JOURNAL OF SYNCHROTRON RADIATION

ISSN 1600-5775

Received 2 April 2016
Accepted 16 May 2016

Edited by P. A. Pianetta, SLAC National Accelerator Laboratory, USA

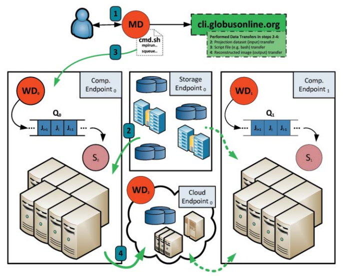
Keywords: tomography; scientific workflows; performance modeling.

Optimization of tomographic reconstruction workflows on geographically distributed resources

Tekin Bicer,[a]* Doğa Gürsoy,[b] Rajkumar Kettimuthu,[a,c] Francesco De Carlo[b] and Ian T. Foster[a,c,d]

[a]Mathematics and Computer Science Division, Argonne National Laboratory, USA, [b]Advanced Photon Source, X-ray Science Division, Argonne National Laboratory, USA, [c]Computation Institute, University of Chicago and Argonne National Laboratory, USA, and [d]Department of Computer Science, University of Chicago, USA. *Correspondence e-mail: bicer@anl.gov

New technological advancements in synchrotron light sources enable data acquisitions at unprecedented levels. This emergent trend affects not only the size of the generated data but also the need for larger computational resources. Although beamline scientists and users have access to local computational resources, these are typically limited and can result in extended execution times. Applications that are based on iterative processing as in tomographic reconstruction methods require high-performance compute clusters for timely analysis of data. Here, time-sensitive analysis and processing of Advanced Photon Source data on geographically distributed resources are focused on. Two main challenges are considered: (i) modeling of the performance of tomographic reconstruction workflows and (ii) transparent execution of these workflows on distributed resources. For the former, three main stages are considered: (i) *data transfer* between storage and computational resources, (i) *wait/queue* time of reconstruction jobs at compute resources, and (iii) *computation* of reconstruction tasks. These performance models allow evaluation and estimation of the execution time of any given iterative tomographic reconstruction workflow that runs on geographically distributed resources. For the latter challenge, a workflow management system is built, which can automate the execution of workflows and minimize the user interaction with the underlying infrastructure. The system utilizes *Globus* to perform secure and efficient data transfer operations. The proposed models and the workflow management system are evaluated by using three high-performance computing and two storage resources, all of which are geographically distributed. Workflows were created with different computational requirements using two compute-intensive tomographic reconstruction algorithms. Experimental evaluation shows that the proposed models and system can be used for selecting the optimum resources, which in turn can provide up to 3.13× speedup (on experimented resources). Moreover, the error rates of the models range between 2.1 and 23.3% (considering workflow execution times), where the accuracy of the model estimations increases with higher computational demands in reconstruction tasks.


MD
cli.globusonline.org
Performed Data Transfers in steps 2-4:
2: Projection dataset (input) transfer
3: Script file (e.g. bash) transfer
4: Reconstructed image (output) transfer
cmd.sh
mpirun..
squeue..
1
2
3
4
WD₀
Comp. Endpoint ₀
Q₀
S₀
Storage Endpoint ₀
WD₁
Comp. Endpoint ₁
Q₁
S₁
WD₂
Cloud Endpoint ₀

1. Introduction

Advances in detector technologies enable increasingly complex experiments and more rapid data acquisition at synchrotron light sources (de Jonge *et al.*, 2014). Current full-field X-ray imaging instruments, including microtomography and high-speed imaging, allow scientists to collect a full three-dimensional (3D) or two-dimensional (2D) dataset at ~1 µm resolution in a fraction of a second (as short as 100 ps). Moreover, the next-generation synchrotron light sources will deliver several orders of magnitude higher temporal resolution in the nanometer range, extending *in situ* and timedependent studies of phenomena to completely new regimes. Similarly, other instruments based on scanning probe techniques will directly benefit from technological advancements, reducing data collection time by a factor of 100, which will result in at least two orders of magnitude more data generation (APS, 2015).

Although data acquisition rates at X-ray light sources are increasing rapidly, the analysis of these data often lags behind, mostly because of the high computational demands of processing and a lack of local compute resources and optimized software tools. These factors significantly hinder the ability to perform near-real-time analysis and visualization. Because experimental setups and data acquisition approaches at light sources are typically configuration-sensitive and scientists need to make timely decisions in order to collect accurate data, short turnaround times for data analysis are crucial. Therefore, efficient and scalable software tools that are tailored to the requirements of synchrotron light sources will be a necessity, especially considering the increased data acquisition rates that will be delivered by the new technological advancements at light sources.

One of the widely desired compute-intensive applications at synchrotron light sources is iterative tomographic image reconstruction (Duke *et al.*, 2015; Gürsoy *et al.*, 2015*a*,*b*). This application enables scientists to observe the internal structure of objects from 3D images. The processing times of iterative reconstruction applications are sensitive to the input dataset size and configuration. Specifically, the number of iterations, 2D projections, sinograms and voxels determine the computational requirements of the reconstruction tasks. Reconstruction of a typical high-resolution tomographic dataset can take days with a well built workstation (Bicer *et al.*, 2015).

In this work, we focus on understanding the performance issues with iterative tomographic reconstruction workflows and improving their end-to-end execution times on geographically distributed resources. In particular, our contributions in this paper are as follows.

(i) We develop performance models for different stages in tomographic reconstruction workflows.

(ii) We present a workflow management system that can execute light source data analysis tasks on distributed resources. Our system utilizes *Globus* for efficient and secure management of workflows and data transfers (Foster, 2011; Allen *et al.*, 2012).

(iii) We evaluate and show our system's accuracy and performance using three high-performance computing (HPC) and two storage resources. During our evaluations, we use two real-world datasets and two iterative reconstruction algorithms.

The remainder of this paper is organized as follows. In §2 we introduce our performance models, which are used for estimating the execution times of the workflows. In §3 we present our workflow management system, in which the execution of tomographic reconstruction workflows is automated on geographically distributed resources. In §4 we evaluate the performance models and workflow management system and use real-world configurations on large-scale HPC resources to assess the accuracy of our model estimations. In §5 we present related work, and in §6 we summarize our conclusions.

2. Performance model

We model the performance of a tomographic reconstruction workflow in three stages: (i) input and output data transfer between storage and compute resources, *data transfer*; (ii) submission of the reconstruction job to computational resource and its *wait/queue time*; and (iii) reconstruction/processing of projection data, *computing*. We assume that the resources used for stages (i) and (iii) may be geographically distant from each other.

2.1. Online estimation of data transfer rate

The tomography datasets are typically stored in local storage resources after their acquisition. Although these resources can provide some level of computational throughput, they are generally limited. Therefore, the data need to be transferred where sufficient computational resources exist. This transfer process can introduce significant overheads depending on the size of the dataset and the status of the network.

Since the network is a shared resource, the available bandwidth between resources changes dynamically. Therefore, we model the data transfer rates in an online fashion. Specifically, we measure the bandwidth from storage to compute resources right before the execution of the workflow.

Our online bandwidth estimation method relies on the bandwidth delay product (BDP) with transfer initialization cost. BDP provides (theoretically) the maximum amount of data that can be in transit before it is acknowledged by the destination resource. It is the product of the theoretical bandwidth between resources and the round-trip time of the packets.

Fig. 1 depicts our online bandwidth estimation method. Here, lines 1 and 2 calculate the sample dataset sizes, DS_1 and DS_2, according to the BDP between resources A and B. Then, the *Generate* function creates these files at resource A. In line 4, the generated sample datasets are transferred from A to B;

```
Require: $RTT_{A\to B}$: Round trip time from resource $A$ to $B$
         $BW_{A\to B}$: Outgoing bandwidth from resource $A$ to $B$
         $c$: Sample dataset replication constant
         $m$: Initialization overhead replication constant ($m > 1$)
         $RD$: Size of the real dataset that is to be transferred

function EstimateDataTransferTime($RD, RTT_{A\to B}, BW_{A\to B}, c, m$)
1:    $DS_1 := RTT_{A\to B} * BW_{A\to B} * c$;
2:    $DS_2 := DS_1 * m$;
3:    $Generate(DS_1, DS_2, A)$;
4:    $t_1 := Transfer(DS_1, A, B)$; $t_2 := Transfer(DS_2, A, B)$;
5:    $Overhead := (m * t_1 - t_2)/(m - 1)$;
6:    $EST_{BW} := DS_2/(t_2 - Overhead)$;
7:    return $RD/EST_{BW} + Overhead$;
```

Figure 1
Algorithm for online bandwidth estimation.

and, according to the transfer times t_1 and t_2, the transfer initialization overhead is calculated. We calculate the current bandwidth (without transfer initialization overhead) in line 6. Using the estimated bandwidth and initialization cost, we then derive the transfer time of the real dataset with size RD. Note that DS_1 and DS_2 are typically much smaller than the original dataset size, hence their transfer costs are minimal.

2.2. Queue time prediction

HPC clusters and supercomputers typically use a queue-based batch scheduler to improve resource utilization and provide fair share between users. These schedulers organize user-submitted tasks/jobs and assign them to the available resources. Depending on the submitted job's resource requirements and the number of jobs in the queue, a job can wait minutes, hours or even days for resources. Therefore, the queue time can significantly extend the end-to-end execution time of a workflow.

We estimate the queue time of a computational resource using a queue simulator. Specifically, first we take a snapshot of the target cluster's queue, Q. This snapshot includes information about previously submitted jobs. Once this information is acquired, we generate a synthetic job, j, that represents the resource requirements of our reconstruction task. This job then is enqueued into the Q. We simulate this new Q with a queue simulator using one of the available scheduling algorithms and observe j's scheduling time.

Depending on the scheduling algorithm, simulated scheduling times can vary significantly. To accommodate the worst-case scenario, we use `conservative backfilling` as our default scheduling algorithm (Mu'alem & Feitelson, 2001). Since we consider the worst-case scenario, we normalize the estimated queue times with a w_s constant. This constant can be derived empirically by scheduling a test job at a target cluster. Specifically, we simulated the current jobs in the target cluster's queue with a test job, which provides a simulation time t_0. At the same time, we submitted the same job to the target cluster's queue, which resulted in t_1 scheduling time. The ratio between t_0 and t_1 represents w_s. Note that w_s needs to be calculated only once; therefore, its computation is negligible.

Although we set `conservative backfilling` as our default scheduling algorithm, other algorithms can also be used. For queue simulations, we used `PYSS` (Python Scheduling Simulator, 2007), a discrete-event simulator that includes 18 different scheduling algorithms.

2.3. Modeling the performance of iterative reconstruction algorithms

In general, a tomography dataset is a 3D array that consists of 2D projections or images of an object. Each projection is a set of line integrals associated with total attenuation of X-ray beams while they pass through the object. During reconstruction, each sinogram is mapped to a slice in a 3D image. The computational complexity of this mapping function depends mostly on the input sinogram size (*i.e.* number of projections); size of the 3D image slice (or output slice); number of iterations; and properties of the reconstruction algorithm, such as communication and data access patterns.

An output slice is typically composed of a grid of size col × col, where each grid cell represents a voxel. Here, col is the width size (x dimension) of a projection. The reconstruction of a slice requires iterating over the X-ray attenuation values in the sinogram, calculating backward and forward projections, and updating the output voxels. The computational demand of the reconstruction depends on the number of intersection points between X-ray and voxel grids. We can estimate the number of intersection points that need to be processed for a projection row as follows,

$$\mathrm{inters}(\theta_i) = \mathrm{col}^2 \left[\left|\sin(\theta_i)\right| + \left|\cos(\theta_i)\right| \right]. \qquad (1)$$

Here, θ_i denotes the rotation of the projection, and col^2 is the number of voxels in the output slice.

Although $\mathrm{inters}(\theta_i)$ can quantify the computational demands of a given projection row (with θ_i rotation), hardware-specific information needs to be incorporated for more precise estimations, especially if the goal is to estimate the execution time. We capture hardware-specific information by introducing a cost function, $\mathrm{cost}(\theta_i)$, and a constant parameter, t_d.

We use $\mathrm{cost}(\theta_i)$ to incorporate overheads due to the data access pattern during reconstruction. For instance, if the rotation of the projection is 0, that is, $\theta = 0$, then the reconstruction algorithm operates on a continuous array of elements (voxels that are in the same row). This type of processing results in high cache utilization and therefore minimum overhead. If the projection rotation is 90° (or $\theta_i = 90$), however, then reconstruction performs strided accesses to the voxels, which in turn introduce low cache utilization and high overhead. We use equation (2) to capture the overhead due to the data access pattern for a given rotation,

$$\mathrm{cost}(\theta_i) = \left[1 + \left|\sin(\theta_i)\right|\right]^k. \qquad (2)$$

In equation (2), the function $\sin(\theta)$ provides the cost trend for the data access overhead, while constant k represents the impact of the overhead. For instance, smaller cache sizes may require larger k value.

Equations (1) and (2) can be used for computing the data access overhead and computational load of reconstructing a sinogram; however, the execution time estimation also depends on the number of instructions that can be executed per second by the running CPU. Therefore, we use the t_d constant to capture the required time (in s) for processing a single intersection point and its corresponding unit time hardware overhead in a compute node.

Assuming that a tomography dataset is composed of n_s sinograms and that each sinogram (S) consists of projection rows with θ_i, namely, $S = \{\theta_1, \theta_2, \ldots, \theta_{|S|}\}$, then we can estimate the reconstruction time of an input tomography dataset using equation (3),

$$n_s \sum_{i=1}^{|S|} \mathrm{inters}(\theta_i)\,\mathrm{cost}(\theta_i)\,t_d. \qquad (3)$$

Note that we can derive all variables and functions in equations (1) and (2) from the input and output datasets. In other words, these parameters and functions can be determined before computation. However, the value of t_d requires reconstruction of a sample sinogram (training data). Fortunately, this process needs to be performed only once for the target hardware (compute node) and typically takes negligible time.

Since our performance model focuses on the computational demand and processing time of a single intersection point, we can estimate the reconstruction times even with large-scale compute resources where a sinogram is being reconstructed by more than one processing unit, such as multicore architectures.

3. Workflow optimization and system design

Execution of geographically distributed workflows requires automation tools to ease synchronization and interaction with resources. In this section, we present the components of our system and discuss its execution.

3.1. System components

Our system consists of four main components: (i) data transfer, (ii) metascheduler, (ii) performance models, and (iv) workflow execution engine. The interactions between these components are shown in Fig. 2 and explained in the following subsections.

3.1.1. Data transfer. The data transfer component is responsible for initiating data transfer requests during workflow execution. It also monitors the status of transfer and triggers actions according to the state of the transmission. For instance, if the data transfer is successful, this component informs the workflow execution engine for the next stage. Otherwise, it dispatches error messages for failure handling.

The metadata information about data transfers and resources is defined in the *resource configuration and metadata* file. This information includes the *type* and (end-point) *address* of the resource, default *input/output directories* for the workflow management system, and the theoretically *available bandwidth*. Here, a resource is a *Globus* end-point, and the type of resource is either storage or compute. If the resource type is compute, it can provide both computational and storage resources, whereas storage resources can be used only for storing data. Input/output directories are the default folders that are being used by the workflow management system. Typically, the input directory defines where input data are stored (for both storage and compute resources), and the output directory is where output data are stored after the completion of data analysis at the compute resource. The available bandwidth is used for estimating data transfer rates (using §2.1).

This component also provides an interface for interacting with *Globus*. The data transfer configuration parameters, such as the number of parallel streams, concurrency and time-out values, can also be set using this interface.

3.1.2. Metascheduler for multisite environments. Our system utilizes geographically distributed resources to minimize the end-to-end execution time. This process requires interacting with various resources and depends on taking into consideration operations such as resource allocations, preparation and scheduling of jobs at the target system.

The metascheduler component creates an abstraction between the resource managers of the compute clusters and the workflow execution engine. Therefore, jobs can be defined in the workflow execution engine (job description) and materialized to real job requests (scripts) in the metascheduler. During execution, these scripts are transferred to the target compute cluster and submitted to its job queue. A job description consists of the job's resource requirements, such as the number of requested processors and allocation time. This information is used for generating a suitable job script for the target computer's resource manager, for example, slurm or torque.

Similar to the data transfer component, information required for job submission is specified in the resource configuration and metadata file. The information includes the type of resource manager and the project name in which the allocation request is going to be charged. Information about the number of nodes, processors and scheduling algorithm of the target resource is also stored in the resource configuration file so that the performance and queue time estimation can be computed.

3.1.3. Performance models and estimations. Our workflow management system uses the models and simulation techniques in §2 to quantify the performance of the resources. Further, it helps in estimating the execution times of different stages. This information can be used for selecting the most efficient resources and minimizing the end-to-end execution time of workflows.

As mentioned earlier, we consider three main stages during the execution of a workflow: data transfer, computation and queue/wait times. The performance models component evaluates these stages according to a given set of resources and user-provided models. Specifically, for the data transfer stage, this component first prepares scripts that generate sample datasets on the target resource. It next initiates the transfer of these sample datasets between resources (using *Globus*). It then collects the required performance information and parameters and calculates the estimated data transfer time by using a performance model. The computation stage requires

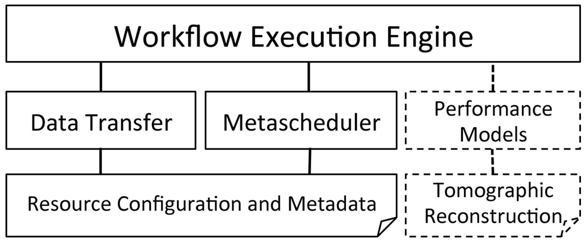

Workflow Execution Engine
Data Transfer
Metascheduler
Performance Models
Resource Configuration and Metadata
Tomographic Reconstruction

Figure 2
Components of the workflow execution engine.

using input data size information (*i.e.* projection data), the t_d value of target compute clusters, number of nodes/processes, and application-specific information such as the number of iterations. This information is then used for setting the parameters in the computational model, and execution times for different compute resources are estimated. For the queue/wait time stage, the system first collects the current states of the queues in the target compute clusters. This state information includes all jobs that are currently running or waiting in the clusters' queues. The performance models component simulates each of these queues with the user-specified reconstruction job and estimates the queue times.

Note that the performance models component extensively uses *Globus* and the resource configuration file for estimations. It also provides an interface in which different light source application models can easily be integrated.

Our system can be used for execution and modeling of large workflows that execute on many geographically distributed resources.

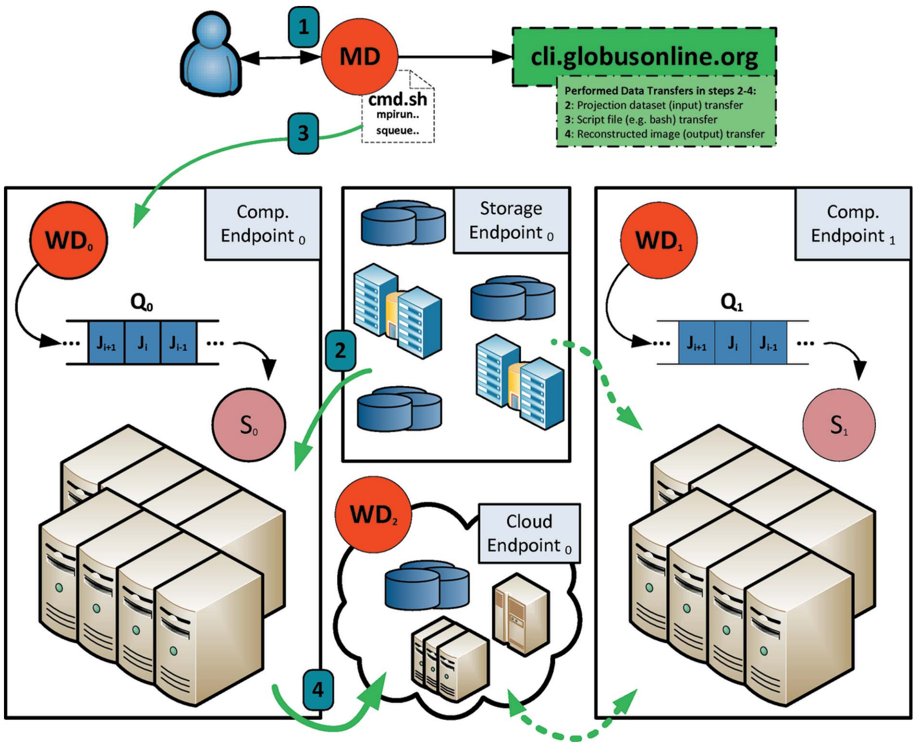


Figure 3
Execution of the workflow management system. Each rectangle (and the cloud) represents a geographically distant location. Steps (1)–(4) illustrate a sample workflow, in which (1) the user initiates an iterative tomographic reconstruction workflow; (2) projection dataset, input, is transferred from Storage Endpoint$_0$ to Comp. Endpoint$_0$; (3) system generated resource allocation script is transferred and submitted to Comp. Endpoint$_0$ for reconstruction; (4) reconstructed image, output, is transferred to Cloud Endpoint$_0$ for visualization and storage.

3.1.4. Workflow execution engine.

The workflow execution engine is the intermediate layer that orchestrates the other components. Users of the workflow management system interact with this layer to initiate workflow executions and performance estimations.

3.2. Execution of workflow management system

Fig. 3 shows the execution flow of our system, which consists of two main processes (daemons) that manage the execution of workflows: `master daemon` (`MD`) and `worker daemon` (`WD`). The user interacts with the `MD` process and initiates the workflow execution. `MD` is responsible for running the previously defined components, namely the data transfer, metascheduler, performance models and workflow execution engine.

Our workflow management system can interact with any end-point that is enabled in *Globus*. In Fig. 3, we show four different end-points: two are compute end-points, one is a cloud end-point, and one is a storage end-point. Since storage resources can be managed easily with *Globus*, `WD` is executed on compute resources only.

Fig. 3 illustrates a sample workflow scenario. Initially, the user interacts with `MD` and (1) initiates the workflow execution. `MD` then establishes a connection with *Globus* and (2) starts transferring data (which need to be processed) from the storage to the compute end-point. While data are being transferred, `MD` (3) creates command scripts and initiates another transfer request from `MD` to the compute resource. After the transmission, `WD` starts executing the retrieved scripts. The scripts consist of different bash commands, such as job submission, sample data generation (for data transfer estimations) or queue status commands (for queue time estimation). In our workflow example, the transferred script has job submission commands; hence `WD` enqueues the job to the cluster's queue. This job is then executed by the cluster's resource manager. Recall that the metascheduler component creates job submission commands depending on the target cluster's environment. Therefore the scripts can transparently be generated by `MD`. Once `WD` finishes executing all the commands in the script, it informs `MD` with a status file. `MD` then initiates another data transfer that (4) sends the processed data from the compute to the data end-point.

4. Experimental results

In this section, we present the experimental evaluation of our performance models and workflow management system. In particular, we compare and analyze the *estimated* and *real* end-to-end execution times of tomographic reconstruction workflows that run on geographically distributed resources.

4.1. Experimental setup

In our experiments, we used three compute clusters, Mira, Stampede and Gordon, for data processing. Table 1 summarizes the technical details of the compute clusters. We also used two storage resources, Chameleon (https://chameleoncloud.org) and Petrel (http://petrel.alcf.anl.gov), for data storage. Petrel is located at Argonne National Laboratory and can provide up to 1.7 PB of storage space. Chameleon is a cloud resource located at the Texas Advanced Computing Center (TACC) and has 1.5 PB of storage space.

We evaluated our computation models using two iterative tomographic reconstruction algorithms: *simultaneous iterative reconstruction technique* (SIRT) and *accelerated penalized maximum likelihood* (APMLR). APMLR is computationally more demanding than SIRT because it incorporates information of neighboring voxels during reconstruction. For data transfer operations and estimations, we used *Globus*. During the data transfer estimations, we set the sample data size replication to 16 (c = 16) and the initialization overhead multiplier to 2 (m = 2). For job queue simulations, we used the `conservative backfilling` algorithm and set its weight in the performance models to 0.04. Finally, the k constant in the cost(θ) function is set to 1.

We used two real-world datasets, 'Seed' and 'Hornby', collected from Advanced Photon Source (APS) beamlines. The Seed dataset was acquired from a seed of *Arabidopsis thaliana*, a flowering plant (Gürsoy *et al.*, 2015*b*); it consists of 720 projections, 512 sinograms and 501 columns (*i.e.* 720 × 2048 × 501 single-precision floating-point numbers). Hornby is X-ray microtomography data from a shale sample (Kanitpanyacharoen *et al.*, 2013); it includes 360 projections, 2048 sinograms and 1024 columns. During our experiments, we varied the number of projections and sinograms of the datasets in order to introduce varying computational demands for reconstruction.

We assumed that a workflow consists of four stages: (1) transferring the projections dataset (input) from storage to the compute resource; (2) submitting the reconstruction job to the queue and wait; (3) reconstructing the tomographic dataset; and (4) transferring the 3D image (output) from the compute resource to storage. During our experiments, we categorized these stages as *data transfer*, *queue* and *computation*.

4.2. Evaluation of performance models on geographically distributed resources

First, we evaluated the estimated and real execution times of the Seed and Hornby datasets using the SIRT reconstruction algorithm. During our evaluation, we varied the experimental configurations and created workflows with different performance characteristics.

Fig. 4 shows the experimental results for the Hornby dataset. For all experiments, 'Real' refers to the actual end-to-end execution times of the workflows, whereas 'Estimated' refers to the estimated execution times after running the performance models. Different dataset sizes were used in order to introduce varying computational demands; therefore, the accuracy of the performance model estimations can be observed. Specifically, we generated several versions of the Hornby dataset and set their dimensions to $P \times 128 \times 2048$, in which P refers to the number of projections in the tomo-

Table 1
Specifications of experimented compute clusters (PFLOPS: peta floating-point operations per second; TFLOPS: tera floating-point operations per seconds).

Cluster	Peak performance	Number of nodes	Number of cores	Memory
Mira†	10 PFLOPS	49125	786432	768 TiB
Stampede‡	9.6 PFLOPS	6400	522080	270 TiB
Gordon§	341 TFLOPS	1024	16384	64 TiB

† Located at ANL: http://www.alcf.anl.gov/mira. ‡ Located at TACC: https://www.tacc.utexas.edu/stampede. § Located at SDSC: http://www.sdsc.edu/services/hpc.

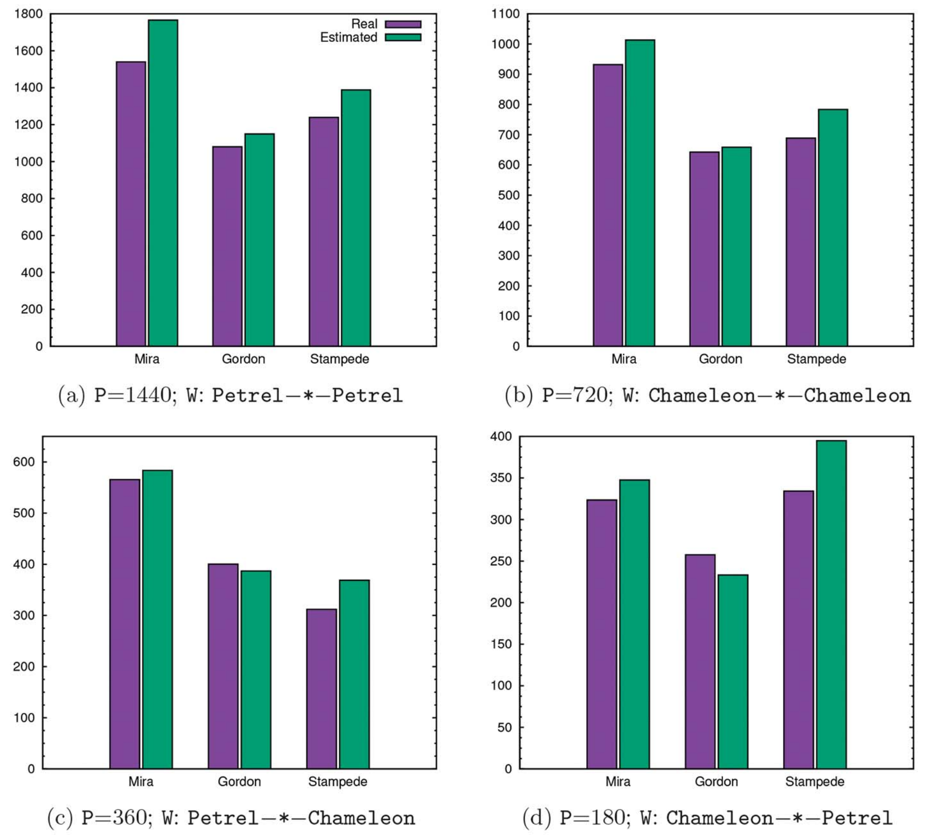


Figure 4
Real and estimated execution times (in s) of Hornby image reconstruction workflow using geographically distributed compute and storage resources.

graphic dataset. We also used different configurations for the storage and compute resources, so that varying data transfer and computational costs were introduced. In particular, each workflow, W, was configured as S_0–C–S_1, where C refers to *compute resource* and S_0 and S_1 are *source* and *destination storage resources*, respectively. Since C = {Gordon, Mira, Stampede} in each experiment, we present C by $*$ in Figs. 4(*a*)–4(*d*). For instance, in Fig. 4(*d*), the Hornby dataset's dimensions are 180 × 128 × 2048 (P = 180), and the dataset is initially stored on Chameleon. For each workflow in Fig. 4(*d*), first the dataset is transferred from Chameleon to one of the computational resources (C). Then, its reconstruction job is submitted to the compute cluster's queue. After the reconstruction, the output dataset (3D image) is transferred to Petrel. For this set of experiments, we used 128, 32 and 32 compute nodes for Mira, Stampede and Gordon, respectively.

Considering the estimated and real execution times in Fig. 4, we see that the error rates range from 1% to 18.2%, with an average of 9.8%. The highest error rate is observed in the P = 180 and C = Stampede configuration. As mentioned before, we consider (and model) three phases in a workflow: data transfer, queue and computation. Among these, our computational model provides the most accurate estimation, mainly because of the explicit allocation of compute nodes. On the other hand, the data transfer and job queue are shared between users, thus introducing noise and performance fluctuations depending on the utilization of the network and compute resources. Because smaller datasets require less computation, errors in data transfer and queue time estimations become more visible in the overall estimation. Specifically, for the P = 180 and C = Stampede configuration, the real and estimated computation times are 137 s and 136 s, respectively. However, the total real time for the queue and data transfer is 197 s, whereas the total estimated time is 258 s.

Among the compute resources, Stampede provides the highest computational throughput per node. When we analyze the execution times in Fig. 4, however, we see that in many cases configurations with Stampede perform poorly. The main reasons are the longer job queues and data transfer times at Stampede. Note that our performance models can provide estimations that can help in selecting the most efficient workflow configurations. If we compare the real execution times in each figure, the speedups in selecting the best configuration *versus* the worst range between 1.41 and 1.95, which indicate that our models can have a significant effect on the end-to-end execution time.

In Fig. 5 we repeat the previous experiments with the Seed dataset. The dimensions of the Seed dataset are P × 512 × 501. Since this dataset is smaller than Hornby's, its computational demands are lower. Therefore, we observe larger error rates. Specifically, considering the real and estimated times, we see that the average error rate is 9.5%, where the rates range between 1% and 23.3%. The largest error rate is observed with the P = 90 and C = Stampede configuration, in which the error rate of the computation time is 6.2% and that of the queue and data transfer is 28.1%. Most of the remaining configurations' error rates are lower than 14.5%.

Similar to the Hornby experiments, if we compare the estimated and real execution times we see that our models can help select the best configuration that can provide the shortest end-to-end execution time. Considering the configurations with the best and worst execution times, the speedups of selecting the right configuration range between 1.89 and 3.13.

Next, in Fig. 6, we focus on the workflow execution times of the Hornby dataset, considering different phases and varying the number of compute nodes. The dimensions of the dataset are configured to 180 × 2048 × 2048; that is, all the sinograms of Hornby are reconstructed. The computations are performed on Stampede, and both the source and target storage resources are set to Chameleon.

Fig. 6(*a*) shows the total execution times. The error rates of the estimated execution times range from 3% to 14.4%.

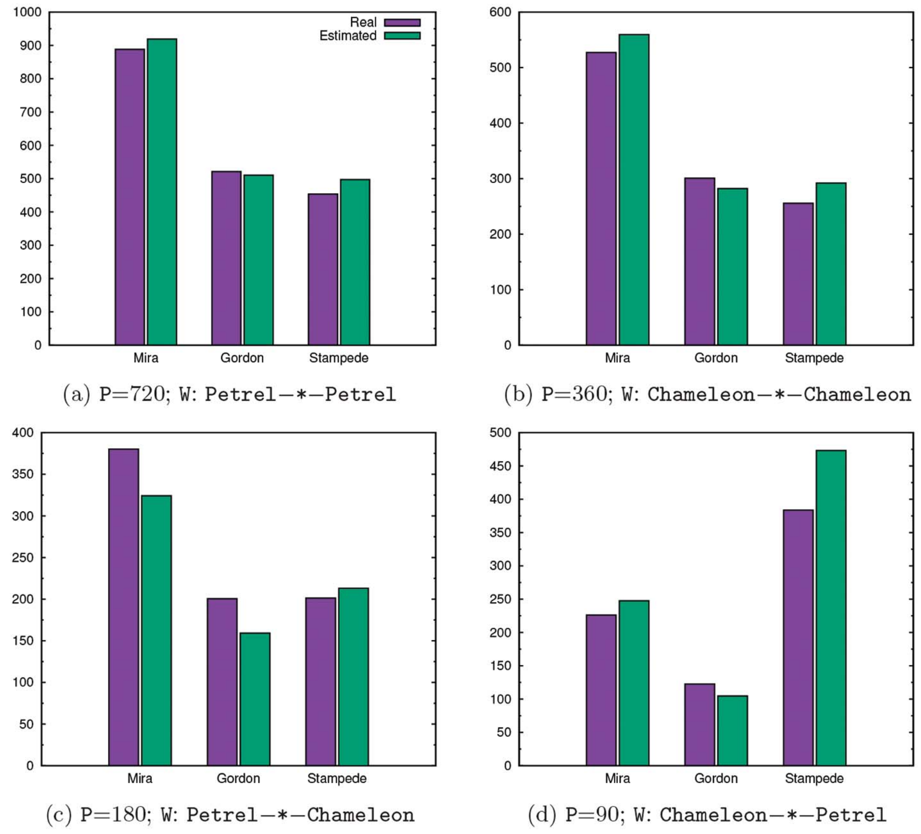


Figure 5
Real and estimated execution times (in s) of Seed image reconstruction workflow using geographically distributed compute and storage resources.

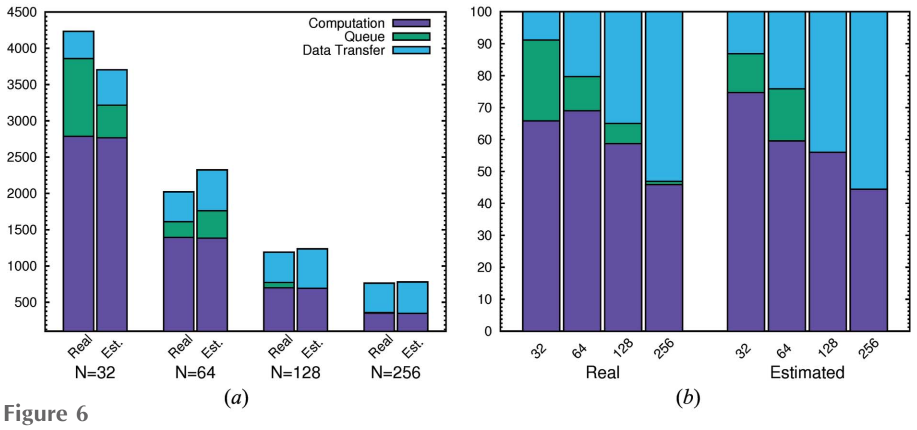


Figure 6
(*a*) Comparison of real and estimated execution times (in s) considering different phases for the Hornby dataset with varying number of compute nodes at Stampede. The reconstruction algorithm is APMLR. The dimensions of the dataset are 180 × 2048 × 2048. Workflow of the reconstruction is configured as *W*: Chameleon–Stampede–Chameleon. (*b*) Distribution (%) of phases with respect to total execution time grouped in real and estimated configurations.

Similar to the previous set of experiments, the estimated times successfully reflect the real execution times. If we focus on the computation times, we observe almost linear scalability; however, the data transfer and queue times do not follow the same trend, which can also be seen in Fig. 6(*b*). For instance, while the queue time of the 32-node configuration takes 25.3% of the execution time, the queue times of the $N = 128$ and $N = 256$ configurations are negligible.

If we analyze the data transfer times, we observe similar results for all the workflows. Because the output dataset size is the same for all configurations, the data transfer takes similar times. Specifically, the data transfer times range from 374 s to 404 s for the real configurations and 431 s to 560 s for the estimated configurations. Because Stampede and Chameleon are located at the same site (TACC), the data transfer is more isolated, and the network can provide better performance.

5. Related work

In recent years, data analysis problems and workflows at synchrotron light sources have gained immense popularity, primarily because of the technological advances in scientific instruments and unprecedented data generation rates at beamlines.

Several activities focus on data management and analysis of light sources (Basham*et al.*, 2015; Hong *et al.*, 2015; Patton *et al.*, 2015). CAMERA is an interdisciplinary project at LBNL (Donatelli*et al.*, 2015). The main goal of the CAMERA project is to investigate problems of DOE user facilities, including synchrotron light sources, and develop fundamental new mathematical solutions. Similarly, the Computational Science Initiative at Brookhaven National Laboratory provides workflow systems to define data-processing tasks for NSLS-II facility users (Computational Science Initiative, 2015). PaNdata is another joint effort that focuses on creating fully integrated information infrastructure for the X-ray and neutron facilities at Europe (Bicarregui*et al.*, 2015).

Tomography is a widely used imaging technique at synchrotron light sources (Gürsoy *et al.*, 2015*a*,*b*; Mohan *et al.*, 2015; Qi & Leahy, 2006; Duke *et al.*, 2015; Chen *et al.*, 2012; Pelt *et al.*, 2016). Iterative tomographic reconstruction typically provides higher-quality images than the filtered-back projection method does, because of the better reduction in noise and in missing wedge artifacts. However, iterative approaches typically require more computational throughput (Sidky *et al.*, 2006; Agulleiro & Fernandez, 2011; Treibig *et al.*, 2013; Beister *et al.*, 2012). Accelerators, such as GPUs, have been used to improve performance of iterative algorithms (Chilingaryan *et al.*, 2010; Mirone *et al.*, 2014; Brun *et al.*, 2015; Vogelgesang *et al.*, 2012), but memory limitations of these accelerators can introduce significant overheads due to the data movement from host to device memory. The TomoPy framework provides a complete set of tools for the portable analysis of tomography datasets, which can run on workstations (Gürsoy *et al.*, 2014). Our workflow management system uses highly parallel versions of TomoPy's reconstruction algorithms to decrease the execution time of reconstruction tasks from days to minutes on high-performance resources (Bicer *et al.*, 2015). Both tools use data exchange models for organization and portability of tomography data (De Carlo *et al.*, 2014).

Scientific workflows have been extensively researched by different communities (Frey *et al.*, 2001; Goecks *et al.*, 2010; Wolstencroft *et al.*, 2013; Taylor *et al.*, 2007). The Pegasus workflow management system maps abstract workflow descriptions to geographically distributed execution environments (Deelman *et al.*, 2015). In this work, we analyze and integrate performance models for light source data analysis workflows. SPOT is another system that focuses on end-to-end analysis of light source data (Deslippe *et al.*, 2014). SPOT enables the execution Advanced Light Source workflows on NERSC's resources. In contrast, our work uses *Globus* for data management and workflow execution, and therefore it can utilize every end-point that is accessible by the user.

6. Conclusion

In this paper, we focused on performance modeling and transparent execution of a synchrotron light source application. The proposed models help to quantify the data transfer and computation performance while considering the queue/wait time of high-performance computing resources. Our models are tailored to iterative tomographic image reconstruction, in which large computational throughput is desired for timely execution. We implemented these models in a workflow management system that minimizes the user interaction while executing reconstruction tasks on geographically distributed resources. We evaluated our models and system using two tomography datasets (from APS beamlines) and two compute-intensive tomographic reconstruction algorithms. Moreover, we used three HPC and two storage resources, all of which are distributed. Our experimental results show that the correct resource selection can provide up to 3.13× speedup. Moreover, the error rates of the models range between 2.1 and 23.3% (considering workflow execution times), where the accuracy of the estimations improves with higher computational demands in reconstruction tasks.

Acknowledgements

This material is based upon work supported by the US Department of Energy, Office of Science, under contract number DE-AC02-06CH11357.

References

Agulleiro, J. & Fernandez, J.-J. (2011). *Bioinformatics*, **27**, 582–583.

Allen, B., Pickett, K., Tuecke, S., Bresnahan, J., Childers, L., Foster, I., Kandaswamy, G., Kettimuthu, R., Kordas, J., Link, M. & Martin, S. (2012). *Commun. ACM*, **55**, 81–88.

APS (2015). *Early Science at the Upgraded Advanced Photon Source.* Technical Report. Advanced Photon Source, Argonne National Laboratory, Argonne, IL, USA.

Basham, M., Filik, J., Wharmby, M. T., Chang, P. C. Y., El Kassaby, B., Gerring, M., Aishima, J., Levik, K., Pulford, B. C. A., Sikharulidze, I., Sneddon, D., Webber, M., Dhesi, S. S., Maccherozzi, F., Svensson, O., Brockhauser, S., Náray, G. & Ashton, A. W. (2015). *J. Synchrotron Rad.* **22**, 853–858.

Beister, M., Kolditz, D. & Kalender, W. A. (2012). *Phys. Med.* **28**, 94–108.

Bicarregui, J., Matthews, B. & Schluenzen, F. (2015). *Synchrotron Radiat. News*, **28**, 30–35.

Bicer, T., Gursoy, D., Kettimuthu, R., De Carlo, F., Agrawal, G. & Foster, I. T. (2015). *Euro-Par 2015: Parallel Processing*, pp. 289–302. Berlin/Heidelberg: Springer.

Brun, F., Pacilè, S., Accardo, A., Kourousias, G., Dreossi, D., Mancini, L., Tromba, G. & Pugliese, R. (2015). *Fund. Inform.* **141**, 233–243.

Chen, R.-C., Dreossi, D., Mancini, L., Menk, R., Rigon, L., Xiao, T.-Q. & Longo, R. (2012). *J. Synchrotron Rad.* **19**, 836–845.

Chilingaryan, S., Kopmann, A., Mirone, A. & dos Santos Rolo, T. (2010). *17th IEEE-NPSS Real Time Conference (RT)*, pp. 1–8.

Computational Science Initiative (2015). *Center for Data-Driven Discovery: Components*, https://www.bnl.gov/compsci/c3d/programs/NSLS.php. [Online accessed November 2015.]

De Carlo, F., Gürsoy, D., Marone, F., Rivers, M., Parkinson, D. Y., Khan, F., Schwarz, N., Vine, D. J., Vogt, S., Gleber, S.-C., Narayanan, S., Newville, M., Lanzirotti, T., Sun, Y., Hong, Y. P. & Jacobsen, C. (2014). *J. Synchrotron Rad.* **21**, 1224–1230.

Deelman, E., Vahi, K., Juve, G., Rynge, M., Callaghan, S., Maechling, P. J., Mayani, R., Chen, W., da Silva, R. F., Livny, M. & Wenger, K. (2015). *Fut. Gen. Comput. Systems*, **46**, 17–35.

Deslippe, J., Essiari, A., Patton, S. J., Samak, T., Tull, C. E., Hexemer, A., Kumar, D., Parkinson, D. & Stewart, P. (2014). *Proceedings of the 9th Workshop on Workflows in Support of Large-Scale Science (WORKS'14)*, pp. 31–40. Piscataway: IEEE Press.

Donatelli, J., Haranczyk, M., Hexemer, A., Krishnan, H., Li, X., Lin, L., Maia, F., Marchesini, S., Parkinson, D., Perciano, T., Shapiro, D., Ushizima, D., Yang, C. & Sethian, J. (2015). *Synchrotron Radiat. News*, **28**, 4–9.

Duke, D. J., Swantek, A. B., Sovis, N. M., Tilocco, F. Z., Powell, C. F., Kastengren, A. L., Gürsoy, D. & Biçer, T. (2015). *SAE Int. J. Eng.* **9**, 2015-01-1873.

Foster, I. (2011). *IEEE Internet Comput.* **15**, 70–73.

Frey, J., Tannenbaum, T., Livny, M., Foster, I. & Tuecke, S. (2001). *Proceedings of the 10th IEEE International Symposium on High Performance Distributed Computing*, pp. 55–63.

Goecks, J., Nekrutenko, A., Taylor, J. & The Galaxy Team (2010). *Genome Biol.* **11**, R86.

Gürsoy, D., Bicer, T., Almer, J. D., Kettimuthu, R., Stock, S. R. & De Carlo, F. (2015*a*). *Philos. Trans. R. Soc. A*, **373**, 20140392.

Gürsoy, D., Biçer, T., Lanzirotti, A., Newville, M. G. & De Carlo, F. (2015*b*). *Opt. Express*, **23**, 9014–9023.

Gürsoy, D., De Carlo, F., Xiao, X. & Jacobsen, C. (2014). *J. Synchrotron Rad.* **21**, 1188–1193.

Hong, Y. P., Chen, S. & Jacobsen, C. (2015). *Proc. SPIE*, **9592**, 95920W.

Jonge, M. D. de, Ryan, C. G. & Jacobsen, C. J. (2014). *J. Synchrotron Rad.* **21**, 1031–1047.

Kanitpanyacharoen, W., Parkinson, D. Y., De Carlo, F., Marone, F., Stampanoni, M., Mokso, R., MacDowell, A. & Wenk, H.-R. (2013). *J. Synchrotron Rad.* **20**, 172–180.

Mirone, A., Brun, E., Gouillart, E., Tafforeau, P. & Kieffer, J. (2014). *Nucl. Instrum. Methods Phys. Res. B*, **324**, 41–48.

Mohan, K., Venkatakrishnan, S., Gibbs, J., Gulsoy, E., Xiao, X., De Graef, M., Voorhees, P. & Bouman, C. (2015). *IEEE Trans. Comput. Imaging*, **1**, 96–111.

Mu'alem, A. W. & Feitelson, D. G. (2001). *IEEE Trans. Parallel Distrib. Syst.* **12**, 529–543.

Patton, S., Samak, T., Tull, C. & Mackenzie, C. (2015). *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pp. 1014–1019.

Pelt, D. M., Gürsoy, D., Palenstijn, W. J., Sijbers, J., De Carlo, F. & Batenburg, K. J. (2016). *J. Synchrotron Rad.* **23**, 842–849.

Python Scheduling Simulator (2007). *PYSS*, https://code.google.com/p/pyss. [Online accessed November 2015.]

Qi, J. & Leahy, R. M. (2006). *Phys. Med. Biol.* **51**, R541–R578.

Sidky, E. Y., Kao, C.-M. & Pan, X. (2006). *J. X-ray Sci. Technol.* **14**, 119–139.

Taylor, I., Shields, M., Wang, I. & Harrison, A. (2007). *Workflows for e-Science*, edited by I. Taylor, E. Deelman, D. Gannon & M. Shields, pp. 320–339. New York: Springer.

Treibig, J., Hager, G., Hofmann, H. G., Hornegger, J. & Wellein, G. (2013). *Intl J. High Performance Comput. Appl.* **27**, 162–177.

Vogelgesang, M., Chilingaryan, S., d, Santos, T. & Kopmann, A. (2012). *Proceedings of the 14th IEEE Conference on High Performance Computing and Communication & The 9th IEEE International Conference on Embedded Software and Systems (HPCC-ICESS)*, pp. 824–829.

Wolstencroft, K., Haines, R., Fellows, D., Williams, A., Withers, D., Owen, S., Soiland-Reyes, S., Dunlop, I., Nenadic, A., Fisher, P., Bhagat, J., Belhajjame, K., Bacall, F., Hardisty, A., Nieva de la Hidalga, A., Balcazar Vargas, M. P., Sufi, S. & Goble, C. (2013). *Nucleic Acids Res.* **41**, W557–W561.