

Deploying *cctbx.xfel* in Cloud Computing and High Performance Computing Environments

Billy K. Poon, Aaron S. Brewster and Nicholas K. Sauter

Molecular Biophysics and Integrated Bioimaging Division, Lawrence Berkeley National Laboratory, Berkeley, CA 94720, USA

Introduction:

The amount of data being generated by protein X-ray crystallography beamlines at photon light sources is growing dramatically, with data sets at the newest X-ray free electron laser (XFEL) facilities currently requiring tens to over a hundred terabytes (TB) for a single serial femtosecond nanocrystallography experiment. This is due to advances in the underlying accelerator technology that can generate more photons more quickly [1], as well as improvements in detectors that allow for more rapid data collection [2]. However, storage technology is not projected to increase at a comparable rate [3]. As a result, in addition to experiments conducted at XFEL facilities, average photon light source users performing crystallography experiments will begin to encounter “Big Data” problems with storing and processing their data in the near future as raw dataset sizes begin to outstrip what users can reasonably purchase and manage on personal computers and small computing clusters.

For traditional protein crystallography, the size of the raw data set is defined by the number of pixels in the detector as well as how many images were collected. Generally, each crystal generates a data set that is several hundred megabytes. With increased brightness at synchrotrons and improved detectors, it is possible at some beamlines to collect the data continuously. That is, instead of stopping at each angular increment for 1 second, the

crystal is rotated continuously and data is collected continuously [4]. Furthermore, with improved robotic sample handling systems [5], it is possible for users to quickly mount and collect many crystals during their beamtime. As a result, while data sets for individual crystals have grown modestly in size due to higher detector pixel counts, the number of data sets that can be collected during a user’s scheduled beam time has increased significantly, resulting in much more data that needs to be processed and analyzed.

But the real dramatic growth in data sizes will be from serial femtosecond nanocrystallography experiments conducted at XFEL facilities [6]. The experimental setup is similar to traditional protein crystallography, but the sample is not mounted or rotated. Instead, a steady stream of nanocrystals is shot in the path of the X-ray beam with each nanocrystal that interacts with the beam resulting in a hit. Several million images are collected, but only a fraction of those images are hits with nanocrystals. Even with better hit-detection algorithms that discard non-hits, several hundred thousand images still need to be stored and processed to produce a data set suitable for structure refinement.

To explore the capabilities of cloud computing and high performance computing (HPC) environments to handle these larger and more numerous datasets, the data

processing software package *cctbx.xfel* [7] was configured to run on Amazon Web Services (AWS) and on Edison, a supercomputer at the National Energy Research Scientific Computing Center (NERSC). A 15.5 TB partial dataset collected from the Linac Coherent Light Source (LCLS) was used for benchmarking the performance of indexing and spotfinding in parallel.

Amazon Web Services:

Amazon Web Services (AWS) provides infrastructure as a service (IaaS) by giving users the ability to store data and run customized virtual machines in Amazon's data centers. Amazon's services are well established and Amazon provides numerous case studies about the effectiveness and cost-savings achieved by companies using the cloud instead of having to manage physical servers [8]. Amazon's servers are divided into separate geographic regions and each geographic region is independent of the others. That is, workloads are not automatically distributed across geographic regions by Amazon. For this test, we will be focusing on a subset of Amazon's offerings: Elastic Compute Cloud (EC2) and Simple Storage Service (S3).

The EC2 service [9] is the basic compute resource provided by Amazon. There are multiple hardware configurations available and the number of instances of each configuration can be dynamically changed to meet the computational demand of the problem being solved. The computing capabilities of the hardware configurations are based on more recent Intel Xeon processors where each physical computing core can handle two threads (i.e. hyperthreading) and each thread corresponds to one vCPU unit. The pricing scales with the

amount of hardware resources (number of vCPUs and memory) in the configuration and the charge is usually for every hour an instance is running. The pricing can be reduced by using reserved instances where an upfront cost is paid, but the hourly cost is lower. Another way to reduce costs is to use spot instances where Amazon offers excess capacity at a lower price that fluctuates depending on demand. The downside to using spot instances is that AWS will reclaim the instance if the price goes higher than the maximum price you are willing to pay per hour (i.e. the bid price).

Amazon has configurations specific for computationally intensive workloads. These cluster compute instances have larger amounts of memory; solid state drives (SSD) for local storage; computing power; and 10 Gigabit network speeds. There are also instances that have graphics processing units (GPU) for computation. As of January 2017, the on-demand price per vCPU hour is \$0.0525 for the c3.8xlarge configuration (32 vCPUs), but can be around \$0.011 per vCPU hour if spot pricing is used. For this test, we had no issues with getting 64 c3.8xlarge spot instances within 5 minutes and keeping the spot instances for the duration of the tests with a bid price of \$1.00 per c3.8xlarge instance hour. The limit was self-imposed for controlling cost, not from some limit imposed by AWS.

The S3 service [10] is the basic data storage resource provided by Amazon. Data are stored as web objects instead of files in a traditional filesystem. The user creates a bucket in a geographic region, and data are stored as objects in the bucket. The S3 service will be used for storing the large data sets because objects have a size limit of five

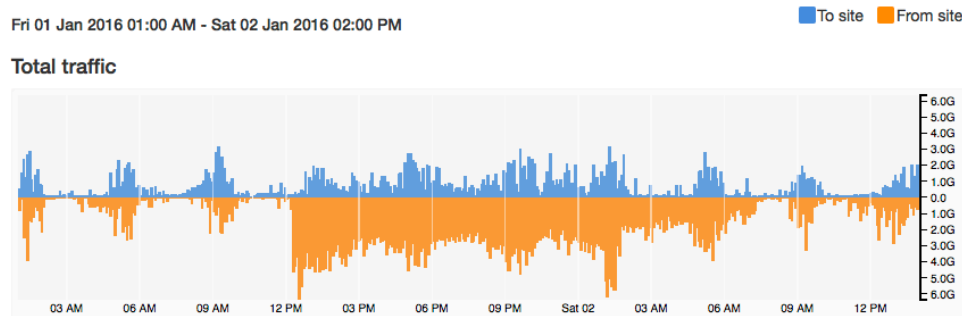


Figure 1: Data transfer rates from LCLS to S3 according to <https://my.es.net>. The transfer started at around 12 pm on January 1, 2016 and ended around 7 am the next day. The orange graph shows the traffic leaving LCLS to S3.

terabytes, but there is no limit to the number of objects that can be stored. The cost is based on the amount of storage used, the amount of data transferred out and the number of requests made for accessing the data. The cost will be dominated by the amount of data stored and, as of January 2017, the price per gigabyte per month is \$0.023. According to Amazon, objects stored in S3 have a durability of 99.999999999% and an availability of 99.99%. The durability is especially important for large data sets because it protects the data against corruption from random events that may flip a bit. During this test, the price per gigabyte per month was \$0.03, so to store all the data in S3, the monthly cost was around \$475. With the more recent pricing, the monthly cost for 15.5 TB drops to around \$364. The results that are stored in S3 only take a few gigabytes, so they do not appreciably change the monthly cost of S3 storage.

The first step for this benchmark is to transfer the 15.5 TB dataset to S3. We chose the us-west-2 region, which is located in Oregon, due to server capacity and the 100 Gbit/s peering between AWS and ESnet [11], the Department of Energy-funded high-speed network used for research. The peering enables a more

direct connection between computers on ESnet (e.g. photon light sources, national laboratories, NERSC) and AWS systems. Figure 1 shows the speed of the transfer over time. Starting at around 12 pm on January 1, 2016, it took approximately 18 hours to transfer 15.5 TB (peak speed ~ 6.0 Gbit/s, average speed ~ 2.0 Gbit/s). The dataset was broken up into 3,303 files of about 5 GB each. The AWS command-line tools were used to copy these files from 2 data transfer nodes (DTN) at LCLS directly into S3. Each DTN has a 10 Gbit/s connection to ESnet. Working with the networking staff at LCLS and a solutions architect from AWS, we determined that the maximum transfer rate per DTN is roughly 3.2 Gbit/s, which matches the peak speed for the data transfer. We also speculated that the shared nature of the LCLS filesystem and DTNs might have lowered the average speed from the maximum.

The second step for this benchmark is to process the data. To do so, we built an Amazon Linux image containing the PSDM software [12] for handling LCLS files and *cctbx.xfel* for analysis. The Open Grid Engine [13] was used for managing the jobs on a cluster of EC2 instances. Table 1 lists the EC2 configurations used for testing. For the

Table 1: EC2 configurations and average costs

Instance Type	CPU	# vCPU	Speed (GHz)	RAM (GB)	SSD (GB)	Instance Cost (\$/h)
c3.8xlarge	E5-2680 v2	32	2.8	60	2 x 320	0.387
c4.8xlarge	E5-2666 v3	36	2.4	60	N/A	0.514
m4.10xlarge	E5-2676 v3	40	2.6	160	N/A	0.426

c4.8xlarge and m4.10xlarge, the Elastic Block Store (EBS) service was used to create 12 GB volumes that can be mounted individually on each node for storing the operating system. As of January 2017, the cost for this service is \$0.10 per GB-month. Since we only use the volume when an instance is running, the cost per node for 12 GB of storage is approximately \$0.00164 per hour. Table 1 also shows the average cost per hour for each instance type based on the actual costs for these tests when using spot instances. For c4.8xlarge and m4.10xlarge, the cost of the EBS volume is included. Using these average costs and the time needed to process 15.5 TB, the EC2 cost of processing the whole dataset is shown in Table 2. However, the cost of S3 is not included in the EC2 costs reported in Table 2 because the entire dataset is held in S3 regardless of any processing. The S3 costs should be tracked separately.

On AWS, the S3 bucket where all the data are stored operates independently from the EC2 instances. So to do the actual processing, we wrapped the normal *cctbx.xfel* analysis with a script that would manage the transfer of data files from S3 onto the EC2 instance and the transfer of results from the EC2 instance back to S3. However, because the analysis is known to be compute-bound, we only have, at most, two data files on each EC2 instance at any given time: the file currently being processed and the next file to be processed (transferred simultaneously with the processing of the

current file). We found through some initial testing that the transfer of one data file (5 GB) from S3 to one of the instance types listed in Table 1 took around 30 s (~ 1.33 Gbit/s). After a node finishes processing its list of files, the results are compressed and transferred back to S3. Because the number of indexable images varies according to which data files are processed, the amount of results varied by node. But this final transfer of results usually took a few minutes and is a small fraction of the total processing time. Lastly, for speed, all the data files and results are cached in RAM.

Figure 2 and Table 2 show the data processing speeds, wall times and EC2 costs for the instance types in Table 1 for different numbers of nodes. It is important to note that the EC2 costs are averaged over a few months and can be higher or lower depending on the current spot market price. Our experience during these tests showed spot pricing that could vary by at most $\pm 25\%$. Also, the cost for each test is relatively independent of the number of nodes because the total number of core hours needed for processing is about the same and the scaling is linear. Some initial processing was performed on all 3,303 files to get a sense of how long each file took to process. The files were distributed in such a way that all the nodes would finish at approximately the same time. As a result, the standard deviations for the average time per node are 1% or smaller.

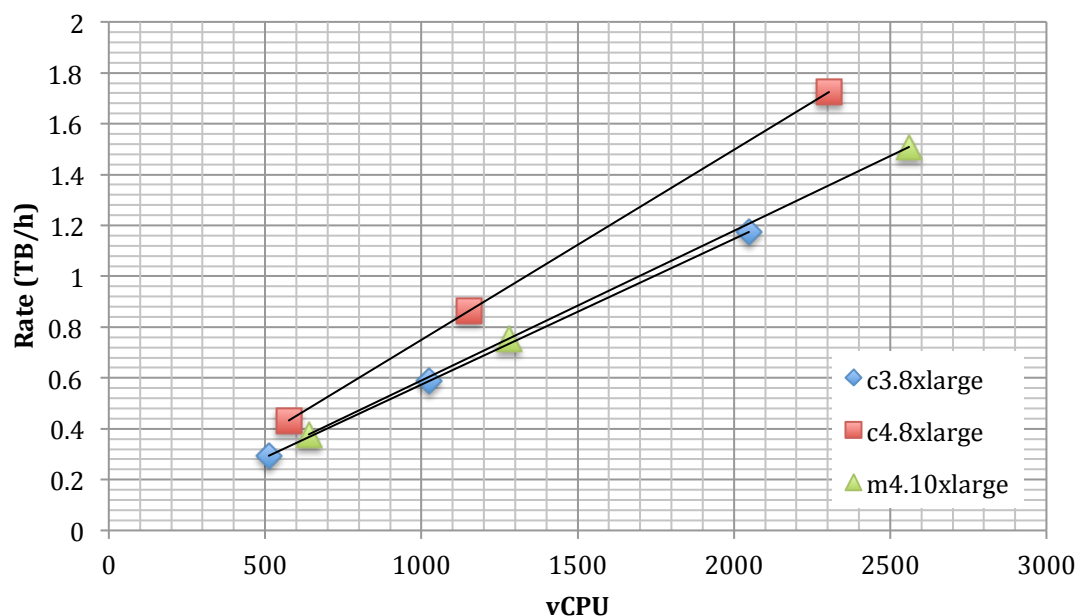


Figure 2: Processing speeds

Edison @ NERSC:

Edison is a Cray XC30 with over 5,500 nodes and 134,000 cores. Edison behaves like a traditional computing cluster with mounted filesystems and a queuing system, but on a much larger scale.

Similar to AWS, the first step is to transfer the dataset to Edison. Using Globus [14], a file transfer management tool, the 3,303 files were transferred from LCLS to the scratch space on Edison in about 10 hours (average

speed ~ 3.7 Gbit/s). Like the AWS transfer, Globus can use multiple DTNs at each endpoint, but Globus hides all of the complexities from the user. Also like the AWS test, the PSDM and *cctbx.xfel* software need to be installed and configured. On Edison, the recommended approach for installing custom software is to use containers. NERSC uses Shifter [15], which is compatible with the widely used Docker container format [16]. Containers are basically virtual machine images packaged into layers so that common

Table 2: Processing times, speeds and EC2 costs for processing 15.5 TB once through.

Instance Type	Nodes	Average Time (s)	σ (s)	Rate (TB/h)	EC2 Cost (\$)
c3.8xlarge	16	189,439	387	0.294	325.76
	32	94,709	242	0.587	325.72
	64	47,390	168	1.17	325.97
c4.8xlarge	16	128,713	482	0.432	278.89
	32	64,399	367	0.864	278.87
	64	32,267	264	1.72	279.24
m4.10xlarge	16	147,376	646	0.378	293.84
	32	73,683	585	0.755	294.04
	64	36,889	397	1.51	294.65

parts (e.g. base operating system) are not replicated for each container. In our case, we use CentOS 6.8 as the base operating system and build PSDM and *cctbx.xfel* on top of that. Schema 3 shows the file used to build the Docker container, which is subsequently transferred to Edison. The actual Docker container can be found in the public Docker repository (hub.docker.com/r/bkpoone/cctbx/tags) under the “aws_test” tag.

Due to time constraints, only one test was performed using 8 nodes (192 physical cores). The 15.5 TB dataset was processed in about two days resulting in a data processing rate of about 0.328 TB/h. The processors in Edison (E5-2695 v2 @ 2.4 GHz) are comparable to the ones in a c3.8xlarge instance (E5-2680 v2 @ 2.8 GHz), but have more physical CPUs per node (12 vs. 8). So even though the closest AWS test used twice as many c3.8xlarge nodes as Edison nodes (16 vs. 8), the AWS test only had about 1/3 more physical cores.

Conclusions and Future Directions:

From these tests, it is possible to effectively run *cctbx.xfel* in cloud computing and HPC environments for production use. On AWS, the performance of independently indexing and spotfinding each diffraction image scales linearly. We speculate that to be true on Edison as well because the application is compute-bound as opposed to I/O-bound and is embarrassingly parallel. That is, the processors are never sitting idle since each core will always have another image queued for analysis. Moreover, we think that the scaling on AWS should extend into multiple thousands of physical cores due to how S3 scales I/O throughput by following the recommended file naming conventions for S3 (i.e. prefixing each file with a random hash) and distributing files across multiple buckets.

On Edison, the same should be true because the scratch filesystem can provide more than 700 GB/s of throughput. This linear scaling is especially crucial for handling the increased detector sizes and data collection rates.

For this particular test dataset, the files were constructed in a way so that they were all approximately 5 GB. This made distributing images to the compute nodes much easier and uniform, especially on AWS. However, this is generally not true for files from LCLS. But with additional software engineering, it would be possible to read blocks of images from any file and distribute the blocks to the compute nodes.

In terms of data transfer, Globus made the process very simple and fast between LCLS and NERSC. Using the AWS command-line tools for transferring to S3 was not as easy and required more expertise for setting up the transfer across 2 DTNs. We did create a Globus endpoint for the S3 bucket and we tried to use Globus to manage the transfer to S3, but the transfer rate was less than 1.0 Gbit/s. Preliminary attempts to improve the transfer rate were inconclusive, however, it is thought that this is an area that can be improved and is important for making cloud computing more accessible to researchers who have large datasets.

For software deployment, the Docker container greatly simplifies the process. During the AWS test, the Amazon EC2 Container Service was still in beta so a full virtual image was built instead. However, if the AWS tests were to be repeated today, we would use the same Docker container as was used on Edison after conversion into the Shifter format. This ability to use the same container for both AWS and NERSC makes

```

FROM centos:6.8
MAINTAINER "Billy Poon" bkpoon@lbl.gov
ENV container docker

# arguments
ARG NCPU=4
# software versions
ENV PSDM_VER 0.15.5
ENV CCTBX_SRC xfel_20151230.tar.xz

# upgrade OS and install base packages for psdm
# https://confluence.slac.stanford.edu/display/PSDM/System+packages+for+rhel6
RUN yum update -y && \
    yum -y install alsa-lib atk compat-libf2c-34 fontconfig freetype gsl \
    libgfortran libgomp libjpeg libpng libpng-devel pango postgresql-libs \
    unixODBC libICE libSM libX11 libXext libXft libXinerama libXpm \
    libXrender libXtst libXxf86vm mesa-libGL mesa-libGLU gtk2 \
    xorg-x11-fonts-Type1 xorg-x11-fonts-base xorg-x11-fonts-100dpi \
    xorg-x11-fonts-truetype xorg-x11-fonts-75dpi xorg-x11-fonts-misc \
    tar xz which gcc gcc-c++ mysql libibverbs openssh-server openssh \
    gcc-gfortran

# install psdm
# https://confluence.slac.stanford.edu/display/PSDM/Software+Distribution
ADD http://pswww.slac.stanford.edu/psdm-repo/dist_scripts/site-setup.sh \
    /reg/g/psdm/
RUN sh /reg/g/psdm/site-setup.sh /reg/g/psdm
ENV SIT_ROOT=/reg/g/psdm
ENV PATH=/reg/g/psdm/sw/dist/apt-rpm/rhel6-x86_64/bin:$PATH
ENV APT_CONFIG=/reg/g/psdm/sw/dist/apt-rpm/rhel6-x86_64/etc/apt/apt.conf
RUN apt-get -y update && \
    apt-get -y install psdm-release-ana-${PSDM_VER}-x86_64-rhel6-gcc44-opt && \
    ln -s /reg/g/psdm/sw/releases/ana-${PSDM_VER} \
        /reg/g/psdm/sw/releases/ana-current

# use old HDF5 (1.8.6) for compatibility with cctbx.xfel
ADD https://www.hdfgroup.org/ftp/HDF5/releases/hdf5-1.8.6/bin/linux-x86_64/hdf5-1.8.6-
linux-x86_64-shared.tar.gz .
#COPY ./hdf5-1.8.6-linux-x86_64-shared.tar.gz .
RUN tar -xf hdf5-1.8.6-linux-x86_64-shared.tar.gz && \
    mkdir -p /reg/g/psdm/sw/external/hdf5/1.8.6 && \
    mv hdf5-1.8.6-linux-x86_64-shared \
        /reg/g/psdm/sw/external/hdf5/1.8.6/x86_64-rhel6-gcc44-opt

# =====
# Install mpich for NERSC
# https://github.com/NERSC/shifter/blob/master/doc/mpi/mpich_abi.rst
WORKDIR /usr/local/src
ADD http://www.mpich.org/static/downloads/3.2/mpich-3.2.tar.gz /usr/local/src/
RUN tar xf mpich-3.2.tar.gz && \
    cd mpich-3.2 && \
    ./configure && \
    make -j ${NCPU} && make install && \
    cd /usr/local/src && \
    rm -rf mpich-3.2

# build myrelease
WORKDIR /reg/g
RUN source /reg/g/psdm/etc/ana_env.sh && \
    newrel ana-${PSDM_VER} myrelease && \
    cd myrelease && \
    source sit_setup.sh && \
    newpkg my_ana_pkg

```

Schema 3: Dockerfile for building container for Edison. The most recent version can be found at https://github.com/bkpoon/cctbx_docker

```

# copy cctbx.xfel from local tarball
RUN mkdir -p /reg/g/cctbx
WORKDIR /reg/g/cctbx
COPY ./${CCTBX_SRC} /reg/g/cctbx/${CCTBX_SRC}
RUN tar -Jxf ./${CCTBX_SRC}

# build cctbx.xfel
# make needs to be run multiple times to ensure complete build (bug)
ENV CPATH=/reg/g/psdm/sw/releases/ana-${PSDM_VER}/arch/x86_64-rhel6-gcc44-opt/geninc
#:/reg/g/psdm/sw/releases/ana-${PSDM_VER}/arch/x86_64-rhel6-gcc44-opt/geninc/hdf5
ENV LD_LIBRARY_PATH=/reg/g/psdm/sw/releases/ana-${PSDM_VER}/arch/x86_64-rhel6-gcc44-opt/lib
RUN source /reg/g/psdm/etc/ana_env.sh &&\
  cd /reg/g/myrelease &&\
  sit_setup.sh &&\
  cd /reg/g/cctbx &&\
  python ./modules/cctbx_project/libtbx/auto_build/bootstrap.py build \
  --builder=xfel --with-python=`which python` --nproc=${NCPU} &&\
  cd build &&\
  make -j ${NCPU} &&\
  make -j ${NCPU}

# finish building myrelease
RUN source /reg/g/psdm/etc/ana_env.sh &&\
  cd /reg/g/myrelease &&\
  source /reg/g/psdm/bin/sit_setup.sh &&\
  source /reg/g/cctbx/build/setpaths.sh &&\
  cd my_ana_pkg &&\
  ln -s /reg/g/cctbx/modules/cctbx_project/xfel/cxi/cspad_ana src &&\
  cd .. &&\
  scons

# recreate /reg/d directories for data
RUN mkdir -p /reg/d/psdm/cxi &&\
  mkdir -p /reg/d/psdm/CXI

```

Schema 3: Dockerfile for building container for Edison. The most recent version can be found at https://github.com/bkpoon/cctbx_docker (continued)

running the software on a large-scale much easier. Moreover, Docker containers can run on macOS, Windows and Linux, so one software installation in a container can behave consistently on multiple platforms at multiple scales.

An additional benefit of containers is that the software becomes independent of the host operating system. This can avoid installation issues where the host operating system might not have the required libraries for running the software. One recent example where this is true involves the structure refinement program Phenix and its interaction with the modeling program Rosetta. The newest

versions of Rosetta require that the compiler supports C++11, but Phenix does not. So in order to run phenix.rosetta_refine, users would need to upgrade their compiler or operating system (to get a newer compiler version) if C++11 support is not available. A container solves this issue by letting the user choose the base operating system that has the required features. In this test, we chose CentOS 6.8 because it is a supported operating system for PSDM. But the version of gcc that comes with CentOS 6.8 does not support C++11. If we were to build a container for Phenix and Rosetta, we could easily modify the Dockerfile to use the latest version CentOS

7, which does have C++11 support. And after the container is built, we could run Phenix and Rosetta on any operating system regardless of C++11 support, as long as it can run Docker. This can be an important feature for users who run their jobs on production servers maintained by other people or who cannot easily upgrade their operating systems.

Acknowledgements:

The authors would like to thank AWS (Kevin Jorissen, Dave Cuthbert, Jamie Baker, Cindy Hewitt, Jamie Kinney) for providing the funds (EDU_R_FY2015_Q2_LawrenceBerkeleyNationalLaboratory_Poon) for running these large-scale jobs. We would like to give a special thanks to Dave Cuthbert for helping with troubleshooting various issues on AWS.

We would also like to thank Duilio Cascio for providing the test dataset and the following

people grouped by their organizations and key area of assistance:

- LCLS (troubleshooting) - Amedeo Perazzo, Chris O'Grady, Igor Gaponenko and Antonio Ceseracciu
- LBNL (Globus endpoint on S3) - Craig Tull, Shreyas Cholia, Krishna Muriki
- NERSC (Shifter container) - David Skinner, Deborah Bard, Joaquin Correa, Doug Jacobsen
- ESnet (data transfer) - Eli Dart, Mike Sinatra
- Globus (data transfer) - Rachana Ananthakrishnan

N.K.S. acknowledges National Institutes of Health grant GM117126 for data processing methods. Use of the LCLS at SLAC National Accelerator Laboratory, is supported by the US Department of Energy, Office of Science and Office of Basic Energy Sciences under contract no. DE-AC02-76SF00515. This research used resources of NERSC, a User Facility supported by the Office of Science, DOE, under Contract No. DE-AC02-05CH11231.

References:

1. Emma, P, Akre, R, Arthur, J, et al. First lasing and operation of an Ångström-wavelength free-electron laser. *Nature Photonics*, 2010; 4: 641-647.
2. Strüder, L, Epp, S, Rolles, D, et al. Large-format, high-speed, X-ray pnCCDs combined with electron and ion imaging spectrometers in a multipurpose chamber for experiments at 4th generation light sources. *Nucl. Instrum. Methods Phys. Res. A*, 2010; 614: 483-496.
3. National Energy Research Scientific Computing Center (NERSC) Strategic Plan for FY2014-2023, 2013. (http://www.nersc.gov/assets/pubs_presos/NERSCplan-FY2014-2023.pdf)
4. Hasegawa, K, Hirata, K, Shimizu, T, et al. Development of a shutterless continuous rotation method using an X-ray CMOS detector for protein crystallography. *J. Appl. Cryst.*, 2009; 42: 1165-1175.
5. Snell, G, Cork, C, Nordmeyer, R, et al. Automated sample mounting and alignment system for biological crystallography at a synchrotron source. *Structure*, 2004; 12(4): 537-545.
6. Boutet, S, Lomb, L, Williams, GJ, et al. High-resolution protein structure determination by serial femtosecond crystallography. *Science*, 2012; 337: 362-364.
7. Hattne, J, Echols, N, Tran, R, et al. Accurate macromolecular structures using minimal measurements from X-ray free-electron lasers. *Nature Methods*, 2014; 11: 545-548
8. AWS Case Studies (<http://aws.amazon.com/solutions/case-studies/>)
9. AWS EC2 (<http://aws.amazon.com/ec2/>)
10. AWS S3 (<http://aws.amazon.com/s3/>)
11. ESnet (<http://es.net/>)

12. PSDM Software Distribution
(<https://confluence.slac.stanford.edu/display/PSDM/Software+Distribution>)
13. Open Grid Engine (<http://gridscheduler.sourceforge.net>)
14. Globus (<https://www.globus.org>)
15. Shifter (<https://www.nersc.gov/research-and-development/user-defined-images/>)
16. Docker (<https://www.docker.com>)