Los Alamos NATIONAL LABORATORY

ASC Data Science Infrastructure Project

Project Lead: James Ahrens

Project Team Members: Divya Banesh, Hugh Greenberg, Jesus Pulido, Christine Sweeney, Terry Turton, Quincy Wofford

May 2022

LA-UR-22-24474

Motivation for the talk

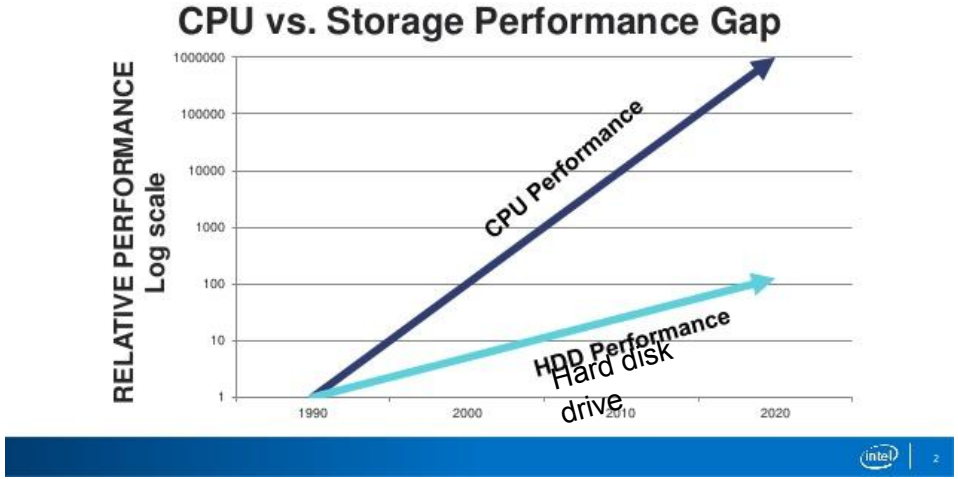
1. Inspired by request to reduce the time to answer questions
2. The observation that:
   - **storing <u>simulation and other data results in a large persistent database</u>** could help to answer questions quickly
3. The idea that:
   - Increasing FLOPS (compute-driven) driver (tera, peta, exa) for the ASC program may have culturally led us away from a data driven approach to answering questions

How does a compute driven approach support answering questions?

- Fundamentally the basis is we will run more simulations
  - And we count on specific knowledge from expert(s) and personally stored results

- To answer faster, in a compute-driven approach we would run the simulation faster…
- As we increase the number of processors --- FLOPS driver (tera, peta, exa)
  - **Strong scaling** - speedup for a fixed problem size
    - This is what we would focus on if we wanted to answer questions quickly
  - **Weak scaling** - speedup for a scaled problem size
    - We typically focus on improved spatial resolution and put our focus on largest run possible

Data storage in a compute-centric world



- Growing decadal gap between CPU performance and storage performance
  - Burst buffers help with user-observed write speed
- Limited percentage of the budget is allocated to storage
  - Some this budget now goes to burst buffer – overall relative reduced capacity

Data storage in a compute-centric world

Challenge	Best HPC practices are compute-centric not data-centric
Focus on files not data	POSIX file permission for access security
Lack of requirement & place for storing meta-data	Unix files do not have/require meta data
Lack of data model standards	Community does not agree on defined data models
Sharing is difficult because of lack of meta-data, data model, temporary file-based approach	
Permanent data store/database does not exist	Batch-oriented scheduling Data store is oriented towards handling massive sizes • 1000s of results will fill up the filesystem • storage is meant to be temporary and is called SCRATCH • users are encourage to move their data off of it to tape
Interactivity and querying is not available - reduces opportunity for explainability	

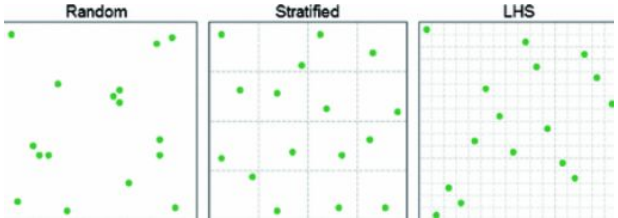
Promise of a data-driven approach

Technique	Purpose in terms of answering questions
1. Statistical experimental design	Coverage
2. Metadata	Organization, findability
3. Creating shared basis for comparison	Sharing, reuse, increased knowledge
4. Automation and improved workflow	Positive reduction in staff time, reproducibility
5. Persistent database	Sharing, reuse, interactive discovery
6. Machine learning	Fast interpolation and classification

This pattern of techniques is extremely interconnected. Productivity gains are non-linear.

1. Statistical experimental design & 2. Metadata

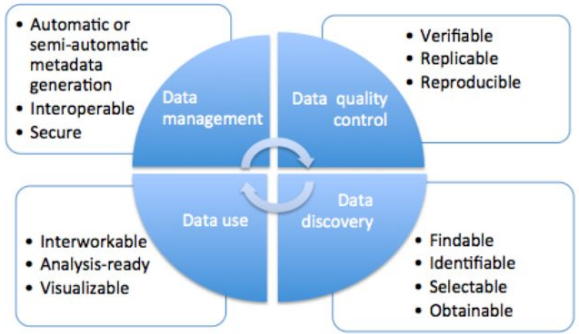
- Experimental Design economically maximizes information
  - In an “ensemble”, we deliberately change one or more input variables in order to observe the effect the changes have on one or more output variables.
  - The statistical design of experiments is an efficient procedure for planning “ensemble” so that the data obtained can be analyzed to yield valid and objective conclusions.
  - Well chosen experimental designs maximize the amount of "information" that can be obtained for a given amount of effort.



Type of Metadata

- Provenance: dates, versions, producers of ensembles, simulation, input and output data, databases
- Use: Statistics and features of data
- Structural: Types of the data
- Access: Description of restrictions on data
- …

Metadata services



3. Create a shared basis for comparison

- Standards for input decks and output results
  - Units of measure, ranges
  - Mapping of variables between different simulations
- The basis of comparison/analysis is likely feature based
- What features are important?
  - How do we calculate these features
  - Can we reduce data in a way that allows flexibility exploration later on?

4. Automation and workflow optimization

- Typically this done via scripting
- Needs to interface to HPC scheduling system
- This process is typically fragile and does not connect to broader data techniques such as meta data, experimental design etc.

5. Persistent database / data store

- Interactively, searchable with a query language, with meta data
- Shared, reusable, persistent
- Similar in size to existing data store

6.) Machine Learning/Statistical Emulation, Data and Simulations

Machine Learning/ Statistical Emulator Step	Output Data Size	Compute Time
Run simulations from carefully designed set of input	Large - Reduced data/derived variable from 100 simulations	Very Long - 100s of simulations
Train an emulator to approximate the simulator	Med - emulator	Long
Predict output with the emulator help assess prior uncertainty and parameter sensitivity.	Small – answer	Quick

Use case: analysis, multiscale simulation – quickly reproduce derived variable

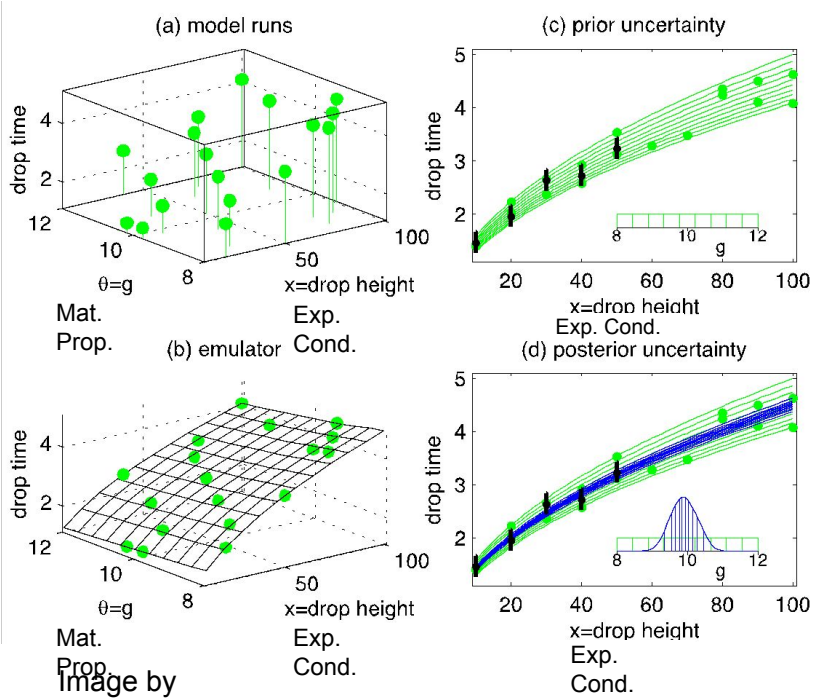
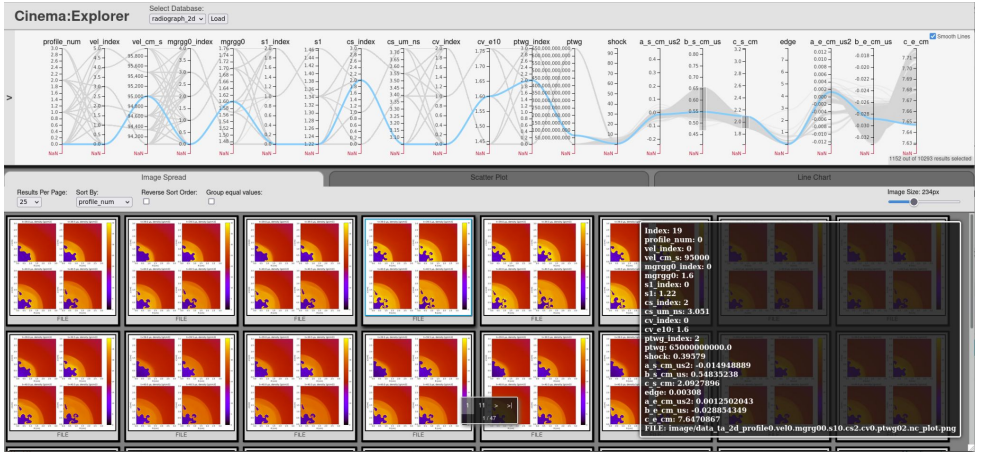


Image by Higdon

Data Science Infrastructure (DSI) Goals

DSI is developing searchable databases and workflows for simulation and experimental data derived from ASC clients.



Short term goal

- Make data easily accessible through metadata indexing and querying while respecting data permissions

Longer term goal

- Use this data for data science activities

Cross-cutting project (CSSE, FOUS, PEM, IC, V&V)

DSI Use Cases & Observed Challenges

Use Case	Challenges
CMF Users	• Diverse user base • No mechanism to track previous runs to prevent waste of scientist time & compute resources
ML Users	• Large ensembles with manual sorting into 100s of directories • Data sharing challenges • Metadata doesn't always travel with relevant files • No mechanism to query metadata • ML workflow itself provides metadata – training & testing sets
HPC Users	• Performance & Test data • Data products link to performance & test data • Manual processes in place to capture performance & test data

DSI Solutions

Ingest, Querying, Access, Automation

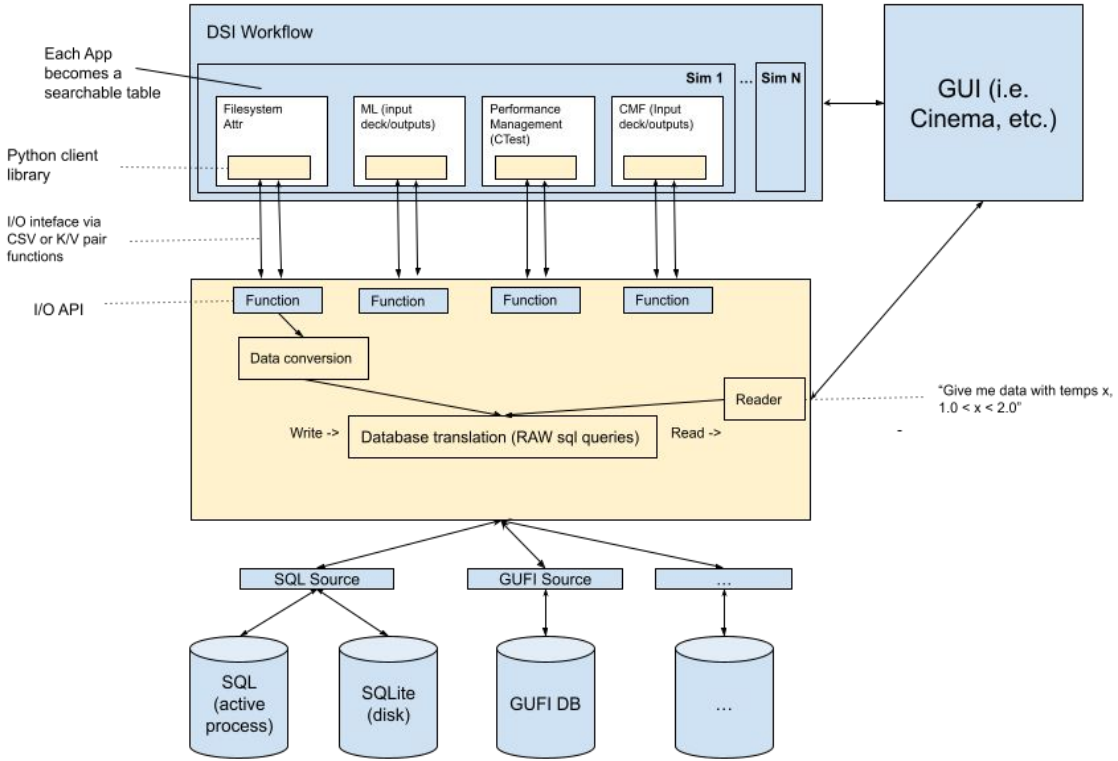
Approach: Be pragmatic as possible including being data format & system agonistic

- Front-end/User-facing API
  - Ingest – “Readers” / Parsers
    - Gather metadata, copy of data and put in DSI system
  - Querying and data access API
    - GUI or Command Line
    - These provides a simplified interface to a relational representation
      - Spreadsheet for your metadata and data
      - Columns are meta data and data (labels and types)
      - Rows are entries
    - Queries supported simplified query interface or native SQL
    - Access to metadata, data via Back-end APIs
- Back-end/Storage-facing API
  - Abstract details such as where the data is located /scratch, project space, home space, tape archive
  - Backends
    - 1. Grand Unified File Index
    - 2. SQLite (relational database in a unix file) plus long term campaign
    - 3. Cloud/data oriented “Hadoop-style” store
- Services API
  - Automated runs to populate metadata and data (CMF, ML, HPC), containers to simplify this process
- Our current backend focus is #2
  - Use SQLite – POSIX semantics address security
  - Metadata and pointers to data stored in a SQL schema as a relation
  - Store copied data as pointers to protected simulation files on long term campaign storage

General DSI Workflow storing searchable metadata

API Write Example

```
# Define path of the database
dbpath = "cmf.db"
# Initialize database
store = fs.store(dbpath)
data_type = fs.data_type()
# Declare the table name
data_type.name = "filesystem"
# Define the schema
data_type.properties["file"] = fs.STRING
data_type.properties["st_size"] = fs.DOUBLE
data_type.properties["st_atime"] = fs.DOUBLE
data_type.properties["st_mtime"] = fs.DOUBLE
data_type.properties["st_ctime"] = fs.DOUBLE
store.put_artifact_type(data_type)
artifact = fs.artifact()
# Add the data into the database, iterating
through all rows
for file,st in zip(file_list,st_list):
   artifact.properties["file"] = str(file)
   artifact.properties["st_size"] = st.st_size
   artifact.properties["st_atime"] = st.st_atime
   artifact.properties["st_mtime"] = st.st_mtime
   artifact.properties["st_ctime"] = st.st_ctime
   store.put_artifacts(artifact)
```



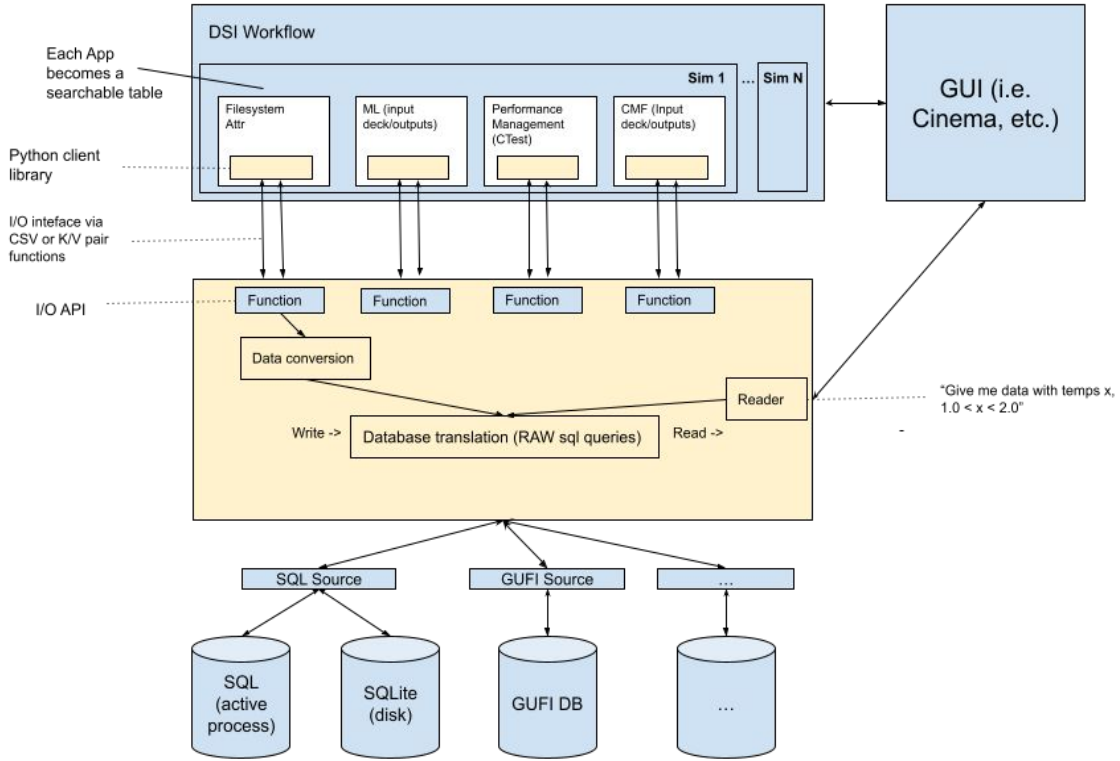
General DSI Workflow for storing and searching metadata

General DSI Workflow storing searchable metadata

API Read Performance

Filesystem Benchmark:
HPC Snow - Yellow
ML Density Radiograph Data
shell_implosion_1d (160k files)

Method	Real Time
scratch4 (lustre)	1.393 s
SQLite	**0.360 s**
project space (nfs)	37.142 s



General DSI Workflow for storing and searching metadata

DSI Use Case 1: CMF Users High dimensional Vis/Querying

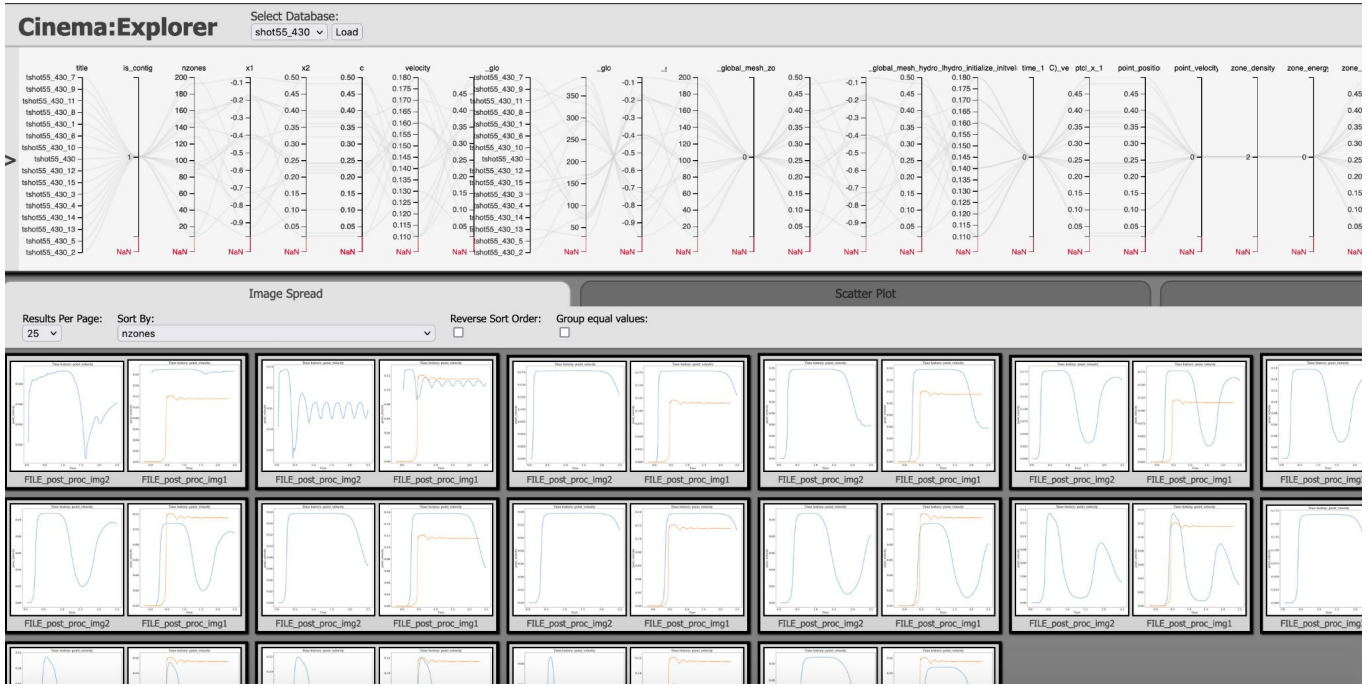
Ensemble of flyer plate experiments:

- Be flyer on Be disc
- LAPA authority, shot_55_430
- VISAR detector to record the impact shock

Vary:

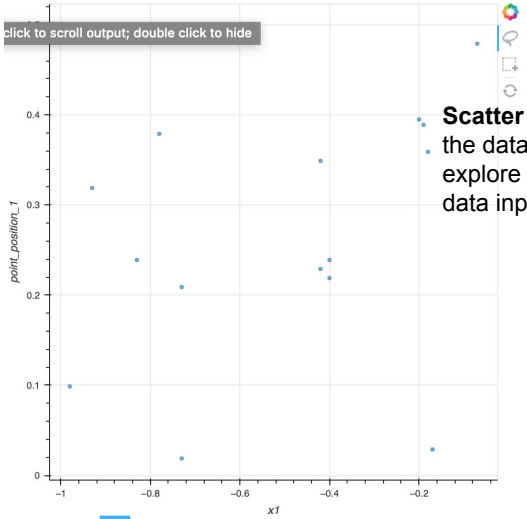
- Flyer thickness
- Flyer radius
- Init velocity of flyer
- Target thickness
- Target radius

Cinema Explorer visualization of 16 shot_55_430 runs: Select properties/metadata are displayed across the parallel coordinate axes: recorded from input decks, tensor output files, and output images recording the visar shock impact.

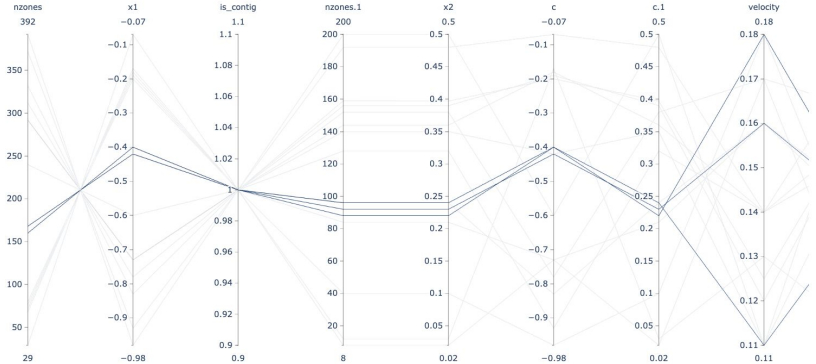


DSI Use Case 1: CMF Users Interactive Visual Query

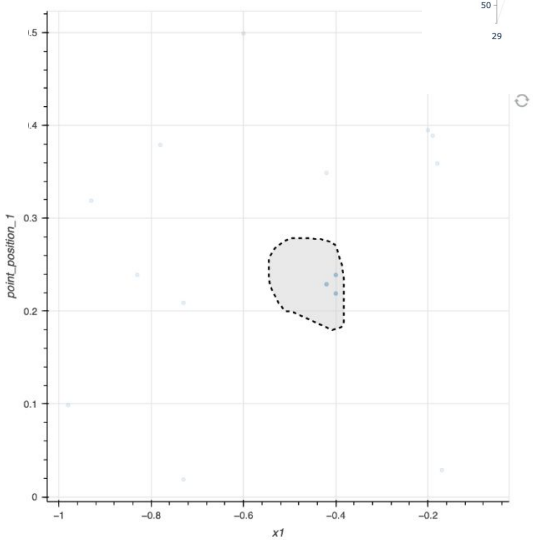
Jupyter notebook workflow leveraging Bokeh for data exploration and visualization



Scatter plot: parameters from the database can be selected to explore the relationship between data inputs and outputs.



Point selection using the 'lasso' tool. Users can select points of interest on the scatter plot for an in-depth analysis.



Parallel Coordinates Plot Viewer. Selected entries from the scatter plot are highlighted in a parallel coordinates plot viewer in Jupyter notebook.

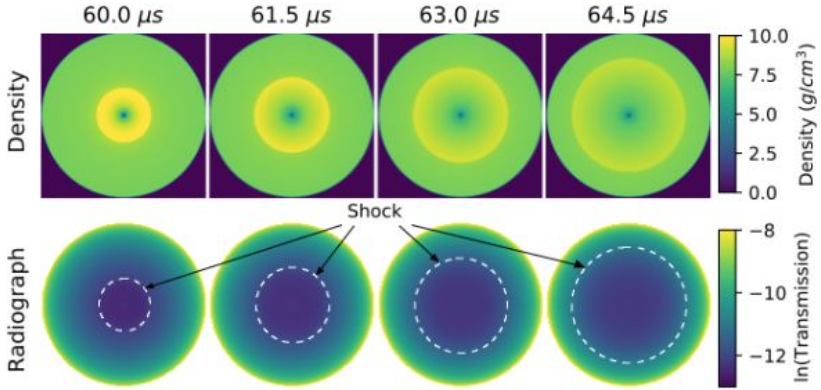
DSI Use Case 2: Machine Learning Workflows Predicting Density Fields from Simulated Radiographs

Contacts: Marc Klasky and Oleg Korobkin

ML-based technique to reconstruct density fields in dynamic systems based on a temporal sequence of simulated radiographic images [1,2] similar to what are obtained experimentally at DARHT.

Data: Ensembles of hydrodynamics simulations (via CTH)

- 160K and 10K simulation result files for 1D and 2D simulations respectively
- Configuration file and one or more plot files per simulation
- Feature file collecting analysis results over the entire ensemble



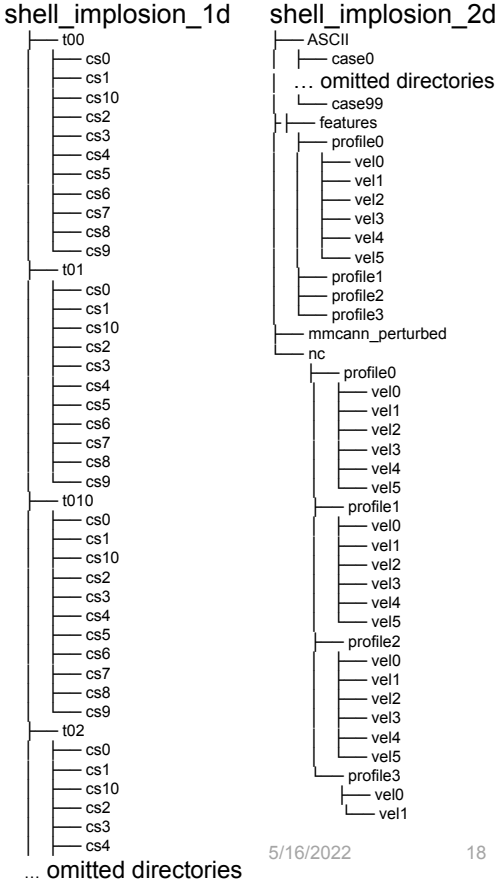
[1] Zhishen Huang, Marc Klasky, Trevor Wilcox, and Saiprasad Ravishankar (2021) "Physics-driven learning of Wasserstein GAN for density reconstruction in dynamic tomography" https://opg.optica.org/ao/fulltext.cfm?uri=ao-61-10-2805&id=470826 .

[2] M. Hossain, B. Nadiga et al. (2022), "High-precision inversion of dynamic radiography using hydrodynamic features" Optics Express, Vol. 30, Issue 9, pp. 14432-14452. https://opg.optica.org/oe/fulltext.cfm?uri=oe-30-9-14432&id=471435

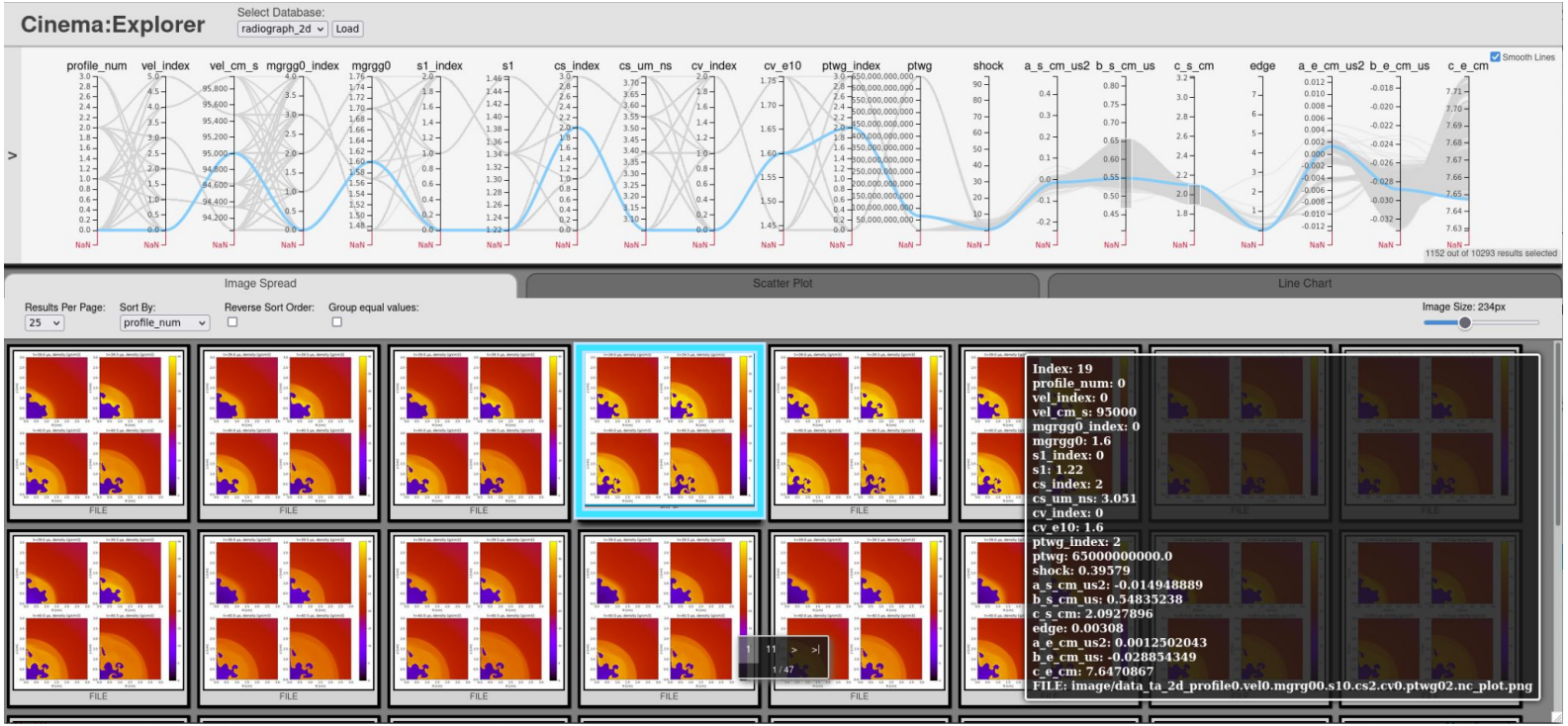
Image at left from [2]

Predicting Density Fields from Simulated Radiographs

- Interviewed ASC ML workflow experts. Chose radiography workflow.
- Built workflow to ingest (custom) metadata from CTH 1D and 2D radiograph simulations into SQLite database, via file crawling and file parsing.
- Created plots for each simulation to enable better exploration.
- Built tool configuration file from SQLite metadata query to enable scientists to do visual selection for training dataset (next slide)
- Queried SQLite database to return file for training dataset. (next next slide)



Predicting Density Fields from Simulated Radiographs



Gray curved lines through parallel coordinates plot at top show the metadata values for 2D radiograph simulations that match the visual query. Each four-plot image is for four 2D radiograph simulation time points. The blue curved line in the parallel coordinates plot at top shows the parameters and results for the selected image (with blue box around it, 4th from the left on the top row in the bottom panel). The black text box at lower right shows the exact metadata values for the selected image.

Predicting Density Fields from Simulated Radiographs

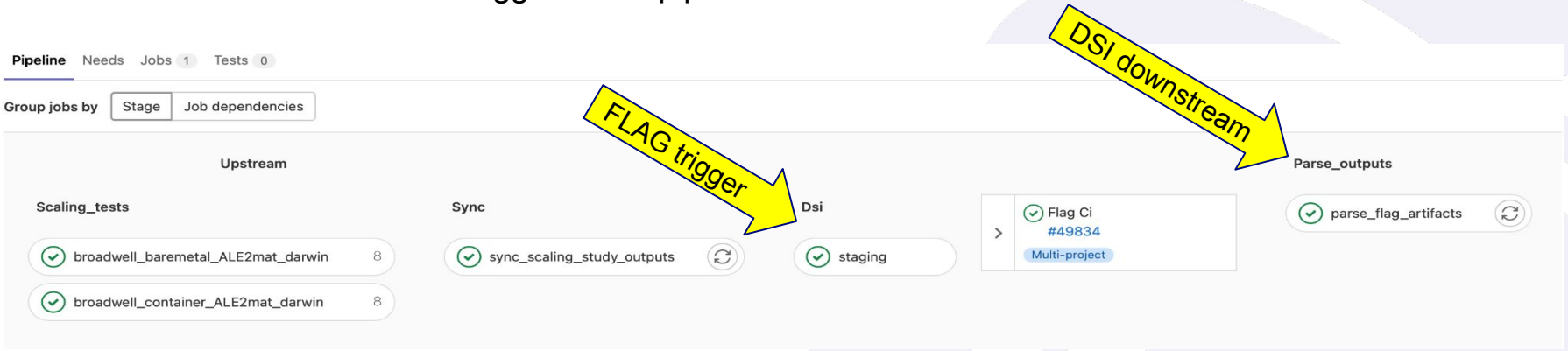
Text-based SQL query of 2D Radiograph Metadata

```
Number rows: 10293

SELECT sim_file_path from radiograph_sims WHERE b_s_cm_us > 4.7283884e-01 AND b_s_cm_us < 6.7283884e-01 AND c_s_cm > 1.8797820
e+00 AND c_s_cm < 2.0797820e+00
Number rows: 1156

('/yellow/usr/projects/dsi_re/data/shell_implosion_2d/nc/profile0/vel0/data_ta_2d_profile0.vel0.mgrg00.s10.cs0.cv0.ptwg00.nc',
)
('/yellow/usr/projects/dsi_re/data/shell_implosion_2d/nc/profile0/vel0/data_ta_2d_profile0.vel0.mgrg00.s10.cs0.cv1.ptwg00.nc',
)
('/yellow/usr/projects/dsi_re/data/shell_implosion_2d/nc/profile0/vel0/data_ta_2d_profile0.vel0.mgrg00.s10.cs0.cv2.ptwg00.nc',
)
('/yellow/usr/projects/dsi_re/data/shell_implosion_2d/nc/profile0/vel0/data_ta_2d_profile0.vel0.mgrg00.s10.cs3.cv0.ptwg00.nc',
)
('/yellow/usr/projects/dsi_re/data/shell_implosion_2d/nc/profile0/vel0/data_ta_2d_profile0.vel0.mgrg00.s10.cs3.cv1.ptwg00.nc',
)
('/yellow/usr/projects/dsi_re/data/shell_implosion_2d/nc/profile0/vel0/data_ta_2d_profile0.vel0.mgrg00.s10.cs3.cv2.ptwg00.nc',
)
('/yellow/usr/projects/dsi_re/data/shell_implosion_2d/nc/profile0/vel0/data_ta_2d_profile0.vel0.mgrg00.s11.cs2.cv0.ptwg00.nc',
)
('/yellow/usr/projects/dsi_re/data/shell_implosion_2d/nc/profile0/vel0/data_ta_2d_profile0.vel0.mgrg00.s11.cs2.cv1.ptwg00.nc',
--More--
```

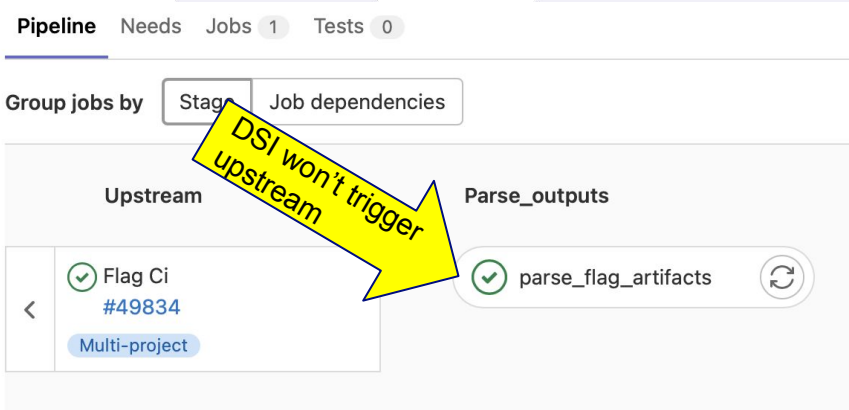
Text-based SQL query for file names of the simulations that match the previous visual query.

DSI Use Case 3: LAP FLAG Performance & Test data

Data Collection – automate to trigger a DSI pipeline to insert data into DSI store



Flexibility built-in: If the DSI parser needs to change to add new metadata to the query-able collection, DSI can trigger the pipeline downstream without involving upstreams. Changes can propagate backwards through time if requested, to make a new query possible on archived data.



DSI Use Case 3: LAP FLAG Performance & Test data

Data Analysis UI & Workflow:

- SQLite database supports raw CLI SQL queries
- Leverage Pandas in Jupyter Notebook workflow

```
def show_scatter_comparison(inputfile_name,df=df,ax=ax):
    df['ranks']=df['node_cnt']*df['ranks_per_node']
    df_container = df[df['runtime_type']=='container']
    df_baremetal = df[df['runtime_type']=='baremetal']
    df_container[['date','ranks','total_problem_run_time']].plot(ax=ax,
                                                                 x='ranks',
                                                                 y='total_problem_run_time',
                                                                 kind='scatter',
                                                                 label='container ({})'.format(inputfile_name),
                                                                 color='blue')
    df_baremetal[['date','ranks','total_problem_run_time']].plot(ax=ax,
                                                                 x='ranks',
                                                                 y='total_problem_run_time',
                                                                 kind='scatter',
                                                                 label='baremetal ({})'.format(inputfile_name),
                                                                 color='orange')
fig = plt.figure(figsize=(25,10))
subplot_base=241
for idx,inputfile_name in enumerate(df['inputfile_name'].unique()):
    ax = fig.add_subplot(subplot_base+idx)
    show_scatter_comparison(inputfile_name,ax=ax,df=df[df['inputfile_name']==inputfile_name].copy())
plt.suptitle('\nFLAG strong scaling for selected problems, container vs baremetal, Darwin CTS-1',fontsize=30)
```

Data Collection UI & Workflow:

- Crawler scripts to parse FLAG outputs
- Transform into SQL query-able format

```
echo $(cat $0 | grep \# | sed 's/\# //g' | grep -v \$0) | sed 's/ /,/g'
for i in "$@"
do
  # date
  printf $(date +"%m-%d-%y"),
  # runtime_type
  printf $(echo $i | awk -F_ '{print $2}' | awk -F. '{print $1}'),
  # node_cnt
  printf $(echo $i | awk -F_ '{print $1}' | sed 's/N//'),
  # ranks_per_node
  printf $(cat $i | grep 'NUMBER OF PROCESSORS' | awk -F: '{print $2}'),
  # performance_dimension
  printf $(cat $i | grep 'Problem dimension' | awk -F: '{print $2}'),
  # processors_per_dimension
  printf $(cat $i | grep 'Processors per dimension' | awk '{print $5"/"$6"/"$7}'),
  # global_zones_per_dimension
  printf $(cat $i | grep 'Global zones per dimension' | awk '{print $6"/"$7"/"$8}'),
  # geometry
  printf $(cat $i | grep 'Geometry' | awk -F: '{print $2}' | sed 's/ //g'),
  # inputfile_md5
  printf $(cat $i | grep 'input file' | awk -F: '{print $2}'),
  # number_of_zones
  printf $(cat $i | grep 'number of zones' | head -1 | awk -F: '{print $2}'),
  # number_of_cycles
  printf $(cat $i | grep -E 'All.*cycles' | tail -1 | awk '{print $2}'),
  # total_problem_run_time
  printf $(cat $i | grep 'total problem run time' | awk -F= '{print $2}')
  echo
done
```

DSI Use Case 3: LAP FLAG Performance & Test data

User: Joe Zerr – LAP CS lead

Typical User Queries:

When was the last time simulation X ran correctly? [testing]

When was the last time X ran well? [performance]

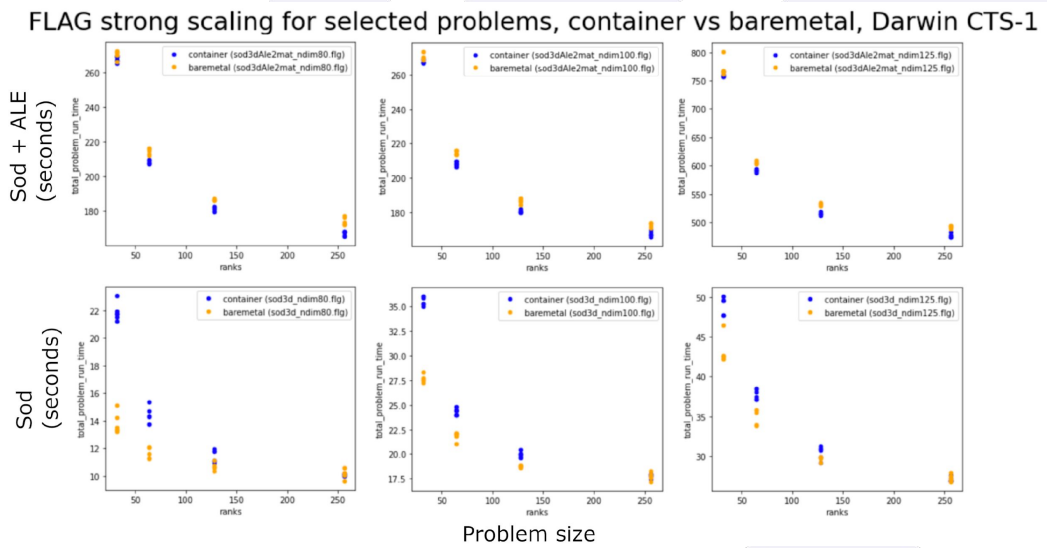
When was the last time X ran correctly and well? [testing, performance]

Requires: capture and archive performance/test data in a format compatible with both the DSI back-end and the queries users wish to run.

Q: *Do containers pose overhead for FLAG in distributed parallel?*

A: It depends

Leverages existing FLAG performance CI with workflow triggers, but CI not required.



DSI Interfacing to other systems

- Leverage from industry? Existing systems evaluated
  - MLFlow: https://github.com/mlflow/mlflow/
  - ModelDB: https://github.com/VertaAI/modeldb

Project	Servers Required	UI	Datastores	Client/Library Language	OSS
MLFlow	Yes	Yes	Multiple SQL	Python	Yes
ModelDB	Yes	Yes	Multiple SQL	Python	Yes
MLMD	No	No	Multiple SQL	Python	Yes

MLMD

- Closest to requirements as it doesn't require servers
- Straightforward API – types are first defined by the user and then instances
- Abstracts queries to python and advanced queries are difficult
- SQL tables abstracted into types and instances
  - Raw queries are hard to do unless you know the internals

Interfaces to other LANL efforts

- Experiments – MarkLogic (OODB), Granta Database (Engineering)
- Library – Exploro

EN
D