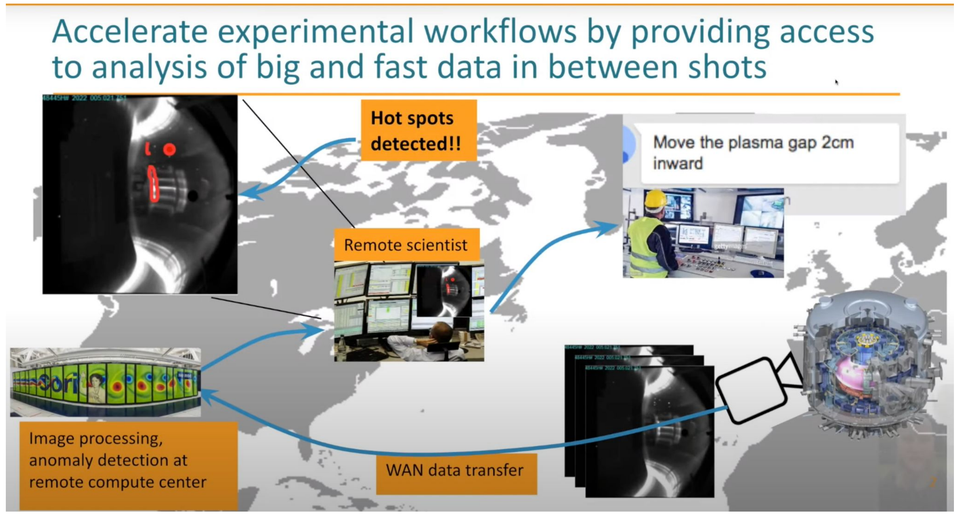
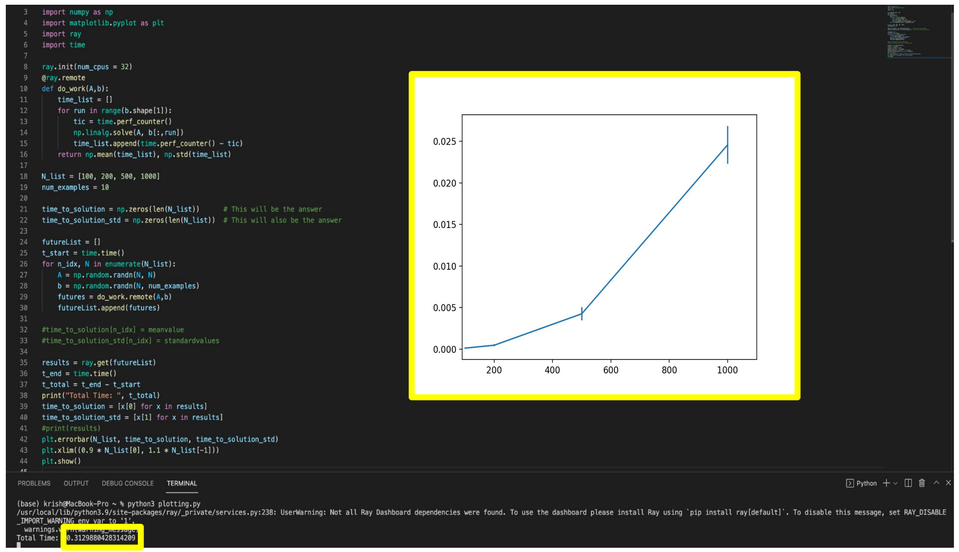
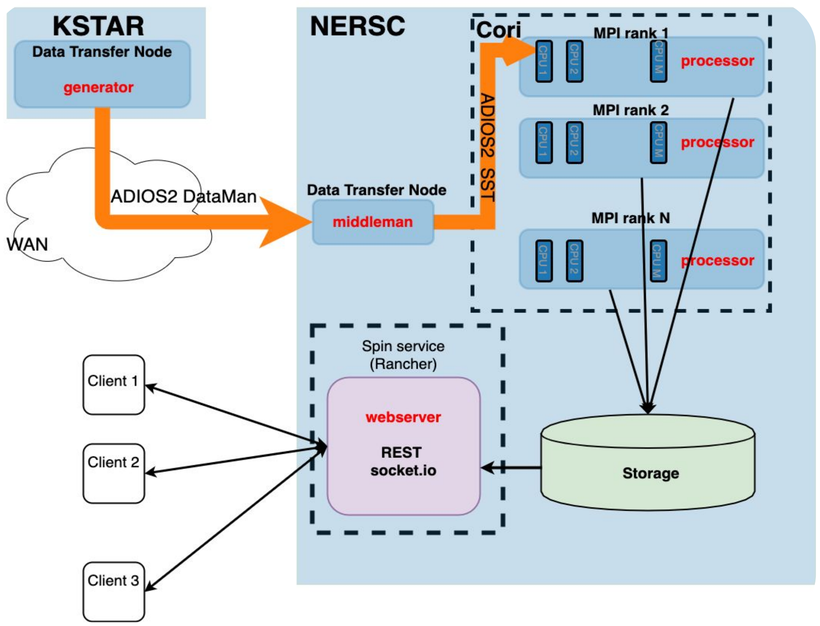
“Facilitating near real-time streaming and analysis of big fusion data on remote HPC resources”

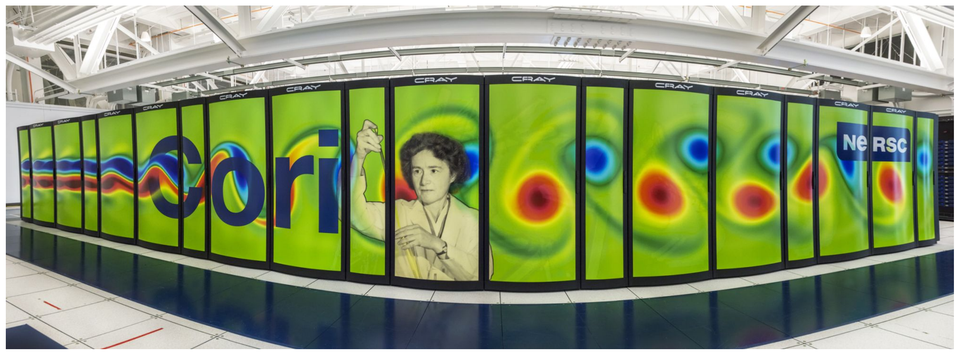
Background:

- During my time at PPPL, I worked on DELTA with Dr. Ralph Kube
- The adaptive near real-time analysis framework facilitates near real-time streaming and analysis of big fusion data on remote HPC resources. It implements software components for loading and staging measurement data, streaming, analysis, and storage. Delta combines these components into a framework for adaptive near-real time analysis of streaming data, targeted towards use-cases in fusion energy research. It consists of multiple executables that send, receive, and process data on different machines.
- Delta first has a generator that runs at the experimental site. The generator stages the experimental data for streaming and sends it to the high-performance computing facility (in our case NERSC). Delta uses the ADIOS2 framework to help stream data (shown by the orange arrows in the image). Following the arrows, we arrive at the HPC center where the data stream is received by a “middleman” running on a NERSC data transfer node. The only job of the middleman is to receive packages from the data stream and forward it to a processor. Finally, the processor receives the data stream and performs data analysis on the distributed computing resources. The analysis results are then stored on a server that is accessible to externally facing services (either file system or database). The web server for visualization runs through NERSC which uses rancher services. The web server connects visualization requests from web clients to the database storage to provide the clients with analysis results in real-time.









RAY

What I Learned:

- During my internship, I was helping create the processors which execute data analysis kernels on a supercomputer. To prepare for creating the processors, I worked with my mentor on creating python programs that will analyze a set of data, give me an output, and use multiple computing nodes simultaneously. To do this, I had to learn a system called Ray. My first step to reaching my final goal was creating a program that used linear algebra to solve a system of linear matrices from a predetermined system size. The time of solving the system would also be recorded. Then, the time needed to solve the system would be graphed as a function of the system size. This program first ran this program on the computer and then ran it on a singular node on Cori. I mostly compared the time differences of a system being solved on a normal computer and a supercomputer. After this experiment, I worked on parallelizing the python code to work on multiple nodes. Since this can only work on Cori, I decided to increase the sample size of each trial run to see how far I can push the limits of my code and the machine. Near the end of my internship, I started tweaking the original code added GPU support to the program which allowed me to process the data even faster. I hope that some of the principles I learned during my internship will be used when creating the processors for Delta.

https://github.com/rkube/delta
https://delta-fusion.readthedocs.io/
https://github.com/NERSC/slurm-ray-cluster
https://docs.nersc.gov/jobs/
https://docs.ray.io/en/master/walkthrough.html
https://docs.nersc.gov/jobs/affinity/#mpiopenmp-example-2
https://my.nersc.gov/script_generator.php
https://docs.nersc.gov/jobs/#srun
https://github.com/NERSC/slurm-ray-cluster/blob/master/submit-ray-cluster.sbatch