Near real-time streaming analysis of big fusion data

R. Kube[1], R.M. Churchill[1]. J.Y. Choi[2], J. Wang[2], L. Stephey[3], E. Dart[4] , M. Choi[5], J. Park [5], CS Chang[1], and S. Klasky[2]

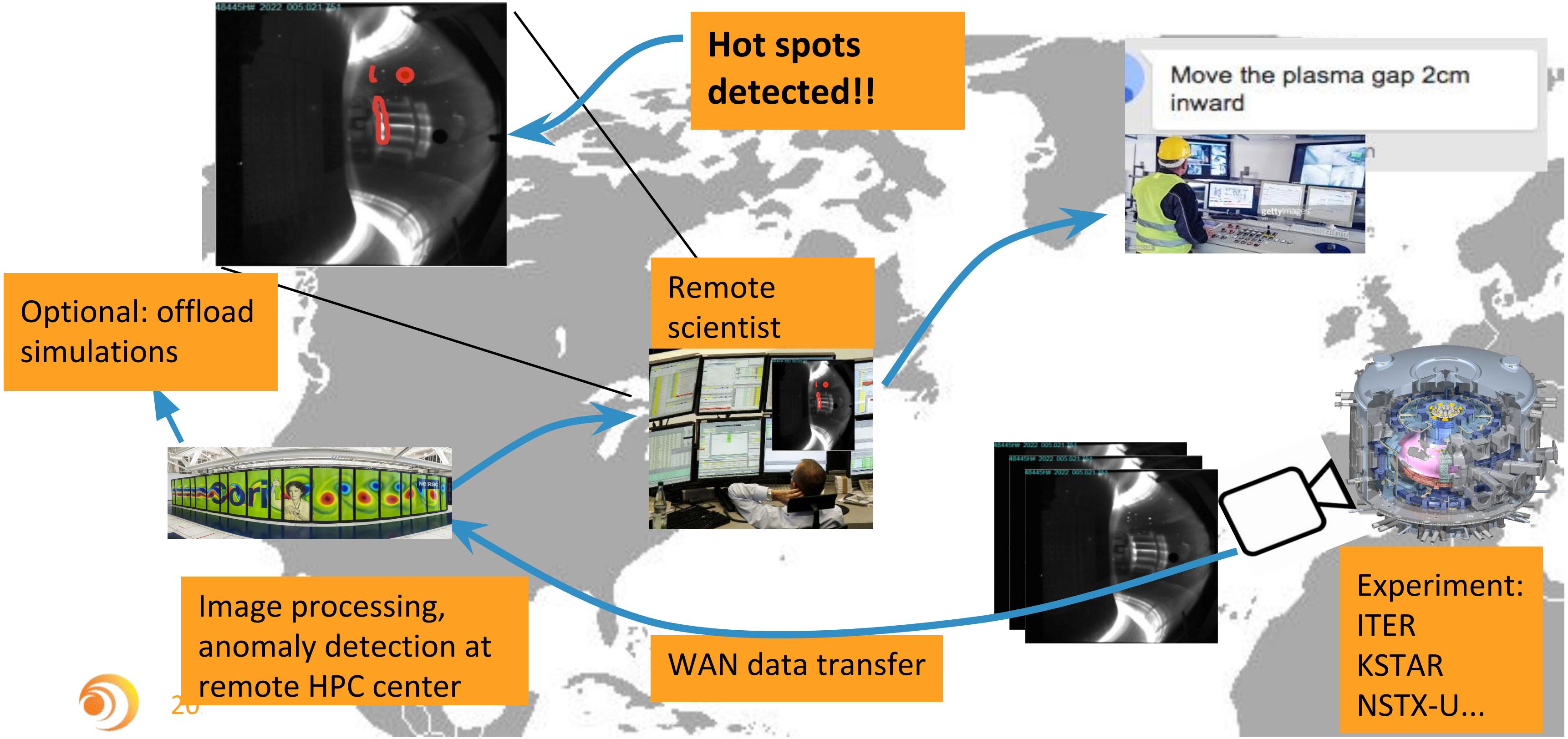
[1]Princeton Plasma Physics Laboratory [2]Oak Ridge National Laboratory

[3] NERSC [4] ESNet [5] Korean Institute of Fusion Energy

PRINCETON PLASMA PHYSICS LABORATORY

PRINCETON UNIVERSITY

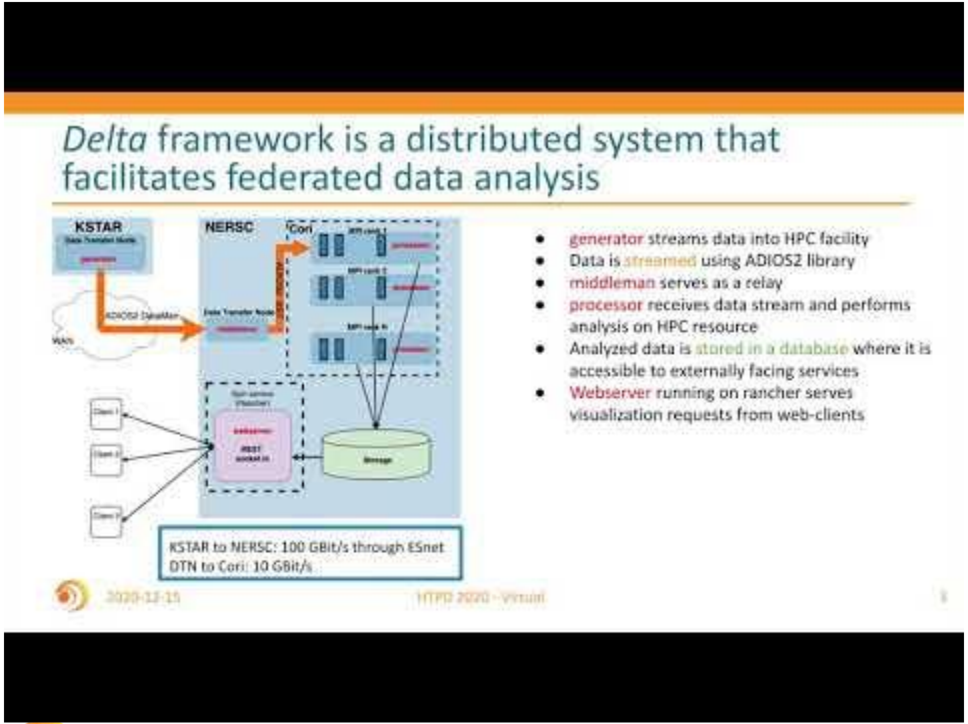
Facilitate remote near real-time streaming analysis of big fusion data



Why?

- Long-pulse fusion devices will generate bigger data sets
  - ITER: Expected Petabytes / day
  - Increasing spatial / temporal resolution from diagnostics
  - Overnight analysis -> In-between shot analysis -> Intra-shot analysis
- AI/ML algorithms require big data sets
  - Systematically collect analyzed fusion data
- Compute environments are changing
  - Increase in FLOPS is driven by accelerators (GPU/TPU/ASIC)
- Fusion relies on international collaborations:
  - Watch data analysis results in real time from remote, just like being in the control room

Demo: remote analysis of KSTAR ECEI data at NERSC



Screen layout:

Network bandwidth	Network bandwidth	CPU utilization
KSTAR Data generator	**NERSC** relay	**Cori**(NERSC) Data analysis

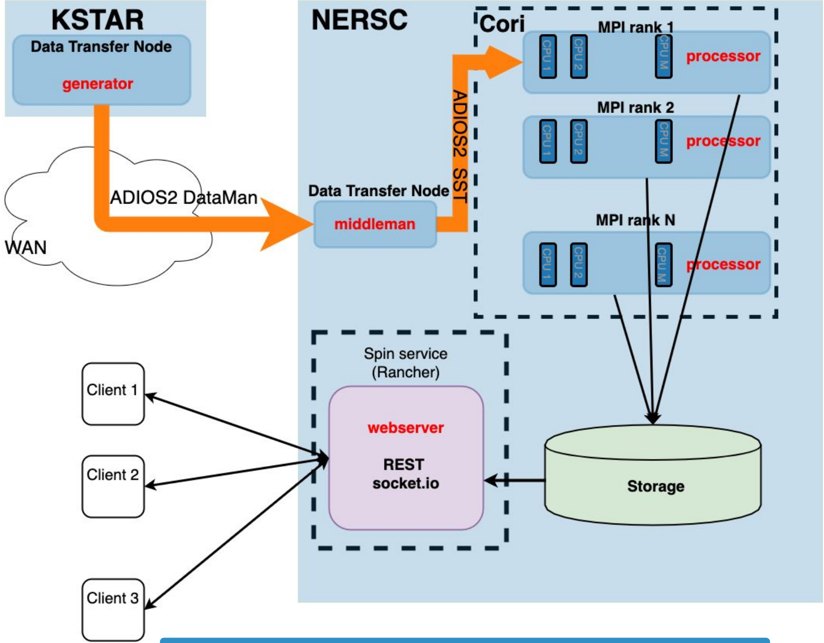
Outline

1. Design and implementation details of the *Delta* framework
2. Benchmark results
3. Web-based live visualization
4. Conclusions and future work

Outline

1. Design and implementation details of the *Delta* framework
2. Benchmark results
3. Web-based live visualization
4. Conclusions and future work

Delta framework is a distributed system that facilitates streaming data analysis



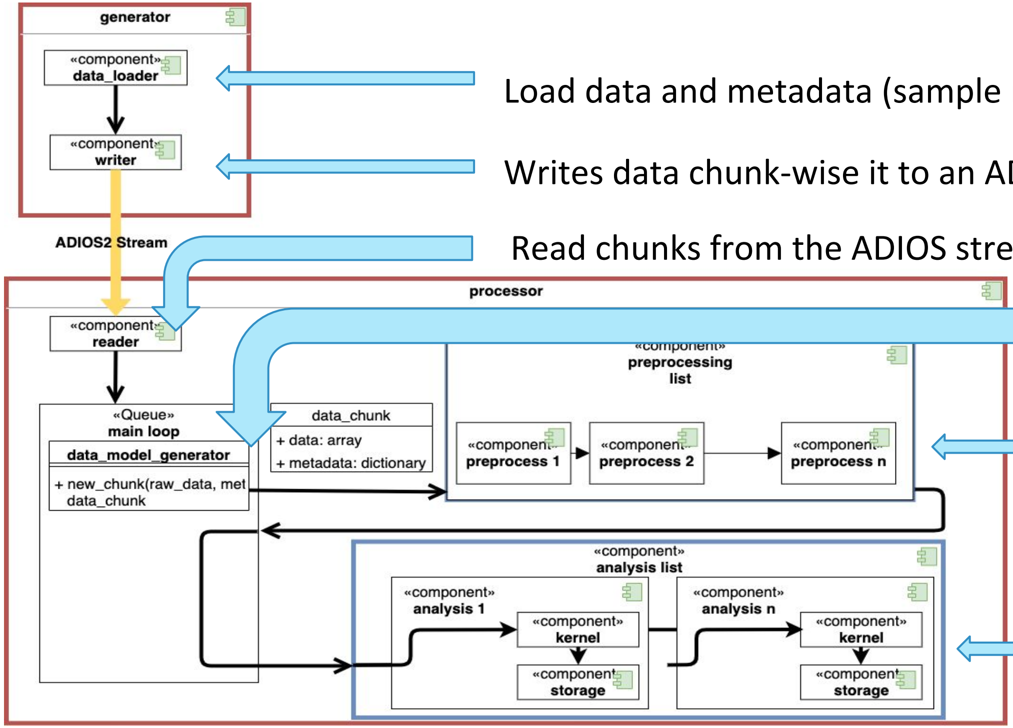
- generator streams data into HPC facility
- Data is streamed using ADIOS2 library
- middleman serves as a relay
- processor receives data stream and performs analysis on HPC resource
- Analyzed data is stored in a database where it is accessible from externally facing services
- Webserver running on Spin serves visualization requests from web-clients

KSTAR to NERSC: 100 GBit/s through ESnet
DTN to Cori: 10 GBit/s

ADIOS2: High-performance parallel I/O library for HPC environments: https://github.com/ornladios/ADIOS2

- Pub-sub streaming: Allows multiple processors listening to a generator.
- File-based I/O: custom binary-pack format (similar to HDF5)

Data flow through the *Delta* framework



Load data and metadata (sample rate, diagnostic setup etc) into memory

Writes data chunk-wise it to an ADIOS stream

Read chunks from the ADIOS stream and put them in a Queue

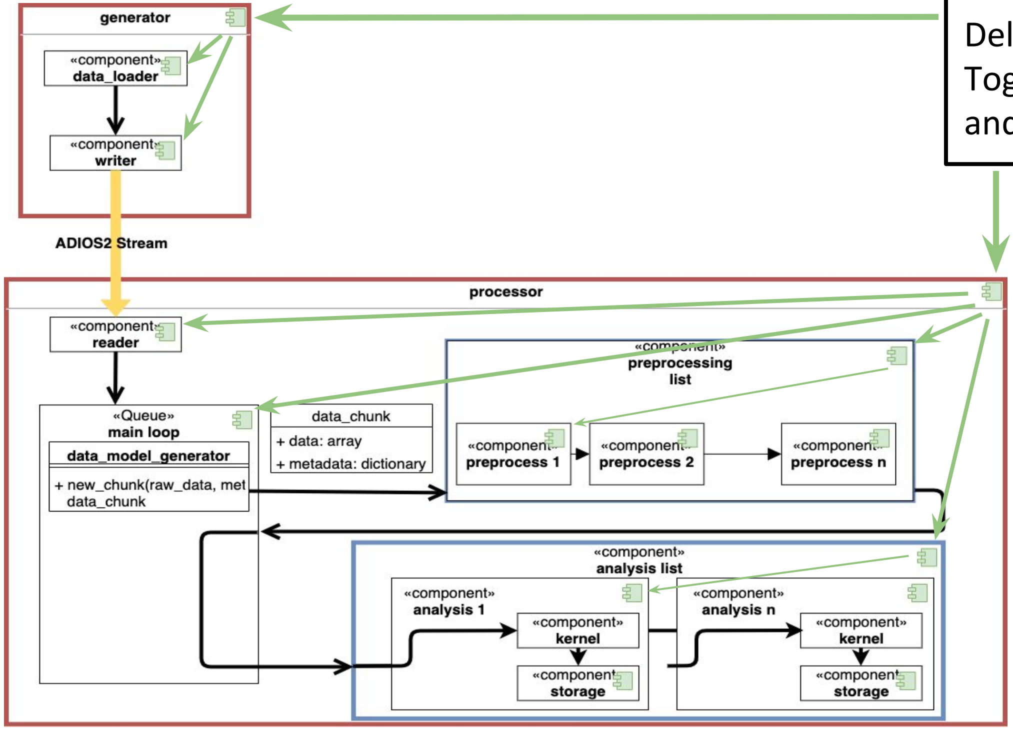
In separate threads: consume items from the Queue and instantiate data_chunk objects

Successively apply a series of preprocessing filters on the data_chunk

Perform data analysis tasks on the filtered data_chunk in parallel. Immediately store results.

8

Reproducible data analysis results guaranteed through shared configuration file.



Delta components share one configuration file. Together with software version (git commit number), and metadata, all analysis results are reproducible.

9

Delta uses a queue and threading to concurrently perform streaming I/O and data analysis

```
def main():
  …
  my_reader = reader(config)
  attrs = my_reader.get_attrs(“stream_attrs”)
  while True:
    stepStatus = my_reader.BeginStep()
    if StepStatus:
      stream_data = my_reader.Get(SSSSS_ECEI_NN)
      data_chunk = new_chunk(stream_data, attrs, cfg)
      Q.put_nowait(data_chunk)
      my_reader.EndStep()
```

A pool of worker threads consume queue items

```
def consume(Q, task_list):
  while True:
    try:
      m = Q.get()
    except queue.Empty:
      break
    m = preprocess.submit(m)
    analysis.submit(m)
  Q.task_done()
```

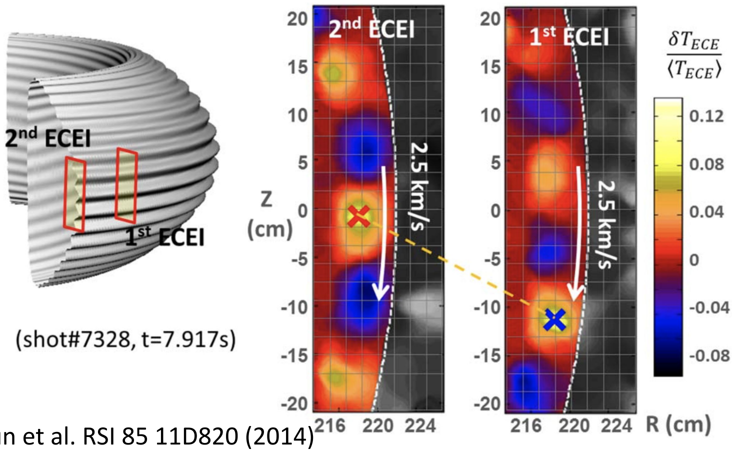
- Reader fetches raw data and attributes from queue
- new_chunk() constructs a data_chunk object for the specific data at hand, e.g. ECEI frames.
- preprocessing and analysis routines interface with data_chunk

Outline

1. Design and implementation details the *Delta* framework
2. Benchmark results
3. Web-based live visualization
4. Conclusions and future work

Benchmark workflow: Perform spectral analysis of ECEI data from KSTAR

ECEI visualizes large scale 2.5d plasma structures



(shot#7328, t=7.917s)

Yun et al. RSI 85 11D820 (2014)

- KSTAR ECE diagnostic: samples 24*8=192 channels with MHz sampling rate
- Diagnostics produces image time-series with about 1GB/sec

ECE benchmark workflow:

Estimate the power spectrum of each channel. Then calculate:

Cross-power: $S_{XY}(\omega) = E[X(\omega)Y^{\dagger}(\omega)]$

Coherence: $C_{XY}(\omega) = |S_{XY}(\omega)| / S_{XX}(\omega)^{½} S_{YY}(\omega)^{½}$

Cross-phase: $P_{XY}(\omega) = \arctan(\mathrm{Im}(S_{XY}(\omega)/\mathrm{Re}(S_{XY}(\omega)))$

Cross-correlation: $R_{XY}(t) = \mathrm{IFFT}(S_{XY}(\omega))$

for all $\binom{192}{2} = 18336$ channel pair combinations (X,Y).

Channel time series are divided into chunks of 10,000 samples.

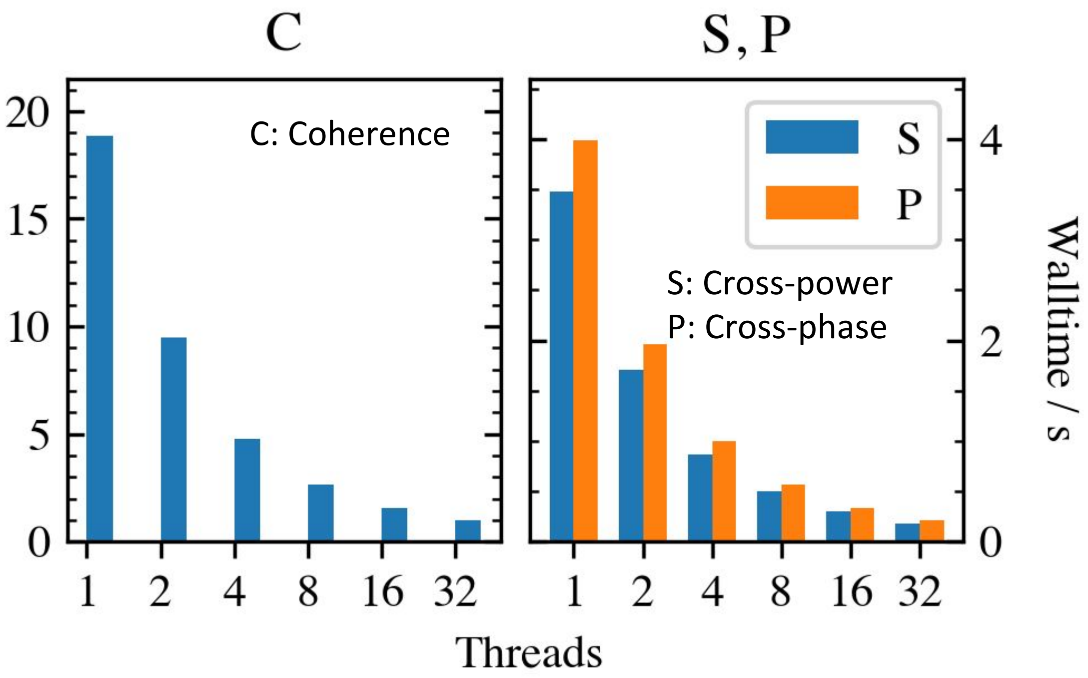
→ Use HPC to analyze many small, independent tasks.

Use cases:

Estimation of local dispersion relation (flow velocity), 2d characterization of T_e turbulence, identification of avalanche-like T_e transport events:

Choi et al. NF 57 126058 (2017); Choi et al. NF 59 086027 (2019);

Multi-threaded implementation of analysis kernels show strong scaling on Cori



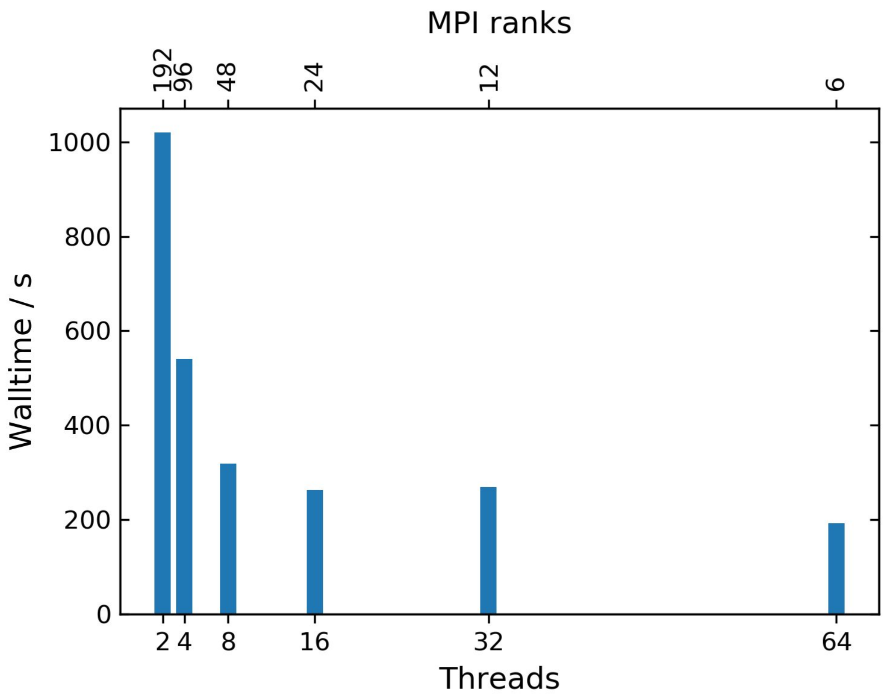
- Spectral analysis kernels C, S, P.
- Using cython to circumvent pythons global interpreter lock
- Execution walltime decreases linearly up to 16 Threads

More threads mean less MPI ranks:

For a constant allocation: $N_{CPU} = N_{MPI} N_{THR}$

- Increasing number of threads N_{THR} decreases number of available MPI ranks N_{MPI} →Less communication

Delta executes the benchmark workflow in between shots



Benchmarks ran on 6 Cori nodes - Xeon Haswell with 64 cores, 128GB RAM (limitation of the real-time queue)

- 192 MPI ranks / 2 threads: Too much communication, CPU cores are not effectively utilized.
- Little speedup when using more than 16 cores
- 6 MPI ranks / 64 threads: Shortest walltime, about 190s.

Time between shots: approx. 10 minutes
Fastest execution time: 190 seconds.
Caveats:

- Data is read from filesystem, no streaming.
- Data analysis results are not stored

Walltime does not change much when streaming + storage is added.

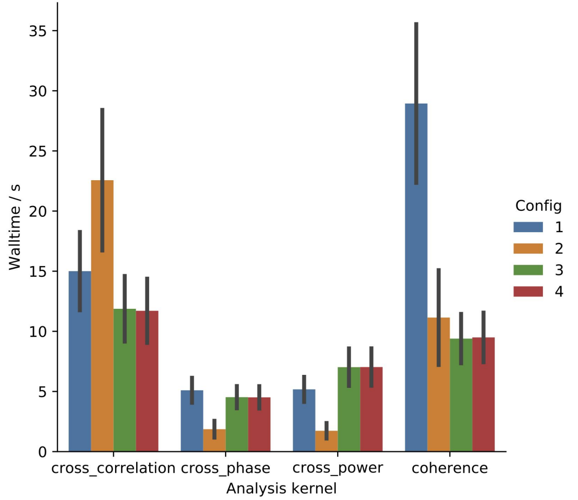
Performing data analysis on GPUs decreases overall walltime by about 35%

Benchmark workflow executed on traverse

- 4 nodes
- 32 MPI ranks / 4 threads per rank

Caveats: Using a drop-in GPU implementation. Not optimized

Scenario	pre-process	Analysis	Avg. walltime / s
1	Host	Serial	933.9
2	Host	Threads	805.7
3	Host	GPU	609.4
4	GPU	GPU	605.3



Kernel execution time measured in each scenario, averaged over 3 runs.

Outline

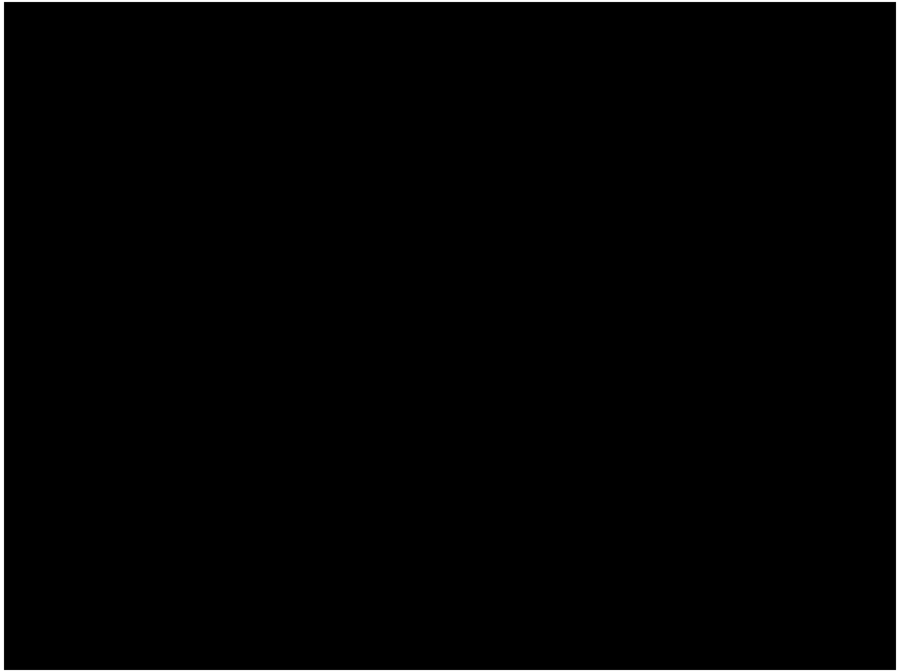
1. Design and implementation details of the *Delta* framework
2. Benchmark results
3. Web-based live visualization
4. Conclusions and future work

Visualization: pull-based

A web server connects
- Database that has analyzed data stored
- Clients which wishes to receive that data

Web server runs on NERSCs spin service as a containerized application.

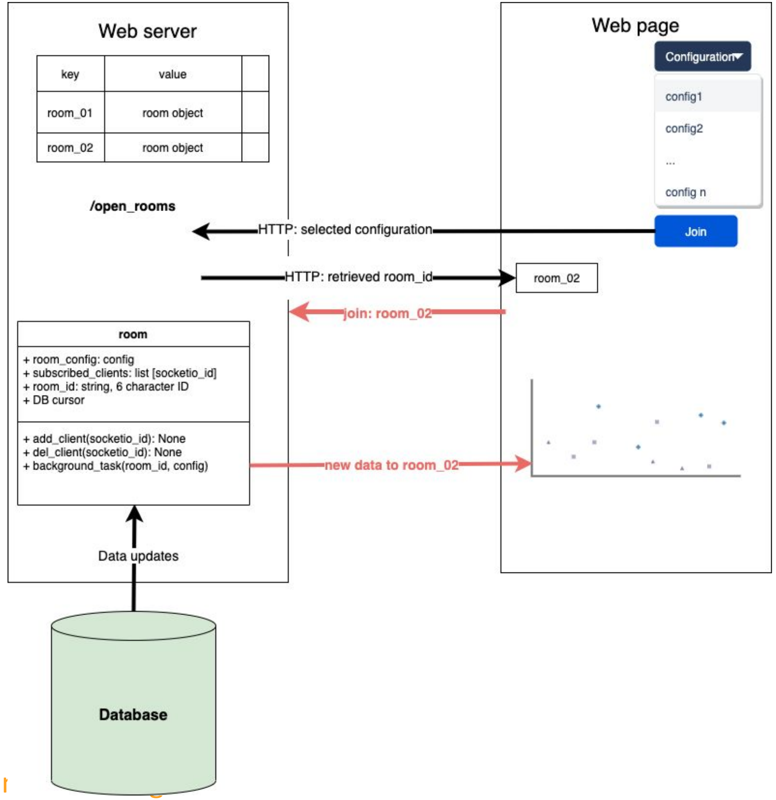
Workflow:
- Client queries a shot
- Server queries the database and returns list of available data chunks
- Client selects a data chunk
- Server retrieves chunk from database, performs post-processing and forwards data
- Client receives data and updates the plot



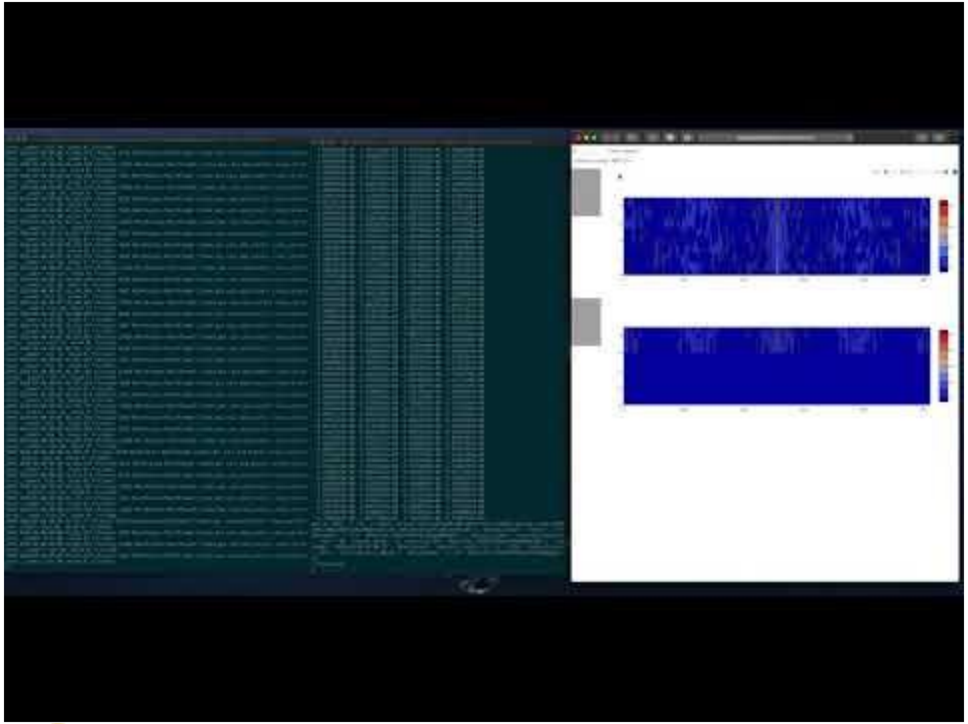
Implementing real-time data visualization

A web server connects
- Database cursors, that receive new analyzed from Delta processor
- Web-clients which wish to receive that data

Implementation:
- Web-client registers interest in a certain data-stream with the server
- Web-server opens a cursor to the database, or associates the client with an existing cursor
- Whenever new data arrives on the cursor, push it to all web-clients. No polling.
- Clients update a plot with the newly received data.



Streaming Visualization (proof-of-concept)



Left: Delta processor on Cori:
analyzes data and stores result in database

Middle: Webserver automatically receives updates from the database and forwards them to the client.

Right: Website (showing 2 clients plots) reactively update the plot when they receive their respective update

Workflow: Select what data to plot. No user-interaction required to receive updates.

Outline

1. Design and implementation details of the *Delta* framework
2. Benchmark results
3. Web-based live visualization
4. Conclusions and future work

Conclusions and future work

We developed the Delta framework which facilitates

- Fast, reliable streaming of big fusion data from experiment to remote HPC centers
- Analysis of data on distributed compute resources, both CPU and GPU
- Storage of analyzed data together with metadata in a database
- Web-based, interactive visualization of analyzed data

Example spectral analysis workflow can be performed in between shots.

Building a database of analyzed fusion data will aid in training data-intensive algorithms

Future work will explore

- Use of ML algorithms to automatically detect and label MHD phenomena (f.ex. unsupervised such as clustering or conv-nets for feature detection)
- Tighter coupling of data-producers to the streaming framework, for example ingesting data directly from digitizers and bypassing the file-system.
- More flexible visualization options such as jupyter notebooks
- Explore use of established streaming data processing software, such as Apache Beam/Kafka.