Real-Time XFEL Data Analysis at SLAC and NERSC: a Trial Run of Nascent Exascale Experimental Data Analysis

Johannes P. Blaschke
National Energy Research Scientific Computing Center,
Lawrence Berkeley National Laboratory,
1 Cyclotron Road, Berkeley, CA 94720, USA
Email: jpblaschke@lbl.gov

Aaron S. Brewster
Molecular Biophysics and Integrated Bioimaging Division,
Lawrence Berkeley National Laboratory,
1 Cyclotron Road, Berkeley, CA 94720, USA
Email: asbrewster@lbl.gov

Daniel W. Paley
Molecular Biophysics and Integrated Bioimaging Division,
Lawrence Berkeley National Laboratory,
1 Cyclotron Road, Berkeley, CA 94720, USA
Email: dwpaley@lbl.gov

Derek Mendez
Molecular Biophysics and Integrated Bioimaging Division,
Lawrence Berkeley National Laboratory,
1 Cyclotron Road, Berkeley, CA 94720, USA
Email: dermen@lbl.gov

Asmit Bhowmick
Molecular Biophysics and Integrated Bioimaging Division,
Lawrence Berkeley National Laboratory,
1 Cyclotron Road, Berkeley, CA 94720, USA
Email: abhowmick@lbl.gov

Nicholas K. Sauter
Molecular Biophysics and Integrated Bioimaging Division,
Lawrence Berkeley National Laboratory,
1 Cyclotron Road, Berkeley, CA 94720, USA
Email: nksauter@lbl.gov

Wilko Kröger
SLAC National Accelerator Laboratory,
2575 Sand Hill Road, Menlo Park, CA 94025, USA
Email: wilko@slac.stanford.edu

Murali Shankar
SLAC National Accelerator Laboratory,
2575 Sand Hill Road, Menlo Park, CA 94025, USA
Email: mshankar@slac.stanford.edu

Bjoern Enders
National Energy Research Scientific Computing Center,
Lawrence Berkeley National Laboratory,
1 Cyclotron Road, Berkeley, CA 94720, USA
Email: benders@lbl.gov

Deborah Bard
National Energy Research Scientific Computing Center,
Lawrence Berkeley National Laboratory,
1 Cyclotron Road, Berkeley, CA 94720, USA
Email: djbard@lbl.gov

Abstract—X-ray scattering experiments using Free Electron Lasers (XFELs) are a powerful tool to determine the molecular structure and function of unknown samples (such as COVID-19 viral proteins). XFEL experiments are a challenge to computing in two ways: i) due to the high cost of running XFELs, a fast turnaround time from data acquisition to data analysis is essential to make informed decisions on experimental protocols; ii) data collection rates are growing exponentially, requiring new scalable algorithms. Here we report our experiences analyzing data from two experiments at the Linac Coherent Light Source (LCLS) during September 2020. Raw data were analyzed on NERSC's Cori XC40 system, using the Superfacility paradigm: our workflow automatically moves raw data between LCLS and NERSC, where it is analyzed using the software package CCTBX. We achieved real time data analysis with a turnaround time from data acquisition to full molecular reconstruction in as little as 10 min – sufficient time for the experiment's operators to make informed decisions. By hosting the data analysis on Cori, and by automating LCLS-NERSC interoperability, we achieved a data analysis rate which matches the data acquisition rate. Completing data analysis with 10 mins is a first for XFEL experiments and an important milestone if we are to keep up with data collection trends.

I. INTRODUCTION

X-ray scattering experiments using Free Electron Lasers (XFELs) are a powerful tool to determine the molecular structure and function of unknown samples, such as COVID-19 viral proteins. The X-ray light produced by XFELs is particularly useful as a tool for probing microscopic samples as it is coherent and intense – allowing teams of scientists to probe structural details that leave only a weak trace signal [1]. However all of this comes at a significant cost: XFEL facilities require specialized equipment and large teams to operate. To operate efficiently, it is essential that the experimental
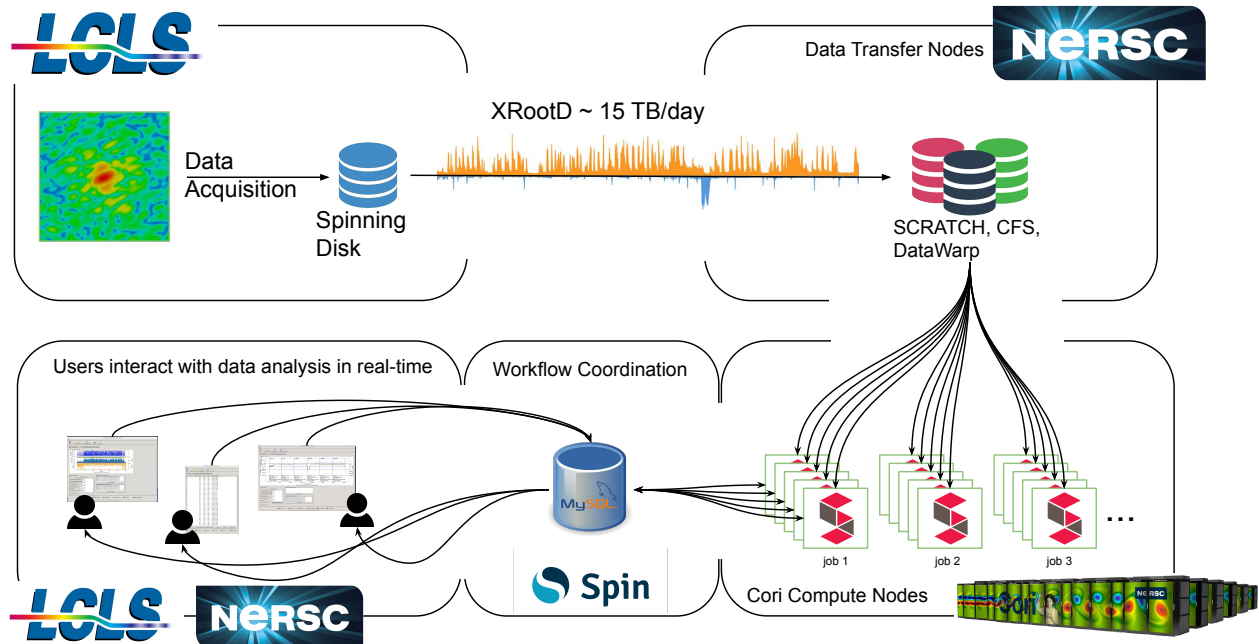
Fig. 1. Sketch of the super-facility workflow: *Top:* Data are automatically transferred from the LCLS spinning-disk storage system via XRootD to NERSC's Scratch file system (the orange and blue spikes show the data transfer rate – approx. 1.3-2.6 GB/s – over the ESNet network, with each spike being a completed run). *Bottom*: On NERSC the CCTBX workers (running in Shifter containers on the Cori compute nodes) automatically analyze new data on Scratch, using the DataWarp burst buffer as a cache. Users at LCLS and NERSC connect to a mySQL database hosted at NERSC to orchestrate the workers, review the data analysis and iterate analysis parameters.

investigators have immediate feedback from data analysis in order to make informed decisions about their experiments in real time. By 2025 the next generation of XFEL experiments will more than double the detector resolution, and increase the rate at which measurements are taken by a factor of over $400\times$ compared to existing facilities. This will require computational intensity levels to escalate from petascale to exascale, for data analysis to keep pace with data collection.

To rise to these challenges, the Linac Coherent Light Source (LCLS) at SLAC has partnered with the National Energy Scientific Computing center (NERSC) at LBNL using a "Superfacility" model [2]: data collected at SLAC are immediately transferred to NERSC (via ESnet) where they are analysed on the Cori XC40 supercomputer. The results are then reported back to the experiment's operators in real time. In this paper, we demonstrate the usefulness of this approach by reporting our experiences from two experiments in September 2020: LV95, which consisted of small molecules related to materials science; and P175, which consisted of COVID-19 viral proteins and potential bound ligands. These experiments needed to test many samples during limited beam-time. In order to know when to move on to the next sample and to make changes to experimental protocol, a complete (or near complete) analysis of the collected data needs to happen at the same rates at which the data are collected.

II. ANALYSING LCLS DATA AT NERSC

Data were collected at a peak rate of 120 images/second (approx. 1/42 of the data collection rate expected in 2023), totalling 15 TB/day. A total of 130 TB of raw data comprising 28 million images were collected during the experiments described in this paper. This is too much data to manage manually, therefore we use the super-facility paradigm: our workflow automatically moves raw data between LCLS and NERSC, where it is analyzed using the CCTBX software package[1] [3], [4]. By running on 64 Haswell nodes, we achieved real time data analysis with a 10 min peak turnaround time from data acquisition to full molecular reconstruction – sufficient time for the experiment's operators to make informed decisions between data-collecting runs. At this computational intensity, the data analysis rate matches the data acquisition rate. This demonstrates the usefulness of the Superfacility approach: by automating job submission and data management, we where able to analyze critical measurements in under 20 mins, a first for XFEL experiments and an important milestone if we are to keep up with instrument data collection trends.

In this paper we give a detailed step-by-step description showing how our workflow is deployed on NERSC's systems; how it coordinates data movement (between SLAC and

[1]The scripts to build CCTBX at NERSC, and the Docker image used for the data processing jobs are available here: https://gitlab.com/NERSC/lcls-software/-/tree/beamtime-2020-09/cctbx-production

Experiment	LV95		P175	
Spotfinding	17M	6%	11M	49%
Indexing	2M	25%	582K	7%
Refinement	564K	99%	46K	85%
Integrating	559K	99%	33K	97%
Total CPU utilization	22663 core-hr		31167 core-hr	

TABLE I

NERSC, discussed in section II-A) and data analysis (via batch jobs at NERSC, discussed in section II-C); and how CCTBX enables interactive data analysis with several human operators in the loop (discussed in section II-B). CCTBX (specifically the *cctbx.xfel* subpackage) is a fully-automatic pipeline management system that: i) tracks new incoming data, and relates it to experimental parameters ("tags") provided by the scientists; ii) automatically submits new analysis jobs (using containerized workers) as new data come in; and iii) reports analysis results via a database hosted on NERSC's "Spin" micro-services platform in real time (discussed in section II-D). This allows a team of scientists to work on the same data via one integrated GUI, while CCTBX coordinates a "swarm" of workers behind the scenes. Fig. 1 illustrates this workflow.

A. Transferring Data to NERSC

The LCLS data movers are responsible for the transfer of the data between the different storage resources used by LCLS. Within the LCLS systems the data are moved from the data acquisition storage to the high performance fast-feedback storage and the large long term analysis storage. The movers also perform the data transfer to the remote HPC sites currently supporting NERSC and the SLAC Shared Scientific Data Facility (SDF). The data mover is a component of the LCLS data management systems and communicates with other components by publishing and subscribing to streams of events using Kafka. The main events for the data-mover are subscribing to new-files-created events and publishing that files have been transferred to a particular storage resource. For the remote transfers the XRootD data server is used. Each remote site exports its shared file system through XRootD which runs on multiple data transfer nodes that each site provides. All servers at a site are clustered into a single system using XRootD's clustering functionality. The data mover uses the XRootD transfer tool *xrdcp* in third party copy mode. The data are directly transferred between an XRootD server at the source and destination, without involving the node the mover is running on. In this instance the destination pulls the data from the source. Fig. 2 shows the NERSC and LCLS XRootD setup. The main entry point into each cluster is the redirector
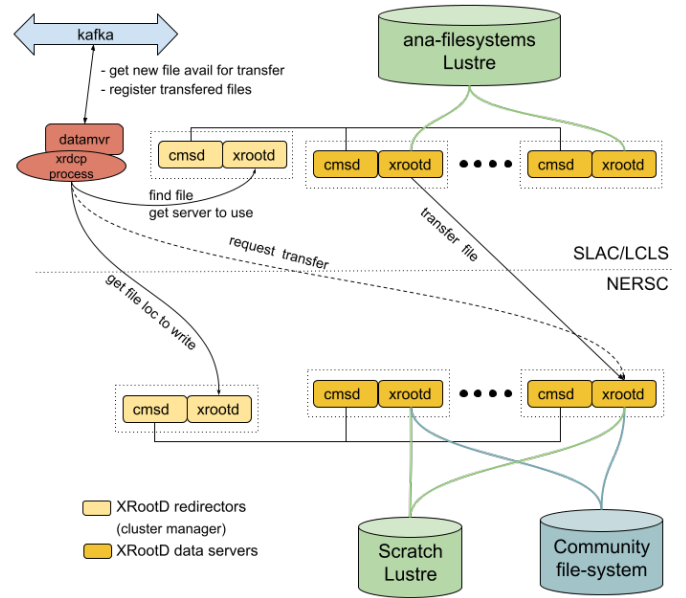

Fig. 2. Schematic of how the data mover transfers data using the NERSC – LCLS XRootD clusters. *Top:* Kafka + data mover pipeline at LCLS together with the XRootD cluster used to send data (via ESNet) to the corresponding cluster at NERSC. *Bottom:* XRootD cluster deployed on two data transfer nodes at NERSC. Once a new file is created, and logged as a file creation event in Kafka (the LCLS data "logbook" service), the data mover initiates a data transfer using the XRootD cluster running at LCLS. The data is transferred via ESnet to its counterpart at NERSC, where the data is deposited in the *SCRATCH* Lustre file system. Once a file has been transferred, it's status in Kafka is recorded as "available at NERSC" – allowing *cctbx.xfel* to begin data analysis.

(aka cluster manager). It redirects the client to the data server that should be used for reading and writing the data.

The data mover is a Python application whose main task is to perform many transfers in parallel. It has two options to discover which files to transfer: either monitor the experiment folder for new files or subscribe to a Kafka stream (the LCLS data "logbook" service) which signals that new files have been created. As new files are created the mover adds them to its internal persistent queue. The files are sorted by run and the oldest runs are transferred first. A run typically consists of 12-18 files. A third of these files contain the detector data and are between a few to 100 GB in size. For each of these files there are two index files that allow random access to the detector data. The size of the index files is less than 1% of the detector data files. The observed peak ESNet tranfer speeds showed bursts of 2.6 GB/s whenever runs where completed. This includes the Lustre I/O speed, both at LCLS and at NERSC. The bursts are due to the data only being transferred once the runs are "concluded" in Kafka.

The Kafka + data mover + XRootD pipeline is fully automated and scalable. Once an experimental run is concluded (a "run" is usually 5-30 mins worth of data collection) the raw data files, as well as index files and calibration data, are automatically recorded in Kafka as "ready to be transferred" and this pipeline will begin the transfer to NERSC. The transfer is usually completed within 3 minutes and the status is updated
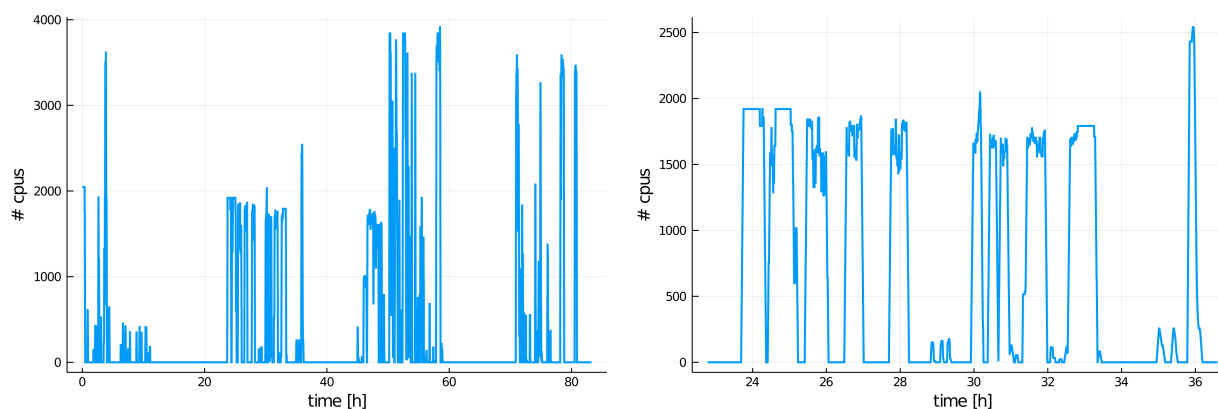
Fig. 3. CPU usage for the P175 experiment. *Left:* CPU usage on Cori Haswell for the whole duration of the experiment. Only the day shifts collected data, therefore no data analysis was needed at night. *Right:* CPU usage for one day shift (on the second day of the experiment). We see the "bursty" CPU utilization that results from urgent computing: whenever new data are available they need to be analyzed as quickly as possible. Once data have been analyzed, the CPUs on Cori go idle, while waiting for new data.

in Kafka as "available at NERSC". The XRootD cluster is fully scalable, allowing us to transfer all the files generated in one run at once.

B. Pipeline Management

Typical XFEL experiments involve collecting multiple datasets for the same or similar samples, potentially moving them through some reaction condition and capturing the structural changes as the reaction proceeds, or screening proteins with a variety of ligands that are biologically or pharmacologically relevant, such as in the case for the COVID-19 viral proteins from experiment P175. Samples therefore accrue a great deal of metadata, and each run needs to be associated with this metadata so datasets can be produced from the right subsets of diffraction images. Therefore, the first task the user completes in the *cctbx.xfel* GUI is tagging runs with short, descriptive terms, such as "batch1", "reactionstate2", or "ligand3". Multiple tags can be added to a data set.

Next, the user needs to provide processing parameters for each dataset. These parameters include details needed to extract reflection data, the experimental geometry such as the location of the detector in 3D space, and known crystal properties. These parameters will need to be updated (with better estimates) as the experiment progresses, and so they are organized by trials, in which the user can change the parameters and re-process the data. This organization into trials is particularly helpful when keeping track of which parameters were used during re-processing.

Finally, the user specifies which tags will form a dataset, mixing and matching them as needed. With these properties in place, the GUI will run through a cycle of determining which tasks are needed to be performed on which data, and submitting these tasks to the cluster to be processed. The GUI monitors the state of each task and continues to submit new jobs as data arrive or as processing tasks finish, allowing downstream tasks to be submitted on upstream results.

The *cctbx.xfel* GUI therefore provides the experiment's operators with a complete pipeline management tool, which lets

multiple users simultaneously specify analysis parameters and view analysis results. When new data or analysis parameters are detected *cctbx.xfel* automatically builds Slurm jobscripts and input files, and submits these to a set of reserved compute nodes. Please see the supplements[2] for a run-through of the *cctbx.xfel* GUI. By acting as the interface with the supercomputer, the *cctbx.xfel* pipeline management system allows scientists to treat HPC as a reactive element. Fig. 3 shows a time series of the CPU utilization during the P175 experiment. This usage pattern is typical of the *cctbx.xfel* workflow: whenever new data are available, they need to be analyzed as quickly as possible resulting in a sudden need for up to 64 Cori Haswell nodes.

C. Processing Data on Cori Compute Nodes

Data was processed on up to 64 Haswell nodes (each consisting of two 16-core Intel Haswell CPUs) on NERSC's Cori XC40 system. The computational workload is highly variable (*cf.* Fig. 3) depending on the nature of the data being collected. XFEL data analysis follows several sequential stages: i) Identifying Bragg spots in a diffraction image (*spotfinding*); ii) Associating each Bragg spot with a Miller index (*indexing*); iii) Refining unknown model parameters (*refinement*); and iv) combining and integrating data for the same Miller indices collected over several images (*integrating*). Table I shows that each stage is lossy: not every image contains data of sufficient quality (*ie.* not enough high-intensity Bragg spots) to conclusively analyze. This means that each subsequent stage processes fewer data – and therefore needs fewer computational resources. Hence stages (iii) and (iv) require much smaller jobs on Cori than stages (i) and (ii).

The computational motif is identical (except for number of images) for each stage. *cctbx.xfel* uses MPI to distribute work over up to 64 Cori Haswell nodes. The work is distributed using a producer-consumer model, where each image is processed largely independently. Fig. 4 sketches this computational motif.

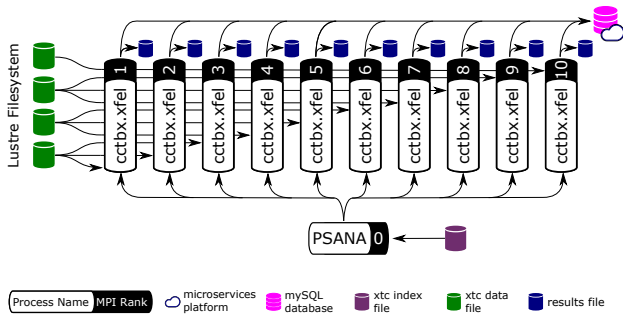[2]Video available on slide 5, or here: https://docs.google.com/file/d/1AXlobhQuEECvVY_DhUqadoolKHx3kmWy/preview

Fig. 4. Structure of an analysis worker running on the Cori Haswell nodes. We rely on MPI parallelism to distribute work between nodes (openMP is also available, but was not needed to achieve the desired throughput). We employ a producer-consumer model to distribute work and achieve load balancing. Data is provided by *psana*, which runs on the first MPI rank. *psana* reads an index file and distributes work to the *cctbx.xfel* workers. The resulting program is a flat tree of MPI ranks with data analysis ranks located at leaves. Workers access data directly by reading the raw data files using offsets provided by the "PSANA" (root) tree node. Finally, the *cctbx.xfel* workers save their results to disk (local to each MPI rank, using the DataWarp burst buffer) and report the analysis progress to a mySQL database hosted on NERSC's Spin micro-services platform. Arrows indicate the overall flow of data.

Fig. 6. Probability distribution of the time taken to perform different data analysis tasks. While most processing steps complete within a few seconds, data analysis can occasionally take significantly longer. Due to this variability, our workflow uses producer-consumer parallelism – which is automatically load-balanced.
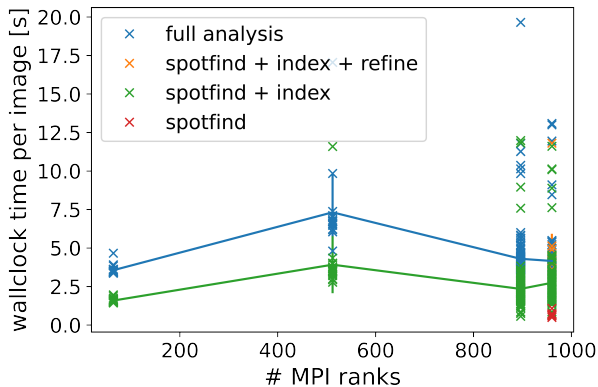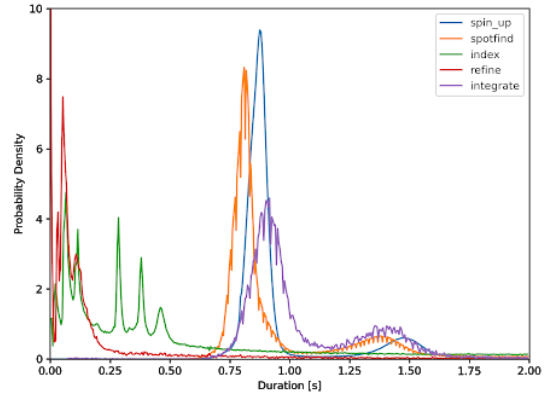

Fig. 5. The average time to process an image remains constant with the number of MPI ranks used. Colors show the different stages of the data analysis pipeline. We also see that the variability grows with number of MPI ranks, in part due to increased resource contention. However, the vast majority of images can be processed with near-constant time, achieving weak scaling on the Cori Haswell nodes.

We use *psana* [5] to read the raw data files. In Fig. 4 we show an example configuration where rank 0 distributes work to available ranks. The producer rank uses MPI to distribute offsets into the raw data files (green "buckets" in Fig. 4). The worker ranks then process each image independently, by accessing the data files (each run's data is stored across several files) using an offset and applying the detector calibration in memory. From here on stages (i) – (iv) are applied without communicating with any other ranks. Finally results are stored to the DataWarp burst buffer (blue buckets) and a mySQL database (pink database icon in Fig. 4).

Fig. 5 shows the average time to process an image. We see that *cctbx.xfel* achieves near-ideal weak scaling, regardless of whether a partial data analysis (red, and green symbols), or a complete data analysis is being performed. The performance

variability in the wallclock per image does increase with the number of MPI ranks. This is primarily due to shared resource contention such as I/O and network latency. Fig. 6 shows the probability density function of the wallclock time for each step. This variability can have two sources: 1) algorithmic: e.g. the peaks in the green curve show different indexing algorithms being applied to the data; and 2) resource contention. The distributions are strongly-peaked, and therefore the vast majority of images are analyzed within 7s. However as it is not possible to predict exactly how long it will take to analyze a batch of images, we use a producer-consumer workflow as it is automatically load-balancing.

A helpful tool to identify performance variability due to resource contention is the computational weather plot as shown in Fig. 7. The MPI ranks are enumerated on the y-axis and wallclock time is plotted on the x-axis. Each worker is plotted as a collection of horizontal lines (a new line for each image). As different images are analyzed, the horizontal line is given different colors: initialization and I/O (red); spot finding (green); indexing (blue); model refinement (dark green); and integration (black). MPI communication happens only when images are assigned to a particular worker and therefore those regions are not plotted (*ie*. they are the white regions between images). Results are stored at the end of each processing step (raw data files are only read during the initialization step).

To demonstrate this powerful diagnostic tool, the left and central panels shown in Fig. 7 show two different forms of contention. The left panel shows an MPI communication-bound job: most time was spent between images, waiting for new work (which is distributed using MPI). The central panel shows an example of I/O contention: at the end of each processing step data is written to the Lustre *SCRATCH* file system, which resulted in several nodes hanging while trying to open files simultaneously. The right panel shows the same setup where each rank caches results to the DataWarp burst buffer instead
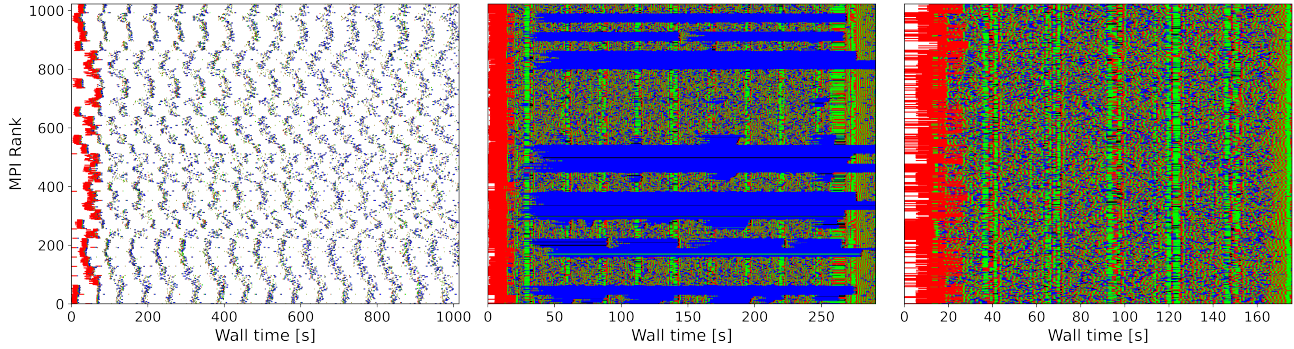
Fig. 7. Computational weather plot illustrating two barriers to scaling (left and center), as well as near-optimal performance (right). Weather plots show resource contention by plotting the data processing timeline of each rank. The colors represent different processing steps: initialization and I/O (red); spot finding (green); indexing (blue); model refinement (dark green); and integration (black). MPI communication is not profiled and is included in the white areas. The left plot shows an MPI communication-bound setup. After optimizing the MPI communication code almost all white space disappears. However this reveals I/O contention (note that all steps open files when saving intermediate results and for logging) on the SCRATCH file system (central plot) as shown by some nodes working normally, while others appear stuck. Switching to the DataWarp burst-buffer resolves this I/O contention resulting in near optimal performance (right plot).
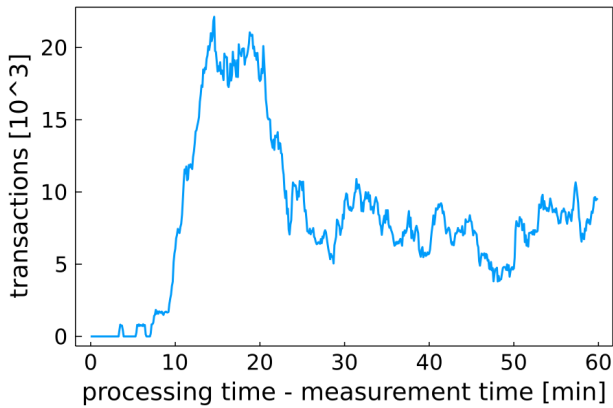

Fig. 8. Histogram of the time between recording an event and first data processing completing at least one step of data processing for the P175 experiment. We find that a few images (those at the end of a run) are processed within 3.5 minutes – given that a data transfer usually takes approx 3 minutes, these images where processed only a few seconds after arriving at NERSC. Most images where processed within approx. 10-20 minutes. The tails in the processing time after 20mins are due to data reprocessing (*cf.* section III-A).

of *SCRATCH*.

D. Workflow Orchestration

We use a relational database implemented via MySQL to maintain the associations between data and results in various stages of processing. In addition, since processing results are logged to the database quickly, we can access those results from the experiment control room and display them to on-site users for rapid feedback on the data they are collecting.

We selected Spin, a NERSC microservices platform for container-based services, to host our MySQL server reliably and scalably[3]. Spin hosts services that can be accessed from the Cori compute and login nodes. Having access to both kinds of node is essential because the *cctbx.xfel* GUI runs

on the login nodes and needs to be able query the database in order to display the progress of data processing jobs, as well as determine which new jobs to submit. The workers do not query the database, instead they commit the status of the images they are processing (e.g. number of spots found per image, the rate at which they are indexed, etc). Hence, even though thousands of ranks will be committing status updates to the MySQL database, these transactions are light weight, with the MySQL service handling them well. We found that database connections and transactions consumed between 1% and 3% of total runtime. This includes latencies caused by accessing Spin via the (slower) TCP network. Furthermore, Spin is scalable, which enables us to flexibly increase the number of connections the database service can efficiently manage as we scale to ever larger data processing workloads.

Fig. 8 shows the effectiveness of this approach by plotting a histogram of the difference between the data collection time and the processing time. We see that some images (a few thousand) where processed within 3.5 minutes – this includes the transfer time of approx. 3 minutes. The majority of images are processed between 10-20 minutes after data is collected. While this does not include reprocessing, or interpreting the results, it does demonstrate that cross-site automation is crucial for fast turn-around.

III. HPC CHALLENGES

While this is a relatively modest computing footprint compared to traditional HPC workloads, real-time data analysis requires the coordination of many moving parts ranging from traditional computing to networking and IO. Data sets are expected to grow at least 3000-fold with increasing detector resolution, beam intensity, and measurement rate. Therefore, the performance profiling data we collected represents an important benchmark, allowing us to extrapolate the overall performance of this Superfacility workflow and predict future bottlenecks which would prevent scaling.
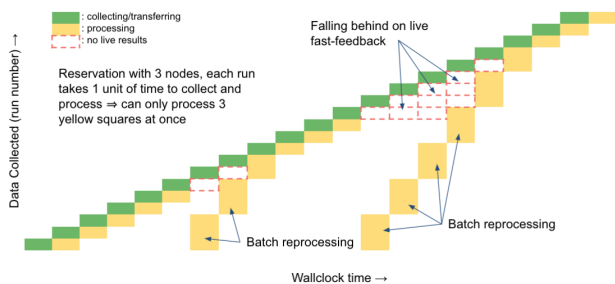
[3]The scripts to deploy a mySQL database using Spin are available here: https://gitlab.com/NERSC/lcls-software/-/tree/beamtime-2020-09/spin/mysql-p175

Fig. 9. An illustration of the XFEL urgent computing needs. The x-axis represents time, and y-axis represents the number of data sets collected. To keep up with processing, as data from runs arrive (green boxes), processing jobs are submitted as soon as possible (yellow boxes). For simplicity we assume that it takes roughly the same amount of time to process a data set as it takes to collect it (green and yellow boxes are the same size). Furthermore, to illustrate the problem of limited reservation sizes, we assume that our reservation has a maximum size of 3 nodes (3 yellow boxes). When new parameters are discovered, all data must be re-processed in a batch, and on a limited reservation, this can lead to delays in live feedback (red boxes). Furthermore, the burden of reprocessing grows with the data set size. Therefore a reservation would potentially need to be as large as the final data set.
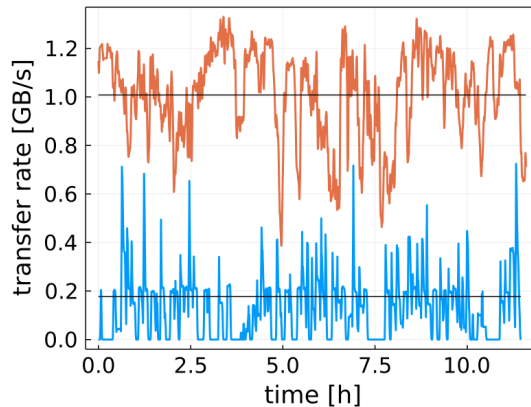

Fig. 10. Data transfer rate between the LCLS and NERSC (Lustre *SCRATCH*) file systems. This rate includes disk read and write speeds, which ultimately limited the rate at which data can be transferred to NERSC. Horizontal black lines show average transfer rates. The orange line shows a representative "good" data transfer speed. However, depending on contention in the Lustre file system at NERSC, this transfer rate can be $5 - 6\times$ lower – shown by the blue line.

A. Urgent and Real-time Computing

XFEL data reduction challenges computing clusters in two ways 1) unequal data processing needs per frame and 2) stochastic (*cf.* Fig. 6) and bursty (*cf.* Fig. 3) computational needs. Together these result in a demand on the job scheduler where computational resources are urgently needed (the urgency is due to the need for fast real-time data processing), with little advance warning (only after all the data has been processed, do we know how many images resulted from "good" measurements).

At NERSC we have enabled time-sensitive computing by allowing nodes on Cori to be *reserved* ahead of time. These nodes will then be kept clear of jobs not explicitly submitted to this reservation.

1) Unequal Data Processing Time: In each run, thousands of image frames are recorded, but how far each frame makes it through the processing pipeline varies widely. A frame could be a complete miss, without a crystal. A crystal may not be of sufficient quality to be processed, and even if it is, it may not be isomorphous with the rest of the data. At each step, the image can be rejected for a variety of reasons. This means the amount of time needed to process an image can vary wildly. This is illustrated in Table I: each processing stage (row) has a finite "success" rate, and therefore only a fraction of images go onto the next stage.

We solve this problem by splitting the pipeline into tasks and using fewer cores for downstream tasks. For example, during P175, for indexing and integration, we used 28 nodes per job, but for scaling and merging which does not read the pixel data, we only used 1-2 nodes per job.

Further, for indexing and integration we use a producer/consumer approach, where a root MPI rank sends images to the other ranks. Each rank reports back when they finish an image and receive a new one to process. In this way, all the ranks are kept busy until the images have all been processed.

2) Stochastic and Bursty Compute: Ideal processing parameters are rarely known when an experiment begins, and midway through data collection, new parameters can be discovered which obviate all previous processing results. In classical computing scheduling this leads to two inefficiencies (Fig. 9).

First, if the set of reserved compute nodes is big enough to accommodate processing needs plus an additional safety margin, then when data is not being collected or when typical processing patterns are being observed, the cluster can be underutilized. Second, when batch-reprocessing needs to occur due to the addition of new parameters, real-time processing can fall behind.

These problems necessitate different scheduling systems than reservations or first in-first out. Our experiences with real-time data processing for LV95 and P175 have shown that reservations are able to guarantee enough computational resources for time-sensitive data processing. However reservations alone can be a wasteful solution: any time the reservation goes unused (*eg.* between measurements) will result in idle compute nodes.

These systems are being developed, but could include a mix of reservations plus real-time priority access to compute resources, which can be released to lower-priority jobs when not being immediately used. Furthermore, preemption is a promising solution to allow underutilized reservations to be filled by preemptible jobs until the compute nodes are needed for urgent computing tasks. Preemptible jobs are programs that listen for a system interrupt (*eg.* SIGINT), and – upon receipt – gracefully save and quit. At NERSC we have developed a reservation system by which preemptible jobs can enter a reservation. These will then be stopped if new jobs are submitted directly to the reservation (after a warning period during which SIGINT is used to request that the job saves and quits).
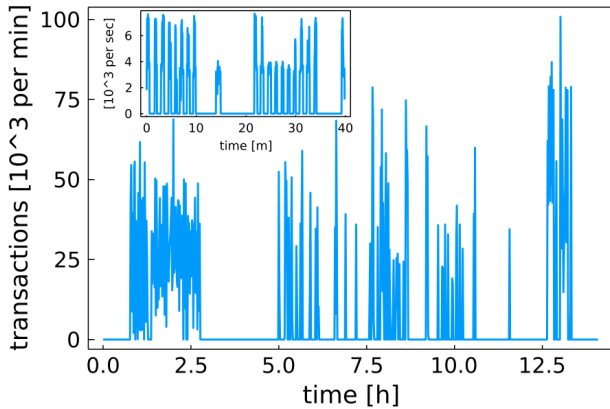
Fig. 11. Rate of database transactions during live data processing. The main plot shows the number (in thousands) of database transactions per minute during a 12 hour shift. The inset shows a 40-min snapshot of number (in thousands) per second. We see that the database receives up to 8000 commits/second, whenever data processing takes place (the "bursts" in the inset show individual data analysis jobs). Despite this heavy load, the Spin microservices platform was capable of handling this load level.

B. I/O and Network Performance

The 100 Gb/s network connection (hosted by ESnet) between LCLS and NERSC made it possible to transfer most raw data files within 3 minutes after concluding the run. Bandwidth on the ESNet link was reserved ahead of time using the SENSE API [6]. The XRootD clusters at LCLS and at NERSC performed well, and could be scaled easily to accommodate more files if a backlog occurred.

In fact, the I/O speeds of the Lustre file systems at LCLS and at NERSC were the rate limiting factor. Fig. 10 shows the end-to-end data transfer rate from LCLS to NERSC. The different lines are measurements taken on two different days. This makes it clear that there are "good" and "bad" days for file system utilization. The 5-6 fold difference is due to a bug in NERSC's *SCRATCH* file system, where some of Lustre's OSTs have a slow write speed. On a "bad day" the slow Lustre write speed can become the dominant bottleneck in the data processing pipeline, where the data for run N is not transferred before run $N + 1$ commences. This highlights that reliable high-performance I/O is crucial for experimental science workflows.

C. Workflow Orchestration

Workflow orchestration at scale is always a challenge, as potentially hundreds of thousands of tasks need to be coordinated from a central place. In our workflow manager, the database takes on the role orchestrating the distributed data processing. Therefore database communication is a potential single point of failure and a bottleneck when experiments are scaled up to the kHz regime with thousands of MPI ranks reporting results simultaneously. Fig. 11 shows that a Spin-hosted MySQL database was able to accommodate the load of approx. 8000 transactions/sec.

While the MySQL database server was selected with scaling in mind, some further optimizations became necessary when performing large-scale analysis runs.

1) Limiting concurrent connections: In some configurations, the usable number of MPI ranks was limited by the concurrent connections that our database could support. We refactored our database communication to cache all database queries for a small set of images before flushing the cache via a single temporary database connection. This reduced the peak concurrent database connections to 1 per 10 MPI ranks.

2) Transactions: For processing in the kHz regime during a different experiment we encountered another bottleneck when many small queries had to be executed sequentially, with later queries depending on earlier ones. Without access to the Spin system, we were overloading the MySQL server we were using, to the point where logging 50000 images could take over an hour. Using the MySQL statement LAST_INSERT_ID() we were able to combine many queries into a single transaction. With this approach, we could log these images using a single MySQL query comprising 130K lines that takes 0.07 seconds.

A related challenge to workflow orchestration is the variable processing time per image. Fig. 6 shows the variability due to *algorithmic* differences between images (*eg.* the peaks in the green line are due to the indexing algorithm "trying" different approaches to find a solution). Therefore we employ a producer-consumer model to distribute parallel tasks across MPI ranks while maintaining a balanced workload (*cf.* section II-C). As the data analysis for each image can have a subtly different call tree, this can have a subtle impact on optimizing performance and diagnosing errors: we can not expect each logical task to take roughly the same amount of time. We observe that between 2% and 3% of images take significantly longer than 2s to process. Using the *hatchet* tool [7] we where able to compare the profiles for jobs with different call trees. Hatchets allows us to analyze each job's call tree hierarchically, and compare common sub-graphs. We found that the slow jobs where a result of I/O contention while reading data, saving results and logging progress. This highlights an important difference to many simulation codes: data analysis workflows often have branching source codes, and invoke many libraries – it is therefore not always possible to optimize the overall run time by merely focusing on a hand full of subroutines that are called over and over.

IV. SUPERFACILITY API

Over the years, NERSC staff has observed how many research workflow operations fall into natural patterns of recurring actions that are carried out when analyzing data. The traditional approach for HPC centers is to provide human-readable interfaces and also to design the experience to meet the interactive expectations of a human user. However this design collapses with workflows that need to run at larger scale or at faster rates such as automated, machine-driven workflows initiated at external facilities such as LCLS-II. We expect this mode of operation to become more prevalent in the future as more and more DOE facilities intend to link into ASCR

computing infrastructure to address their data and computing needs. Providing machine-readable APIs for HPC resources is the logical prerequisite to make this connection happen. It is also particularly fitting these days as the workflows community comes to together to discuss common needs which, in turn, can inform the development of such APIs [8] .

Providing a modern API into NERSC is a central component of the Superfacility project [2] at Lawrence Berkeley National Laboratory (LBNL), which aims to lay the basis for a more unified, seamless environment that combines hardware solutions, application software, and data management tools to deliver breakthrough science. Automation is a key component of the Superfacility concept, which envisions science teams at experiment facilities orchestrating automated data analysis pipelines which move data from the instrument to the computing site, perform analysis, and disseminate results – all without any human in the loop.

The SF API provides RESTful API interfaces to resources and takes inspiration from work at various HPC centers [9], [10] as well as from NERSC's first API, the *NE*RSC *W*eb development *T*oolkit (NEWT) [11]. While NEWT was designed to serve primarily as backend service for web science gateways, the new SF API is more targeted at workflows and provides a modern, token-based authentication mechanisms as well as asynchronous task execution. The SF API service itself is built as a set of Docker[4] containers and runs in Spin[5], NERSC's Containers-as-a-Service platform. By and large, it orchestrates connections to backend systems and databases, asynchronously manages any long-running tasks, handles authentication and authorization, and hosts its own documentation. Currently, the API provides the endpoints described in table II. As the API is in active development, the most up to date documentation can be obtained online at the automatically generated Swagger page.[6]

Enumerating all of the use cases for the API would be too much to cover in this manuscript as NERSC envisions *all* of the common interactions with its systems to become automatable. Instead, we close with describing two use cases, where one

---

[4]https://www.docker.com

[5]https://www.nersc.gov/systems/spin/

[6]Superfacility API documentation generated using the Swagger toolset, available at https://api.nersc.gov/api/v1.2/

TABLE II
API ENDPOINTS.

/meta	information the API installation at the HPC center
/account	retrieve allocation info for a user or project
/utilities	browse, upload, and download files or a free form command
/storage	move data between sites with Globus, or between NERSC storage tiers
/status	retrieve system health status, including planned outages
/compute	submit and manage jobs, check job status
/tasks	information about pending and completed tasks

describes the abstract case of checking system health before a file transfer and the other describes a current application of the SF API in the AutoSFX pipeline of LCLS-II (a similar pipeline as *cctbx.xfel* for serial femtosecond crystallography data analysis).

A. Example: Checking system health before data transfer.

Because the demand for compute capacity is driven by detector output which can vary cyclically, experiments often need HPC-scale computing at short notice. Some experiments may have even arranged for multiple compute sites to be available to handle workloads in a given time period. To build a truly automated and resilient workflow, scientists need to be able to query the health and status of a facility and make decisions based on the response; for example, if a file system is unavailable, the workflow pipeline should choose not to send data to it. To assess the status of a NERSC resource, the API provides the **/status/** endpoint. Keeping with the example of an imminent file transfer, the workflow could query **/status/dtns** and **/status/community_filesystem** in order to find out the health of NERSC's data transfer nodes and the community file system, respectively. A json-formatted return of one of those queries would look like this:

```
{"name": "dtns",
 "full_name": "Data Transfer Nodes",
 "description": "System is active",
 "system_type": "filesystem",
 "notes": [],
 "status": "active",
 "updated_at": "2021-05-21T07:55:00"}
```

A status indicated as "active" would now inform the workflow that the resources is operational and that it could start the data transfer. It could use its own tools for these transfers, but the API also provides the **/storage** endpoint to move data between Globus-enabled[7] sites and between the NERSC storage tiers. For planning further ahead, a query to **/status/outages/planned** would provide any scheduled outage in the future and would enable the workflow manager to choose an alternative destination or date for the transfer.

B. Example: Using the SF API in the LCLS AutoSFX pipeline.

The LCLS data management system invokes the SF API to integrate its automation engine (ARP) with NERSC computing resources. Data management events (start/end runs, file transfers etc) automatically trigger analysis jobs, which are then initiated, monitored and managed at NERSC using **/compute/jobs/cori** calls. Runtime progress bar updates from the jobs, in addition to job statuses from **/compute/jobs/cori**, are then pushed to the browser and dynamically update the web UI. The entire AutoSFX workflow, consisting of multiple index/merge steps, is expressed as an

---

[7]https://globus.org

AirFlow Directed Acyclic Graph (DAG). Each node in the DAG is executed by the ARP by composing **/utilities** and **/compute/jobs/cori** calls (see table II). Summary results (for example, electron density maps) are copied back to the experiment folders using **/utilities/download** calls and displayed in the web UI. As many of these calls target asynchronous endpoints (e.g. **/compute/jobs**) where each POST call generates a task, the workflow frequently queries the **/tasks** to inquire the status of those tasks in order to advance in the DAG.

V. CONCLUSION

In this paper we have demonstrated the power and possibility of using on-demand HPC to analyse data in real time for a running XFEL experiment at LCLS. This will provide a new mode of sustainable operations for high data-rate experiments (over $400\times$ the rate of today's experiments) expected to come online in 2025. To achieve on-demand and real-time feedback for experiment control, we have addressed scaling problems in the application, work scheduling, data management and workflow management. We have identified areas for future development based on a series of carefully-profiled experiments performed in late 2020, which achieved the goal of having the analysis keep up with the experiment operation. Most importantly, the experiments described in this paper were not one-off demonstrations, but the start of a regular mode of joint operations between an experimental user facility and an HPC user facility that is both sustainable and scalable. HPC centers are increasingly being used for this kind of experiment-driven workflow, and the tools and techniques developed in this work were designed to be generalizable to other science areas.

VI. ACKNOWLEDGEMENTS

REFERENCES

[1] N. K. Sauter, J. Kern, J. Yano, and J. M. Holton, "Towards the spatial resolution of metalloprotein charge states by detailed modeling of XFEL crystallographic diffraction," *Acta Crystallographica Section D: Structural Biology*, vol. 76, no. 2, pp. 176–192, 2020. [Online]. Available: https://doi.org/10.1107/S2059798320000418

[2] B. Enders, D. Bard, C. Snavely, L. Gerhardt, J. Lee, B. Totzke, K. Antypas, S. Byna, R. Cheema, S. Cholia, M. Day, A. Gaur, A. Greiner, T. Groves, M. Kiran, Q. Koziol, K. Rowland, C. Samuel, A. Selvarajan, A. Sim, D. Skinner, R. Thomas, and G. Torok, "Cross-facility science with the Superfacility project at LBNL," in *2020 IEEE/ACM 2nd Annual Workshop on Extreme-scale Experiment-in-the-Loop Computing (XLOOP)*, 2020, pp. 1–7. [Online]. Available: https://doi.org/10.1109/XLOOP51963.2020.00006

[3] R. W. Grosse-Kunstleve, N. K. Sauter, N. W. Moriarty, and P. D. Adams, "The Computational Crystallography Toolbox: Crystallographic algorithms in a reusable software framework," *Journal of Applied Crystallography*, vol. 35, no. 1, pp. 126–136, 2002.

[4] N. K. Sauter, J. Hattne, R. W. Grosse-Kunstleve, and N. Echols, "New Python-based methods for data processing," *Acta Crystallographica Section D*, vol. 69, no. 7, pp. 1274–1282, Jul 2013. [Online]. Available: https://doi.org/10.1107/S0907444913000863

[5] D. Damiani, M. Dubrovin, I. Gaponenko, W. Kroeger, T. Lane, A. Mitra, C. O'Grady, A. Salnikov, A. Sanchez-Gonzalez, D. Schneider, and C. H. Yoon, "Linac coherent light source data analysis using psana," *Journal of Applied Crystallography*, vol. 49, 04 2016. [Online]. Available: https://doi.org/10.1107/S1600576716004349

[6] I. Monga, C. Guok, J. MacAuley, A. Sim, H. Newman, J. Balcas, P. DeMar, L. Winkler, T. Lehman, and X. Yang, "Sdn for end-to-end networked science at the exascale (sense)," in *2018 IEEE/ACM Innovating the Network for Data-Intensive Science (INDIS)*, 2018, pp. 33–44. [Online]. Available: https://doi.org/10.1109/INDIS.2018.00007

[7] A. Bhatele, S. Brink, and T. Gamblin, "Hatchet: Pruning the overgrowth in parallel profiles," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '19. New York, NY, USA: Association for Computing Machinery, 2019. [Online]. Available: https://doi.org/10.1145/3295500.3356219

[8] R. F. da Silva et. al., "Workflows Community Summit: Advancing the State- of-the-art of Scientific Workflows Management Systems Research and Development," Jun. 2021. [Online]. Available: https://doi.org/10.5281/zenodo.4915801

[9] R. Dooley, S. R. Brandt, and J. Fonner, "The Agave platform: An open, science-as-a-service platform for digital science," in *Proceedings of the Practice and Experience on Advanced Research Computing*, ser. PEARC '18. New York, NY, USA: Association for Computing Machinery, 2018. [Online]. Available: https://doi.org/10.1145/3219104.3219129

[10] F. A. Cruz, A. J. Dabin, J. P. Dorsch, E. Koutsaniti, N. F. Lezcano, M. Martinasso, and D. Petrusic, "FirecREST: a RESTful API to HPC systems," in *2020 IEEE/ACM International Workshop on Interoperability of Supercomputing and Cloud Technologies (SuperCompCloud)*, 2020, pp. 21–26. [Online]. Available: https://doi.org/10.1109/SuperCompCloud51944.2020.00009

[11] S. Cholia, D. Skinner, and J. Boverhof, "Newt: A restful service for building high performance computing web applications," in *2010 Gateway Computing Environments Workshop (GCE)*, 2010, pp. 1–11. [Online]. Available: https://doi.org/10.1109/GCE.2010.5676125