frontiers | Frontiers in High Performance Computing

TYPE Original Research
PUBLISHED 06 June 2025
DOI 10.3389/fhpcp.2025.1550855

OPEN ACCESS

EDITED BY
Martin Berzins,
The University of Utah, United States

REVIEWED BY
Xiao Wang,
Oak Ridge National Laboratory (DOE), United States
Peng Chen,
RIKEN, Japan

*CORRESPONDENCE
Hai Duc Nguyen
✉ hai.nguyen@anl.gov

RECEIVED 24 December 2024
ACCEPTED 12 May 2025
PUBLISHED 06 June 2025

CITATION
Nguyen HD, Bicer T, Nicolae B, Kettimuthu R, Huerta EA and Foster IT (2025) Resilient execution of distributed X-ray image analysis workflows.
Front. High Perform. Comput. 3:1550855.
doi: 10.3389/fhpcp.2025.1550855

Resilient execution of distributed X-ray image analysis workflows

Hai Duc Nguyen[1]*, Tekin Bicer[1], Bogdan Nicolae[1], Rajkumar Kettimuthu[1], E. A. Huerta[1,2] and Ian T. Foster[1,2]

[1]Argonne National Laboratory, Lemont, IL, United States, [2]University of Chicago, Chicago, IL, United States

Long-running scientific workflows, such as tomographic data analysis pipelines, are prone to a variety of failures, including hardware and network disruptions, as well as software errors. These failures can substantially degrade performance and increase turnaround times, particularly in large-scale, geographically distributed, and time-sensitive environments like synchrotron radiation facilities. In this work, we propose and evaluate resilience strategies aimed at mitigating the impact of failures in tomographic reconstruction workflows. Specifically, we introduce an asynchronous, non-blocking checkpointing mechanism and a dynamic load redistribution technique with lazy recovery, designed to enhance workflow reliability and minimize failure-induced overheads. These approaches facilitate progress preservation, balanced load distribution, and efficient recovery in error-prone environments. To evaluate their effectiveness, we implement a 3D tomographic reconstruction pipeline and deploy it across Argonne's leadership computing infrastructure and synchrotron facilities. Our results demonstrate that the proposed resilience techniques significantly reduce failure impact—by up to 500×—while maintaining negligible overhead (<3%).

KEYWORDS

HPC, producer-consumer workflows, tomographic reconstruction, resilience, fault tolerance, checkpointing, load-balancing

1 Introduction

Large-scale, long-running X-ray image analysis workflows are crucial in synchrotron radiation facilities, supporting data-intensive experimentation across diverse scientific domains, including life sciences (Rawson et al., 2020), energy (Finegan et al., 2015; Liu et al., 2019a), geology (Leu et al., 2014; Butler et al., 2020), and materials science (Vásárhelyi et al., 2020; Ravi et al., 2022). These workflows enable advanced imaging techniques, such as high-speed, time-resolved tomography (Zwanenburg et al., 2021; Maire and Withers, 2014). Central to these workflows is tomographic reconstruction, which converts partial, low-dimensional data captured from various angles into detailed, high-dimensional representations (Withers et al., 2021), providing critical insights for downstream, domain-specific analyses (Hsieh and Flohr, 2021; Tang et al., 2021).

Data Intensive Tomographic Reconstruction. Iterative tomographic reconstruction methods are commonly employed due to their ability to balance reconstruction quality and computational efficiency (Stiller, 2018). These techniques iteratively refine image estimates by comparing them to actual data collected from scientific instruments. At each iteration, the differences between the estimates and the measurements are used to progressively improve the reconstruction until a predefined quality criterion is met. While computationally intensive, these methods are particularly

well-suited for large, high-resolution datasets (Hidayetoğlu et al., 2019; Wang et al., 2017). Thus, an important goal is to reduce the duration of the reconstruction. A key observation is that the reconstructed image and the corresponding raw data can be partitioned into slices, which can be independently processed in parallel. This partitioning facilitates significant speed-up, as parallel tasks can process slices concurrently, with performance scaling proportionally to the number of available compute nodes.

Fault Tolerance is a Significant Challenge. The adoption of parallelization in tomographic reconstruction introduces significant challenges in fault tolerance. These workflows often run on high-performance computing (HPC) systems, which consist of thousands of compute nodes (Wang et al., 2017; Bicer et al., 2017). The large number of components, including CPUs, GPUs, memory, communication links, and storage systems, makes these systems highly susceptible to failures, ranging from software crashes to complete system outages (Cappello et al., 2014; Dubey et al., 2024). In such environments, failures are the norm rather than the exception (Cappello et al., 2014; Dubey et al., 2024). When a failure occurs, the progress of the affected task is typically lost unless appropriate recovery mechanisms are in place (Schlichting and Schneider, 1983). This issue is exacerbated when spare resources are unavailable, forcing surviving tasks to complete their slices while leaving the final reconstructed image incomplete. Restarting failed tasks from scratch can recover the image but at a high cost, potentially doubling runtimes and leading to inefficient resource utilization as many nodes sit idle for extended periods. In extreme cases, the failure rate may surpass the system's ability to make progress, trapping the workflow in an endless cycle of recomputation.

Limitations of Current Approaches. Current HPC fault tolerance mechanisms typically rely on checkpoint-restart techniques (Elnozahy et al., 2002). These systems periodically save the state of each process, allowing failed tasks to be restarted from the last checkpoint. While this reduces the amount of lost computation, it introduces I/O overhead due to frequent writes to parallel file systems (PFS), which can become a bottleneck in high-concurrency environments. Advanced checkpointing systems, such as VeloC (Nicolae et al., 2021) and SCR (Mohror et al., 2014), alleviate this issue by asynchronously writing checkpoint data to local storage and deferring PFS writes. However, these systems are optimized for tightly coupled parallel tasks, which require all tasks to restart simultaneously in the event of a failure. Tomographic reconstruction, in contrast, is an embarrassingly parallel problem. The failure of a single task does not require the restart of surviving tasks, which do not have dependencies on the failed tasks, eliminating unnecessary synchronization overhead. Recovery options include (i) restarting failed tasks sequentially after surviving tasks finish, resulting in long runtimes and inefficient resource usage, or (ii) redistributing checkpointed slices among surviving tasks. While redistribution reduces runtime and improves resource utilization, the lack of spare resources complicates recovery and load balancing. Furthermore, slices cannot be arbitrarily divided, making efficient redistribution non-trivial to implement.

Contributions. To address the limitations of existing fault tolerance mechanisms, we propose a checkpoint-restart solution specifically optimized for data-parallel patterns and thus directly applicable to parallelized tomographic reconstruction. Our approach incorporates two key innovations. First, we introduce an *asynchronous, non-blocking checkpointing* mechanism. This mechanism operates independently for each task and supports variable-sized checkpoints that reflect the evolving critical state required for recovery. As a result, tasks can continue without being blocked while critical state data is saved progressively, ensuring minimal disruption during execution. Second, in the event of a failure, we employ a *lazy, balanced redistribution of failed tasks*. Surviving tasks are allowed to continue their processing without interruption, while the checkpointed slices of the failed tasks are redistributed among the survivors in the background. We design an optimized redistribution strategy that minimizes the expensive read overheads associated with parallel file systems (PFS) by relying on direct communication between surviving tasks. This strategy ensures efficient resource utilization and maintains balanced load distribution across the available tasks. In brief, our contributions are:

- We formulate a performance model to quantify the impact of failures on parallel tomographic reconstruction time. We demonstrate that frequent failures can increase reconstruction time by up to four orders of magnitude without proper resilience mechanisms (Section 2).
- We present a detailed description of our resilient solutions, incorporating asynchronous checkpointing and lazy redistribution. These strategies allow for large-scale reconstruction with minimal overhead, even in failure-prone environments. Specifically, checkpointing reduces failure impact by up to 100×, and dynamic redistribution further improves performance by an additional 4.3× (Section 4).
- We provide a comprehensive experimental evaluation, including controlled studies and diverse failure scenarios. Our results show that our solutions reduce reconstruction time sensitivity to failure timing and frequency, minimize recovery overhead, and maintain balanced workloads across tasks. In total, these approaches reduce failure impact by up to 500×, with minimal increase in reconstruction time (no more than 3×) and <3% overhead (Section 5).

2 Background

Here, we introduce a computed tomography (CT) image analysis pipeline employed at synchrotron light sources, followed by a discussion of the computational workflow for imaging and reconstruction.

2.1 Tomographic reconstruction

Figure 1 illustrates a typical tomographic experimental setup along with its data acquisition, sinogram generation, and reconstruction stages. During a tomography experiment, a target sample is placed on a rotating stage and illuminated by an X-ray

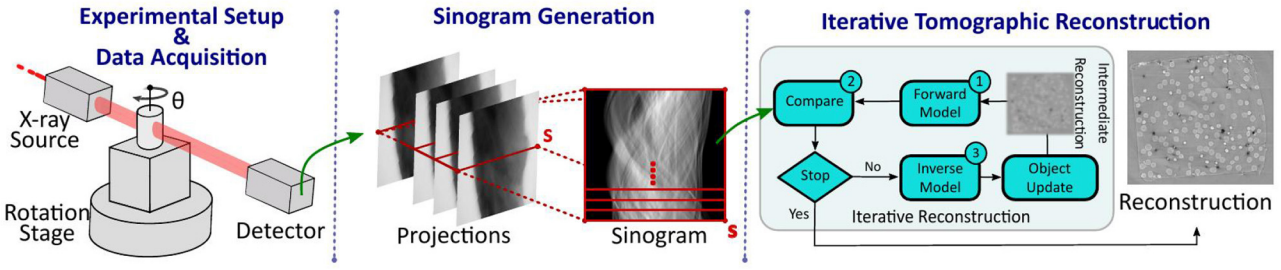

FIGURE 1
Semantics of a typical tomographic data acquisition and reconstruction pipeline (Liu et al., 2019b): 2D projections obtained during data acquisition are decomposed into independent sinograms for parallel, iterative reconstruction. The pipeline can also include a feedback loop, where reconstructed images are analyzed to refine and dynamically control the ongoing experiment.

source. As the X-rays pass through the sample, they are attenuated depending on the sample's thickness and density—denser regions cause greater attenuation, resulting in lower detector readings compared to less dense areas. The attenuated X-rays are then measured by a photon detector, producing a reading known as a *projection*.

Projections are collected at various angles of rotation, θ, typically with a fixed exposure time for each. Ideally, the experiment captures projections that fully cover the sample from all necessary angles. The attenuation of the X-ray intensity is modeled by the Beer–Lambert law:

$$I_\theta(s) = I_0(s)\exp[-p_\theta(s)]\,, \tag{1}$$

where $I_0(s)$ represents the initial X-ray intensity, $I_\theta(s)$ represents the detector readings at angle θ, and $p_\theta(s)$ corresponds to the cross-sectional projection of the sample, known as a *sinogram* (depicted in red in Figure 1). For parallel beam geometry, which is the common data acquisition technique at synchrotron radiation facilities, the sinogram contains measurements corresponding to a specific cross-section of the target sample.

The goal of tomographic reconstruction is to recover, from a set of sinograms, 2D cross-sectional images of the sample. Iterative reconstruction techniques, such as ART-family or more advanced Model-based iterative reconstruction (MBIR) techniques, are popular for recovering 2D images as they offer advantages over traditional analytical methods, especially when dealing with dynamically evolving features or limited/noisy data (Mohan et al., 2015; Nikitin et al., 2019). They are also well suited for fast reconstruction tasks with limited data, provided adequate computational resources are available (Bicer et al., 2016; Wang et al., 2016).

The iterative reconstruction techniques can also be used for the reconstruction of streaming X-ray projections, where the reconstruction tasks can process unbounded data streams and provide low-latency feedback to the downstream tasks, such as experimental control systems or domain/beamline scientists (Bicer et al., 2020). Although such tightly coupled experimentation and online data analysis pipelines are difficult to establish—mainly due to their computational, latency, and output quality requirements—they are useful for long-running experiments and real-time science-relevant data acquisitions. In this work, we focus on offline reconstruction workflows, in which the full measurement data is available during the reconstruction. However, our resiliency techniques can also be extended to online workflows.

2.2 Offline reconstruction workflow

We briefly describe the general structure of an offline tomographic reconstruction. As shown in Figure 2, the offline tomographic reconstruction workflow consists of four components.

Data acquisition collects projections from the experimental setup as a sequence of 2D images, each corresponding to a specific projection angle, θ. The size of the sequence is characterized by $X \times Y \times K$ where $X \times Y$ is X-ray projection dimensions, and K is the number of acquired projections. These parameters depend on the experimental setup and data acquisition configuration. For instance, image resolution can range from $(X \times Y) = (1{,}024 \times 1{,}024)$ to $(4{,}096 \times 4{,}096)$ pixels, depending on the target spatial and temporal resolutions, detector's specification, and sample/phenomena being observed. Further, large samples can be imaged via multiple data acquisition sessions and stitching X-ray projections, which can result in very large projections, e.g., $(20{,}000 \times 20{,}000)$ mouse brain data (Hidayetoğlu et al., 2020). The total number of projections (or the rotation angles) depends on the sample/phenomena that is being imaged and can be in the thousands range.

Data distributor generates sinograms from acquired projections and assigns them to their corresponding reconstruction tasks. The chunking is based on the rows, so sinograms are generated by slicing projections horizontally with the finest granularity. For example, if the resolution of a projection is $(X, Y) = (1{,}920 \times 1{,}080)$, then the data distributor will split it into 1,080 sinograms, each with dimension $1{,}920 \times 1$. Sinograms are identified by their row indexes. After slicing, the data distribution map sinogram row indexes to the reconstruction task, try to equalize the number of rows assigned to a task for load balancing.

Reconstruction tasks perform iterative tomographic reconstruction on assigned sinograms as described in Section 2.1. The reconstruction is performed in parallel by N tasks $T_1, T_2, ..., T_N$. The complete 3D construction is a cube of size $X \times Y \times Z$, where $X \times Y$ are the width and height of the

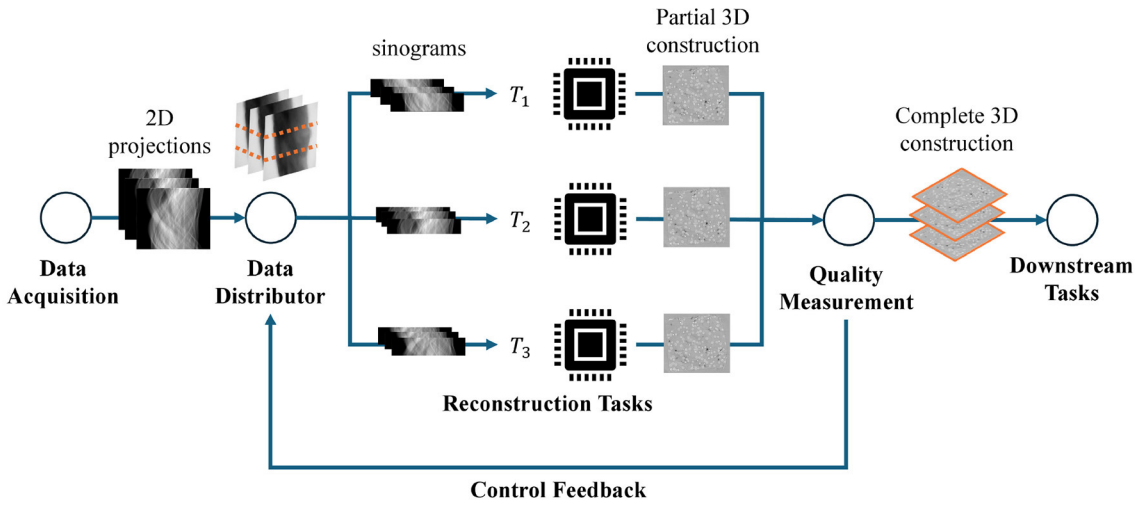

FIGURE 2
The tomography reconstruction workflow includes four components: (1) a data acquisition task, (2) a data distribution task, (3) N reconstruction tasks (here, $N = 3$), and (4) a quality measurement task.

reconstruction, identical to the dimension of projections used by the reconstruction, and Z is its depth. Reconstruction tasks create the 3D independent construction by row. That is, all sinograms with the same row index are assigned to the same tasks. These sinograms together form a complete input dataset to reconstruct a $X \times 1 \times Z$ 3D slice of the 3D construction. These slices are collected by the downstream process (e.g., quality measurement) and stacked by their row indexes to form the complete reconstruction.

***Quality measurement*:** reconstructed slices generated by reconstruction tasks are aggregated to form a complete 3D image. A quality metric, such as the Structural Similarity Index Measure (SSIM) or Mean Squared Error (MSE), is calculated to assess whether the reconstruction meets the desired quality, determining whether the process can be concluded.

2.3 Problem formulation

We now use the general workflow described above to formulate the problem in terms of computation structure, performance, and the impact of failure on reconstruction execution. Notations introduced in this section (and employed in the rest of the paper) are summarized in Table 1. Let M be the number of iterations of the reconstruction process. In each iteration, each pixel of the $X \times Y \times Z$ 3D construction is generated by adjusting the old generated pixel with the ground truth measurement collected from K projections. The computation is performed independently in parallel by N tasks. Thus, the overall reconstruction time is

$$R(N, M) = \frac{M}{N} O(XYZK), \tag{2}$$

where N and M are configurable parameters that the reconstruction workflow can adjust depending on the objective. For example, having more tasks (big N) will speed up the reconstruction time, while performing more iterations (big M) will improve reconstruction output at the cost of more computation and thus increased reconstruction time. Selecting N and M leads to an optimization problem where we want to minimize the execution time to achieve a good enough reconstruction quality. In this paper, we assume N and M have been configured optimally and focus on minimizing the impact of failures on the end-to-end reconstruction time for those optimal parameter values.

TABLE 1 Notation employed in this paper.

Symbol	Explanation
Computation workflow modeling	
X	The width of 3D reconstruction (or the number of columns of a 2D projection)
Y	The height of 3D reconstruction (or the number of rows/sinograms of a 2D projection)
Y_o	Number of outstanding sinograms
Z	The depth of 3D construction
K	Number of projections
M	Number of reconstruction iterations
N	Number of reconstruction tasks
N_s	Number of surviving tasks after failure
T_i	The i-th reconstruction task ($1 \leq i \leq N$)
$R(N, M)$	Ideal reconstruction time (no failure, see Equation 2)
Failure and resilience modeling	
μ	Per-task mean time to failure (MTTF)
λ	Per-task failure frequency ($1/\mu$)
μ_P	Application's mean time between failure (MTBF)
$C(N)$	Overhead of creating a checkpoint of current reconstruction progress
W	Checkpointing period

Failures can occur at any level of the data processing pipeline, including hardware failures, software crashes, and network anomalies (Canal et al., 2020). These failures vary in nature: some are transient, while others are unrecoverable. Some silently corrupt the workflow state, whereas others immediately interrupt execution. In this paper, we focus on *fail-stop* failures (Schlichting and Schneider, 1983), in which a task halts execution upon failure; in tomographic reconstruction, such a failure results in the loss of all progress made on the task's assigned sinograms. These failures are unpredictable and can occur at any time.

Fail-stop failures are often modeled as random processes (Benoit et al., 2022), where tasks fail independently with identical probabilities. These failures are characterized by the mean time to failure (MTTF), denoted as μ, which represents the average time a single task runs before failing. Task failure probability is typically described using an exponential distribution Exp(λ), where $\lambda = \frac{1}{\mu}$ represents the failure occurrence frequency (Dongarra et al., 2015). Consequently, the failure probability of the *entire* pipeline also follows an exponential distribution Exp($N\lambda$). The mean time between failures (MTBF) for the whole pipeline is given by

$$\mu_p = \frac{\mu}{N}. \tag{3}$$

Equation 3 shows that increasing reconstruction parallelism by adding more tasks reduces the mean time between failures (MTBF). This is intuitive, as more tasks introduce more potential points of failure, causing failures to occur more frequently. This phenomenon poses a significant challenge for achieving resilient execution at large scales. For instance, if the MTTF of a single task is approximately one week, a reconstruction pipeline with 1,000 tasks would, on average, fail in about 10 min—often shorter than the runtime of many workflows of this size (Bicer et al., 2015). Moreover, even with compute nodes designed to operate reliably for years, the sheer scale of large systems amplifies failure rates. Studies have shown that large-scale deployments, with thousands of compute nodes, can experience failures several times per day due to the aggregated likelihood of individual node failures (Cappello et al., 2014; Ferreira et al., 2011).

Current reconstruction workflow deployments, like traditional HPC applications (Benoit et al., 2022), restart the entire pipeline when a failure occurs. While this approach simplifies management, it becomes inefficient at large scales where failures are common rather than rare (as shown by Equation 3). Frequent restarts can introduce substantial overhead. Worse, since most computation infrastructure operates in shared environments, resources needed for reconstruction tasks may not always be immediately available, leading to unbounded waiting times.

To address these challenges, we want to continue reconstruction with the surviving tasks instead of restarting the entire pipeline. Achieving this resilient capability requires two additional steps:

1. *Data redistribution:* the data distributor reassigns the sinograms from the failed task to surviving tasks, ensuring that all sinograms are eventually reconstructed.
2. *Progress recovery:* the surviving tasks recover the progress made by the failed task and continue processing both the recovered sinograms and their own. This step ensures all sinograms are processed sufficiently to achieve the desired reconstruction quality.

While these extra steps mitigate the impact of failures, they introduce additional overhead to the end-to-end reconstruction process, which, as we will demonstrate below, can be significant.

2.4 Impact of failures on reconstruction time

We discuss the impact of failure on reconstruction time from two perspectives: the time required to repeat the computation performed by the failed task prior to failure and increased time due to reallocation of work from the failed task following failure.

2.4.1 Expensive progress recovery

Figure 3a shows a naive approach to progress recovery where a surviving task takes over the data of a failed task. Since the failed task's progress is lost, the surviving task must recompute from the beginning up to the point where the failed task left off. This effort is referred to as *recomputation*, and the time spent on recomputation is *wasted*, as no new results are generated during this time. To quantify the recomputation impact on end-to-end reconstruction time, we run a reconstruction simulation of 256 sinograms (640 × 640 pixels each) over 10 iterations. Figure 3b shows the average reconstruction time under various mean times to failure (MTTF), with failures injected according to the failure model presented above.

In an ideal scenario without failures, reconstruction time decreases linearly with increased parallelism. A single task needs 15,360 s (∼4.2 h) to complete. Adding more parallel tasks reduces this time significantly, achieving an ideal reconstruction time of 240 s (∼4 min) with 64 tasks (black line). However, the presence of failures drastically alters this performance. At MTTF = 10,000 s (∼2.8 h), the reconstruction time increases by 1.5× to 2× across all parallel configurations. Alarmingly, as the failure rate increases, the impact on reconstruction time grows exponentially. For instance, with one task, reducing the MTTF from 10,000 s to 1,000 s introduces 10× more failures but extends the reconstruction time by over 100× (green and blue bars on the left). Even at high parallelism, failures remain devastating. With 64 tasks, reducing the MTTF from 10,000 s to just 10 s (a three-orders-of-magnitude decrease) results in reconstruction times soaring by more than four orders of magnitude! These results highlight the catastrophic impact of frequent failures on reconstruction performance and, thus, the critical need for robust failure-resilience solutions.

A conventional approach to minimize expensive recomputation is to use checkpoints. In this method, reconstruction tasks periodically save snapshots of their progress on assigned sinograms to the underlying parallel file system. These snapshots, called *checkpoints*, record the state of task progress throughout the reconstruction pipeline. If a task fails, the remaining tasks can load the latest checkpoint to recover lost

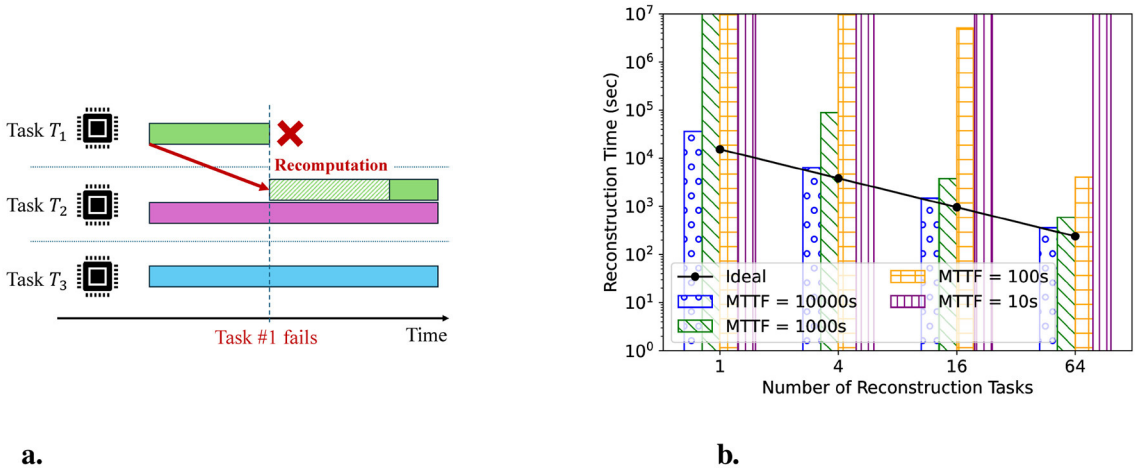

FIGURE 3
Naive resilience incurs expensive recomputation, significantly prolongs reconstruction at failures. **(a)** Progress recovery triggers expensive recomputation when failure occurs. **(b)** Impact of recomputation on execution time.

progress, skipping prior computations and significantly reducing recovery time.

However, determining the optimal checkpoint frequency is challenging. Checkpoints introduce overhead, including the time to save progress to permanent storage and communication costs to ensure consistency across tasks. Taking checkpoints too frequently increases this overhead, potentially outweighing the time savings from reduced recovery. Conversely, infrequent checkpoints—especially the extreme case of no checkpoints at all—lead to longer recomputation times, as seen earlier.

Research has long focused on balancing this trade-off between checkpointing overhead and recovery time. In a general setting, the Young/Daly formula (Daly, 2006; Young, 1974) provides an optimal checkpoint period W_{YD}, given by

$$W_{YD} = \sqrt{2\mu_p C(N)}, \tag{4}$$

where $C(N)$ is the checkpoint overhead whose complexity depends on the number of tasks participating in creating the checkpoints N and the size of the checkpoint (a constant, $X \times Y \times Z$, in our formulation).

While this formula is widely used in traditional HPC applications (Bautista-Gomez et al., 2024), it has a significant limitation in our setting. It assumes that failures result in application-wide crashes, where all tasks are equally affected. However, in our resilient framework, the pipeline continues to execute when a task fails, leading to dynamic changes in (i) the number of active tasks, N_s; (ii) the load per task; and (iii) the time spent on computation and checkpointing, $C(N)$. These dynamics render the static checkpointing period derived from the Young/Daly formula suboptimal. To address this limitation, we need a new adaptive strategy that accounts for execution-time dynamics, allowing the checkpointing configuration to adjust as failures occur.

2.4.2 Load imbalance

Throughout the reconstruction execution, the total amount of computation remains unchanged, so when a task fails, the surviving tasks must take on additional work. Since each task has limited resources, this added workload slows down their progress, significantly reducing overall reconstruction efficiency. Figure 4a illustrates this issue using three reconstruction tasks (T_1, T_2, T_3). If T_1 fails, its data is redistributed to T_2. However, due to T_2's limited resources, two issues arise: (1) T_2 must recover T_1's lost progress, stalling the reconstruction of its own data, and (2) after recovery, T_2 must process both T_1's data and its own, and thus takes much longer to complete than would have T_3. As a result, T_3 finishes early and sits idle, waiting for T_2, leading to significant resource waste.

This example highlights the urgent need for robust load-balancing and computation scheduling strategies to mitigate stragglers and reduce resource waste. However, implementing such strategies in tomographic reconstruction presents unique challenges due to the workflow's structure. *First*, per-sinogram reconstruction requires the full details of each sinogram, which prevents splitting a single sinogram across multiple tasks for parallel processing. This imposes a lower bound on how finely the data can be divided, limiting the granularity of redistribution needed for load balancing. *Second*, tomographic reconstruction relies on multiple iterations that must proceed in a strict sequential order. This prevents redistributing data across tasks in the temporal dimension by duplicating sinograms and parallelizing iterations. As a result, all sinograms must progress at a similar pace. If progress becomes uneven, some sinograms can turn into *stragglers*, which may delay the entire reconstruction workflow.

To quantify the impact of such constraints, we consider a scenario in which 10 reconstruction tasks work on 10 sinograms, each of size 640 × 640, for 10 iterations. Failures are injected immediately after the reconstruction starts, and the number of tasks stopped by failures varies from one to nine. The reconstruction

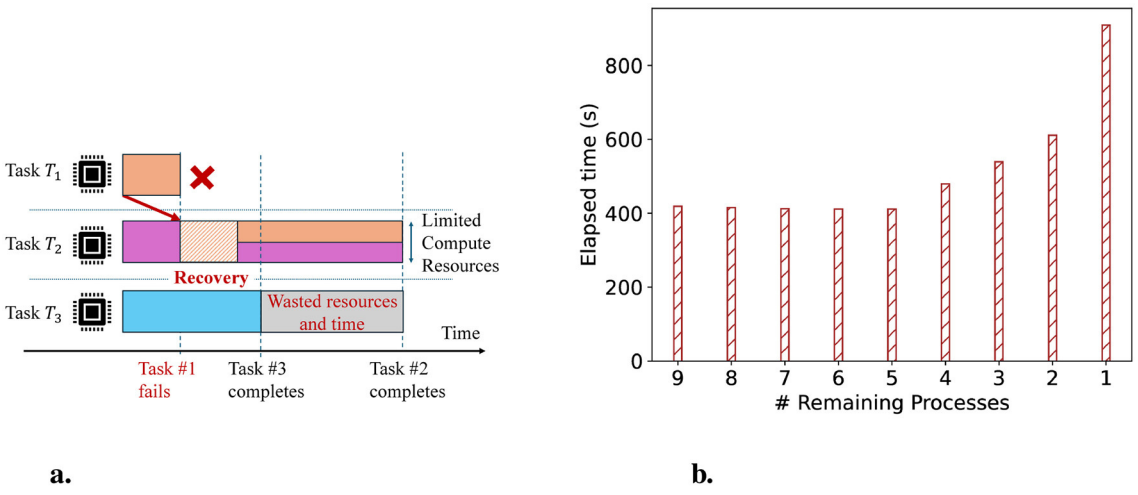

FIGURE 4
Failures potentially make load unevenly distributed among surviving tasks, negatively impacting reconstruction time. **(a)** Leftover failure tasks may cause load imbalance among remaining tasks. **(b)** Impact of load imbalance on reconstruction time.

times, shown in Figure 4b, reveal a striking pattern: there is no performance difference when the number of surviving tasks is 9, 8, 7, 6, or 5. This counterintuitive result occurs because, in all these cases, at least one surviving task must process two sinograms while the others handle only one. Due to the sinogram restriction, we cannot split or redistribute excess sinograms across underloaded tasks to balance the workload. As depicted in Figure 4a, the overloaded tasks become stragglers, determining the overall reconstruction time, while underloaded tasks finish early and sit idle, wasting resources. The inefficiency is particularly severe when considering resource usage: there is a 1.8× difference in total resources between the cases of nine and five surviving tasks, yet this does not result in any improvement in reconstruction performance. This example highlights the significant resource inefficiency caused by load imbalance and the inability to redistribute work under the constraints of tomographic reconstruction.

3 Related work

We review related work in tomographic reconstruction and HPC checkpoint-restart.

3.1 Related and similar applications

Tomographic reconstruction algorithms have been well-researched, and many advanced techniques exist (Kak and Slaney, 2001; Crowther et al., 1970; Withers et al., 2021; Fessler et al., 2000; Mohan et al., 2014). These techniques can be grouped into two categories: one-pass analytical and iterative statistical techniques. One-pass algorithms can provide fast feedback, but are prone to measurement noise and error and can generate reconstructions with significant artifacts (Bicer et al., 2017). Iterative techniques, in contrast, can provide better quality with limited/partial data, typically at the cost of additional computation (Hidayetoğlu et al., 2019; Bicer et al., 2015; Mohan et al., 2015). Methods for parallelizing these reconstruction algorithms have advanced considerably in recent years, motivated by the demands of larger problems and experimental time constraints (Hidayetoğlu et al., 2020; Wang et al., 2016; Chen et al., 2019).

Long-running experiments that generate large experimental datasets require executing large-scale distributed workflows, potentially on geographically distributed resources (Bicer et al., 2016, 2021). AI/ML models have been incorporated into such workflows to accelerate processing (Kamilov et al., 2023, 2015; Lin et al., 2024; Liu et al., 2019b; Mohan et al., 2024; Bouvier et al., 2024). However, although such workflows have been researched in the context of synchrotron radiation data analysis pipelines (Babu et al., 2023; Benmore et al., 2022; Vescovi et al., 2022), their resilient execution has seen little attention.

3.2 HPC checkpoint-restart

Scalable Checkpoint/Restart (SCR) (Mohror et al., 2014), introduces multi-level resilience strategies that take advantage of the multiple storage levels: it supports local storage, partner replication, and XOR encoding on remote nodes in addition to flushing the checkpoint data to the PFS. Fault Tolerant Interface (FTI) (Bautista-Gomez et al., 2011) is another related effort that offers similar support while adding Reed-Solomon (RS) encoding (Reed and Solomon, 1960). Both offer limited support for asynchronous checkpoint flushes between the levels and they adopt a one-file-per-process approach, which results in a large number of files that often overwhelms a parallel file system. Aggregation approaches such as MPI-IO (Thakur et al., 1999) and GenericIO (Habib et al., 2016) solve this problem for synchronous I/O. They collect the checkpointing data to a smaller number of proxy compute nodes, which in turn interact with the parallel file system and write a smaller number of files. Other approaches

such as Gossman et al. (2024) introduce aggregation support for asynchronous checkpointing.

VeloC (Nicolae et al., 2021) is a checkpointing system that improves on the multi-level resilience strategies introduced by SCR and FTI. Specifically, it focuses on providing efficient asynchronous support to mask the overhead of the multi-level resilience strategies through a background engine that implements a modular checkpointing pipeline designed to streamline the entire life-cycle of the checkpoints, from serialization of checkpointing data, to additional transformations, integrity verification and checksumming, caching, prefetching, and GPU support (Maurya et al., 2023). It mitigates interference with the application through a series of multi-threading policies that prioritize application resource utilization under competition. VeloC supports checkpointing of both embarrassingly parallel (i.e., independent checkpointing) and tightly coupled (i.e., collective checkpointing) applications. Specific optimizations can be applied for checkpointing AI models (Nicolae et al., 2020; Mohan et al., 2021; Maurya et al., 2024).

An important in the context of checkpointing systems is versioning, which is critical in the case when the latest checkpoint is not necessarily trusted. This can happen when anomalies (silent data corruption, convergence instability due to spikes, etc.) cannot be immediately detected and may be captured in the latest checkpoint, prompting the need to restart from older checkpoints. In this case, versioning needs to maintain an entire history of checkpoints, which can quickly explode to large sizes. Incremental techniques (Nicolae et al., 2011; Tan et al., 2023; Underwood et al., 2024) that store only differences in order to compact the history (Nicolae, 2022) can be used in order to improve I/O performance, scalability, and space efficiency.

4 Resilient solutions

This section introduces strategies to (i) minimize recomputation overhead and (ii) resolve resource waste and load imbalances among surviving tasks after failures. Finally, we discuss how to integrate these strategies into the reconstruction workflow to enable failure-resilient capabilities for the application.

4.1 High-level ideas

4.1.1 Checkpointing

To reduce recomputation during failure recovery, we implement checkpointing, which periodically saves snapshots of the reconstruction state to the underlying parallel file system. Additionally, we leverage the structure of the reconstruction workflow to apply advanced checkpointing techniques that minimize checkpoint creation overhead during failure-free execution:

- *Asynchronous checkpointing*: per-sinogram reconstructions are independent; each sinogram is processed independently, requiring no communication or synchronization with others to produce proper results. We take advantage of this by making checkpoints *self-contained*, embedding all necessary information to restore progress (e.g., completed iterations, sinogram indices, etc.). This design allows surviving tasks to manage checkpoints independently, eliminating computation and synchronization overhead during failure recovery. This lightweight approach simplifies the recovery process and enhances efficiency.
- *Non-blocking*: tomographic reconstruction is computation-intensive, whereas checkpointing is I/O-intensive. To minimize checkpoint creation overhead, we *overlap* these processes by offloading checkpoint creation from the reconstruction's critical path. When a task needs to create a checkpoint, it specifies the memory regions containing the checkpoint data and delegates this information to a separate process. The task then resumes its computation immediately while the checkpoint is written asynchronously in the background. This approach effectively overlaps computation with I/O, reducing the impact of checkpointing on reconstruction runtime.

We now address the critical question of how frequently checkpoints should be taken to minimize the combined overhead of checkpoint creation during failure-free execution and wasted computation when failures occur. Using the techniques described earlier, we significantly reduce checkpoint creation overhead, making it negligible compared to reconstruction time in ideal conditions (see Section 5.2.3). In such cases, the optimal strategy is to checkpoint as frequently as possible. For our setting, the highest feasible frequency is once per iteration, as reconstruction at each iteration is atomic.

However, real-world computational environments are shared, and I/O bandwidth is often contested by multiple co-located applications. This contention introduces uncertainty and can severely degrade I/O performance, making checkpoint creation overhead significant—even exceeding the time required for per-iteration reconstruction. In these cases, frequent checkpointing can block the reconstruction progress. To handle this challenge, we dynamically adjust the checkpoint period W based on a modified Young/Daly formula as follows

$$W = \sqrt{2\frac{\mu}{N_s}C(N_s)}\,, \tag{5}$$

recall $N_s \leq N$ is the number of surviving tasks at the time of reconfiguration, μ is the mean time to failure of a single task, and $C(N_s)$ is the empirical checkpoint overhead, calculated as the average overhead observed since the last reconfiguration.

This formula assumes task failures are independent and identically distributed, following an exponential (memoryless) distribution. Consequently, the mean time between failures (MTBF) for the entire pipeline is determined by the sum of the MTTFs of all surviving tasks. According to the original Young/Daly formula (Equation 4), reconfiguration based on this formula gives the optimal checkpoint period for N_s tasks. To ensure optimal checkpointing throughout execution, we reconfigure the checkpoint period W dynamically whenever a failure occurs. This adaptive approach accounts for changes in task availability and I/O performance, ensuring that checkpointing remains efficient and minimizes overhead across the entire pipeline execution.

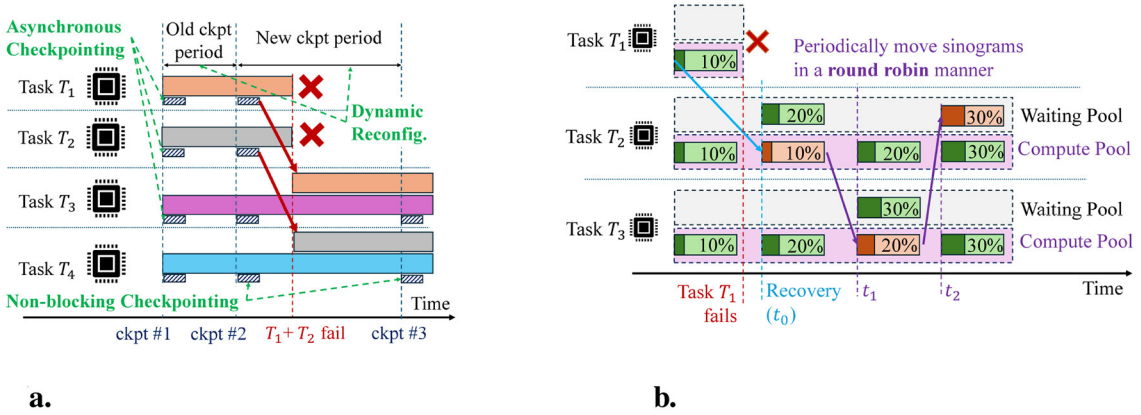

FIGURE 5
Resilient solution ideas: **(a)** Asynchronous, non-blocking checkpoint mechanism with dynamic reconfiguration avoids costly recomputation with minimum checkpointing overhead; **(b)** Dynamic data redistribution balances load among surviving tasks.

Figure 5a illustrates our proposed checkpointing solutions using an example with four reconstruction tasks: T_1, T_2, T_3, and T_4. The non-blocking checkpoint mechanism enables checkpoints to be created in the background, allowing reconstruction to proceed seamlessly during failure-free operation. When a failure occurs, tasks T_1 and T_2 stop working, but their progress is preserved in their most recent checkpoints, avoiding costly recomputation. The asynchronous checkpointing mechanism further enhances efficiency by allowing surviving tasks to recover independently and in parallel. In this example, T_3 recovers the progress of T_1 while T_4 recovers T_2, with zero inter-task communication and synchronization, minimizing recovery overhead. After recovery, the number of active tasks is halved, prompting a reconfiguration of the checkpointing period to adapt to the new task count. This dynamic adjustment ensures that both checkpoint overhead and wasted computation are minimized, maintaining overall efficiency even in the face of failures.

4.1.2 Dynamic redistribution and lazy recovery

To optimize resource usage and balance the workload among reconstruction tasks, we implement a dynamic redistribution approach based on two key ideas:

- *Lazy recovery*: when a failure occurs, surviving tasks do not immediately recover lost progress from checkpoints. Instead, they treat the checkpointed progress as the most recent state of the failed task. Recovery is embedded into future reconstruction iterations, avoiding resource locking and enabling tasks to utilize their resources more efficiently for ongoing computations.
- *Dynamic load balancing*: static data assignment can create stragglers, leading to inefficiencies. To address this, we divide data into the smallest manageable units (e.g., single sinograms) and dynamically redistribute them among reconstruction tasks. This ensures that the computational workload is evenly distributed over time, allowing all tasks to progress at a similar pace and avoiding bottlenecks for efficient reconstruction.

We implement two pools within each reconstruction task: a compute pool for sinograms currently under reconstruction and a waiting pool for completed sinograms or excess outstanding sinograms from overloaded tasks. Sinograms are assigned to these pools as follows: completed sinograms are moved to the waiting pool, while the remaining sinograms are sorted by progress. The top R sinograms with the least progress are assigned to the compute pool, and the rest go to the waiting pool. The value of R, a global parameter determined by the load distributor, is calculated as:

$$R = \lfloor \frac{Y_o}{N_s} \rfloor, \tag{6}$$

where Y_o is the total number of outstanding sinograms, and N_s is the number of surviving tasks. During computation, tasks process only the sinograms in their compute pool. Thus, distributing outstanding sinograms close to R across compute pools ensures equal computation demand and eliminates stragglers.

If Y_o is divisible by N_s, the distribution is straightforward. Otherwise, we dynamically shift the remainder sinograms from Y_o/N_s among tasks in a round-robin manner. Specifically, whenever Y_o or N_s changes, the data distributor recalculates R using Equation 6 and determines $E = Y_o - R \cdot N_s$. It then ensures each task holds at least R sinograms, assigning the E excess sinograms to the E surviving tasks with the lowest indexes. After each iteration, tasks update the progress of their sinograms. Tasks with excess sinograms retain only the R sinograms with the highest progress and shift the remainder to the next E tasks in a cyclic order, wrapping around to the initial indexes as needed. This dynamic redistribution maintains balanced workloads and ensures efficient reconstruction throughout the process.

To clarify the idea, we use an example illustrated in Figure 5b. Initially, there are $Y_o = 3$ outstanding sinograms, each being processed by one of the three active tasks T_1, T_2, and T_3 ($N_s = 3$). The reconstruction consists of 10 iterations, with each iteration advancing the progress of a sinogram by 10%. At the start, all three sinograms are at 10% progress.

Before completing the first iteration, Task T_1 fails, leaving its sinogram stuck at 10% progress. By the time the failure is detected

at t_0, T_2 and T_3 have proceeded to the next iteration. The failure triggers recovery, including updating $N_s = 2$ and $R = \lfloor \frac{Y_o}{N_s} \rfloor = 1$ (Equation 6). This means each surviving task, T_2 and T_3, will hold one sinogram each, and the sinogram left by T_1 will be periodically shifted between them based on the round-robin rule. Thus, at t_0, T_2 restores T_1's sinogram, but due to lazy recovery, the sinogram retains its checkpointed progress (10%) rather than triggering recomputation immediately, while the other sinograms are already at 20%. Since the recovered sinogram has the least progress, it is placed in T_2's compute pool and is processed alongside the sinogram held by T_3.

After completing the iteration at t_1, T_2 has two outstanding sinograms but $R = 1$, so it must shift a sinogram with the lowest progress to T_3. Since both sinograms held by T_2 are at 20%, it just randomly picks one of them; here, we assume the recovered sinogram is shifted. Then, both tasks proceed to the next iteration, where T_3 places the shifted sinogram (now at 20%) into its compute pool for further processing, advancing its progress to 30%. Meanwhile, the sinogram originally assigned to T_3 remains in the waiting pool at 30%. In the next iteration (t_2), the recovered sinogram is shifted back to T_2, and the reconstruction continues in this pattern.

By t_2, all sinograms have reached equal progress (30%) despite the recovered sinogram initially lagging behind due to T_1's failure. This example illustrates how dynamic redistribution prioritizes sinograms with the least progress, gradually equalizing their progress while keeping all tasks active. Thus, this approach eliminates stragglers, optimizes resource utilization, and ensures the reconstruction pipeline operates efficiently.

4.2 Unified algorithm

Building on the illustrative examples above, we now describe how we integrated our resilient solutions into the tomographic reconstruction pipeline. Our approach comprises two components: the resilient data distributor and reconstruction tasks.

```
while true do
  if Y_o or N_s change then
    if N_s changes then                ▷ Failure detected
      Trigger Recovery
      Collect recovered sinograms
      Reassign recovered sinograms
      Send recovered sinograms to assigned tasks
      Reconfigure Checkpoint Period W
    end if
    if Y_o = 0 then
      complete()
    else
      R ← ⌊Y_o / N_s⌋
      SendToTasks(Y_o, R)
    end if
  end if
end while
```

Algorithm 1. Resilient solution implementation at data distributor.

Algorithm 1 outlines the pseudo-code for the resilient data distributor implementation. The distributor operates in an infinite control loop, responding to changes in the number of outstanding sinograms (Y_o) or surviving tasks (N_s), triggered by either a completed sinogram reconstruction or task failures, respectively.

When a task fails (i.e., N_s changes), the distributor initiates recovery by instructing surviving tasks to load checkpoints of data lost due to the failure. These tasks then return the checkpointed data to the distributor for load balance-aware redistribution. Tasks are assigned at least $R = \lfloor \frac{Y_o}{N_s} \rfloor$ sinograms, with any excess ($E = Y_o - R \cdot N_s$) distributed to the first E surviving tasks with the smallest indexes. This strategy ensures near-optimal load balancing, minimizing the difference in workload across tasks. The distributor then transfers the assigned sinograms back to the reconstruction tasks accordingly to resume processing. Finally, the checkpoint period W is adjusted with the new N_s to minimize the overhead of upcoming checkpoint creations.

If no failures are detected, the distributor updates Y_o and the compute pool size R to assist reconstruction tasks in efficiently shifting sinograms (discuss below). Once $Y_o = 0$, all sinogram processing is complete, and the data distributor halts, concluding the process.

```
while true do
  if RecoveryIsTriggered() then
    Recover lost sinograms from checkpoints
    Send recovered sinograms to data distributor
    Receive sinograms from distributor
    Reconfigure checkpoint period W
  end if
  Y_o, R ← ReceiveFromDataDistributor()
  if Y_o = 0 then
    complete()
  else
    ComputePool, WaitingPool ← Assign(R)
    Reconsruction(ComputePool)
    UpdateProgress(ComputePool)
    if current time − last ckpt time ≥ W then
      Checkpoint(ComputePool, WaitingPool)
    end if
    ExceedSinograms ← collectExceedSinograms(R)
    Shift(ExceedSinograms)
  end if
end while
```

Algorithm 2. Resilient solution implementation in reconstruction tasks.

Algorithm 2 outlines the resilient implementation for reconstruction tasks. The reconstruction is an iterative process. At the start of each iteration, tasks check for failures reported by the data distributor. If a failure is detected, the data distributor broadcasts the indexes of failed tasks. Let S be the number of failed tasks whose checkpoints must be read for recovery. If $S \geq N_s$, each surviving task loads checkpoints from one failed task in parallel, updates S, and repeats this process until $S < N_s$. Then, the first S tasks with the smallest indexes load the remaining checkpoints.

These loaded checkpoints are sent to the data distributor for redistribution as described above.

The distributor then sends newly assigned sinograms to surviving tasks. To proceed, each task updates the load balance factor R from the data distributor and divides its sinograms into compute and waiting pools. The top R sinograms with the least progress are assigned to the compute pool, while the rest go to the waiting pool. Tasks process the compute pool, update progress, notify the data distributor upon sinogram completion, and create checkpoints. If a task has more than R outstanding sinograms, it shifts the excess (those with the least progress) to the next surviving tasks with the smallest cycled indexes. This shifting serves to (1) move low-progress sinograms to underloaded tasks for continued computation, ensuring sinograms are constructed at a similar pace, and (2) balance workloads across tasks, preventing stragglers and resource waste, thereby minimizing reconstruction time.

4.3 Resilient implementation

We developed a minimalistic workflow to generate a tomographic reconstruction pipeline for resiliency tests,[1] which is based on the Trace reconstruction engine (Bicer et al., 2017, 2020). The pipeline supports three iterative reconstruction algorithms: Algebraic Reconstruction Technique (ART) (Gordon et al., 1970), Simultaneous Iterative Reconstruction Technique (SIRT) (Kak and Slaney, 2001), and Maximum Likelihood Expectation Maximization (MLEM) (Dempster et al., 1977). Trace parallelizes the execution of these algorithms via partitioning sinogram and using block iterations (on projections). The workflow can continuously update reconstruction data as more data becomes available. The code is CPU-based and is optimized for shared and distributed memory parallelism. While this application employs X-ray imaging data generated at the Advanced Photon Source (APS) for tomographic tasks, it is generic in capturing a wide class of tomography applications, including scanning transmission electron microscopy (STEM) (Al-Najjar et al., 2022).

Our workflow design is based on the architecture described in Figure 2. It comprises three main components: data acquisition, `DAQ`, for simulating data acquisition from experimental setup; distributor, `DIST`, responsible for normalizing incoming data from `DAQ` and distributing to parallel reconstruction tasks; and the reconstruction task, `SIRT`, which performs partial 3D volume reconstruction using the data received from `DAQ`. In our workflow, `DAQ` and `DIST` components have only one instance, whereas there can be thousands of `SIRT` instances, depending on the scale of the reconstruction (Bicer et al., 2017). In our implementation, we containerize these three workflow components, for portability, scalability, and easy (resiliency) testing.

The resilient features are implemented in C++, utilizing the Message Passing Interface (MPI) for data exchange between workflow components. For non-blocking checkpointing, we use VeloC (Nicolae et al., 2019). In each reconstruction task, we allocate dedicated memory regions for the reconstruction output and register them with VeloC whenever the checkpoint configuration changes.

When a task is about to create a checkpoint, it populates the registered memory regions with the most up-to-date data needed for recovery. This data includes (i) the reconstruction results, (ii) the ID of corresponding sinograms in the results, and (iii) progress indicators, such as the number of completed iterations for each sinogram. After updating the memory regions, the task sends a write signal to VeloC and immediately resumes the reconstruction process. VeloC handles the checkpointing asynchronously by using a separate process to flush the registered memory regions to permanent storage (e.g., a parallel file system). Each saved checkpoint is identified by a tuple (T_i, v) where T_i is the ID of the task that created the checkpoint, and v is an auto-incrementing number indicating the version of the checkpoint.

When a failure occurs, the data distributor will notify the surviving tasks of the IDs of the failed tasks. If a surviving task is designated to recover a failed task's data, it allocates memory for recovery and registers this memory with VeloC. The task then sends a read request to VeloC, specifying the ID of the task to be recovered. VeloC searches for the latest version of the checkpoint corresponding to that task ID, loads it into the registered recovery memory region, and notifies the task. The task can then proceed with data redistribution as described in Section 4.1.2.

5 Evaluation

We perform experiments to evaluate the performance of the resilient solution discussed in Section 4. We will first present the evaluation methodology, including objectives, workloads, and execution environment configurations in Section 5.1, followed by the results in Section 5.2.

5.1 Methodology

5.1.1 Objectives

We perform experiments with two objectives. *First*, we create well-controlled execution scenarios to gain insights into the impact of failures on reconstruction execution as well as to quantitatively evaluate our resilient solution ideas and implementations. *Second*, we generate failures based on standard randomization processes while varying experimental conditions, including reconstruction sizes, algorithms, and projection datasets, to assess the applicability and robustness of our resilient solutions in diverse real-world settings.

5.1.2 Workloads

We used two sample datasets from TomoBank (De Carlo et al., 2018) to create workflow loads. The first, **Spheres**, comprises 1,500 X-ray projections (i.e., rotation angles) of borosilicate glass spheres encased in a polypropylene matrix (Spheres, 2024). Each projection is an image with dimensions (2,048 × 2,048), resulting

1 The code base can be accessed at https://github.com/diaspora-project/aps-mini-apps.

in an overall dataset size of (1,500, 2,048, 2,048). The second dataset is the **Chip** dataset corresponding to a portion of an electronic circuit (Nano CT, 2024) with 1,204 (2,448 × 2,448) projections. In our first controlled experiments, we down-sampled the datasets to manage the execution scale, simulate various failure scenarios, and reduce experimentation time. We used the full-scale datasets in later experiments to evaluate our resilient solutions in practical settings.

5.1.3 System configurations

All experiments are conducted on the Polaris supercomputer at Argonne National Laboratory. To ensure full control over the experimental environment, the reconstruction pipelines are deployed on dedicated compute nodes with no co-located applications. Each compute node features a single AMD EPYC "Milan" processor with 64 physical cores and 512 GB of memory. Each reconstruction task is assigned to one physical core, and the compute nodes are interconnected by a 200 Gbps high-speed network. Checkpoints are managed by VeloC operating in asynchronous mode (Nicolae et al., 2019) and stored in a Luster File System, making them accessible to all reconstruction tasks.

5.1.4 Tomographic deployment approaches

We run experiments using the following tomographic reconstruction pipelines.

- *Naive (baseline)*: naive resilient implementation. When a failure is detected, recovery starts immediately, reassigning lost data of a failed task to an arbitrary surviving task. The selected task reconstructs the lost data from scratch before continues making progress.
- *Ckpt*: similar to *Naive*, except the selected task reconstructs the lost progress from the latest checkpoint. Checkpoints are created and configured following ideas presented in Section 4.1.1.
- *Balance-aware*: similar to *naive*, except lost data are spread evenly among surviving tasks according to our dynamic load balancing algorithm discussed in Section 4.1.2.
- *Lazy*: similar to *Naive*, except the progress recovery does not start immediately but gets deferred according to our lazy recovery algorithm, as discussed in Section 4.1.2.

Note that *Ckpt*, *Balance-Aware*, and *Lazy* implementation are orthogonal, so they can be stacked to yield better performance. Thus, we will conduct controlled experiments with them separately in Sections 5.2.1, 5.2.2 to evaluate their effectiveness and integrate them latter in Section 5.2.3 according to our resilient algorithm in Section 4 to see their combined efficiency in practice.

5.1.5 Metrics

We evaluate resilient solutions based on four metrics.

- *Elapsed time:* the end-to-end reconstruction time to measure the performance of a resilient solution under failure.
- *Number of failures:* the number of failures that occur throughout the reconstruction process. The metric indicates the impact of failure on the construction process.
- *Balance:* calculated as follows

$$\text{Balance} = \frac{L - S}{S}, \tag{7}$$

where L is the largest number of sinograms assigned to a task and S is the smallest number of sinograms assigned to a task at the same moment or interval (assuming the sinogram-to-task assignment does not change throughout the interval). The metric allows us to investigate the balance of load among tasks with Balance $= 0$ indicating perfect even load distribution.

- *Overhead:* The additional time that a reconstruction pipeline spends without making progress (e.g., recomputation, making checkpoint, or communication for failure recovery). The metric indicates the required cost to achieve resilient goals.

5.2 Experimental results

We present the experimental results in two stages. First, we evaluate the reconstruction pipeline in a controlled environment with deterministic failure injections to analyze their impact on reconstruction time and assess the effectiveness of our solutions in mitigating these effects (Sections 5.2.1, 5.2.2). Second, we introduce randomized failure injections to evaluate the practical efficiency of our solutions under realistic conditions (Section 5.2.3).

5.2.1 Checkpointing effectiveness

We evaluate the impact of failures on reconstruction time by manually injecting failures into a pipeline processing 64 sinograms (640×640 pixels each) with 64 parallel reconstruction tasks (one sinogram per task) over ten iterations. Failures are introduced at varying points in the pipeline's progress (measured as the fraction of total completion) and by varying the fraction of tasks affected (up to 64 tasks). The results, shown in Figure 6, compare recomputation overhead (normalized against ideal execution without failures) in two scenarios: without checkpointing (Figure 6a) and with checkpointing (Figure 6b).

As expected, failures affecting more tasks lead to higher overhead. For instance, when a failure occurs at 10% progress, the overhead increases by 6.8× as the fraction of affected tasks rises from 10 to 90%. Additionally, the timing of the failure significantly impacts the results. Without checkpointing, failures occurring late in the pipeline incur drastically higher overhead compared to earlier failures. As shown in Figure 6a, a failure at 90% progress results in an overhead ranging from 83 to 221%.

In contrast, checkpointing dramatically reduces overhead and minimizes sensitivity to failure timing. As illustrated in Figure 7a, for most cases, the overhead with checkpointing remains below 22%, even for failures at 90% progress—delivering performance four to ten times faster than the no-checkpointing scenario. This improvement stems from checkpointing's ability to preserve progress: after a failure, only computations after the last checkpoint need to be recomputed, while earlier progress is retained. In comparison, without checkpointing, all progress made by affected

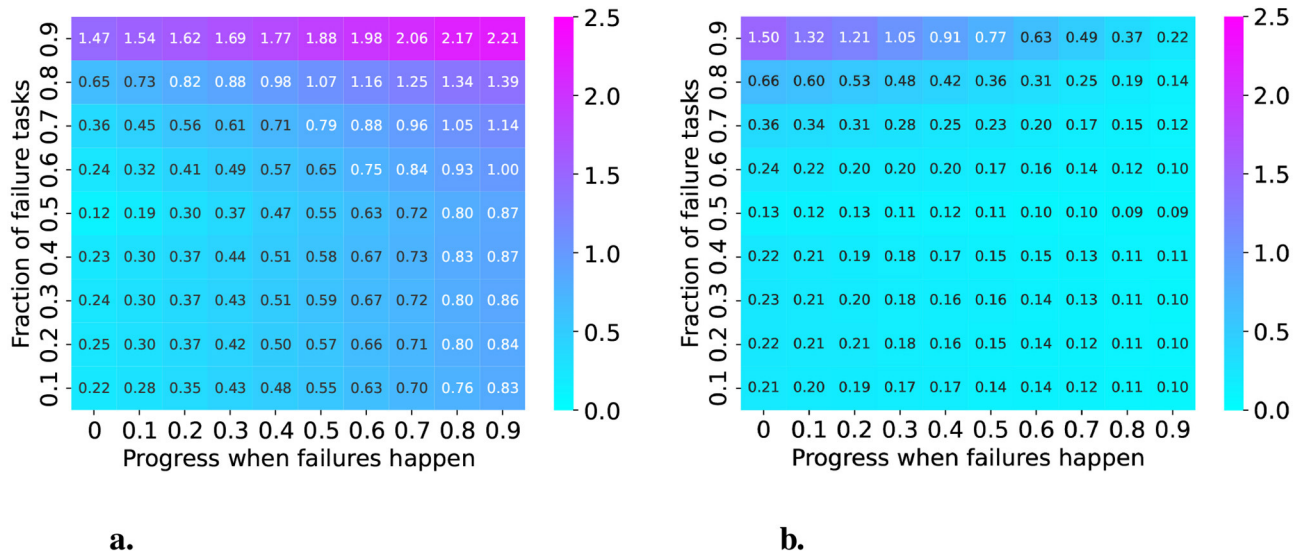

FIGURE 6
Checkpointing efficiency: overhead is bounded by checkpointing recovery, while without checkpointing, the later the failure, the worse the effect on reconstruction time. **(a)** Without checkpointing. **(b)** With checkpointing.

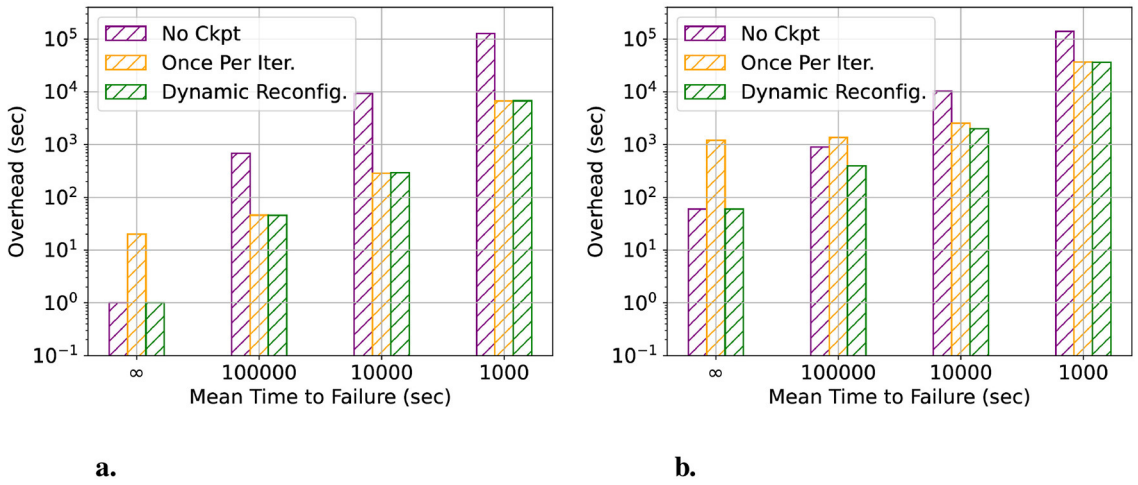

FIGURE 7
Checkpoint reconfiguration efficiency: By dynamically adjusting the checkpoint period, the combined checkpoint overhead and wasted computation are minimized under various MTTF scenarios, even when the reconstruction is under I/O contention. **(a)** Without I/O contention. **(b)** With I/O contention.

tasks is lost, requiring full recomputation of the lost data and significantly increasing overhead, especially for failures occurring late in the process.

Next, we explore the effectiveness of the dynamic checkpoint reconfiguration by manually injecting failures following an exponential distribution into a reconstruction pipeline processing 64 sinograms (each 640 × 640 pixels) with 64 parallel reconstruction tasks (one sinogram per task) over 20 iterations. To handle failures, the pipeline implements two checkpointing strategies: (i) our proposed *dynamic reconfiguration* strategy (described in Section 4.1.1), and (ii) *static checkpoint* based on a preconfigured period with two extreme configurations: no checkpointing (infinity) and checkpointing at the highest frequency (once per iteration). Figure 7 presents the combined overhead and wasted computation under varying MTTF (μ) for two scenarios: (1) no I/O contention and (2) with I/O contention (where each checkpoint creation overhead is manually increased to match per-iteration reconstruction time).

Under failure-free execution (MTTF = ∞), checkpointing adds unnecessary overhead, making the No Checkpoint strategy optimal. However, as failures begin to occur with decreasing MTTF, checkpoints become critical to reduce wasted computation. At MTTF = 1,000 s, static checkpointing at the highest frequency (once per iteration) achieves the lowest overhead.

When there is no I/O contention, our checkpointing implementation introduces insignificant overhead (more in

Section 5.2.3), so wasted computation dominates the combined overhead, and frequent checkpointing is optimal. As shown in Figure 7a, static checkpointing at the highest frequency achieves the best results. However, when I/O contention emerges, the situation changes dramatically. Checkpointing overhead becomes significant, and blindly checkpointing at the highest frequency is counterproductive. For example, at MTTF = 100,000 s (in Figure 7b), frequent checkpointing performs as poorly as having no checkpoint at all due to I/O bottlenecks. The results demonstrate that static checkpointing, while simple, is insufficient for handling dynamic execution environments. A configuration that works optimally in one scenario becomes inefficient when execution conditions change (e.g., task failures or I/O contention).

In contrast, our dynamic reconfiguration strategy overcomes these challenges by dynamically adjusting the checkpoint frequency based on decisive factors, including task MTTF, the number of surviving tasks after failures, and real-time checkpoint overhead influenced by I/O contention. This adaptability allows the strategy to maintain efficiency across varying execution conditions.

As shown in Figure 7, the Dynamic Reconfiguration strategy consistently matches or outperforms the best static checkpointing across all MTTF values and I/O contention scenarios, remarkably achieving at least 3× lower overhead compared to the worst static configuration. The results confirm the essential of a self-adaptive strategy like ours to efficiently balance checkpointing overhead and failure recovery in unpredictable, large-scale systems.

Takeaway: *Checkpointing avoids recovering lost progress from scratch, significantly reduces recomputation overhead and mitigates sensitivity to failure timing. However, determining the optimal checkpointing period is heavily influenced by uncertain execution conditions, requiring dynamic reconfiguration to maintain efficient and resilient execution.*

5.2.2 Dynamic distribution effectiveness

We evaluate the effectiveness of dynamic redistribution by focusing on two key aspects: (i) load-balancing-aware data redistribution and (ii) lazy recovery. To measure the impact of load balancing, we conduct an experiment with 64 tasks, reconstructing 256 sinograms (640 × 640 pixels each) over 10 iterations. At 10% progress, we manually inject a failure, varying the fraction of affected tasks. By introducing the failure early, we ensure that the remaining 90% of the reconstruction process is influenced by the pipeline's response to failure, providing a clear assessment of the redistribution approaches.

Upon detecting a failure, the pipeline either (i) naively redistributes data from failed tasks to remaining tasks using a one-to-one mapping (i.e., the *Naive* approach) or (ii) employs the balance-aware dynamic redistribution approach described in Section 4.1.2 (i.e., the *Balance-Aware* approach).

The results of our experiments, shown in Figure 8a, allow us to compare the reconstruction times and load balance (see Equation 7) achieved by the two approaches as a function of the fraction of tasks that are failed at 10% progress. We find that the balance-aware approach effectively maintains an even workload distribution, with its balance indicator consistently close to zero. Thus, it ensures efficient resource utilization and minimizes reconstruction time. In contrast, the naive approach assigns sinograms without considering workload disparities, resulting in significantly higher balance indicators and up to 38% longer reconstruction times. These results highlight the effectiveness of the balance-aware approach in optimizing performance and handling task failures.

Next, we evaluate the effectiveness of lazy recovery by periodically injecting failures into a reconstruction pipeline with 64 tasks processing 64 sinograms (640 × 640 pixels each) over 10 iterations. This setup highlights the impact of recovery overhead, as reconstruction stalls during recovery due to resources being spent on loading checkpoints and recomputing lost progress. Long recovery increases the pipeline's vulnerability to additional failures, compounding the recovery overhead and further delaying reconstruction. Worse, if failures frequently occur during recovery, the overhead stacks, creating a vicious cycle that significantly prolongs reconstruction time.

The results of this second set of experiments are given in Figure 8b, which shows reconstruction time for different MTTF values in two scenarios: (i) *immediate recovery*, where lost progress is restored from checkpoints immediately after failure detection, and (ii) *lazy recovery*, where progress is gradually restored during subsequent reconstruction iterations. When a failure occurs, one surviving task is terminated, and the solid lines in the figure indicate the total number of failures introduced. If all tasks are terminated, the pipeline restarts from the most recent checkpoints.

When the time between failures is above eight seconds, the number of terminated tasks remains low (fewer than 10), and failures have minimal impact on reconstruction time. However, when failure intervals shorten, the pipeline becomes increasingly prone to overlapping failures during recovery. This overlap destroys partial recovery progress, requiring repeated recomputation of the same data and exponentially increasing recovery overhead.

The results clearly show the advantage of lazy recovery in such cases. Even with time between failures as short as four seconds, lazy recovery reduces reconstruction time by up to 4× compared to immediate recovery, as it avoids the compounding effects of stacked recovery overhead. This demonstrates the robustness of lazy recovery in mitigating the impact of frequent failures and ensuring efficient pipeline performance.

Takeaway: *Dynamic redistribution and lazy recovery ensure balanced workloads and reduce sensitivity to frequent failures, delivering up to 4× improvement over naive resilient solutions.*

5.2.3 Resilient solution efficiency

Next, we evaluate the combined efficiency of all resilient solutions by integrating them into the reconstruction pipeline and testing them under an uncertain environment with random failure occurrences modeled using a Poisson process with task failures generated by identical, independent Exponential distributions. Figure 9a shows the reconstruction time of 64 tasks processing

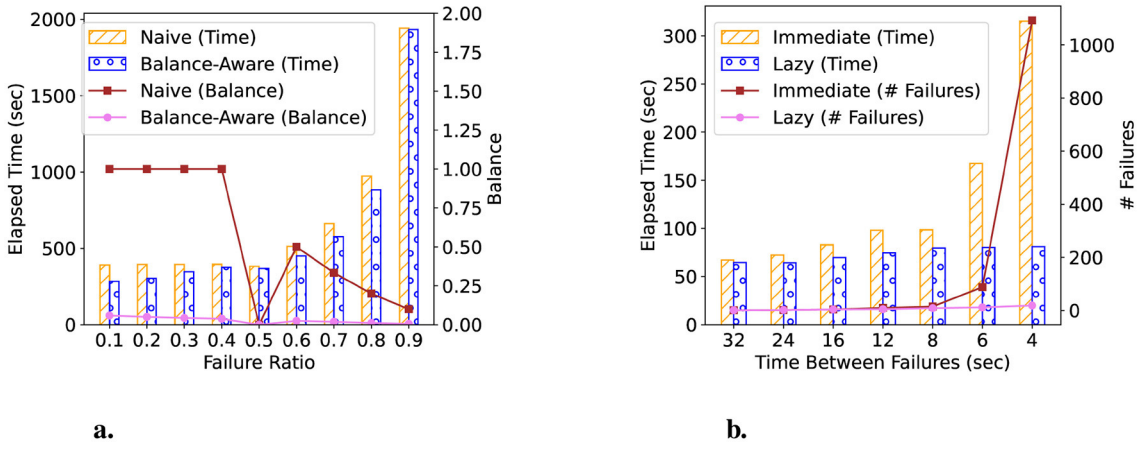

FIGURE 8
Effectiveness of two methods for responding to failures. *Balance-aware dynamic distribution* keeps load evenly distributed among surviving tasks while *lazy recovery* eliminates the compounding effect of high-frequent failure occurrences. They improve resilient efficiency by up to 38% and 4×, respectively. **(a)** Balance-aware redistribution. **(b)** Lazy recovery.

64 sinograms (640 × 640 pixels each) over 10 iterations under varying MTTF μ. We compare four pipeline implementations: (1) no resilience, (2) reconstruction with checkpointing (+*Ckpt*), (3) reconstruction with checkpointing and load balancing (+*Ckpt* +*Balance*), and (4) reconstruction with checkpointing, load balancing, and lazy recovery (+*Ckpt* +*Balance* +*Lazy*).

Without resilient solutions, reconstruction time increases exponentially as MTTF decreases. Starting from 60 s in a failure-free environment, the reconstruction time surges to over 100,000 s (>27 h, a four-order-of-magnitude slowdown) at MTTF of 10 s. Introducing checkpointing dramatically reduces this overhead by preserving progress and avoiding full recomputation of lost work. At MTTF of 10 s, checkpointing reduces reconstruction time to under 1,000 s—a 100× improvement compared to no resilience.

Adding dynamic redistribution with balance-aware assignment and lazy recovery further enhances performance, reducing reconstruction time by an additional 4.3×. Under the extreme condition of an MTTF of 10 s, 6× shorter than the pipeline's ideal execution time, the fully resilient pipeline completes the reconstruction in just 184 s, only 3× slower than ideal execution and over 500× faster than the naive resilient baseline. The results confirm the efficiency of the proposed resilient solutions in practical settings. Furthermore, the solutions are complementary and work seamlessly together. The observed improvements align with those reported in controlled experiments, showing that stacking these techniques consistently enhances overall efficiency. This confirms that combining these solutions can provide robust and scalable resilience.

These remarkable improvements come with minimal overhead. As shown in Figure 9b, the combined checkpointing and communication overhead for dynamic redistribution remains consistently below 3% of the ideal reconstruction time across all failure scenarios. This negligible cost is vastly outweighed by the dramatic reductions in reconstruction time, especially under extreme failure conditions.

Takeaway: *Integrating checkpointing, balance-aware dynamic redistribution, and lazy recovery into reconstruction pipelines dramatically improves resilience, achieving over 500× faster reconstruction under extreme failure conditions with minimal overhead (<3% of ideal runtime), ensuring robust and practical failure handling.*

5.2.4 Applicability

Finally, we evaluate the applicability of our resilient solutions by integrating all proposed mechanisms—checkpointing, dynamic load balancing, and lazy recovery—to support reconstruction workflows across diverse execution scenarios. The objective is to ensure our solutions maintain efficiency under different conditions, demonstrating their viability for real-world tomographic reconstruction. To achieve this, we begin with a base configuration (64 reconstruction tasks processing 64 sinograms from the Spheres dataset in 10 iterations using the ART algorithm) and systematically alter key parameters, including sinogram size, dataset, and reconstruction algorithm. We then compare end-to-end reconstruction time and cost against the base setting and ideal failure-free execution to assess robustness and applicability.

Robustness against data size. We reuse the efficiency evaluation setup (Section 5.2.3), employing the ART algorithm with 64 tasks reconstructing a 3D volume from 64 sinograms over 10 iterations. Sinogram sizes vary among 640×640, 1,280 × 1,280, and 2,048 × 2,048 to assess scalability. The end-to-end reconstruction time and overhead are depicted in Figure 10. Changes in sinogram size have minimal impact on the efficiency we previously observed in Section 5.2.3.

In terms of *performance*, Figure 10a illustrates the actual reconstruction time under failure, normalized by the ideal runtime without failure. Our solution effectively mitigates failure across all sinogram sizes. As the mean time to failure (MTTF) decreases

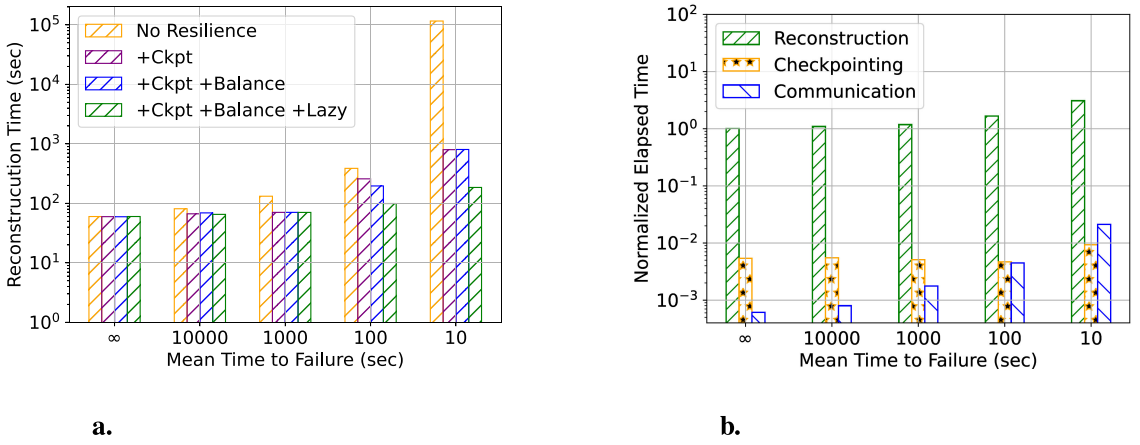

FIGURE 9
Overall efficiency **(a)** and overhead breakdown **(b)** for our resilient application as a function of mean time to failure. Our resilient solutions significantly reduce the impact of failure on reconstruction time with a negligible overhead.

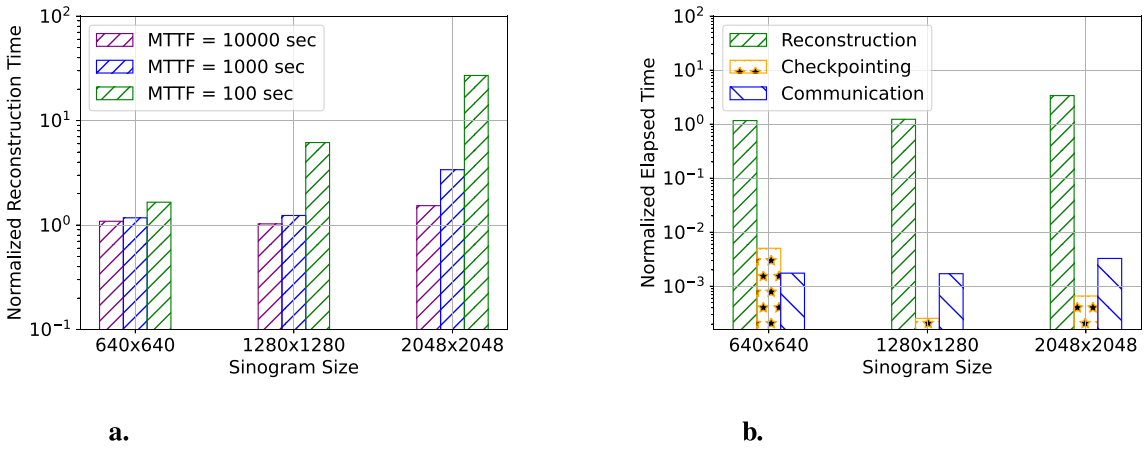

FIGURE 10
Reconstruction time and overhead under various failure scenarios, for varying sinogram sizes. In all scenarios, our solutions ensure completion at an acceptable reconstruction time with insignificant overhead. **(a)** Reconstruction time. **(b)** Overhead (MTTF = 1,000).

from 10,000 to 100 s (a 100× severity increase), the reconstruction time increases by an order of magnitude slower, only 1.5× to 16×. Notably, the failure impact prolongs reconstruction time by at most 6.2×, except for the outlier case of 2,048 × 2,048 sinograms at MTTF = 100 s, where the impact reaches 27× (rightmost green column). However, even in this extreme case, the prolonged reconstruction time is a necessary trade-off. Without resilience, completing the reconstruction would be virtually impossible. The ideal reconstruction time in this setting is 2,400 s, 24× the MTTF. This means the probability for a task to complete without failure, according to the Poisson process, is only $p = \exp(-\frac{2,400}{100}) = 3.78\times10^{-9}\%$, making successful execution of a 64-task pipeline infeasible. Our solution ensures completion, despite high failure rates, at an acceptable reconstruction time.

In terms of *cost*, Figure 10b breaks down time spent on checkpointing, communication, and computation at MTTF = 1,000, normalized by the ideal reconstruction time. As seen in Figure 9a, resilience mechanisms contribute to less than 1% of meaningful computation. Interestingly, as sinogram size increases from 640×640 to 1,280×1,280, the overhead ratio decreases by approximately 3.5×. This is because larger sinograms require longer computation, allowing our asynchronous, non-blocking mechanisms to overlap checkpointing with computation more effectively, improving cost efficiency.

Reconstruction algorithms support. We further evaluate how well our resilience mechanisms generalize across different reconstruction algorithms. For the sake of generality, we consider algorithms for two popular beam geometries: parallel-beam and cone-beam.

For the *parallel beam*, we evaluate three algorithms: ART, SIRT, and MLEM. We reconstruct 64 sinograms (2,048 × 2,048) with 64 tasks over 10 iterations. Figure 11a presents the reconstruction times under various MTTF values. Differences among algorithms are negligible, as both MLEM and SIRT, despite implementation

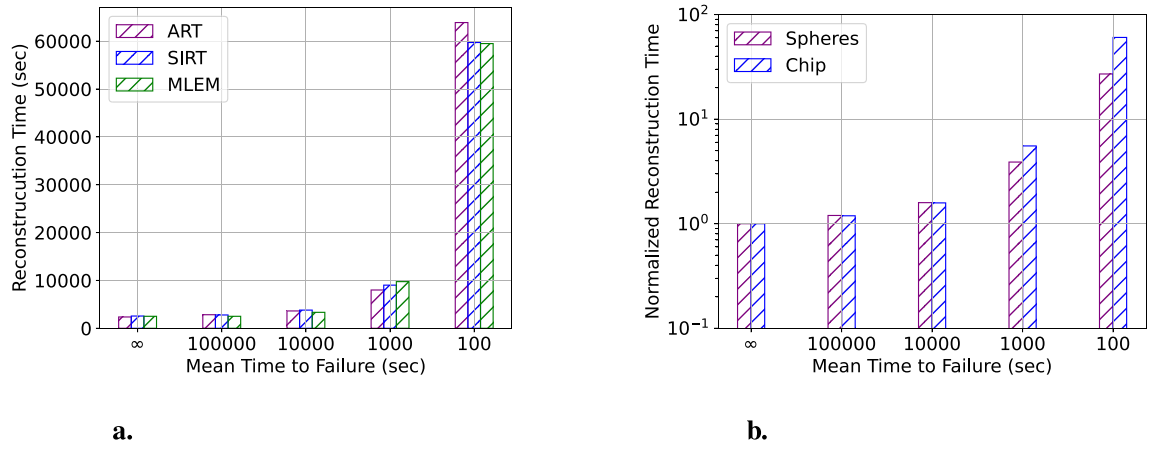

FIGURE 11
Reconstruction times under varying failure rates for different reconstruction algorithms **(a)** and datasets **(b)**. Despite changes in algorithm and input data, the reconstruction times exhibit only minor variations.

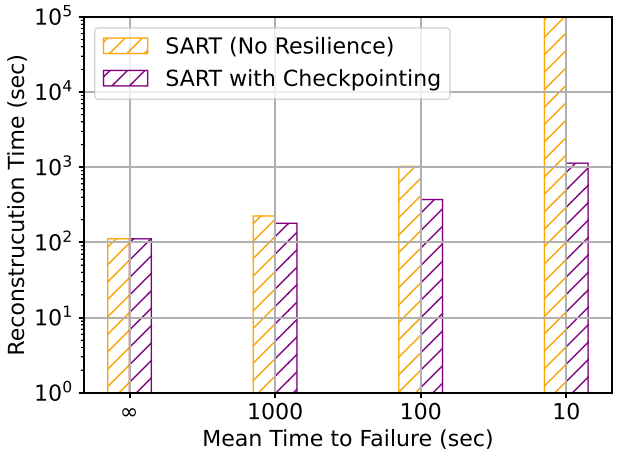

FIGURE 12
The reconstruction time of cone-beam geometry, with and without checkpointing, varying mean time to failure. Our checkpointing solution dramatically reduces reconstruction time under failures by up to 100×.

differences, follow an iterative pattern where intermediate states serve as checkpoints. This makes our checkpointing mechanism just as effective for these algorithms as it is for ART. Furthermore, like ART, MLEM and SIRT are embarrassingly parallel, benefiting equally from dynamic load balancing and lazy recovery.

In the *cone-beam* setup, cross-sinogram dependencies prevent us from directly applying load-balancing and lazy recovery techniques for resiliency. As a result, among the three solutions evaluated in Section 5.2.3, only checkpointing can be applied without compromising the correctness of the original algorithm. Figure 12 presents the reconstruction time for 360 projections processed by 16 tasks over 10 iterations, using a 100,000-s timeout (i.e., reconstruction is forcefully terminated if execution exceeds 100,000 s, ∼27 h, or 1,000× the ideal execution time of 110 s). We evaluate two (block-iterative versions of) SART algorithms: (i) a naive MPI-based implementation of the SART algorithm derived from the RTK toolkit (RTK, 2025) (*No Resilience*) and (ii) the same MPI implementation enhanced with our checkpointing support. We used a simple parallelization technique for the SART, in which the volumes are replicated and projections are partitioned among reconstruction processes. The volumes are synchronized at the end of each iteration, which translates to block iterations on the projection dimension.

Without failure (Mean time to failure = ∞), both implementations complete the reconstruction in comparable time, demonstrating the minimal overhead of our checkpointing solution, consistent with earlier results. However, once failures are introduced, the original implementation experiences severe penalties: each failure forces a complete restart, causing the reconstruction time to grow exponentially. Specifically, at MTTF = 10 s, the original reconstruction exceeds the timeout, indicating infeasibility with at least 1,000× slowdown relative to the ideal runtime. In contrast, our checkpointing solution dramatically mitigates the impact of failures. By checkpointing every iteration, we reduce reconstruction time penalties by 25% at MTTF = 1,000 s and by a factor of at least 100× at MTTF = 10 s—an extreme scenario where failures occur 11× faster than the ideal reconstruction time. This reduction keeps the overhead under 10× relative to the failure-free case, demonstrating that our solution sustains practical and predictable reconstruction performance even under highly failure-prone conditions, ensuring feasibility where naive execution would become entirely impractical.

Robustness against datasets. Lastly, we assess the applicability of our solution to different datasets. Using the same experimental setup (64 tasks processing 64 sinograms over 10 iterations with ART), we reconstruct full-scale *Spheres* and *Chip* datasets. Their reconstruction times under various failure conditions are shown in Figure 11b. The variation in reconstruction time between datasets is attributed to differences in resolution (2,048×2,048 vs. 2,448×2,448). However, our resilience mechanisms remain effective: at MTTF = 100,000 s, reconstruction times are nearly identical to ideal cases. Even at MTTF = 100 s, where failure

severity increases by 1,000×, reconstruction time increases by at most 60×, a significantly lower impact than the raw failure rate increase, demonstrating robustness across datasets.

Takeaway: *Proposed resilient solutions demonstrate robustness across diverse datasets, varying data sizes, and multiple reconstruction algorithms, ensuring their practical applicability to a wide range of tomographic reconstruction pipelines.*

6 Discussion

Our resilient solutions were initially designed for data-parallel iterative tomographic reconstruction, but their core ideas rely on two key properties common to many reconstruction pipelines—making them broadly applicable. Specifically, these properties are: (i) the *iterative reconstruction pattern*, in which the reconstructed object represents an intermediate state that can serve as a progress checkpoint to avoid expensive recomputation, and (ii) *partial updates with data-parallelism*, where the data can be partitioned (e.g., sinograms) and processed by independent tasks, enabling efficient dynamic rebalancing and recovery when failures occur.

Any tomographic reconstruction workflow that exhibits one or both of these properties can leverage our resilient solutions. Furthermore, as demonstrated in Sections 5.2.3 and 5.2.4, our techniques are designed to work in an orthogonal manner—if one part of the solution is not applicable, the remaining mechanisms retain their efficiency and still significantly reduce the impact of failures. Based on this observation, we now discuss how our approach can be adapted to a wide range of reconstruction methods and hardware platforms beyond the specific setting presented previously in the paper.

6.1 Reconstruction methods

There is a vast number of tomographic reconstruction techniques developed with different properties. For instance, analytical reconstruction techniques, such as GridRec (Dowd et al., 1999) or filtered-back projection (FBP) (Herman, 2009) can provide fast reconstruction, however, they are typically prone to measurement noise and can result in suboptimal image quality with limited measurements. On the other hand, the iterative techniques provide higher quality images at the cost of additional computational demand. In this work, we focus on providing resiliency with iterative techniques for the following reasons: iterative techniques can provide reasonable and actionable reconstructions with partial/streaming data; the system checkpoint can be taken by saving the reconstructed volume throughout the execution, e.g., between iterations; the reconstruction process can continue after detection and redistribution of the workload.

The parallelization of (iterative) reconstruction techniques has been an active research area, and many advanced methods have been developed for large-scale, fine-grained parallel tomographic reconstruction (Wu et al., 2024; Chen et al., 2021; Wang et al., 2017, 2016; Hidayetoğlu et al., 2020). In this work, we focus on the resiliency aspect of the iterative reconstruction applications and we exploit the fact that partially reconstructed images represent the intermediate state of the execution, which can be saved between the iterations. This unique insight allows us to apply our resiliency method to any iterative reconstruction process, irrespective of underlying parallelization technique, algorithm, or data acquisition geometry. Although, we focus on parallel beam geometry in our experiments, the other reconstruction workflows can benefit from our resiliency techniques as far as they follow the same analysis pattern.

6.2 Underlying hardware

Although our resilient solutions are currently implemented and evaluated on CPU-based systems, their design is independent of any specific hardware architecture. As illustrated in Algorithm 2, the core reconstruction computation is encapsulated in the *Reconstruction*() function, which acts as a black box within our resiliency framework. This abstraction enables the same strategies to be applied to other platforms without requiring significant modifications to the underlying reconstruction implementation.

For example, our approach can be adapted for GPU-based reconstruction pipelines with only minor modifications. The *Reconstruction*() function can be replaced by a GPU-specific implementation that incorporates additional data transfers between the GPU memory and the host to facilitate resilient control. These transfers include: (i) periodically extracting the current output from the GPU at the end of each iteration for checkpointing, and (ii) refreshing the GPU memory to update the reconstruction state in response to dynamic load redistribution and recovery.

In summary, our resilient solutions are flexible and highly applicable. They can be seamlessly integrated with a variety of reconstruction methods—whether employing parallel-beam or cone-beam geometries—and across different hardware platforms, from CPUs to GPUs. This versatility ensures that our approach can effectively address the challenges of diverse and evolving tomographic reconstruction environments.

7 Conclusion and future work

We have presented resilient solutions to enhance the reliability and performance of distributed X-ray image analysis workflows, particularly focusing on tomographic reconstruction. We introduced an asynchronous, non-blocking checkpointing mechanism and a dynamic redistribution technique with lazy recovery to mitigate the impact of failures. These methods enabled progress preservation, balanced load distribution, and low-cost recovery reconstruction in error-prone environments. Our evaluations, conducted using a 3D tomographic reconstruction deployed across Argonne's leadership computing infrastructure and synchrotron facilities, demonstrated the effectiveness of the proposed strategies in both controlled and real-world scenarios, reducing failure impact by up to 500× while maintaining negligible overhead (<3%).

This work creates many exciting future research directions. For instance, while the current study focuses on task failures, real-world scenarios involve other critical challenges, such as data transmission losses, file corruptions, and storage failures. How

can the proposed solutions be integrated with existing approaches to address these issues and further enhance the application's resilience? Moreover, with the growing demand for real-time reconstruction, which requires data to be processed and available within strict deadlines, how can the proposed solutions be adapted or extended to meet these stringent timing requirements?

There are also significant opportunities for improvement. For example, our current formulation assumes that resources associated with failed tasks are lost permanently. In practice, workflows could retain resources and spawn new tasks to replace failed ones. In elastic environments like the cloud (Foster and Gannon, 2017), workflows might dynamically scale the number of tasks across geographically distributed resources to accelerate reconstruction as needed. While these advancements present exciting possibilities for deploying efficient pipelines at large scales, they also pose challenging questions: How should data be redistributed upon task restarts or scaling? When and where should tasks be started or stopped to optimize performance and resilience?

Data availability statement

Publicly available datasets were analyzed in this study. Both code and datasets can be accessed at: https://github.com/diaspora-project/aps-mini-apps.

Author contributions

HN: Conceptualization, Formal analysis, Methodology, Software, Visualization, Writing – original draft. TB: Conceptualization, Data curation, Investigation, Methodology, Software, Writing – original draft. BN: Conceptualization, Formal analysis, Investigation, Methodology, Writing – original draft. RK: Writing – review & editing. EH: Writing – review & editing. IF: Funding acquisition, Project administration, Supervision, Writing – review & editing.

Funding

The author(s) declare that financial support was received for the research and/or publication of this article. This work was supported by the U.S. Department of Energy (DOE) under Contract No. DE-AC02-06CH11357, including funding from the Office of Advanced Scientific Computing Research (ASCR)'s Diaspora project and the Laboratory Directed Research. Additionally, this work was partially supported by the National Science Foundation (NSF), Office of Advanced Cyberinfrastructure, under Grant CSSI-2411386/2411387.

Acknowledgments

This research used resources of the Argonne Leadership Computing Facility, a U.S. DOE Office of Science user facility at ANL, and is based on research supported by the U.S. DOE Office of Science ASCR Program under Contract No. DE-AC02-06CH11357.

Conflict of interest

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

The author(s) declared that they were an editorial board member of Frontiers, at the time of submission. This had no impact on the peer review process and the final decision.

Generative AI statement

The author(s) declare that no Gen AI was used in the creation of this manuscript.

Publisher's note

References

Al-Najjar, A., Rao, N. S., Sankaran, R., Ziatdinov, M., Mukherjee, D., Ovchinnikova, O., et al. (2022). "Enabling autonomous electron microscopy for networked computation and steering," in *IEEE 18th International Conference on e-Science (e-Science)* (Salt Lake City, UT: IEEE), 267–277. doi: 10.1109/eScience55777.2022.00040

Babu, A. V., Bicer, T., Kandel, S., Zhou, T., Ching, D. J., Henke, S., et al. (2023). AI-assisted automated workflow for real-time X-ray ptychography data analysis via federated resources. *Electron. Imaging* 35, 1–6. doi: 10.2352/EI.2023.35.11.HPCI-232

Bautista-Gomez, L., Benoit, A., Di, S., Herault, T., Robert, Y., Sun, H., et al. (2024). *A* survey on checkpointing strategies: should we always checkpoint à la Young/Daly? *Future Gener. Comput. Syst.* 161, 315–328. doi: 10.1016/j.future.2024.07.022

Bautista-Gomez, L., Tsuboi, S., Komatitsch, D., Cappello, F., Maruyama, N., Matsuoka, S., et al. (2011). "FTI: high performance fault tolerance interface for hybrid systems," in *ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis* (Seattle, WA: ACM), 1–32. doi: 10.1145/2063384.2063427

Benmore, C., Bicer, T., Chan, M. K., Di, Z., Gürsoy, D., Hwang, I., et al. (2022). Advancing AI/ML at the advanced photon source. *Synchrotron Radiat. News* 35, 28–35. doi: 10.1080/08940886.2022.2112500

Benoit, A., Du, Y., Herault, T., Marchal, L., Pallez, G., Perotin, L., et al. (2022). "Checkpointing à la Young/Daly: an overview," in *14th International Conference on Contemporary Computing* (New York, NY: ACM), 701–710. doi: 10.1145/3549206.3549328

Bicer, T., Gürsoy, D., Andrade, V. D., Kettimuthu, R., Scullin, W., Carlo, F. D., et al. (2017). Trace: a high-throughput tomographic reconstruction engine for large-scale datasets. *Adv. Struct. Chem. Imaging*. 3, 1–10. doi: 10.1186/s40679-017-0040-7

Bicer, T., Gursoy, D., Kettimuthu, R., De Carlo, F., Agrawal, G., and Foster, I. T. (2015). "Rapid tomographic image reconstruction via large-scale parallelization," in *European Conference on Parallel Processing* (Cham: Springer), 289–302. doi: 10.1007/978-3-662-48096-0_23

Bicer, T., Gürsoy, D., Kettimuthu, R., De Carlo, F., and Foster, I. T. (2016). Optimization of tomographic reconstruction workflows on geographically distributed resources. *J. Synchrotron Radiat*. 23, 997–1005. doi: 10.1107/S1600577516007980

Bicer, T., Nikitin, V., Aslan, S., Gürsoy, D., Kettimuthu, R., and Foster, I. T. (2020). "Tomographic reconstruction of dynamic features with streaming sliding subsets," in *Annual Workshop on Large-scale Experiment-in-the-Loop Computing* (IEEE/ACM). doi: 10.1109/XLOOP51963.2020.00007

Bicer, T., Yu, X., Ching, D. J., Chard, R., Cherukara, M. J., Nicolae, B., et al. (2021). "High-performance ptychographic reconstruction with federated facilities," in *Smoky Mountains Computational Sciences and Engineering Conference* (Cham: Springer International Publishing), 173–189. doi: 10.1007/978-3-030-96498-6_10

Bouvier, T., Nicolae, B., Costan, A., Bicer, T., Foster, I., Antoniu, G., et al. (2024). Efficient distributed continual learning for steering experiments in real-time. *Future Gener. Comput. Syst*. 162:107438. doi: 10.1016/j.future.2024.07.016

Butler, I., Fusseis, F., Cartwright-Taylor, A., and Flynn, M. (2020). Mjölnir: a miniature triaxial rock deformation apparatus for 4D synchrotron X-ray microtomography. *J. Synchrotron Radiat*. 27, 1681–1687. doi: 10.1107/S160057752001173X

Canal, R., Hernandez, C., Tornero, R., Cilardo, A., Massari, G., Reghenzani, F., et al. (2020). Predictive reliability and fault management in exascale systems: state of the art and perspectives. *ACM Comput. Surv*. 53, 1–32. doi: 10.1145/3403956

Cappello, F., Geist, A., Gropp, W., Kale, S., Kramer, B., Snir, M., et al. (2014). Toward exascale resilience: 2014 update. *Supercomput. Front. Innov* 1, 5–28. doi: 10.14529/jsfi140101

Chen, P., Wahib, M., Takizawa, S., Takano, R., and Matsuoka, S. (2019). "iFDK: a scalable framework for instant high-resolution image reconstruction," in *International Conference for High Performance Computing, Networking, Storage and Analysis* (New York, NY: ACM), 1–24. doi: 10.1145/3295500.3356163

Chen, P., Wahib, M., Wang, X., Hirofuchi, T., Ogawa, H., Biguri, A., et al. (2021). "Scalable FBP decomposition for cone-beam CT reconstruction," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (New york, NY: ACM), 1–16. doi: 10.1145/3458817.3476139

Crowther, R. A., DeRosier, D., and Klug, A. (1970). The reconstruction of a three-dimensional structure from projections and its application to electron microscopy. *Proc. R. Soc. Lond. A Math. Phys. Sci*. 317, 319–340. doi: 10.1098/rspa.1970.0119

Daly, J. T. (2006). A higher order estimate of the optimum checkpoint interval for restart dumps. *Future Gener. Comput. Syst*. 22, 303–312. doi: 10.1016/j.future.2004.11.016

De Carlo, F., Gürsoy, D., Ching, D. J., Batenburg, K. J., Ludwig, W., Mancini, L., et al. (2018). TomoBank: a tomographic data repository for computational X-ray science. *Meas. Sci. Technol*. 29:034004. doi: 10.1088/1361-6501/aa9c19

Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977). Maximum likelihood from incomplete data via the em algorithm. *J. R. Stat. Soc. Ser. B (Methodol.)*, 39, 1–22. doi: 10.1111/j.2517-6161.1977.tb01600.x

Dongarra, J., Herault, T., and Robert, Y. (2015). *Fault Tolerance Techniques for High-Performance Computing*. Cham: Springer. doi: 10.1007/978-3-319-20943-2_1

Dowd, B. A., Campbell, G. H., Marr, R. B., Nagarkar, V. V., Tipnis, S. V., Axe, L., et al. (1999). "Developments in synchrotron X-ray computed microtomography at the national synchrotron light source," in *Developments in X-ray Tomography II, Vol. 3772* (Denver, CO: SPIE), 224–236. doi: 10.1117/12.363725

Dubey, A., Jauhri, A., Pandey, A., Kadian, A., Al-Dahle, A., Letman, A., et al. (2024). The Llama 3 herd of models. *arXiv* [Preprint]. arXiv:2407.21783. doi: 10.48550/arXiv.2407.21783

Elnozahy, E. N. M., Alvisi, L., Wang, Y.-M., and Johnson, D. B. (2002). A survey of rollback-recovery protocols in message-passing systems. *ACM Comput. Surv*. 34, 375–408. doi: 10.1145/568522.568525

Ferreira, K., Stearley, J., Laros, I. I. I., Oldfield, J. H., Pedretti, R., Brightwell, K., et al. (2011). "Evaluating the viability of process replication reliability for exascale systems," in *International Conference for High Performance Computing, Networking, Storage and Analysis* (New York, NY: ACM), 1–12. doi: 10.1145/2063384.2063443

Fessler, J. A., Sonka, M., and Fitzpatrick, J. M. (2000). Statistical image reconstruction methods for transmission tomography. *Handb. Med. Imaging* 2, 1–70. doi: 10.1117/3.831079.ch1

Finegan, D. P., Scheel, M., Robinson, J. B., Tjaden, B., Hunt, I., Mason, T. J., et al. (2015). In-operando high-speed tomography of lithium-ion batteries during thermal runaway. *Nat. Commun*. 6:6924. doi: 10.1038/ncomms7924

Foster, I., and Gannon, D. B. (2017). *Cloud Computing for Science and Engineering*. Cambridge, MA: MIT Press.

Gordon, R., Bender, R., and Herman, G. T. (1970). Algebraic reconstruction techniques (ART) for three-dimensional electron microscopy and X-ray photography. *J Theor. Biol*. 29, 471–481. doi: 10.1016/0022-5193(70)90109-8

Gossman, M. J., Nicolae, B., and Calhoun, J. C. (2024). Scalable I/O aggregation for asynchronous multi-level checkpointing. *Future Gener. Comput. Syst*. 160, 420–432. doi: 10.1016/j.future.2024.06.003

Habib, S., Pope, A., Finkel, H., Frontiere, N., Heitmann, K., Daniel, D., et al. (2016). HACC: simulating sky surveys on state-of-the-art supercomputing architectures. *New Astronomy* 42, 49–65. doi: 10.1016/j.newast.2015.06.003

Herman, G. T. (2009). *Fundamentals of Computerized Tomography: Image Reconstruction From projections*. Berlin: Springer Science and Business Media.

Hidayetoğlu, M., Bicer, T., De Gonzalo, S. G., Ren, B., De Andrade, V., Gursoy, D., et al. (2020). "Petascale XCT: 3D image reconstruction with hierarchical communications on multi-GPU nodes," in *International Conference for High Performance Computing, Networking, Storage and Analysis* (Atlanta, GA: IEEE), 1–13. doi: 10.1109/SC41405.2020.00041

Hidayetoğlu, M., Biçer, T., De Gonzalo, S. G., Ren, B., Gürsoy, D., Kettimuthu, R., et al. (2019). "MemXCT: Memory-centric X-ray CT reconstruction with massive parallelization," in *International Conference for High Performance Computing, Networking, Storage and Analysis* (New York, NY: ACM), 1–56. doi: 10.1145/3295500.3356220

Hsieh, J., and Flohr, T. (2021). Computed tomography recent history and future perspectives. *J. Med. Imaging* 8:052109. doi: 10.1117/1.JMI.8.5.052109

Kak, A. C., and Slaney, M. (2001). *Principles of Computerized Tomographic Imaging*. Philadelphia, PA: Society for Industrial and Applied Mathematics. doi: 10.1137/1.9780898719277

Kamilov, U. S., Bouman, C. A., Buzzard, G. T., and Wohlberg, B. (2023). Plug-and-play methods for integrating physical and learned models in computational imaging: theory, algorithms, and applications. *IEEE Signal Process. Mag*. 40, 85–97. doi: 10.1109/MSP.2022.3199595

Kamilov, U. S., Papadopoulos, I. N., Shoreh, M. H., Goy, A., Vonesch, C., Unser, M., et al. (2015). Learning approach to optical tomography. *Optica* 2, 517–522. doi: 10.1364/OPTICA.2.000517

Leu, L., Berg, S., Enzmann, F., Armstrong, R. T., and Kersten, M. (2014). Fast X-ray micro-tomography of multiphase flow in Berea sandstone: a sensitivity study on image processing. *Transp. Porous Media* 105, 451–469. doi: 10.1007/s11242-014-0378-4

Lin, Y., Feng, S., Theiler, J., Chen, Y., Villa, U., Rao, J., et al. (2024). Physics and deep learning in computational wave imaging. *arXiv* [Preprint] arXiv:2410.08329. doi: 10.48550/arXiv.2410.08329

Liu, D., Shadike, Z., Lin, R., Qian, K., Li, H., Li, K., et al. (2019a). Review of recent development of *in situ*/operando characterization techniques for lithium battery research. *Adv. Mater*. 31:1806620. doi: 10.1002/adma.201806620

Liu, Z., Bicer, T., Kettimuthu, R., and Foster, I. (2019b). "Deep learning accelerated light source experiments," in *2019 IEEE/ACM Third Workshop on Deep Learning on Supercomputers (DLS)* (Denver, CO: IEEE), 20–28. doi: 10.1109/DLS49591.2019.00008

Maire, E., and Withers, P. J. (2014). Quantitative X-ray tomography. *Int. Mater. Rev*. 59, 1–43. doi: 10.1179/1743280413Y.0000000023

Maurya, A., Rafique, M., Tonellot, T., AlSalem, H., Cappello, F., and Nicolae, B. (2023). "Gpu-enabled asynchronous multi-level checkpoint caching and prefetching," in *HPDC'23: The 32nd International Symposium on High-Performance Parallel and Distributed Computing* (Orlando, FL), 73–85. doi: 10.1145/3588195.3592987

Maurya, A., Underwood, R., Rafique, M., Cappello, F., and Nicolae, B. (2024). "Datastates-llm: lazy asynchronous checkpointing for large language models," in *HPDC'24: The 33nd International Symposium on High-Performance Parallel and Distributed Computing* (Pisa). doi: 10.1145/3625549.3658685

Mohan, J., Phanishayee, A., and Chidambaram, V. (2021). "CheckFreq: frequent, fine-grained DNN checkpointing," in *FAST'21: The 19th USENIX Conference on File and Storage Technologies* (Boston, MA: USENIX Association), 203–216.

Mohan, K., Venkatakrishnan, S., Gibbs, J., Gulsoy, E., Xiao, X., De Graef, M., et al. (2015). TIMBIR: a method for time-space reconstruction from interlaced views. *IEEE Trans. Comput. Imaging* 1, 96–111. doi: 10.1109/TCI.2015.2431913

Mohan, K. A., Ferrucci, M., Divin, C., Stevenson, G. A., and Kim, H. (2024). Distributed stochastic optimization of a neural representation network for time-space tomography reconstruction. *arXiv* [Preprint]. arXiv:2404.19075. doi: 10.48550/arXiv.2404.19075

Mohan, K. A., Venkatakrishnan, S., Drummy, L. F., Simmons, J., Parkinson, D. Y., Bouman, C. A., et al. (2014). "Model-based iterative reconstruction for synchrotron X-ray tomography," in *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (Florence: IEEE), 6909–6913. doi: 10.1109/ICASSP.2014.6854939

Mohror, K., Moody, A., Bronevetsky, G., and de Supinski, B. R. (2014). Detailed modeling and evaluation of a scalable multilevel checkpointing system. *IEEE Trans. Parallel Distrib. Syst*. 25, 2255–2263. doi: 10.1109/TPDS.2013.100

Nano CT (2024). *Chip Dataset*. Available online at: https://tomobank.readthedocs.io/en/latest/source/data/docs.data.nano.html#psf (accessed May 26, 2025).

Nicolae, B. (2022). "Scalable multi-versioning ordered key-value stores with persistent memory support," in *IPDPS 2022: The 36th IEEE International Parallel and Distributed Processing Symposium* (Lyon), 93–103. doi: 10.1109/IPDPS53621.2022.00018

Nicolae, B., Antoniu, G., Bouge, L., Moise, D., and Carpen-Amarie, A. (2011). Blobseer: next-generation data management for large scale infrastructures. *J. Parallel Distrib. Comput*. 71, 169–184. doi: 10.1016/j.jpdc.2010.08.004

Nicolae, B., Li, J., Wozniak, J., Bosilca, G., Dorier, M., Cappello, F., et al. (2020). "Deepfreeze: towards scalable asynchronous checkpointing of deep learning models," in *CGrid'20: 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing* (Melbourne, VIC), 172–181. doi: 10.1109/CCGrid49817.2020.00-76

Nicolae, B., Moody, A., Gonsiorowski, E., Mohror, K., and Cappello, F. (2019). "VeloC: Towards high performance adaptive asynchronous checkpointing at large scale," in *IEEE International Parallel and Distributed Processing Symposium* (IEEE), 911–920. doi: 10.1109/IPDPS.2019.00099

Nicolae, B., Moody, A., Kosinovsky, G., Mohror, K., and Cappello, F. (2021). "Veloc: very low overhead checkpointing in the age of exascale," in *SuperCheck'21: The First International Symposium on Checkpointing for Supercomputing, Virtual Event*.

Nikitin, V. V., Carlsson, M., Andersson, F., and Mokso, R. (2019). Four-dimensional tomographic reconstruction by time domain decomposition. *IEEE Trans. Comput. Imaging* 5, 409–419. doi: 10.1109/TCI.2019.2898088

Ravi, N., Chaturvedi, P., Huerta, E., Liu, Z., Chard, R., Scourtas, A., et al. (2022). Fair principles for ai models with a practical application for accelerated high energy diffraction microscopy. *Sci. Data* 9:657. doi: 10.1038/s41597-022-01712-9

Rawson, S. D., Maksimcuka, J., Withers, P. J., and Cartmell, S. H. (2020). X-ray computed tomography in life sciences. *BMC Biol*. 18, 1–15. doi: 10.1186/s12915-020-0753-2

Reed, I. S., and Solomon, G. (1960). Polynomial codes over certain finite fields. *J. Soc. Ind. Appl. Math*. 8, 300–304. doi: 10.1137/0108018

RTK (2025). *Reconstruction Toolkit (RTK)*. Available online at: https://www.openrtk.org/ (accessed May 26, 2025).

Schlichting, R. D., and Schneider, F. B. (1983). Fail-stop processors: an approach to designing fault-tolerant computing systems. *ACM Trans. Comput. Syst*. 1, 222–238. doi: 10.1145/357369.357371

Spheres (2024). Spheres dataset. Available online at: https://tomobank.readthedocs.io/en/latest/source/data/docs.data.spheres.html (accessed May 26, 2025).

Stiller, W. (2018). Basics of iterative reconstruction methods in computed tomography: a vendor-independent overview. *Eur. J. Radiol*. 109, 147–154. doi: 10.1016/j.ejrad.2018.10.025

Tan, N., Luettgau, J., Marquez, J., Terianishi, K., Morales, N., Bhowmick, S., et al. (2023). "Scalable incremental checkpointing using gpu-accelerated de-duplication," in *ICPP'23: The 52nd International Conference on Parallel Processing* (Salt Lake City: 665–674). doi: 10.1145/3605573.3605639

Tang, F., Wu, Z., Yang, C., Osenberg, M., Hilger, A., Dong, K., et al. (2021). Synchrotron X-ray tomography for rechargeable battery research: fundamentals, setups and applications. *Small Methods* 5:2100557. doi: 10.1002/smtd.202100557

Thakur, R., Gropp, W., and Lusk, E. (1999). "Data sieving and collective i/o in Romio," in *Proceedings. Frontiers '99. Seventh Symposium on the Frontiers of Massively Parallel Computation* (Annapolis, MA), 182–189. doi: 10.1109/FMPC.1999.750599

Underwood, R., Madhyastha, M., Burns, R., and Nicolae, B. (2024). "Evostore: towards scalable storage of evolving learning models in *HPDC'24: The 33nd International Symposium on High-Performance Parallel and Distributed Computing* (Pisa). doi: 10.1145/3625549.3658679

Vásárhelyi, L., Kónya, Z., Kukovecz, Á., and Vajtai, R. (2020). Microcomputed tomography-based characterization of advanced materials: a review. *Mater. Today Adv*. 8:100084. doi: 10.1016/j.mtadv.2020.100084

Vescovi, R., Chard, R., Saint, N. D., Blaiszik, B., Pruyne, J., Bicer, T., et al. (2022). Linking scientific instruments and computation: patterns, technologies, and experiences. *Patterns* 3:100606. doi: 10.1016/j.patter.2022.100606

Wang, X., Sabne, A., Kisner, S., Raghunathan, A., Bouman, C., Midkiff, S., et al. (2016). High performance model based image reconstruction. *ACM SIGPLAN Notices* 51:2. doi: 10.1145/3016078.2851163

Wang, X., Sabne, A., Sakdhnagool, P., Kisner, S. J., Bouman, C. A., Midkiff, S. P., et al. (2017). "Massively parallel 3D image reconstruction," in *International Conference for High Performance Computing, Networking, Storage and Analysis* (Denver, CO), 1–12. doi: 10.1145/3126908.3126911

Withers, P. J., Bouman, C., Carmignato, S., Cnudde, V., Grimaldi, D., Hagen, C. K., et al. (2021). X-ray computed tomography. *Nat. Rev. Methods Prim*. 1:18. doi: 10.1038/s43586-021-00015-4

Wu, D., Chen, P., Wang, X., Lyngaas, I., Miyajima, T., Endo, T., et al. (2024). "Real-time high-resolution X-ray computed tomography," in *Proceedings of the 38th ACM International Conference on Supercomputing* (New York, NY: ACM), 110–123. doi: 10.1145/3650200.3656634

Young, J. W. (1974). A first order approximation to the optimum checkpoint interval. *Commun. ACM* 17, 530–531. doi: 10.1145/361147.361115

Zwanenburg, E., Williams, M., and Warnett, J. M. (2021). Review of high-speed imaging with lab-based X-ray computed tomography. *Meas. Sci. Technol*. 33:012003. doi: 10.1088/1361-6501/ac354a