A mathematical programming- and simulation-based framework to evaluate cyberinfrastructure design choices

Zhengchun Liu*, Rajkumar Kettimuthu*, Sven Leyffer*, Prashant Palkar†, and Ian Foster*‡
* Mathematics and Computer Science Division, Argonne National Laboratory, 9700 Cass Ave., Lemont, IL 60439, USA
{zhengchun.liu, kettimut, leyffer, foster}@anl.gov
† Industrial Engineering and Operations Research, Indian Institute of Technology Bombay, Mumbai, MH, India 400076
prashant.palkar@iitb.ac.in
‡ Department of Computer Science, University of Chicago, Chicago, IL 60637, USA

***Abstract*—Modern scientific experimental facilities such as x-ray light sources increasingly require on-demand access to large-scale computing for data analysis, for example to detect experimental errors or to select the next experiment. As the number of such facilities, the number of instruments at each facility, and the scale of computational demands all grow, the question arises as to how to meet these demands most efficiently and cost-effectively. A single computer per instrument is unlikely to be cost-effective because of low utilization and high operating costs. A single national compute facility, on the other hand, introduces a single point of failure and perhaps excessive communication costs. We introduce here methods for evaluating these and other potential design points, such as per-facility computer systems and a distributed multisite "superfacility." We use the U.S. Department of Energy light sources as a use case and build a mixed-integer programming model and a customizable superfacility simulator to enable joint optimization of design choices and associated operational decisions. The methodology and tools provide new insights into design choices for on-demand computing facilities for real-time analysis of scientific experiment data. The simulator can also be used to support facility operations, for example by simulating the impact of events such as outages.**

I. Introduction

New experimental modalities can generate extremely large quantities of data. The ability to make effective use of these new capabilities depends on the linking of data generation with computation, for example to filter out important results, identify experimental misconfigurations, or select the next experiment. X-ray light sources—crucial tools for addressing grand challenge problems in the life sciences, energy, climate change, and information technology [1–3]—illustrate these challenges well. A typical light source facility hosts dozens of beamlines, each with multiple instruments. Scientists who run experiments at beamlines want to collect the maximum information possible about the sample under study in as little time as possible. The ability to analyze data produced by detectors in near-real time can enable optimized data collection schemes, on-the-fly adjustments to experimental parameters, early detection of and response to errors (saving both beam time and scientist time), and ultimately improved productivity [4–10].

At experimental science facilities such as Argonne's Advanced Photon Source (APS), data collected at beamline experiments are typically processed either locally on workstations or at a central facility cluster—portions of which may be dedicated to specific beamlines. Inevitably, demand often outstrips supply, slowing analysis and/or discouraging the use of advanced computational methods. In this study, our goal is to ensure that compute resources are available on-demand to provide rapid turnaround time, which is crucial to ensure that data collected are useful and usable. With current state-of-the-art detectors, many beamlines already struggle to derive useful feedback rapidly with the locally available dedicated compute resources. Data rates are projected to increase considerably at many facilities in the next few years. Therefore, the ability to use high-performance computing (HPC) will be critical for these facilities.

Real-time analysis is challenging because of the vast amounts of generated data that must be analyzed in short amounts of time. Detector technology is improving at rates far greater than Moore's law, and modern detectors can generate data at multiple gigabytes per second. The computational power required to extract useful information from these streaming datasets in *real time* almost always exceeds the resources available at a beamline. Thus the use of remote computing facilities for analysis is no longer a luxury but a necessity.

The use of remote computing facilities for real-time analysis requires effective and reliable methods for on-demand acquisition of compute and network resources, efficient data streaming from beamline to compute facility, and analysis at data generation rates so that timely decisions can be taken. These tasks are complicated by the potential for competing load from different experiments. Researchers have performed a variety of preliminary studies on these problems, including methods for online analysis of data from microtomography [4, 5], high energy diffraction microscopy [6, 7], diffuse scattering [8], continuous motion scan ptychography [9], and linking of ultrafast three-dimensional x-ray imaging and continuum simulation [10]. These studies illustrate that users of light source

facilities can understand their samples sufficiently during beam time experiments to adjust the experiment in order to increase scientific output—if enough CPU and network resources can be marshaled on demand. But it is far from clear how one should design a cyberinfrastructure that would allow these science workflows to marshal enough CPU and network resources on demand.

In this work, we develop a framework to evaluate design choices for such a cyberinfrastructure. Our framework consists of a mathematical model, an optimization engine, and a discrete event simulator. It allows the users to identify optimal design choices for a variety of objective functions and constraints (using the mathematical optimization for a smaller problem space) and to evaluate how well they will work in practice (using simulation for a larger problem space). The objective function could be a combination of constraints (e.g., maximum processing delay, minimum resource utilization) and scientific output (e.g., in terms of timeliness of data processing), system reliability, and system cost. Our framework can be used to explore interesting tradeoffs.

II. Motivation

Various groups have demonstrated the connection of experimental facilities with remote supercomputers using high-speed network connections such as ESnet to process data from experimental facilities in near-real time [5, 7, 8, 11]. These demonstrations typically involved special arrangements with compute facility and network operators to obtain the compute and network resources on demand. But such special arrangements are not efficient or scalable. Moreover, the batch queue systems at supercomputers cannot meet the requirement of these experimental science workflows, which often require compute resources within minutes or even seconds of data becoming available.

New approaches are needed to provide the compute resources required by experimental facilities. For example, we may build resources dedicated for a class of experimental science workflows (e.g., light source workflows) and schedule experiments to maximize their utilization. Or, we may add resources to the existing supercomputers, change scheduling policies to meet the needs of experimental science workflows, and share resources with batch jobs. Which approach is better? In either case, where do we place the new resources? Should we put all resources in one place or distribute them across the different sites? If distributed, how do we distribute them? How much additional network capacity is needed? Other approaches are also possible: for example, commercial clouds leverage the law of large numbers and some degree of overprovisioning to enable immediate scheduling of at least modest-sized tasks.

Our goal is to develop a framework that will support a wide range of analytical and simulation studies designed to understand the pros and cons of these different approaches and the inevitable tradeoffs between cost, performance, and other factors.

III. The framework

When designing a cyberinfrastructure that caters to the on-demand requirements of one or more science communities or projects, several factors should be considered. One key factor is flexibility. For example, an experiment schedule that has some flexibility can be adjusted to minimize the peak demand. Thus, our framework has a demand optimization component that uses the flexibility in the demand to optimize the demand. Finding an optimal design that meets the user requirements with the least budget would be ideal. Theoretically, one could use mathematical programming for this purpose, but this is an NP-complete problem. Mathematical programming also has other restrictions such as a simplified data transfer model and ideal scheduling based on a global vision of jobs. Thus, we include in our framework both mathematical programming and a discrete event simulator, so that mathematical programming can be applied on a subset of the problem and the top few most-optimal solutions can be evaluated on the original (bigger) problem by using simulation (that is closer to reality) to identify a design that is most promising for a real-world environment. Figure 1 shows the architecture of our framework.

A. Mathematical problem description

We are interested in formulating a mixed-integer programming (MIP) model that allows us to schedule jobs from scientific facilities at HPC resources, taking into account the network capacities and the resource capacities while ensuring that slowdown for a given percentile of the jobs is below a certain threshold. The slowdown for a job is defined as the ratio of the time length between the start and finish of a job and the run time of the job.

1) Definition of Sets: We use calligraphic letters to describe the sets in our mathematical model.

Sites, $\mathcal{S}$, is the set of scientific facilities/sites that create compute jobs.

Resources, $\mathcal{R}$, is the set of HPC resources where the jobs are processed.

JobIds, $\mathcal{I}$, is a set of unique job identifiers (we use integers).

Jobs, $\mathcal{J}$, is the set of jobs, where $\mathcal{J} \subseteq \mathcal{I} \times \mathcal{S}$.

Time, $\mathcal{T}$, is the set of (discretized) time intervals (again, integers for us).

Nodes, $\mathcal{N}$, is the set of network nodes through which traffic is routed.

Links, $\mathcal{L}$, is the set of links in the network, see Figure 9.

Paths, $\mathcal{P}(s, r)$, is the set of (some) paths from site $s \in \mathcal{S}$ to resource $r \in \mathcal{R}$. Each path is described by a set of links that connect s to r.

2) Fixed Parameters and Constants: We use the following parameters in the model. All are defined by using capital letters.

CreateTime, $T^0_{js} \in \mathcal{T}$, is the time when job $(j, s) \in \mathcal{J}$ is created.

NumNodes, N_{js}, is the number of nodes required to process a job $(j, s) \in \mathcal{J}$.

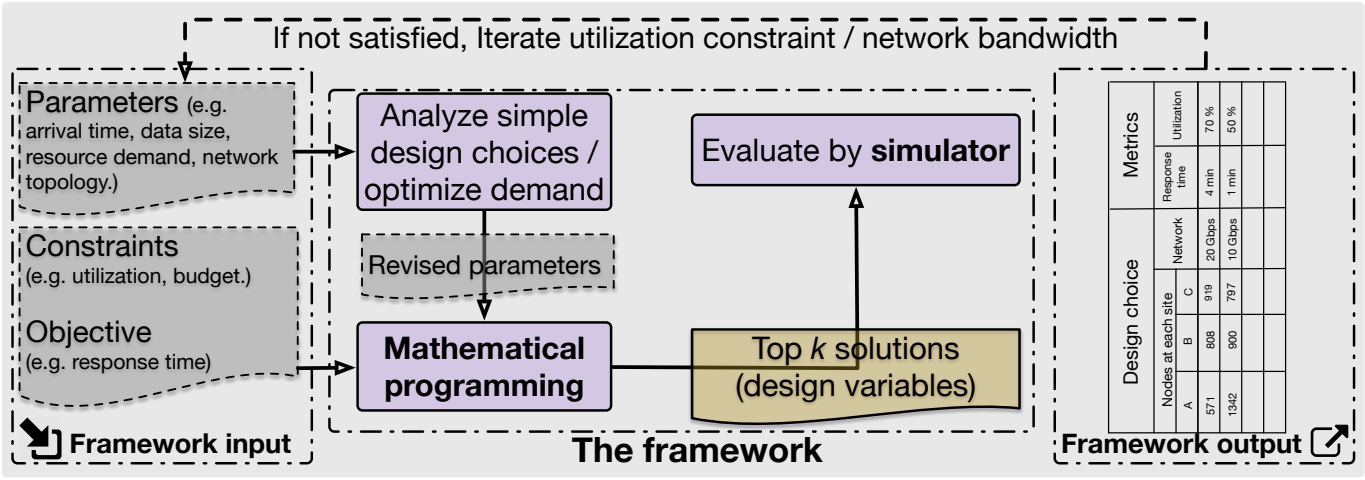

Fig. 1: A framework to evaluate cyberinfrastructure design choices.

RunTime, T^R_{js}, is the run time of job $(j,s) \in \mathcal{J}$.
DataSize, D_{js}, is the data size of job $(j,s) \in \mathcal{J}$.
Capacity, C_{kl}, is the maximum capacity of the link $(k,l) \in \mathcal{L}$.
ComputeBudget, B, is the maximum compute budget, measured in number of nodes.
SlowDown, S, is the maximum slowdown for a predefined percentile of jobs.
Percentile, P, is the percentile of jobs that must have slowdown within S.
BigM, M, is a sufficiently large constant used for modeling purposes, defined as follows.

$$M = \max_{(j,s)\in\mathcal{J}} \frac{|\mathcal{T}| - T^0_{js}}{T^R_{js}}$$

3) Independent Problem Variables: We are interested in both sizing the resources and allocating jobs to the resources. For better readability, we define the sets $\mathcal{T}_{js} = \{(T^0_{js}+1)\dots(|\mathcal{T}| - T^R_{js})\}$ and $\bar{\mathcal{T}}_{js} = \{(T^0_{js})\dots(|\mathcal{T}| - T^R_{js}-1)\}$, in other words, the set of time periods during which job (j,s) could run and the data corresponding to it could arrive, respectively. Next, we introduce the following sets of variables.

$c_r \geq 0$, an integer, is the maximum design capacity of resource $r \in \mathcal{R}$.
$x_{jsrt} \in \{0,1\}$, a binary variable, is 1 iff job $(j,s) \in \mathcal{J}$ is sent to resource $r \in \mathcal{R}$ at time $t \in \mathcal{T}_{js}$
$\lambda_{jsrpt} : 0 \leq \lambda_{jsrpt} \leq 1$ is the fraction of the data of job $j \in \mathcal{I}$ that is sent from site $s \in \mathcal{S}$ to resource $r \in \mathcal{R}$ along path $p \in \mathcal{P}(s,r)$ at time $t \in \bar{\mathcal{T}}_{js}$.
$b_{kl} : 0 \leq b_{kl} \leq C_{kl}$ is the capacity of the link $(k,l) \in \mathcal{L}$.
$z_{jsrp} \in \{0,1\}$, a binary variable, is 1 iff job $(j,s) \in \mathcal{J}$ is sent from site $s \in \mathcal{S}$ to resource $r \in \mathcal{R}$ along path $p \in \mathcal{P}(s,r)$.
$w_{js} \in \{0,1\}$, a binary variable, is 1 iff job $(j,s) \in \mathcal{J}$ has a slowdown $\leq S$.

4) Objective Function: The objective is to minimize the aggregate slowdown in scheduling of jobs.

$$\underset{c,b,x,\lambda,z,w}{\text{minimize}} \sum_{(j,s)\in\mathcal{J}} \frac{(\sum_{r\in\mathcal{R}} \sum_{t\in\mathcal{T}_{js}} t \cdot x_{jsrt}) - T^0_{js} + T^R_{js}}{T^R_{js}} \tag{III.1}$$

5) Constraints: The constraints fall into three sets: job-scheduling constraints, network constraints, and slowdown constraints.

a) Job-Scheduling Constraints: We start by describing the constraints that model the scheduling of the jobs. The first constraint states that every job is scheduled.

$$\sum_{r\in\mathcal{R}, t\in\mathcal{T}_{js}} x_{jsrt} = 1, \quad \forall (j,s) \in \mathcal{J} \tag{III.2}$$

Next, we model the compute capacity, c_r, for every resource and at all time instances.

$$\sum_{(j,s)\in\mathcal{J}} N_{js} \sum_{\tau\in\mathcal{T}_{js}: t-T^R_{js}+1\leq\tau\leq t} x_{jsr\tau} \leq c_r, \quad \forall r \in \mathcal{R}, \forall t \in \mathcal{T} \tag{III.3}$$

The following constraint limits the total compute budget.

$$\sum_{r\in\mathcal{R}} c_r \leq B \tag{III.4}$$

b) Network Constraints: The second set of constraints models the fact that data cannot arrive at the compute resource, $r \in \mathcal{R}$, after the job $(j,s) \in \mathcal{J}$ has started.

$$\sum_{p\in\mathcal{P}(s,r)} \lambda_{jsrpt} \leq \sum_{\tau\in\mathcal{T}_{js}:\tau\geq(t+1)} x_{jsr\tau}, \quad \forall (j,s) \in \mathcal{J},\ \forall r \in \mathcal{R},\ \forall t \in \bar{\mathcal{T}}_{js} \tag{III.5}$$

Next, we write the capacity constraint for all links.

$$\sum_{(j,s)\in\mathcal{J}} \sum_{r\in\mathcal{R}} \sum_{\substack{p\in\mathcal{P}(s,r):(k,l)\in p,\\ t\in\bar{\mathcal{T}}_{js}}} D_{js}\lambda_{jsrpt} \leq b_{k,l}, \quad \forall (k,l) \in \mathcal{L},\ \forall t \in \mathcal{T}. \tag{III.6}$$

The next two constraints model the fact that data can be transmitted only on a unique path and only if it is processed at the target resource. First, we select a path.

$$\sum_{p \in \mathcal{P}(s,r)} z_{jsrp} = \sum_{t \in \mathcal{T}_{js}} x_{jsrt}, \quad \forall (j,s) \in \mathcal{J},\ \forall r \in \mathcal{R} \quad \text{(III.7)}$$

Then, we constrain the transmission along links that are not present in the selected path.

$$\sum_{t \in \bar{\mathcal{T}}_{js}} \lambda_{jsrpt} = z_{jsrp}, \quad \forall (j,s) \in \mathcal{J},\ \forall r \in \mathcal{R},\ \forall p \in \mathcal{P}(s,r) \quad \text{(III.8)}$$

c) Slowdown Constraints: The third set of constraints models that at least a fixed percentile, P, of jobs must be within the slowdown limit, S.

$$\sum_{(j,s) \in \mathcal{J}} w_{j,s} \geq \lceil P|\mathcal{J}| \rceil \quad \text{(III.9)}$$

The following constraint enforces the condition that w_{js} is 1 for a job $(j,s) \in \mathcal{J}$ if and only if the slowdown corresponding to this job is within S.

$$\sum_{r \in \mathcal{R}, t \in \mathcal{T}_{js}} t \cdot x_{jsrt} - T^0_{js} \leq T^R_{js}(S w_{js} + M(1 - w_{js})), \quad \forall (j,s) \in \mathcal{J} \quad \text{(III.10)}$$

d) Full Model: The complete mixed-integer optimization problem is then given as

$\underset{c,x,y,\lambda}{\text{minimize}}$	(III.1)	objective
subject to	(III.2) – (III.4)	job scheduling constraints
	(III.5) – (III.8)	network constraints
	(III.9) – (III.10)	delay constraints

We used AMPL [12, 13] to implement this mixed-integer model. The model is not specific to any particular superfacility: parameters such as network topology, connectivity, HPC sites, and scientific facilities are customizable through a configuration file. The full code is open source and available at https://github.com/ramsesproject/superfacility-mip.

B. A discrete event simulation

Finding an optimal solution by using mathematical programming for large problem sizes may not be feasible. Therefore, our framework uses mathematical programming on one or more representative subsets of the problem and uses simulation to evaluate the solutions on large problem.

A discrete event simulation (DES) models the operation of a system as a discrete sequence of events in time. Each event occurs at a particular instant in time and marks a change of state in the system [14]. Between consecutive events, no change in the system is assumed to occur; the simulation can directly jump in time from one event to the next. The DES approach thus contrasts with continuous simulation in which the simulation continuously tracks the system dynamics over time. DES is called an activity-based simulation; time is divided into small time slices, and the system state is updated according to the activities that occur in each time slice. This type of simulation is ideal for modeling the execution process of workflow over distributed infrastructure.

We have built a discrete event simulator that encompasses experiments at light source facilities, data transfers from experiment facilities to HPC sites, and job scheduling and execution on HPC sites. It focuses on the process of the distributed workflow. The simulator is implemented in the Python programming language with the ns-3 DES engine [15]. The code and a tutorial are available at https://github.com/ramsesproject/superfacility.

1) Distributed workflow: Figure 2 shows the flow of simulation events. All actions are driven by events through callback functions. Compared with a continuous implementation that updates state variables at each time step, this DES implementation runs much faster because it does not have to simulate every time step.

At each beamline of each light source facility, a job is characterized by its arrival time (t_{ar}), experiment setup time (T_s), data acquisition time (T_{daq}), raw dataset size (S_{raw}), number of computing nodes required for analysis (N_{node}), data analysis time (T_{run}), and size of the results (S_{res}). In simulation, we obtain these job characteristics by sampling from the statistical data that we collect from each light source facility. More specifically, in the beamline simulation module (on the top of Figure 2), when a previous experiment is done or at the beginning of simulation, a new job is initialized based on data we collect at each facility if the current time is in the operation time period (as shown in Figure 5). Then, an event associated with a callback function *setupFinishCB()* is scheduled to happen after T_s seconds; that function updates state variables and schedules another event associated with callback function *daqFinishCB()* to finish the data acquisition (i.e., right after T_{daq} seconds). The callback function *daqFinishCB()* either triggers a new experiment if the current time is in the operating period or schedules an event associated with a callback function to trigger a new experiment. Thus, each beamline of the light source facility keeps generating jobs during its operating period (shown in Figure 5).

The job will be sent to the global scheduler once its data are ready to be analyzed. Then the global scheduler will choose an HPC site for the job based on network and computing resource availability. We show in Figure 3 the job scheduling and network bandwidth reservation and sharing algorithm.

2) Scheduling: The scheduler or resource manager has a global view of resource availability and is responsible for finding the best resource for each job. Scheduler policies have a great deal of influence on job slowdown and resource utilization. Currently we have implemented only one scheduling policy; other policies could be added, allowing the use of the simulator to study the effectiveness of different scheduling policies before deployment.

Our current scheduling policy (also shown in Figure 3) assumes that there is a global scheduler that knows the availability of the wide-area network and each HPC computing

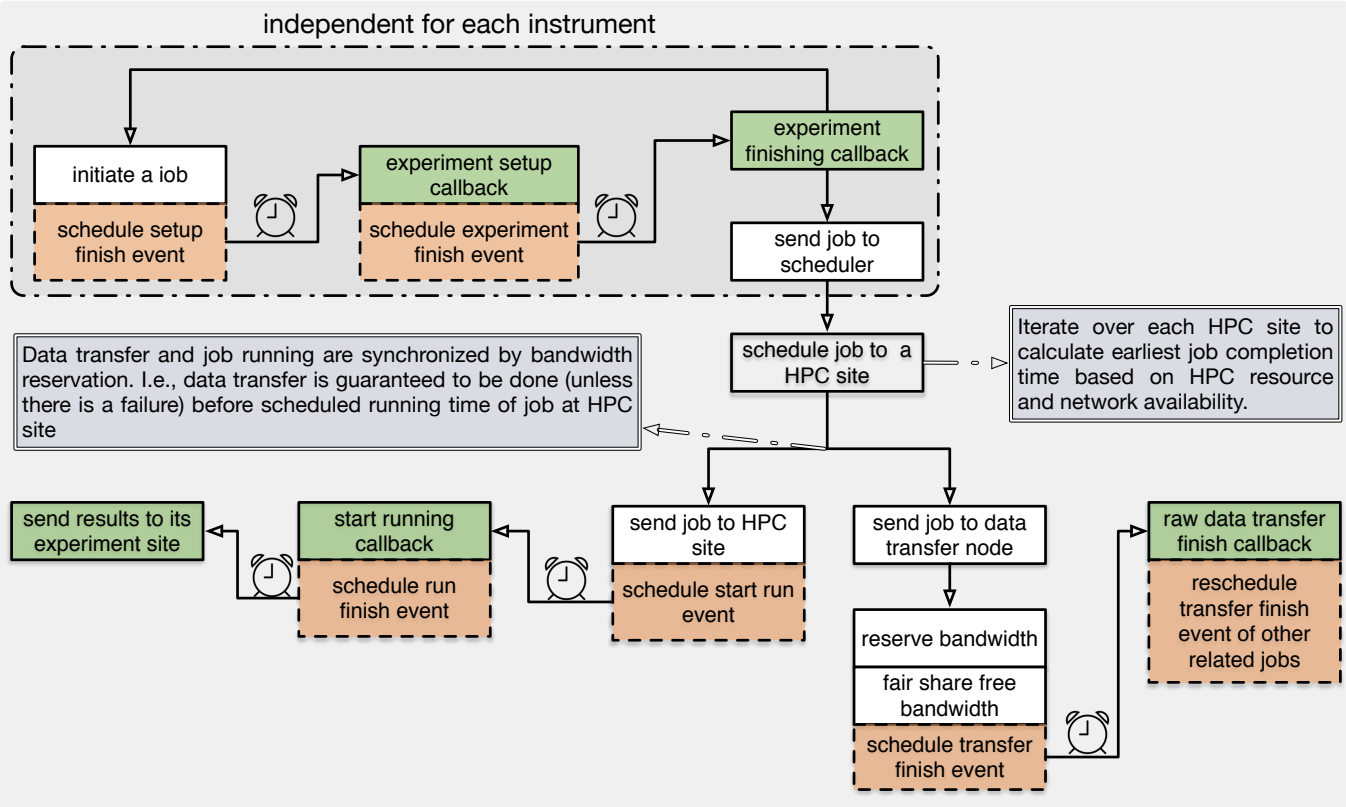

Fig. 2: Discrete event simulation flow chart of jobs over the superfacility. The detailed job scheduling algorithm is shown in Figure 3.

resource. When a new job is sent to the scheduler, we iterate over each HPC site to compute the earliest time when sufficient resources becomes available. Specifically, to calculate the earliest time, we first check the bandwidth availability from the light source facility to the tentative target HPC site in order to compute the shortest data transfer time T_{trs}. Then we check the computing resource availability from T_{trs} seconds from that moment. The scheduler iterates over all HPC sites and selects the one with the earliest computing resource available time.

3) Data transfer service: We assume a guaranteed data transfer time, as might be obtained by using a bandwidth reservation system such as OSCARS [16] over ESnet to reserve a specific amount of bandwidth from experiment site E to HPC site C for a given time period. Thus, the data transfer time will be guaranteed, and the job can be started to run on the HPC site as scheduled.

We also assume that the remaining (unreserved) bandwidth of each physical link is fair shared by all ongoing transfers. If, for example, there are 10 transfers over a 100 Gbps link at time t_1 with a total reserved bandwidth of 80 Gbps, then each job will get extra 2 Gbps of bandwidth because of fair sharing of the unreserved 20 Gbps. This mechanism is applied because some new jobs may come at a future time t_2; the new job will have more bandwidth available if some of these 10 transfers are done before t_2 because of the extra 2 Gbps of bandwidth. Thus, in simulation, an callback function that updates the extra bandwidth will be triggered when there is a new transfer or an ongoing transfer has finished.

IV. Case Study

We demonstrate the use of our framework to explore design choices for a superfacility and to evaluate quantitatively the tradeoffs between user metrics (e.g., response time) and system metrics (e.g., utilization). Specifically, we evaluate the design choices for a superfacility that provides on-demand computing resources for five U.S. Department of Energy light source facilities [17]: the Advanced Light Source (ALS), Advanced Photon Source (APS), National Synchrotron Light Source II (NSLS), Linac Coherent Light Source (LCLS), and Stanford Synchrotron Radiation light source (SSRL).

A first factor to consider when studying computing needs at light sources is the facility operating schedules. As shown in Figure 4, each facility alternates between normal operating periods and shutdown periods in which maintenance is performed, with the latter scheduled on both weekly and longer cadences. The shutdown periods of the different facilities overlap a great deal, with aggregate operating hours being particularly reduced in January, September, and December.

A second factor is the nature of the computing tasks that facilities generate when operating. Here we relied on measurements and/or estimates provided by the facilities of experiment setup time, data acquisition time, size of raw data, and computing demand in core-hours. Building on these data, we quantified and summed the hourly computing resource

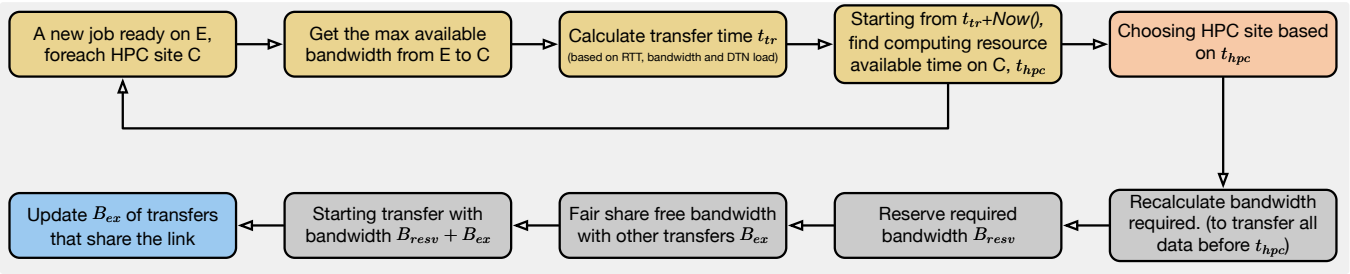

Fig. 3: Simulated job scheduling and data transfer process.

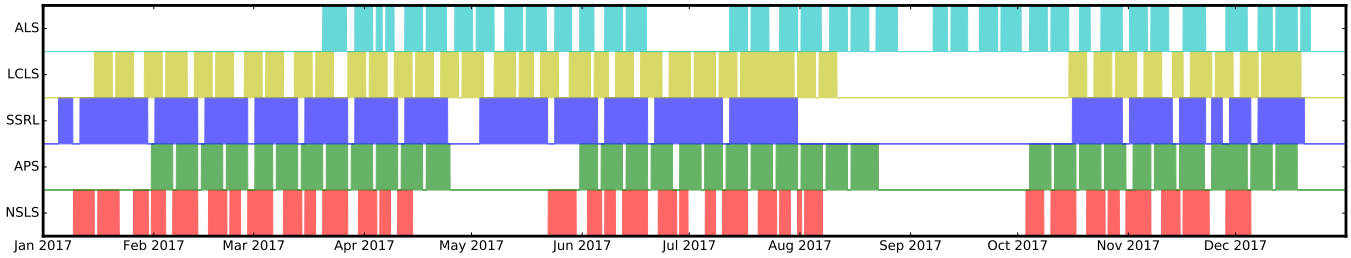

Fig. 4: Shaded periods denote normal operating periods at each of the five DOE light source facilities during 2017. Other time periods are shutdown periods for maintenance.

demands across all five facilities. The resulting numbers are approximate and do not necessarily capture future evolution in demand, but they provide a basis for analysis.

Figure 5 shows the resulting aggregate computing resource demands. The green line indicates the (often substantial) computing resources required to ensure on-demand access to computing during each period; the blue line shows the computing resources required to meet average demand over the year, that is, to ensure that all demands are eventually met. As the figure shows, the difference between peak and average computing demand is high; thus, deploying sufficient resources to be able to meet peak demand at all times will result in low utilization.

A. Analysis of simple design choices

Some design choices can be evaluated (and possibly eliminated) by using basic analysis.

a) Dedicated supercomputer for each facility: The most straightforward way to realize on-demand analysis of light source data is to build a separate, dedicated computing resource at each facility capable of meeting its peak requirements. Doing so, however, will result in overall utilizations of 20.7%. These utilization values are calculated by dividing the actual resource requirement (sum of the compute requirement for each job in core-hours) by the total resource deployed (the peak demand times 8760, which is the number of hours in one year). One reason for the low utilizations is that, as shown in Figure 4, each facility has its own shutdown periods on both long (months) and short (days) time scales. Indeed, NSLS, APS, SSRL, LCLS, and ALS operate for only 46.6%, 56.8%, 63.4%, 57.8%, and 48.3% of the time, respectively, in 2017.

b) One supercomputer for all facilities: Another obvious design choice is to build a single centralized compute facility to meet the aggregate demand of all five light source facilities. If we do so, however, the overall resource utilization is only 28.7%. While better than building a separate computer for each facility, this is still extremely low. The problem is that the difference between the peak and average demand is large, as shown in Figure 5.

c) Optimization of demand: We can improve compute resource utilization by reducing the gap between peak and average demand. For example, we could adjust the shutdown schedule for different facilities. One simple adjustment would be to retain the current sequence of active and shutdown periods at each facility, but to shift certain facilities' annual operating schedules forward in time in a manner that reduces contention for shared compute resources. We explored this option, performing an exhaustive search of potential whole-day shifts across the various facilities. (This search took about 1,000 core-hours; but since this is a one-time optimization, it is not onerous.) We determined that the optimal shift to reduce the variance in demand for each facility would be S_{NSLS} = 0 days, S_{APS} = 159 days, S_{SSRL} = 350 days, S_{LCLS} = 202 days, and S_{ALS} = 318 days. Figure 6 shows the shifted schedule and Figure 7 the computing resource requirements after applying the optimal shift. Comparing with Figure 5, we see that the variance in resource demand is indeed reduced after shifting the operating schedule.

Even with this shifted schedule, however, and if we deploy enough compute resources to provide on-demand access for all jobs, the average resource utilization is still only 34%. This result can be improved by other adjustments. For example, we could adjust experiment schedules during operations. Alterna-

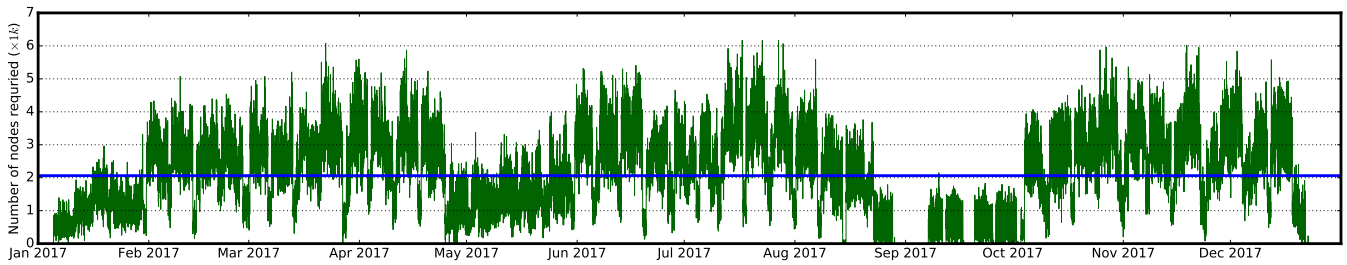

Fig. 5: Aggregate computing requirement for select data- and compute-intensive beamlines at the five facilities.

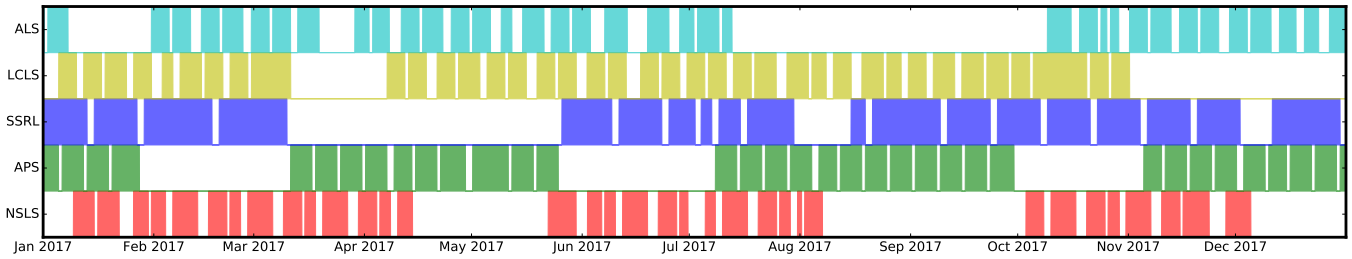

Fig. 6: Operating schedule of five DOE light source facilities after applying optimal shift to each facility's 2017 schedule.

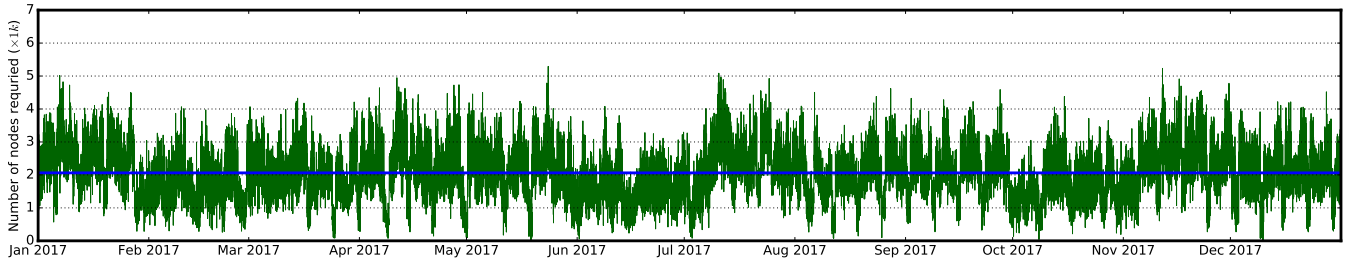

Fig. 7: Aggregate computing demand for data- and compute-intensive beamlines at the five facilities after applying optimal shift.

tively, we could allow certain (presumably lower-priority) jobs to be delayed. If 5% of the jobs can have slowdown greater than one, then resource utilization improves to 61%.

d) Distribution of the compute resources: A single compute facility introduces a single point of failure and potentially also excessive communication costs. Distributing the computing resources to multiple locations can improve availability and reduce network congestion.

Besides the five light source facilities, the DOE Office of Science operates three major HPC sites: the Argonne Leadership Computing Facility (ALCF), Oak Ridge Leadership Computing Facility (OLCF), and National Energy Research Scientific Computing Center (NERSC). These HPC sites and the light source facilities are all connected by the ESnet, a high-speed computer network serving DOE scientists and their collaborators worldwide. To ease HPC resource management, we assume that the new computing resources can be deployed to those three HPC sites. Figure 9 shows the topology of the proposed superfacility.

Table I compares the 75th and 95th percentile slowdown for the regular and shifted schedule with the compute resources distributed equally across the three compute facilities.

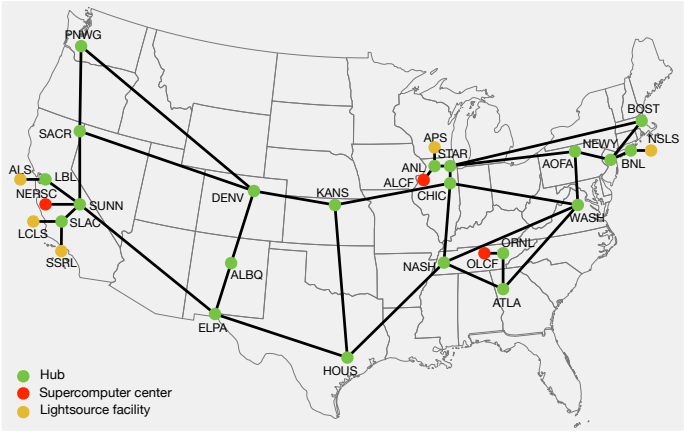

Fig. 9: DOE superfacility topology, showing the five light sources, three HPC facilities, and the ESnet backbone.

B. Joint optimization of design choices: A demonstration

Section IV-A excluded some straightforward design choices through basic analysis. Both analytics in terms of resource utilization, experiment job slowdown, and the influence of

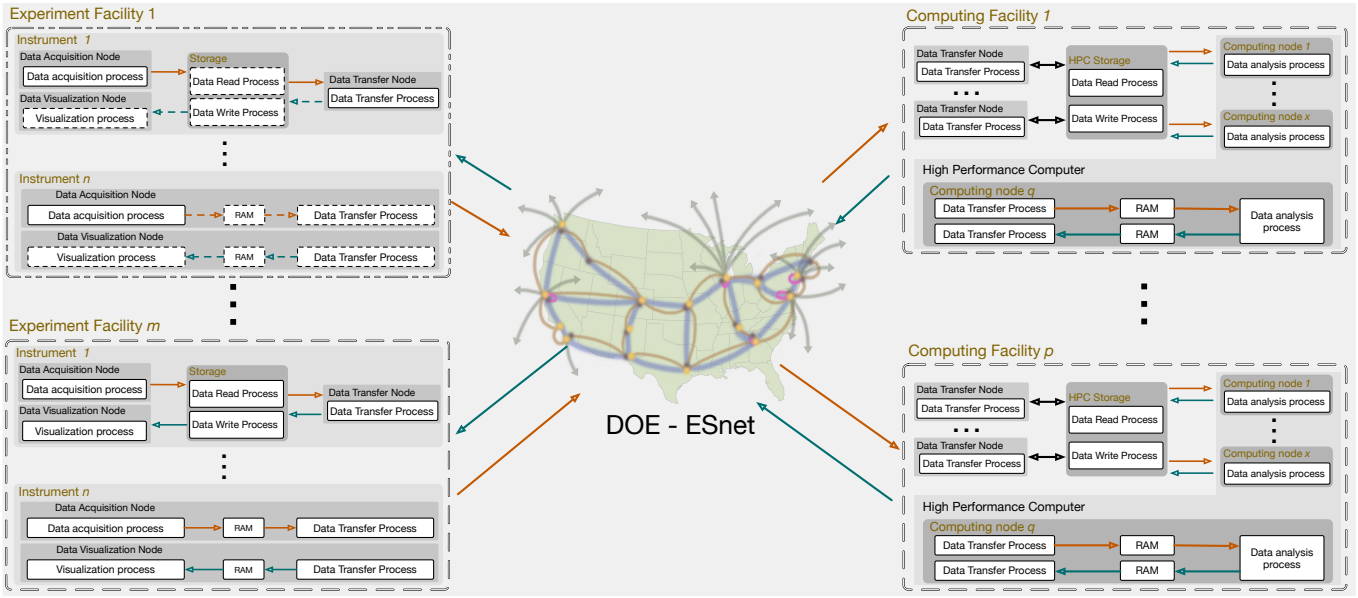

Fig. 8: Sample superfacility over ESnet that supports both streaming and batch workflows.

TABLE I: Comparison of slowdown between regular operating scheduling and optimally shifted operating schedule.

Resource	Regular Schedule		Optimal Schedule	
	Q75	Q95	Q75	Q95
1.00×	986.31	6321.02	457.48	1400.24
1.25×	141.30	991.51	14.71	73.68
1.50×	7.30	82.56	1.41	5.23
1.75×	3.12	10.06	1.12	1.99
2.00×	1.21	2.99	1.04	1.77

unplanned outage imply that distributed-computing resources is a better choice. This section demonstrates the use of our framework to evaluate the design choice.

The time taken by state-of-the-art mixed-integer linear programming algorithms to solve a particular instance may partly depend on its size, that is, on the number of discrete variables and associated constraints. The model presented in this paper has $|\mathcal{J}| \times |\mathcal{R}| \times (\sum_{(j,s)\in\mathcal{J}} |\mathcal{T}_{js}|)$, $|\mathcal{J}| \times |\mathcal{R}| \times (\sum_{s\in\mathcal{S}, r\in\mathcal{R}} |\mathcal{P}(s,r)|) \times (\sum_{(j,s)\in\mathcal{J}} |\mathcal{T}_{js}|)$ and $|\mathcal{J}|$ instances of the x, λ, and w binary variables, respectively. Since sites, resources, and links do not change frequently, the major factors impacting the number of variables and constraints (except (III.9)) are the number of jobs ($|\mathcal{J}|$) and the time period ($|\mathcal{T}|$) of the optimization.

Solving a model with many intervals is a challenge in terms of both compute and memory requirements. For example, when using a step size of 10 seconds, a model for five hours (1,800 intervals, about 200 jobs) will have hundreds of thousands of variables. Therefore, instead of solving the model for a whole year ($\mathcal{T}$), we solve the model for some representative time periods and then use simulation to evaluate the top-k optimal solutions of each time period in order to achieve the feasible optimal choice in practice.

After applying the optimal shift, we have four of the five facilities operating most of the time (see Figure 7). We pick six short (three-hour) representative time periods—one in which all facilities are operating and five in which only four facilities are operating, with each of the latter involving a unique set of four operating facilities. We implemented the MIP model by using AMPL [12, 13], solved with CPLEX optimizer [18] for this subset, and evaluated the top 20 optimal solutions of the MIP model by using simulation.

Table II shows example simulation results. We have designed our framework to output multiple solutions, for two reasons. First, many design choices have similar objective function values (e.g., job slowdown). For example, Figure 10 shows the objective function value of the top 20 solutions. The difference between the optimal objective function value and the 20th optimal is small (only 1.6 %). However, their resource distributions are different. Second, we do not expect that even a good model will capture all design constraints. Thus an apparently suboptimal design choice (i.e., one with a slightly different objective function value) may be subjectively more feasible in practice. By presenting multiple solutions, we allow analysts to consider other factors in their decision making.

V. Related Work

The most straightforward way to evaluate the performance of a new system before it is built is through modeling and simulation. This process can help determine performance bottlenecks inherent in an architecture and provide a basis for refining the system configuration.

Optimal job scheduling is NP-hard because of its combinatorial nature. The study of scheduling algorithms for workflows that run over distributed infrastructure has been an active area of research. Given the nature of most scheduling

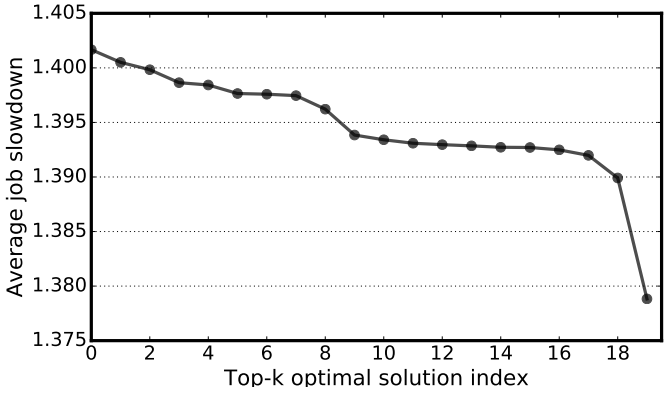

Fig. 10: Objective function values (average job slowdown) of top 20 optimal solutions.

TABLE II: Tradeoff between resource distribution and job slowdown. SD denotes the 75th percentile of job slowdown, UT is the computing resource utilization (%).

Design choice				Metric	
Computer nodes			Network	User (SD)	Provider (UT)
ANL	NERSC	ORNL			
774	669	1653	20 Gbps	2.07	66.7
771	1476	849	20 Gbps	1.54	66.7
938	1050	1108	20 Gbps	1.41	66.7
1220	1050	826	20 Gbps	1.37	66.7
990	757	653	30 Gbps	52.0	86.0
966	778	656	30 Gbps	51.8	86.0
1122	1050	924	30 Gbps	1.13	66.7
798	1184	1114	30 Gbps	1.13	66.7
668	808	1620	30 Gbps	1.16	66.7
571	808	1717	30 Gbps	1.17	66.7

problems, one needs to evaluate and compare their efficacy over a wide range of scenarios. It has thus become necessary to simulate those algorithms for increasingly complex distributed dynamic and workflow applications. For example, Casanova et al. [19, 20] developed SimGrid, a simulation toolkit for the study of scheduling algorithms for distributed applications. Wu et al. [21] used a simulation system to study the execution dynamics of distributed computing workflows and to evaluate the network performance of workflow scheduling or mapping algorithms before actual deployment and experimentation. However, these systems focused on evaluating individual workflows, not infrastructure design.

Mathematical programming is widely used in operations research to minimize or maximize an objective function of many variables, subject to constraints on the variables [12]. It is also an important tool for optimizing design choices, job scheduling, and resource configuration in distributed systems. For example, Chretien et al. [22] introduced an approach based on successive linear programming approximations of a sparse model. They relaxed an integer linear program and used l_p norm-based operators to force the solver to find almost-integer solutions that can be assimilated to an integer solution. Kumar et al. [23] presented a metabroker that schedules multiple jobs on utility grids, with a linear/integer programming model used to schedule jobs. They also proposed a linear-programming-driven genetic algorithm that combines linear programming and genetic algorithms to obtain a metaschedule that minimizes the combined cost of all users in a coordinated manner. However, these studies focused on evaluating scheduling algorithms and did not consider infrastructure constraints, such as networks.

Although researchers have demonstrated near-real-time processing of data from experimental facilities [5, 7, 8, 11], they have not addressed the problem of how to obtain the compute and network resources on demand. The use of numerical optimization and simulation methods to guide design of distributed systems is well established [24, 25]. However, little work has been done on applying such methods to the design of distributed computational facilities. The Models Of Networked Analysis at Regional Centers (MONARC) system [26] applied discrete event simulation methods to evaluate design alternatives for high energy physics computing; their focus was on specialized workflows involving a single data source and many data analysis sites. Others have used simulation to study the performance of distributed computing applications [19, 20], scheduling algorithms [27], or data replication strategies [28], among many other topics, but with a focus on application performance rather than large-scale system design.

VI. Discussion and conclusion

We have presented a framework to evaluate design choices for a cyberinfrastructure. The framework uses a combination of mathematical programming and simulation to find near-optimal resource distributions that can work well in practice. We used mixed-integer programming to find the top k optimal resource distributions for a representative subset of the problem, and we evaluated these solutions using a discrete event simulator that incorporates a number of aspects to mimic the real-world environment. We used the example of a superfacility for analyzing light source data in order to demonstrate how our framework can be used to evaluate different design choices and understand associated tradeoffs. The customizable simulator developed in this work is also useful in superfacility operations, for example (1) to work as a part to decision support system to evaluate proposals for unforeseen scenarios; (2) to prove the feasibility of new project in terms of its computing requirement; and (3) to study scheduling algorithms for workflows run over distributed infrastructure.

We note that we did not consider the use of commercial clouds in our design choices; we plan to do so in future work. Another topic of future work is integrating the simulator (it will work as a function to evaluate design choices) with derivative-free optimization methods in order to directly optimize superfacility design choices.

Acknowledgment

This material was based upon work supported in part by the U.S. Department of Energy, Office of Science, Ad-

vanced Scientific Computing Research, under Contract DE-AC02-06CH11357. The mathematical programming model was solved by using a cluster hosted by the Joint Laboratory for System Evaluation at Argonne National Laboratory. We thank Amedeo Perazzo, Aina Cohen, Dula Parkinson, Alexander Hexemer, Brian Toby, and Nicholas Schwarz for providing us data about the experiment characteristics at light source facilities.

REFERENCES

[1] "Directing Matter and Energy: Five Challenges for Science and the Imagination," https://science.energy.gov/~/media/bes/pdf/reports/files/Directing_Matter_and_Energy_rpt.pdf, 2007.

[2] "Next-Generation Photon Sources for Grand Challenges in Science and Energy," https://science.energy.gov/~/media/bes/pdf/reports/files/Next-Generation_Photon_Sources_rpt.pdf, 2009.

[3] "The Report of the BES Advisory Subcommittee on Future X-ray Light Sources," https://science.energy.gov/~/media/bes/besac/pdf/Reports/Future_Light_Sources_report_BESAC_approved_72513.pdf, 2013.

[4] T. Bicer, D. Gursoy, R. Kettimuthu, F. De Carlo, G. Agrawal, and I. T. Foster, "Rapid tomographic image reconstruction via large-scale parallelization," in *Euro-Par 2015: Parallel Processing. LNCS 9233*, 2015, pp. 289–302. [Online]. Available: http://dx.doi.org/10.1007/978-3-662-48096-0_23

[5] T. Bicer, D. Gürsoy, R. Kettimuthu, F. De Carlo, and I. T. Foster, "Optimization of tomographic reconstruction workflows on geographically distributed resources," *Journal of Synchrotron Radiation*, vol. 23, no. 4, pp. 997–1005, Jul 2016. [Online]. Available: https://doi.org/10.1107/S1600577516007980

[6] J. M. Wozniak, H. Sharma, T. G. Armstrong, M. Wilde, J. D. Almer, and I. Foster, "Big data staging with MPI-IO for interactive X-ray science," in *IEEE/ACM International Symposium on Big Data Computing*. IEEE Computer Society, 2014, pp. 26–34.

[7] J. M. Wozniak, K. Chard, B. Blaiszik, M. Wilde, and I. Foster, "Streaming, storing, and sharing big data for light source science," *Proc. STREAM*, 2015.

[8] I. Foster, R. Ananthakrishnan, B. Blaiszik, K. Chard, R. Osborn, S. Tuecke, M. Wilde, and J. Wozniak, "Networking materials data: Accelerating discovery at an experimental facility," in *Big Data and High Performance Computing*. IOS Press, 2015, pp. 117–132.

[9] J. Deng, Y. S. Nashed, S. Chen, N. W. Phillips, T. Peterka, R. Ross, S. Vogt, C. Jacobsen, and D. J. Vine, "Continuous motion scan ptychography: characterization for increased speed in coherent x-ray imaging," *Optics Express*, vol. 23, no. 5, pp. 5438–5451, 2015.

[10] M. J. Cherukara, K. Sasikumar, W. Cha, B. Narayanan, S. Leake, E. Dufresne, T. Peterka, I. McNulty, H. Wen, S. K. Sankaranarayanan *et al.*, "Ultra-fast three-dimensional X-ray imaging of deformation modes in ZnO nanocrystals," *Nano Letters*, 2017.

[11] L. Ramakrishnan, G. Fox, and S. Jha, "Stream2016: Streaming requirements, experience, applications and middleware workshop," Tech. Rep. LBNL-1006355, 2016, https://pubarchive.lbl.gov/islandora/object/ir%3A1006355/.

[12] R. Fourer, D. M. Gay, and B. W. Kernighan, *AMPL: A modeling language for mathematical programming. Second edition*. INFORMS, 2002.

[13] "AMPL Optimization Inc." 2009, http://ampl.com/about-us/.

[14] S. Robinson, *Simulation: The Practice of Model Development and Use*. John Wiley & Sons, 2004.

[15] G. F. Riley and T. R. Henderson, "The ns-3 network simulator," *Modeling and Tools for Network Simulation*, pp. 15–34, 2010.

[16] ESnet-OSCARS, *On-Demand Secure Circuits and Advance Reservation System*, 2017 (accessed June 3, 2017), https://www.es.net/engineering-services/oscars/.

[17] U.S. DOE Office of Science, *X-Ray Light Sources*, 2017 (accessed June 13, 2017), https://science.energy.gov/bes/suf/user-facilities/x-ray-light-sources/.

[18] "IBM ILOG CPLEX V12.1: Users manual for CPLEX," 2009, International Business Machines Corporation.

[19] H. Casanova, "SimGrid: A toolkit for the simulation of application scheduling," in *1st IEEE/ACM International Symposium on Cluster Computing and the Grid*, 2001, pp. 430–437.

[20] H. Casanova, A. Giersch, A. Legrand, M. Quinson, and F. Suter, "Versatile, scalable, and accurate simulation of distributed applications and platforms," *Journal of Parallel and Distributed Computing*, vol. 74, no. 10, pp. 2899–2917, Jun. 2014. [Online]. Available: http://hal.inria.fr/hal-01017319

[21] Q. Wu and Y. Gu, "Modeling and simulation of distributed computing workflows in heterogeneous network environments," *SIMULATION*, vol. 87, no. 12, pp. 1049–1065, 2011. [Online]. Available: http://dx.doi.org/10.1177/0037549710396920

[22] S. Chretien, J.-M. Nicod, L. Philippe, V. Rehn-Sonigo, and L. Toch, "Job scheduling using successive linear programming approximations of a sparse model," in *18th International Euro-Par Parallel Processing Conference*. Springer, 2012, pp. 116–127. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-32820-6_14

[23] S. K. Garg, P. Konugurthi, and R. Buyya, "A linear programming-driven genetic algorithm for meta-scheduling on utility grids," *International Journal of Parallel, Emergent and Distributed Systems*, vol. 26, no. 6, pp. 493–517, 2011. [Online]. Available: http://dx.doi.org/10.1080/17445760.2010.530002

[24] A. N. Tantawi and D. Towsley, "Optimal static load balancing in distributed computer systems," *Journal of the ACM*, vol. 32, no. 2, pp. 445–465, 1985.

[25] L. Breslau, D. Estrin, K. Fall, S. Floyd, J. Heidemann, A. Helmy, P. Huang, S. McCanne, K. Varadhan, Y. Xu *et al.*, "Advances in network simulation," *Computer*, vol. 33, no. 5, pp. 59–67, 2000.

[26] I. C. Legrand and H. B. Newman, "The MONARC toolset for simulating large network-distributed processing systems," in *32nd Winter Simulation Conference*, 2000, pp. 1794–1801.

[27] H. Arabnejad, J. G. Barbosa, and R. Prodan, "Low-time complexity budget–deadline constrained workflow scheduling on heterogeneous resources," *Future Generation Computer Systems*, vol. 55, pp. 29–40, 2016.

[28] K. Ranganathan and I. Foster, "Identifying dynamic replication strategies for a high-performance data grid," in *Grid Computing. LNCS 2242*. Springer, 2001, pp. 75–86.