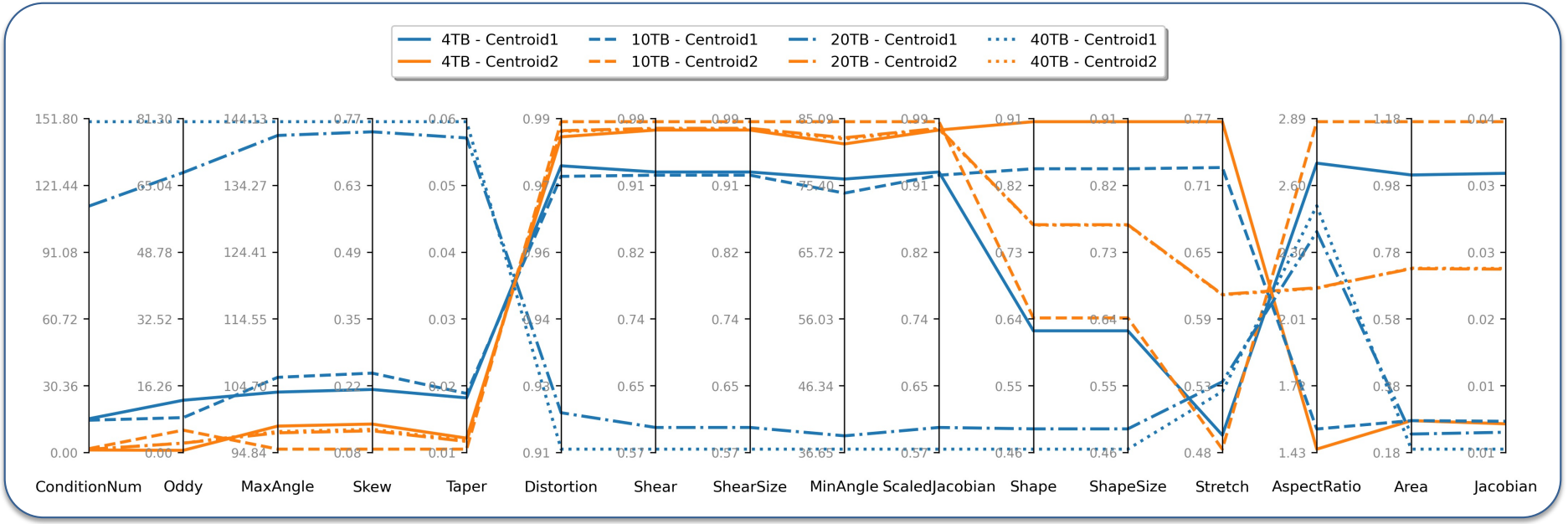
K-means clustering for 20 iterations (stages 8-49)

Cluster quality evaluation for Silhouette score (stages 50-53)

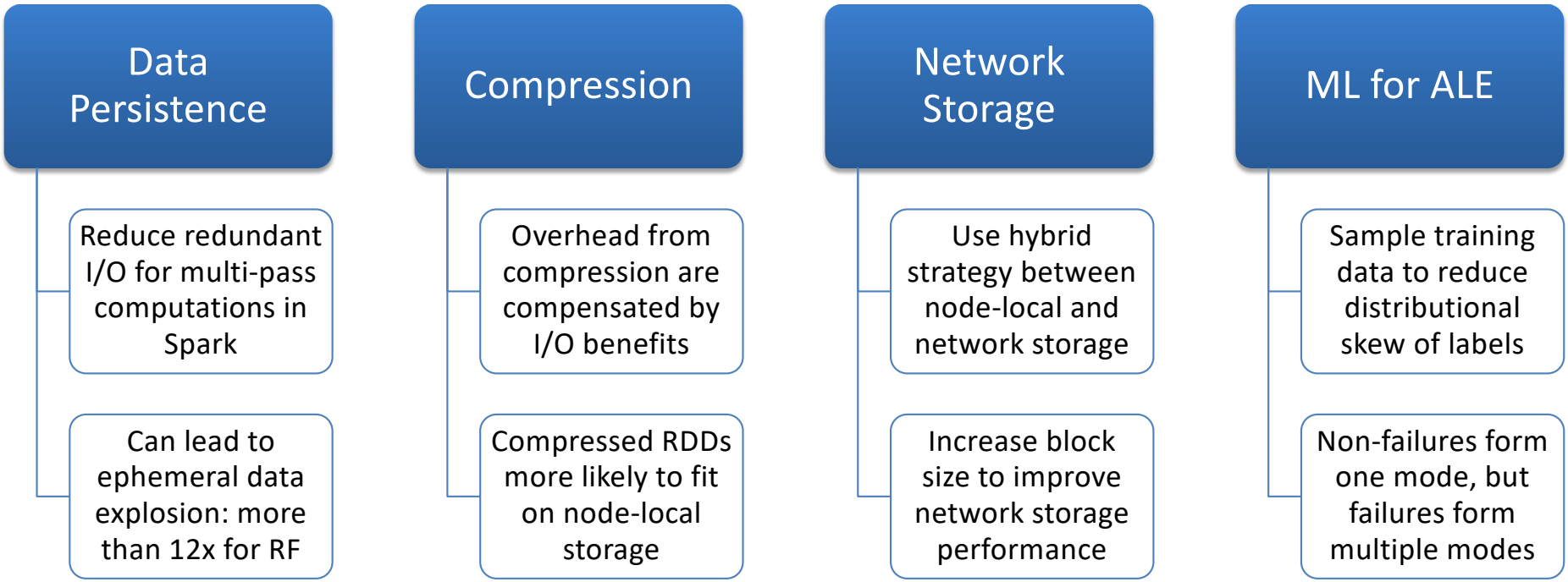
More data improves clustering with different failure modes

	4TB	10TB	20TB	40TB
Silhouette Score	0.183	0.239	0.837	0.852

Silhouette score, a measure of cluster consistency, increases with more data!

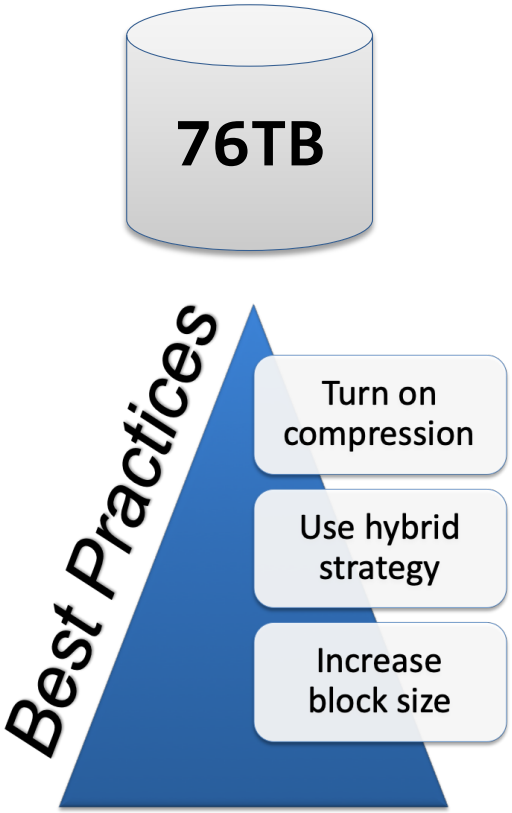


Discussion: lessons learned from running HPC+Spark at scale



Conclusion: Spark can be *tamed* for massive HPC simulation data

- ML-driven simulations are on the rise
  - ALE+ML application produced over a trillion learning examples
- Spark offers off-the-shelf solution for HPC simulation data
  - *Curse of persistence* can lead to ephemeral data explosion
  - State-of-the-practice recommendations are inadequate
- *Best practices* for exploiting Spark for HPC simulation data
  1. Enable RDD compression with ZStd (level-1)
  2. Utilize hybrid strategy for Spark scratch space
  3. Increase block size for network storage persist



[ICMLA 2016]

M. Jiang, B. Gallagher, J. Kallman, and D. Laney, “A Supervised Learning Framework for Arbitrary Lagrangian-Eulerian Simulations,” IEEE International Conference on Machine Learning and Applications (ICMLA), pp. 977–982, 2016.

[HPCAsia 2020]

M. Jiang, B. Gallagher, A. Chu, G. Abdulla, and T. Bender, “Exploiting Spark for HPC Simulation Data: Taming the Ephemeral Data Explosion,” *International Conference on High Performance Computing in Asia-Pacific Region (HPCAsia)*, pp. 150–160, 2020.

Lawrence Livermore National Laboratory