SIAM J. SCI. COMPUT.
Vol. 46, No. 5, pp. A3258–A3281

MODIFYING THE ASYNCHRONOUS JACOBI METHOD FOR DATA CORRUPTION RESILIENCE*

CHRISTOPHER J. VOGL†, ZACHARY R. ATKINS‡, ALYSON FOX†, AGNIESZKA MIĘDLAR§, AND COLIN PONCE†

Abstract. Moving scientific computation from high-performance computing (HPC) and cloud computing (CC) environments to devices on the edge, i.e., physically near instruments of interest, has received tremendous interest in recent years. Such edge computing environments can operate on data in situ, offering enticing benefits over data aggregation to HPC and CC facilities that include avoiding costs of transmission, increased data privacy, and real-time data analysis. Because of the inherent unreliability of edge computing environments, new fault-tolerant approaches must be developed before the benefits of edge computing can be realized. Motivated by algorithm-based fault tolerance, a variant of the asynchronous Jacobi (ASJ) method is developed that achieves resilience to data corruption by rejecting solution approximations from neighbor devices according to a bound derived from convergence theory. Numerical results on a two-dimensional Poisson problem show that the new rejection criterion, along with a novel approximation to the shortest path length on which the criterion depends, restores convergence for the ASJ variant in the presence of certain types data corruption. Numerical results are obtained for when the singular values in the analytic bound are approximated. Additional linear systems are also explored, one with a more dense sparsity pattern and one that includes advection. All results indicate that successful resilience to data corruption depends on whether the bound tightens fast enough to reject corrupted data before the iteration evolution deviates significantly from that predicted by the convergence theory defining the bound. This observation generalizes to future work on algorithm-based fault tolerance for other asynchronous algorithms, including upcoming approaches that leverage Krylov subspaces.

Key words. distributed computing, asynchronous methods, data corruption, Jacobi

MSC codes. 15, 64

DOI. 10.1137/23M1605648

1. Introduction.

Recent years have seen a proliferation of *edge devices,* i.e., streamlined computing devices that provide an entry point to the individual instruments in their vicinity. Modern infrastructure includes a wide range of such devices, from smart residential thermostats to industrial smart grid meters. These devices, along with wearable health care devices and content delivery systems, are motivating a push of computation beyond the walls of high-performance computing (HPC) and cloud computing (CC) facilities onto the edge devices themselves. Consider, as an example, the benefits of enabling smart power grid devices to operate autonomously when the central operator is disabled due to a natural disaster or cyberphysical attack. The capability provided by edge computing environments to operate without a single point of failure or on data in situ is appealing to real-time system operators.

*Submitted to the journal's Numerical Algorithms for Scientific Computing section September 29, 2023; accepted for publication (in revised form) July 23, 2024; published electronically October 9, 2024.

https://doi.org/10.1137/23M1605648

Funding: This work was performed under the auspices of the U.S. Department of Energy by the Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344 and was supported by the LLNL-LDRD Program under project nos. 21-FS-007 and 22-ERD-045, LLNL-JRNL-853510. The work of Agnieszka Mię̨dlar was supported by NSF grants DMS-2144181 and DMS-2324958.

†Lawrence Livermore National Laboratory, Livermore, CA 94550 USA (vogl2@llnl.gov, fox33@llnl.gov, ponce11@llnl.gov).

‡Department of Computer Science, University of Colorado, Boulder, CO 80309 USA (zach.atkins@colorado.edu).

§Department of Mathematics, Virginia Tech, Blacksburg, VA 24060 USA (amiedlar@vt.edu).

Unfortunately, the benefits of edge computing cannot be realized before the inherent unreliability of edge devices is addressed. Modern scientific computing algorithms typically assume that data will not be corrupted as the algorithm is executed. HPC and CC platforms provide such data integrity by utilizing fault management techniques. Checkpointing and redundant computation are cornerstones of fault management techniques in HPC and CC and are an integral part of n-modular redundancy [4], n-version programming [17], majority voting [4], and redundant cloud servers [17] techniques. The frequency of checkpointing is typically chosen to avoid restarting from a checkpoint created long before the fault occurs while keeping the cost of synchronization and storage reasonable. Similarly, the amount of redundancy is typically chosen to avoid having all redundant entities experience a fault at the same time while keeping the cost of storage and flops reasonable. Thus, checkpointing and redundancy are not practical for edge computing environments where synchronization is typically expensive and data storage and/or flops are limited.

One promising alternative fault management strategy is the class of *algorithm-based fault-tolerant* (ABFT) methods. The general idea is to leverage the structure or expected behavior of the algorithm to detect, mitigate, and/or recover from faults such as data corruption. Examples of ABFT schemes include methods for the fast Fourier transform [14], matrix multiplication [23], Krylov-based iterative methods [6], and the synchronous Jacobi method [2]. Focusing on the iterative methods, the work presented in [6] uses the orthogonality of projections onto Krylov spaces for detection of faults, while [2] utilizes the contraction mapping property of stationary iterative methods. Unfortunately, those ABFT approaches are for iterative methods that require frequent synchronizations, making them impractical for edge computing environments due to network latency, heterogeneous nodes, and nonpersistent nodes/links. Asynchronous methods remove the need for global synchronization after each iteration, allowing them to outperform their synchronous counterparts in high-latency environments (see, e.g., [22]). This makes asynchronous methods very appealing for edge computing; however, the authors are aware of only two existing asynchronous methods with ABFT strategies: the robust alternating direction method of multipliers (ADMM) [13] and the robust push-sum algorithm [18].

Solving the nonlinear systems that are being pushed to edge computing environments, such as distribution system state estimation [21] and machine learning for autonomous vehicles [15] and smart agriculture [12], typically requires the solution of one or more linear systems (e.g., direction vectors in the Newton–Rhapson method). To address the resulting need for an asynchronous linear solver with ABFT for such tasks, we modify the asynchronous Jacobi (ASJ) method [8, 11, 22, 2, 5] with a rejection criterion based on the convergence properties of ASJ. In [13], such a rejection criterion is developed where data from neighboring nodes are rejected if the difference between successive data obtained from a neighbor exceeds a bound derived from the convergence theory for ADMM. Here, a similar criterion is used but with a novel bound derived for the ASJ method based on the ASJ convergence theory developed in [11]. Because the bound for ASJ depends on time-dependent global information, specifically, the shortest path length, a novel local approximation to that global information is also developed. It is worth noting that the choice of the ASJ method for modification is motivated by the facts that (i) the authors are not aware of another asynchronous linear solver with established convergence theory; (ii) while the Jacobi method is known to scale poorly to large and ill-conditioned systems, ASJ can be sufficient for small problems that appear in edge environments (e.g., state estimation in a neighborhood); and (iii) the observations and understanding of the resilience

modification for ASJ can be generalized to upcoming asynchronous Krylov-based solvers, such as the one recently developed in [7].

This paper is organized as follows. Section 2 formulates the problem, introduces the notation and important definitions, and discusses the nature of data corruption to be investigated. Section 3 develops our resilience enabling technique. Section 4 presents numerical results verifying the implementation of the method and demonstrating the effectiveness of the proposed rejection technique in the presence of various forms of data corruption. An empirical sensitivity study of the rejection criterion parameters is also presented, with a discussion on potential improvements to the local approximation of global information used in the criterion. The evaluation of the method is extended to two additional test problems, with one involving a more dense sparsity pattern and another involving advection. Section 5 summarizes the outcomes of the paper and discusses how the results might generalize to ongoing and future work.

2. Problem statement. Solutions of linear systems are ubiquitous in modern scientific computing algorithms, defining search directions in both iterative and nonlinear solvers. Thus, consider solving the linear system

$$A\mathbf{x} = \mathbf{b} \tag{2.1}$$

for $\mathbf{x} \in \mathbb{R}^m$, where $A \in \mathbb{R}^{m\times m}$ and $\mathbf{b} \in \mathbb{R}^m$. Assume that A is nonsingular so that a unique solution to (2.1) exists. The ASJ method is an iterative solver for (2.1) in that successive approximations to the solution $\mathbf{x}$ are formed across N computational nodes. Denote by $\{\mathbf{x}_i\}_{i=1}^{i=N}$ the partition of the elements of $\mathbf{x}$ such that $\mathbf{x}_i \in \mathbb{R}^{m_i}$ is the subset of $\mathbf{x}$ that node i is approximating. Let $I \in \mathbb{R}^{m\times m}$ and $D \in \mathbb{R}^{m\times m}$ be the identity matrix and the diagonal matrix containing the diagonal elements of A, respectively. The update equation that defines the successive approximations computed by node i, denoted $\mathbf{x}_i^0$, $\mathbf{x}_i^1$, etc., can now be expressed as

$$\mathbf{x}_i^\kappa = \sum_{j=1}^{N} M_{ij}\mathbf{x}_j^{\psi(i,j,\kappa)} + \mathbf{c}_i, \quad \kappa = 1, 2, \ldots, \tag{2.2}$$

where $\{M_{ij}\}$ is the partition of the elements of the Jacobi iteration matrix $M := I - D^{-1}A$ such that the rows and columns of $M_{ij} \in \mathbb{R}^{m_i\times m_j}$ correspond to $\mathbf{x}_i$ and $\mathbf{x}_j$, respectively, and $\{\mathbf{c}_i\}$ is the partition of the elements of $\mathbf{c} := D^{-1}\mathbf{b}$ such that the elements of $\mathbf{c}_i \in \mathbb{R}^{m_i}$ correspond to $\mathbf{x}_i$. The index function ψ is defined by $\psi(i, j, \kappa) = \lambda$ if node i uses node j's λth approximation in the computation of its κth approximation. All matrix, vector, and matrix-vector operations, including the communication of $\mathbf{x}_i^\kappa$ and $\mathbf{x}_j^{\psi(i,j,\kappa)}$, are to be considered block operations.

The general form of (2.2) defines a class of chaotic or asynchronous iterative methods, first introduced by Chazan and Miranker [5], that generalize classic relaxation methods to allow each compute node to perform a new iteration immediately after the previous iteration has completed. Chazan and Miranker provide the sufficient condition to guarantee convergence of any relaxation scheme of the form (2.2): that the spectral radius of the absolute value of the global iteration matrix, M, is bounded below one, i.e., $\rho(|M|) < 1$, where $|M|$ is defined by taking the absolute value of each element in the matrix. However, the authors in [5] assume that the values of $\mathbf{x}_j^{\psi(i,j,\kappa)}$ sent by node j are the same as those received by node i. Such an assumption can become invalid in emerging computing environments that do not provide the guarantees of current high-performance computing systems. Thus, the goal of this work is to modify (2.2) to ensure (or at least encourage) convergence even if data corruption

TABLE 1
Notation table.

Symbol	Description	Location
N	number of computational nodes	section 2
A	square system matrix ($\mathbb{R}^{m\times m}$) with real components	section 2
$\mathbf{x}$, $\mathbf{b}$	vectors of real components ($\mathbb{R}^m$)	section 2
$\mathbf{x}_i$	subset ($\mathbb{R}^{m_i}$) of the elements of $\mathbf{x}$ that is assigned to the ith computational node	section 2
D	diagonal matrix ($\mathbb{R}^{m\times m}$) whose elements are the diagonal entries of A	section 2
M	Jacobi iteration matrix $I - D^{-1}A$ ($\mathbb{R}^{m\times m}$)	section 2
$M_{i,j}$	subset ($\mathbb{R}^{m_i\times m_j}$) of the elements of M with rows corresponding to $\mathbf{x}_i$ and columns corresponding to $\mathbf{x}_j$	section 2
i,j,k	indexing of partition subsets	section 2
λ, κ	indexing of iterations	section 2
$\mathbf{x}_i^\kappa$	the κth iteration of the approximation to vector $\mathbf{x}$ on node i ($\mathbb{R}^{m_i}$)	section 2
p	the probability of a bit flip in a communicated element	section 2.1
ω_f, ω_r	the time to failure and recovery time	section 2.2
δ	an offset that is sampled from a Gaussian distribution with a positive mean	section 2.2
$\nu_i(t)$	iteration index such that $\mathbf{x}_i^{\nu_i(t)}$ is the most recent approximation of $\mathbf{x}_i$ at time t	section 3
$\widetilde{\mathbf{x}}(t)$	global approximate solution ($\mathbb{R}^m$) at time t such that $\widetilde{\mathbf{x}}_i(t) = \mathbf{x}_i^{\nu_i(t)}$, $i = 1, \ldots, N$	section 3
$\mathbf{x}^*$	global exact solution ($\mathbb{R}^m$) to $A\mathbf{x} = \mathbf{b}$	section 3
$\mathbf{e}(t)$	global error ($\mathbb{R}^m$) at time t such that $\mathbf{e}_i(t) = \widetilde{\mathbf{x}}_i(t) - \mathbf{x}_i^*$	section 3
$\Omega(t) := \Omega(\psi, M, t)$	error operator ($\mathbb{R}^{m\times m}$) such that $\mathbf{e}(t) = \Omega(t)\mathbf{e}(0)$	section 3
$\mathcal{G}(\mathcal{V}, \mathcal{E})$	directed acyclic graph with nodes $\mathcal{V}$ and edges $\mathcal{E}$	section 3
$s(t)$, $l(t)$	shortest and longest paths in $\mathcal{G}$, respectively	section 3
$\psi_{ij}(\kappa) := \psi(i, j, \kappa)$	iteration index of the update from node j, which node i uses to compute its κth update	section 3
$\zeta_{ij}(t)$	$\arg\max_\kappa \psi_{ij}(\kappa) < \psi_{ij}(\nu_i(t))$	section 3
$\tau_{ij}[\kappa]$	time at which the solution approximation existed on node j that will later be used to form $\mathbf{x}_i^\kappa$	section 3
$\sigma_{min}(A), \sigma_{max}(A)$	smallest and largest singular value of A, respectively	section 3
$\tilde{s}_i(t)$	approximate shortest path	section 3
$d\mathbf{x}_i^{\nu_i(t)} := \mathbf{x}_i^{\nu_i(t)} - \mathbf{x}_i^{\nu_i(t)-1}$	difference ($\mathbb{R}^{m_i}$) between two successive solution approximations on node i	section 3
ϵ	a user-defined tolerance for the stopping criteria	section 3

results in either (i) the values of $\mathbf{x}_j^{\psi(i,j,\kappa)}$ received by node i being different than those sent by node j or (ii) the values of $\mathbf{x}_j^\kappa$ stored on node j being altered. As a convenience to the reader, Table 1 summarizes the notation used herein as well as the location where the notation is first mentioned.

2.1. Natural data corruption. The first data corruption model is motivated by bit flips occurring in network hardware memory that alter data as they are in transit. This natural data corruption is modeled as a random process where each component of transmitted data is affected by a bit flip with a fixed probability $p \in (0, 1)$. The bit flips themselves are performed either on IEEE 754 double-precision (64-bit) floating point [1] or on 32-bit signed integer numbers. The affected bit index is sampled from various uniform integer distributions, then the bit flip is performed directly. In extremely rare cases, this method of performing bit flips on double-precision numbers can result in the special floating point values `NaN` or `inf`. It is

worth noting that this data corruption approach mirrors the model of Anzt, Dongarra, and Quintana-Ortí [2], where a fixed number of bit flips are introduced per iteration to the entries of the iteration matrix M during the matrix-vector product in each iteration, which may corrupt up to 1% of updates to the elements of the solution vector. Instead, we choose to corrupt the elements of the transmitted solution vector directly at a fixed probability $p \in (0, 1)$; i.e., corruption is applied with probability p to each transmitted data element.

2.2. Malevolent data corruption. The second data corruption model is motivated by intentional corruption caused by a malicious actor who has gained intermittent access to a device to manipulate the result of a calculation. This malevolent data corruption is modeled as a periodic process where each agent is considered to be in either a "normal" or a "degraded" state. When in a "normal" state, the new approximate solution is unaltered. After ω_f s have passed, the agent is compromised and enters a "degraded" state. While in the "degraded" state, the impacted data on an agent are corrupted by adding an offset to all solution elements. Note that such nontransient corruption, i.e., overwriting of the local solution data, presents a more challenging recovery scenario than transient corruption. This offset is sampled from a Gaussian distribution with a positive mean of δ and a standard deviation of 0.5δ. The repeated application of these offsets will gradually increase the magnitude of the corrupted elements of the solution vector, absent any mitigation strategy. We choose the standard deviation of 0.5δ to ensure that 95% of the sampled offsets will be greater than zero regardless of the choice of δ. After ω_r s have passed, the agent is secured and returns to a "normal" state.

3. Corruption resilience modification. To improve data corruption resilience in the ASJ method (2.2), we take the approach of inspecting incoming data before they are used to form the next approximation $\mathbf{x}_i^\kappa$ in (2.2). If the data are identified as corrupted, they are rejected by being excluded from contributing to the next solution approximation. We will use the convergence theory established by Hook and Dingle [11] to derive our rejection criterion. The authors in [11] derive an error bound using metrics of the evolution of the solution approximations computed by each node and the communication pattern between nodes. They accomplish this task by casting the algorithm evolution as a directed acyclic graph whose vertices are the solution approximations computed by each node and edges indicate when a solution approximation is used to compute a latter solution approximation. Using a similar notation, we will summarize the components used to form the rejection criterion below.

Define $\nu_i(t)$ so that $\mathbf{x}_i^{\nu_i(t)}$ is the solution approximation on node i at time t. A global solution approximation at time t, denoted by $\widetilde{\mathbf{x}}(t)$, is defined blockwise as $\widetilde{\mathbf{x}}_i(t) = \mathbf{x}_i^{\nu_i(t)}$, $i = 1, \ldots, N$. With $\mathbf{x}^*$ being the exact solution of (2.1), the global error at time t is defined as $\mathbf{e}(t) = \widetilde{\mathbf{x}}(t) - \mathbf{x}^*$. Denote the error operator $\Omega(\psi, M, t)$ such that $\mathbf{e}(t) = \Omega(\psi, M, t)\mathbf{e}(0)$. The properties of $\Omega(\psi, M, t)$, denoted herein as $\Omega(t)$, are presented in [11] using a directed acyclic graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ with graph nodes $\mathcal{V}$ and edges $\mathcal{E}$. This is *not* the graph of computational node-to-node connections but is instead a directed acyclic graph representation of the evolution of the collective computation: Each solution approximation at each computational node (i.e., each $\mathbf{x}_j^\kappa$) is an element of $\mathcal{V}$, and there is an edge in $\mathcal{E}$ from $\mathbf{x}_j^\lambda$ to $\mathbf{x}_i^\kappa$ iff $\psi(i, j, \kappa) = \lambda$. Figure 1 presents a simple example of a directed acyclic graph for the algorithm evolution between two nodes. The initial states on nodes 1 and 2 are denoted by $\mathbf{x}_1^0$ and $\mathbf{x}_2^0$, respectively. The initial solution approximation on node 2 is received by node 1 and used to compute the next solution approximation on node 1:

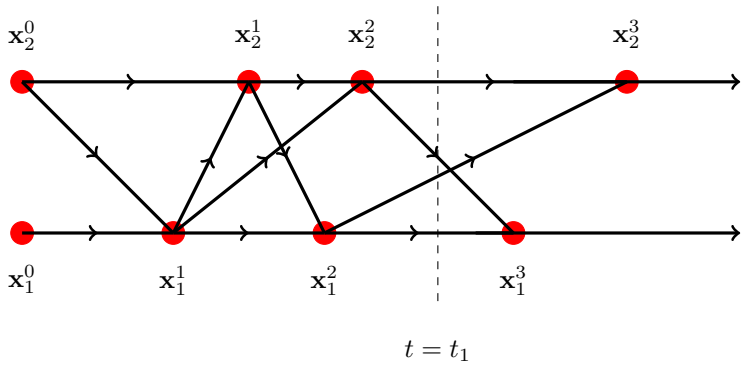

FIG. 1. *Directed acyclic graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ illustrating an example of two-node evolution of the solution approximations $\mathbf{x}_1^{\nu_1(t)}$ and $\mathbf{x}_2^{\nu_2(t)}$.*

$$\mathbf{x}_1^1 = M_{11}\mathbf{x}_1^0 + M_{12}\mathbf{x}_2^0.$$

Node 2, on the other hand, uses $\mathbf{x}_1^1$ instead of $\mathbf{x}_1^0$ to compute the next approximation,

$$\boldsymbol{x}_2^1 = M_{22}\boldsymbol{x}_2^0 + M_{12}\boldsymbol{x}_1^1,$$

as depicted in Figure 1. Hook and Dingle [11, Theorem 1] prove that the error operator $\Omega(\psi, M, t)$ consists of sums over all the paths within $\mathcal{G}(\mathcal{V}, \mathcal{E})$ of the corresponding iteration matrix blocks. For example, the error operator at time t_1 in Figure 1 is

$$\Omega(\psi, M, t_1) = \begin{bmatrix} M_{11}M_{11}M_{11} + M_{12}M_{21}M_{11} & M_{12}M_{22} + M_{11}M_{12} \\ M_{22}M_{21}M_{11} + M_{21}M_{11} & M_{22}M_{22} + M_{21}M_{12} + M_{22}M_{21}M_{12} \end{bmatrix}.$$

The authors further show that the convergence rate is bounded by the slowest propagation of information, defined as the shortest path in $\mathcal{G}(\mathcal{V}, \mathcal{E})$ from an initial solution approximation to a current approximation, leading to the error bound that forms the basis of our rejection criteria. Given a nonnegative iteration matrix M, i.e., all elements of M are nonnegative, Hook and Dingle [11, Theorem 3] show that $\Omega(t)$ is bounded as follows:

$$\|\Omega(t)\|_2 \leq \left\| \sum_{k=s(t)}^{l(t)} M^k \right\|_2, \tag{3.1}$$

where $s(t)$ and $l(t)$ are the lengths of the shortest and longest paths in $\mathcal{G}(\mathcal{V}, \mathcal{E})$ at time t, respectively. The goal now is to use (3.1) to develop a criterion for whether computational node i should accept or reject a new solution approximation $\mathbf{x}_j^{\psi(i,j,\nu_i(t))}$ obtained from node j. For notational brevity, we introduce $\psi_{ij}(\kappa) := \psi(i, j, \kappa)$.

To compare the new solution approximation $\mathbf{x}_j^{\psi_{ij}(\nu_i(t))}$ to the previous solution approximation received by computational node i from computational node j, we define $\zeta_{ij}(t)$ to be the index of the solution approximation on node i that was last directly influenced by a solution approximation from node j. In other words, we seek to denote the two most recent solution approximations received by node i from node j at time t as $\mathbf{x}_j^{\psi_{ij}(\nu_i(t))}$ and $\mathbf{x}_j^{\psi_{ij}(\zeta_{ij}(t))}$, respectively. Formally, $\zeta_{ij}(t) = \arg\max_\kappa\{\psi_{ij}(\kappa) < \psi_{ij}(\nu_i(t))\}$. Now, a bound on $\|\mathbf{x}_j^{\psi_{ij}(\nu_i(t))} - \mathbf{x}_j^{\psi_{ij}(\zeta_{ij}(t))}\|_2$ can be derived using (3.1). Let $\tau_{ij}[\kappa]$ be the time at which the solution approximation existed on computational

node j that would later be used to form $\mathbf{x}_i^\kappa$ so that $\nu_j\big(\tau_{ij}[\zeta_{ij}(t)]\big) = \psi_{ij}(\zeta_{ij}(t))$ and $\nu_j\big(\tau_{ij}[\nu_i(t)]\big) = \psi_{ij}(\nu_i(t))$. Note that $\mathbf{x}_j^{\psi_{ij}(\nu_i(t))}$ can now be expressed as $\mathbf{x}_j^{\nu_j(\tau_{ij}[\nu_i(t)])} = \widetilde{\mathbf{x}}_j\big(\tau_{ij}[\nu_i(t)]\big)$ and $\mathbf{x}_j^{\psi_{ij}(\zeta_{ij}(t))}$ as $\mathbf{x}_j^{\nu_j(\tau_{ij}[\zeta_{ij}(t)])} = \widetilde{\mathbf{x}}_j\big(\tau_{ij}[\zeta_{ij}(t)]\big)$. Thus, the following bound holds:

$$\mathbf{x}_j^{\psi_{ij}(\nu_i(t))} - \mathbf{x}_j^{\psi_{ij}(\zeta_{ij}(t))} = \underbrace{\widetilde{\mathbf{x}}_j\big(\tau_{ij}[\nu_i(t)]\big) - \mathbf{x}_j^*}_{\left[\Omega\left(\tau_{ij}[\nu_i(t)]\right)\mathbf{e}(0)\right]_j} + \underbrace{\mathbf{x}_j^* - \widetilde{\mathbf{x}}_j\big(\tau_{ij}[\zeta_{ij}(t)]\big)}_{\left[-\Omega\left(\tau_{ij}[\zeta_{ij}(t)]\right)\mathbf{e}(0)\right]_j}$$

$$\Rightarrow \|\mathbf{x}_j^{\psi_{ij}(\nu_i(t))} - \mathbf{x}_j^{\psi_{ij}(\zeta_{ij}(t))}\|_2 \le \left[\big\|\Omega\big(\tau_{ij}[\zeta_{ij}(t)]\big)\big\|_2 + \big\|\Omega\big(\tau_{ij}[\nu_i(t)]\big)\big\|_2\right]\|\mathbf{e}(0)\|_2.$$

Assuming that the initial solution approximation is the zero vector, one has $\|\mathbf{e}(0)\|_2 \le \|A^{-1}\|_2\|\mathbf{b}\|_2$. Assuming also that the iteration matrix M is nonnegative so that the Hook and Dingle bound (3.1) can be applied, the following bound is obtained:

(3.2)
$$\|\mathbf{x}_j^{\psi_{ij}(\nu_i(t))} - \mathbf{x}_j^{\psi_{ij}(\zeta_{ij}(t))}\|_2 \le \left[\left\|\sum_{k=s(\tau_{ij}[\zeta_{ij}(t)])}^{l(\tau_{ij}[\zeta_{ij}(t)])} M^k\right\|_2 + \left\|\sum_{k=s(\tau_{ij}[\nu_i(t)])}^{l(\tau_{ij}[\nu_i(t)])} M^k\right\|_2\right]\|A^{-1}\|_2\|\mathbf{b}\|_2.$$

Evaluating the bound (3.2) directly is very difficult in practice, primarily because none of A^{-1}, the τ_{ij} map, or the $s(t)$ and $l(t)$ functions are known a priori. Thus, to obtain a practical version of (3.2), the two individual finite series are bounded by a single infinite series:

$$\|\mathbf{x}_j^{\psi_{ij}(\nu_i(t))} - \mathbf{x}_j^{\psi_{ij}(\zeta_{ij}(t))}\|_2 \le 2\|A^{-1}\|_2\|\mathbf{b}\|_2 \sum_{k=s(\tau_{ij}[\zeta_{ij}(t)])}^{\infty} \|M\|_2^k.$$

Recall that $\|M\|_2$ is equal to the largest singular value of M, denoted as $\sigma_{\max}(M)$, and that $\|A^{-1}\|_2$ is equal to the reciprocal of the smallest singular value of A, denoted as $1/\sigma_{\min}(A)$. Finally, introduce $\tilde{s}_i(t)$ as a lower bound on $\min_{r\neq i} s\big(\tau_{ir}[\zeta_{ir}(t)]\big)$ so that if the geometric series above converges (i.e., if $\|M\|_2 < 1$), then

$$\|\mathbf{x}_j^{\psi_{ij}(\nu_i(t))} - \mathbf{x}_j^{\psi_{ij}(\zeta_{ij}(t))}\|_2 \le 2\,\frac{\|\mathbf{b}\|_2}{\sigma_{\min}(A)}\,\frac{\sigma_{\max}(M)^{\tilde{s}_i(t)}}{1-\sigma_{\max}(M)}. \tag{3.3}$$

The lower bound $\tilde{s}_i(t)$ is obtained in the following manner: Each computational node r sends its current value of $\tilde{s}_r(t)$ along with the current solution approximation to its neighbors. When computational node i receives a value of $\tilde{s}_r(t)$ from node r, that value is stored by node i as $\tilde{s}_r$. Additionally, every time that node i computes a new solution approximation, a separate counter $\tilde{s}_i^0$ is incremented. Once computational node i has received a value from each neighbor r with $M_{ir} \neq 0$, the values for both $\tilde{s}_i(t)$ and $\tilde{s}_i^0$ are set to $\min\big(\tilde{s}_i^0,\, 1 + \min_{r:M_{ir}\neq 0}\tilde{s}_r\big)$. Then the process repeats, with computational node i again collecting updated values for all relevant $\tilde{s}_r(t)$ before updating $\tilde{s}_i(t)$. Note that since the value received from computational node j for $\tilde{s}_j(t)+1$ should never be less than $\tilde{s}_i(t)$, the solution approximation $\mathbf{x}_j^{\psi_{ij}(\nu_i(t))}$ will only be accepted by computational node i if (3.3) is satisfied and the new value of $\tilde{s}_j(t)$ is such that $\tilde{s}_j(t)+1 \ge \tilde{s}_i(t)$. This additional constraint provides some resilience for when the value of $\tilde{s}_j(t)$ is itself corrupted. These two constraints form the rejection criterion for the rejection variant of the ASJ method presented in Algorithm 1.

Algorithm 1. Asynchronous Jacobi rejection variant (ASJ-R).

foreach *node* i=1,2,...,N **do**
  Initialize the algorithm with $\mathbf{x}_i^0 = \mathbf{0}$, $\tilde{s}_i = 0$, and $\tilde{s}_i^0 = 0$. Set $\kappa = 0$ and $\mathcal{S} = \{\}$.
  foreach $\mathbf{x}_j$ *and* $\tilde{s}_j$ *received from node* j **do**
    if $\|\mathbf{x}_j - \mathbf{x}_j^\kappa\|_2 \leq 2\frac{\|b\|_2}{\sigma_{\min}(A)}\frac{\sigma_{\max}(M)^{\tilde{s}_i}}{1-\sigma_{\max}(M)}$ *and* $\tilde{s}_j + 1 \geq \tilde{s}_i$ **then**
      set $\mathbf{x}_j^\kappa = \mathbf{x}_j$
      store $\tilde{s}_j$ in $\mathcal{S}$
      if $\mathcal{S}$ *contains* $\tilde{s}_r$ *for all* r *such that* $M_{ir} \neq 0$ **then**
        set $\tilde{s}_i = \min\{\tilde{s}_i^0, 1 + \min \mathcal{S}\}$
        set $\tilde{s}_i^0 = \tilde{s}_i$
        set $\mathcal{S} = \{\}$
      set $\mathbf{x}_i^{\kappa+1} = \sum_{r=1}^N M_{ir}\mathbf{x}_r^\kappa + \mathbf{c}_i$
      set $\tilde{s}_i^0 = \tilde{s}_i^0 + 1$
      communicate $\mathbf{x}_i^{\kappa+1}$ and $\tilde{s}_i$
      set $\kappa = \kappa + 1$

The development of stopping criteria for asynchronous methods remains an active area of research (see, e.g., the review by Magoulés and Gbikpi-Benissan [16] or the survey by Spiteri [20, section 3]). Hook and Dingle [11] have each node report to a root node when a local stopping criterion is met. Each node then continues iterating until the root node indicates that it has received reports from all nodes. Instead of designating a root node, our approach has each node collect such reports in a decentralized fashion. Each node i checks the following criterion after a new solution approximation $\mathbf{x}_i^{\nu_i(t)}$ is computed:

$$\|D_{ii} d\mathbf{x}_i^{\nu_i(t)}\|_\infty < \epsilon \frac{\|b\|_2}{\sqrt{m}}, \tag{3.4}$$

where $d\mathbf{x}_i^{\nu_i(t)} = \mathbf{x}_i^{\nu_i(t)} - \mathbf{x}_i^{\nu_i(t)-1}$ and $\epsilon > 0$ is a prescribed tolerance. If the bound in (3.4) is satisfied, node i reports to all other nodes that it has locally converged while continuing to iterate. If a successive solution approximation on node i fails to satisfy (3.4), node i reports to all other nodes that it has no longer locally converged. If a node j has locally converged and has received reports that all other nodes have locally converged, it starts a "convergence duration" timer while continuing to iterate. If node j either determines that it has no longer locally converged or receives a report that another node has no longer locally converged, the timer is set back to zero. Each node continues to iterate until either (i) the specifying convergence duration is achieved or (ii) a specified maximum number of iterations is reached. Ideally, the convergence duration is chosen so that the information that a given node has no longer locally converged has time to arrive at all other nodes before those other nodes stop iterating. As such, the proper choice of convergence duration likely depends on parameters that determine the speed at which information propagates across the nodes, e.g., network hardware, sparsity pattern of A, etc.

4. Numerical results. Having derived the modified ASJ in section 3, shown in Algorithm 1, we now numerically evaluate the proposed method. We choose the benchmark problem to have an analytic solution so that we can verify the implementation of Algorithm 1. We then compare the method convergence against that of the

traditional ASJ method when the natural and malevolent data corruptions described in sections 2.1 and 2.2, respectively, are present. The comparison includes results for both when the singular values in the path-length rejection criterion (3.3) are "exact" (computed by each node before the iteration using an eigensolver) and when the singular values are approximated with varying levels of relative accuracy.

Each run is to a convergence tolerance of $\epsilon = 10^{-5}$ and performed on a single 36-core node of the Quartz supercomputer at the Livermore Computing Complex. The "exact" singular values in the rejection criterion (3.3) are computed using the bidiagonal divide-and-conquer singular value decomposition algorithm in the *Eigen* software library [9]. The *Skywing* software platform [19], which is designed to support asynchronous algorithms, is used for execution and communication for all methods. Each run leverages 16 *Skywing* agents to serve as the $N = 16$ computational nodes. More on the motivation and implementation of *Skywing* can be found in [7] and on GitHub at https://github.com/LLNL/Skywing, respectively.

To account for the stochastic nature of both asynchronous algorithms and the corruption model, ensembles of 30 runs are performed for each study. Quantities such as the ensemble wall-clock time and relative solution error are reported as a geometric mean defined as

$$\bar{a} = \exp\left(\frac{1}{s}\sum_{k=1}^{s}\log\left(a_k\right)\right),$$

where a is the quantity of interest and $s = 30$ corresponds to the ensemble of 30 runs. When a is time-dependent, such as when a represents the relative solution error $\|\mathbf{e}(t)\|_2/\|\mathbf{x}^*\|_2$, the values of $\bar{a}(t)$ are obtained using linear interpolants of $a_k(t)$. If $a_k(t)$ contains an IEEE 754 `NaN` or $\pm\infty$ value, as can happen in the relative solution error with data corruption, the corresponding value of $\bar{a}(t)$ is omitted.

The linear system (2.1) solved throughout most of this section is obtained from a finite difference discretization of the following Poisson problem on the unit square:

$$\begin{aligned} -\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) &= f, \quad x \in (0,1), y \in (0,1), \\ u(0,y) = u(1,y) = u(x,0) &= u(x,1) = 0, \end{aligned} \tag{4.1}$$

where the choice of $f(x,y) = -2\pi^2\sin(\pi x)\sin(\pi y)$ results in an analytic solution $u(x,y) = -\sin(\pi x)\sin(\pi y)$. The unit square is uniformly discretized into $\ell+1 \times \ell+1$ squares of length $h = 1/(\ell+1)$. Such a discretization along with the Dirichlet boundary condition in (4.1) leaves the values of $u(x_i, y_j)$ to be determined at the square vertices, where $x_i = (i+1)h$ and $y_j = (j+1)h$ for $i = 0, \ldots, \ell-1$ and $j = 0, \ldots, \ell-1$. Let the kth element of $\mathbf{x} \in \mathbb{R}^{\ell^2}$ and $\mathbf{b} \in \mathbb{R}^{\ell^2}$ in (2.1) be $u(x_i, y_j)$ and $f(x_i, y_j)$, respectively, with $i = (k \mod \ell)$ and $j = k\ell$. With the Laplace operator discretized across the points (x_i, y_j) using centered finite difference, the matrix A in (2.1) is defined as the following $\ell^2 \times \ell^2$ block tridiagonal matrix:

$$A = \begin{bmatrix} L & -I & & & \\ -I & L & -I & & \\ & \ddots & \ddots & \ddots & \\ & & -I & L & -I \\ & & & -I & L \end{bmatrix}, \text{ where } L = \begin{bmatrix} 4 & -1 & & & \\ -1 & 4 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 4 & -1 \\ & & & -1 & 4 \end{bmatrix}$$

and $I \in \mathbb{R}^{\ell \times \ell}$ is the identity matrix. The linear system is evenly distributed across the agents, i.e., $m_1 = \ldots = m_{16}$.

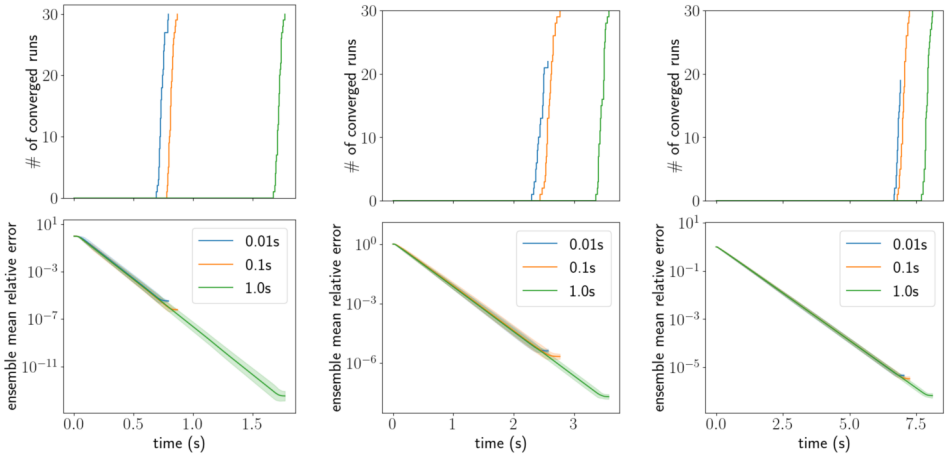

FIG. 2. *Ensemble convergence of ASJ for various convergence durations on Poisson benchmark problem with $m = 144$ (left), $m = 400$ (center), and $m = 784$ (right). All ensemble runs converge for all durations with the smallest system size; however, the larger system sizes indicate that the agents are unable to reach consensus on convergence for the smallest duration.*

4.1. Convergence duration. Recall that the stopping criteria discussed at the end of section 3 involves a user-specified convergence duration for both the traditional ASJ and the modified asynchronous Jacobi (ASJ-R) methods. As such, we first evaluate the impact of the convergence duration on the asynchronous solving of the benchmark problem (4.1) in the absence of data corruption. The number of squares chosen to discretize the unit square domain is selected so that the linear system results in $m = 144$, $m = 400$, and $m = 784$ unknowns, respectively. Figure 2 shows the dependence of the convergence behavior on the convergence duration. For the smallest system, $m = 144$, all runs converge for all three duration values both in the sense that the agents reach consensus on convergence and in the sense that the resulting relative solution error is below the target tolerance. As the system size increases to $m = 400$, however, the shortest duration 0.01 s results in around 30% of the ensemble runs ending with the agents unable to reach consensus on convergence, even though the relative solution error is near or below the target tolerance. With the largest system size $m = 784$, the shortest duration results in slightly more than 30% of the ensemble runs failing to reach consensus. Increasing the duration to 0.1 s remedies the lack of consensus for all three system sizes evaluated. While these results could motivate a duration of 0.1 s for evaluating the methods with corruption present, we choose the 1 s duration that is an order of magnitude longer to increase confidence that convergent results are due to the algorithm being able to continue decreasing the error despite corruption and not the iteration stopping at the right moment between corruption occurrences.

4.2. Path length rejection variant with data corruption. With the convergence duration established, we now evaluate the resilience of the ASJ and ASJ-R methods to both natural and malevolent corruption, as defined in sections 2.1 and 2.2, respectively. To choose a system size, we consider the sizes of the power flow benchmark systems mentioned in [21], which range from ≈ 14 to ≈ 300 unknowns. As such, all corruption studies discretize the unit square such that the linear system has $m = 400$ unknowns ($\ell = 20$) and results in each agent communicating with one or two neighbors in a line configuration. We present time for the corruption studies as

relative to the average time to convergence (≈ 3.5 s) obtained in section 4.1 for $m = 400$; i.e., a time to convergence of 1.0 indicates that the method converged in the same amount of time as it would without corruption present.

Natural data corruption. Our first investigation introduces bit flips to communicated data, similar to the studies performed by Anzt, Dongarra, and Quintana-Ortí [2]. We aim to assess the impact of the probability p of a bit flip on the convergence of both ASJ and ASJ-R. As discussed in section 2.1, corruption is applied at each iteration and on every agent with probability p to all communicated data. The elements of $\mathbf{x}_i^\kappa$ in both ASJ and ASJ-R are stored as IEEE 754 double floating point numbers, whereas the values of the approximate shortest path length $\tilde{s}_i(t)$ in ASJ-R are stored as signed integers. If a given double floating point value is chosen to be corrupted, a bit index out of a given subset of its 64-bit representation is randomly chosen to be flipped. If a given signed integer value is chosen to be corrupted, a bit index out of any of its 32-bit representation is randomly chosen to be flipped.

For our first study, we fix the probability of a bit flip in a given communicated value to be $p = 0.01$. Following Anzt, Dongarra, and Quintana-Ortí [2], we investigate the following double floating point subsets: the lower mantissa $\text{IE}^3([0{-}25])$, the upper mantissa $\text{IE}^3([26{-}51])$, the exponent $\text{IE}^3([52{-}62])$, and the sign bit $\text{IE}^3(63)$. We start with the lower mantissa subset $\text{IE}^3([0{-}25])$, which leads to floating point value corruption ranging from $1/2^{52} \approx 10^{-16}$ to $1/2^{27} \approx 10^{-8}$ relative to the original values. Figure 3 shows the convergence behavior for ASJ and ASJ-R. For both ASJ and ASJ-R, all runs in the respective bit flip ensemble converge with times to solution that are approximately the same as those of the respective baseline (no corruption) ensemble. The indifference of the ASJ convergence behavior to lower-mantissa flips is consistent with Anzt, Dongarra, and Quintana-Ortí [2], where it was found that ASJ convergence behavior is not affected by such bit flips until the relative residual norm is reduced to a very small value. The indifference is due to the corruption caused by lower-mantissa flips being too small to significantly affect the iteration evolution to the tolerance $\epsilon = 10^{-5}$, as relaxation methods are inherently robust to small amounts of corruption.

To introduce larger corruption, we now investigate the sign bit $\text{IE}^3(63)$ subset, which leads to floating point value corruption of 2 relative to the original values. Figure 4 shows the convergence behavior for ASJ and ASJ-R. For ASJ, the effect of the more significant corruption from sign bit flips is evident, as the solution error of all ASJ runs decreases at first but then stagnates at a level well above the convergence tolerance, consistent with the findings of [2]. The introduction of the rejection criterion

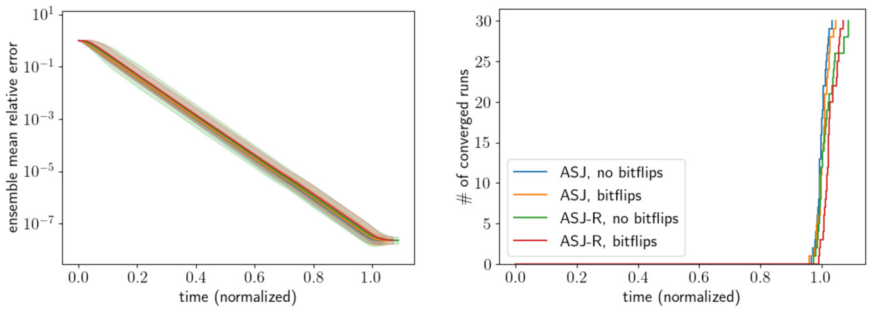

FIG. 3. *Ensemble convergence of ASJ and ASJ-R with bit flip probability $p = 0.01$, with double floating point flips limited to the lower mantissa $\text{IE}^3([0{-}25])$. Convergence is achieved in all ASJ and ASJ-R runs, with times to solution comparable to the respective baseline (no corruption) values.*

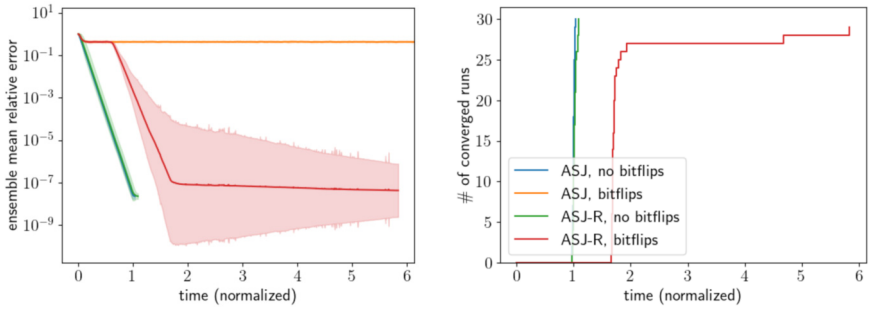

FIG. 4. *Ensemble convergence of ASJ and ASJ-R with bit flip probability* $p = 0.01$*, with double floating point flips limited to the sign bit* IE^3(63)*. Convergence is lost for all of the ASJ runs and achieved for all ASJ-R runs, albeit with longer times to solution.*

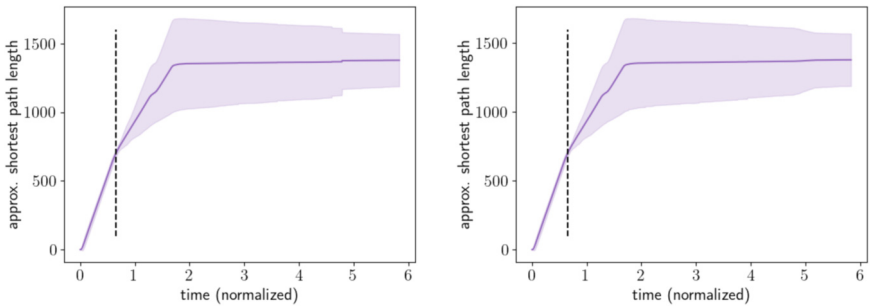

FIG. 5. *Approximate shortest path length* $\tilde{s}_i(t)$ *used by the ASJ-R algorithm with bit flip probability* $p = 0.01$*, with double floating point flips limited to the sign bit* IE^3(63) *(left:* $i = 7$*; right:* $i = 9$*). The approximate shortest path length reaches* 700 *at around the time the stagnation period ends in Figure* 4 *(denoted by dashed black line).*

in ASJ-R, based on (3.3), restores convergence in all of the runs, albeit with longer times to solution: 1.75 times longer for about 80% of the runs and 6 times longer for the slowest run. This increased time to solution is explained starting with the presence of a stagnation period for all ASJ-R runs with bit flips. Given that the ASJ-R error during this stagnation period coincides with the stagnated ASJ error, it can be inferred that the value of the approximate shortest path length $\tilde{s}_i(t)$ in (3.3) during the stagnation period is not yet large enough for ASJ-R to reject the corruption. Figure 5 shows the value of the approximate shortest path length for two agents. The values of $\tilde{s}_7(t)$ and $\tilde{s}_9(t)$ grow roughly linearly with time until the end of the stagnation period around $t = 0.6$. Around $t = 0.65$, the value of $\tilde{s}_i(t)$ is large enough (≈ 700) to start rejecting data containing bit flips so that all 30 runs can resume converging. The values of $\tilde{s}_7(t)$ and $\tilde{s}_9(t)$ continue to grow after $t = 0.65$, albeit at a slower rate due to the rejections.

To introduce corruption with a relative magnitude between the sign bit and lower mantissa subsets, we now investigate the upper mantissa IE^3([26–51]) subset, which leads to floating point value corruption ranging from $1/2^{26} \approx 10^{-7}$ to $1/2$ relative to the original values. In Figure 6, the corruption from upper-mantissa flips is still large enough to prevent convergence in all ASJ runs, with the solution error stagnating at a level above tolerance but lower than with sign bit flips, consistent with the smaller-magnitude value changes and with the findings of [2]. For ASJ-R, convergence is

achieved in all runs with a time to solution around 3 times longer for all but one run that took around 4.25 times longer. There is a stagnation period followed by a return to convergence behavior, as seen with sign bit flips, only the period lasts longer until about $t = 0.7$. There is also a significant reduction in convergence rate starting around $t = 1.5$ that is not addressed until around $t = 2.75$. Both of these differences are explained by the observation that as $\tilde{s}_i(t)$ grows, there is always corruption of a certain magnitude that will not be rejected. Thus, to reject the larger of the magnitude corruption range that upper-mantissa bit flips can cause, $\tilde{s}_i(t)$ must reach a larger value than with sign bit flips before convergence is restored from the stagnation period. This is confirmed in Figure 7, where the value for $\tilde{s}_i(t)$ at $t = 0.7$ is about 750. The degraded convergence from $t = 1.5$ until $t = 2.75$ is explained by the requirement that $\tilde{s}_i(t)$ reach an even larger value before the smaller of the corruption range is rejected. It is worth noting that Anzt, Dongarra, and Quintana-Ortí [2] also see a slower rate of convergence for synchronous Jacobi with bit flips for likely the same reason.

The last subset to investigate is the exponent subset IE^3([52-62]), which leads to floating point value changes ranging from 1 to $2^{1023} - 1 \approx 10^{308}$ relative to the original values. In Figure 8, the corruption from exponent flips is large enough that all ASJ runs result in solution iterates containing nonfinite (i.e., IEEE 754 `NaN`) values after just a few iterations, which is consistent with the findings of [2]. For ASJ-R, convergence

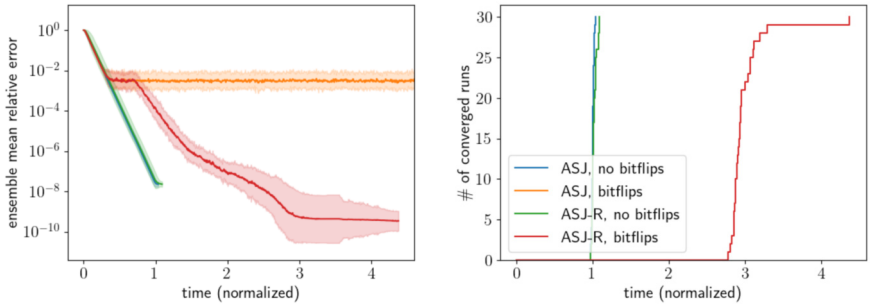

FIG. 6. *Ensemble convergence of ASJ and ASJ-R with bit flip probability* $p = 0.01$*, with double floating point flips limited to the upper mantissa* IE^3([26−51])*. Convergence is lost for all of the ASJ runs and achieved for all ASJ-R runs. The times to solution of the convergent runs are longer than those of sign bit flips (Figure* 4*); however, the times to tolerance* $\epsilon = 10^{-5}$ *are about the same as sign bit flips.*

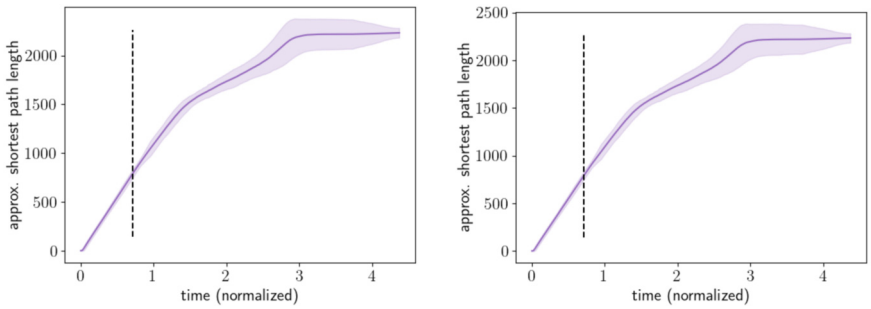

FIG. 7. *Approximate shortest path length* $\tilde{s}_i(t)$ *used by the ASJ-R algorithm with bit flip probability* $p = 0.01$*, with double float point flips limited to the sign bit* IE^3([26−51]) *(left:* $i = 7$*; right:* $i = 9$*). The approximate shortest path length reaches* 750 *at around the time the stagnation period ends in Figure* 6 *(denoted by dashed black line).*

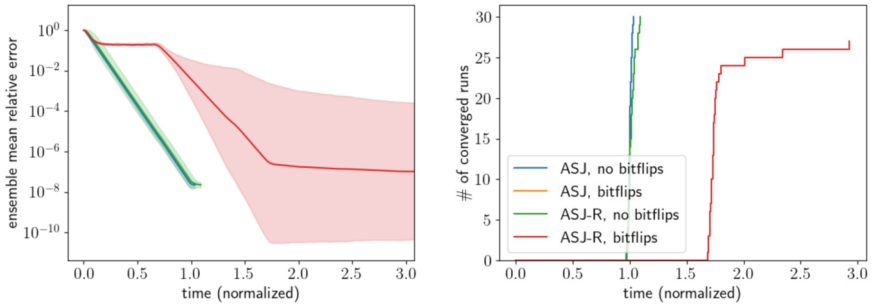

FIG. 8. *Ensemble convergence of ASJ and ASJ-R with bit flip probability* $p = 0.01$*, with double floating point flips limited to the exponent* IE^3([52−62])*. Convergence is lost for all of the ASJ runs and achieved for most of the ASJ-R runs, with the times to solution being mostly comparable to that of sign bit flips (Figure* 4*).*

is achieved in most runs with times to solution similar to that of sign bit flips: around 1.75 times for almost all the runs with one run taking almost 3 times. The slowdown to 1.75 times is explained as with the sign and upper-mantissa bit flips: $\tilde{s}_i(t)$ must reach a certain value before larger-magnitude corruption data are rejected enough to restore convergence. The slowdown to 3 times is explained as with the upper-mantissa bit flips: After the initial stagnation period, $\tilde{s}_i(t)$ must continue to grow before smaller-magnitude corruption is also rejected. The runs that do not converge are, however, not explained by prior observations for sign or upper-mantissa bit flips. In those runs, the corruption that is not rejected during the stagnation period is large enough to cause the evolution of the solution approximations to substantially deviate from that predicted by the convergence theory on which the rejection criterion (3.3) is derived. The deviation is large enough that the updates required to drive the solution approximation back toward the exact solution are now significantly larger than those predicted by the convergence theory, causing the rejection criterion to reject the valid updates that would otherwise restore convergence.

With an understanding of how ASJ and ASJ-R perform on bit flips with probability $p = 0.01$ in subsets of the floating point double, we now investigate flipping any of 64 bits with probability p values of 0.0025, 0.005, 0.01, 0.015, 0.02, and 0.04. All of the runs in any ASJ ensemble corresponding to $p > 0$ quickly saw `NaN` values in the solution approximations. Noting similar occurrence of `NaN` values in Figure 8, one can infer that, even with bit flip probability as low as $p = 0.0025$, the ASJ runs quickly experience the occurrence of one or more exponent bit flips. For ASJ-R, the convergence behavior is shown in Figure 9. All but one of the runs for $p \leq 0.015$ converged, with a large majority of runs still converging for $p = 0.02$ and $p = 0.04$. As one might expect, increasing p results in the times to solution increasing from around 2.25 times to around 4 times, with increasing variability as p increases. Such behavior is explained by the prior observations: that increasing p results in increased likelihood of bit flips causing corruption that is small enough to avoid rejection by the increasing $\tilde{s}_i(t)$ yet large enough to (i) delay convergence until $\tilde{s}_i(t)$ increases enough or (ii) cause the solution approximation to deviate substantially enough that the rejection criterion actually prevents convergence. All in all, the ASJ-R algorithm has a very high probability of converging even when a large number of bit flips occur; e.g., $p = 0.04$ of communicated data are corrupted at each iteration.

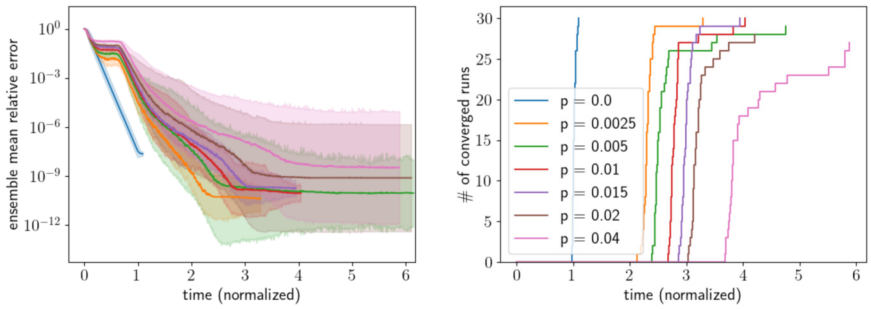

FIG. 9. *Ensemble convergence of ASJ-R with bit flip corruption probabilities ranging from* $p = 0$ *to* $p = 0.04$*, with double floating point flips in any of the* 64 *bits. Convergence is lost for all of the ASJ runs and achieved for a large majority of ASJ-R runs, with increasing* p *resulting in larger times to solution and a larger proportion of ensemble runs that fail to converge.*

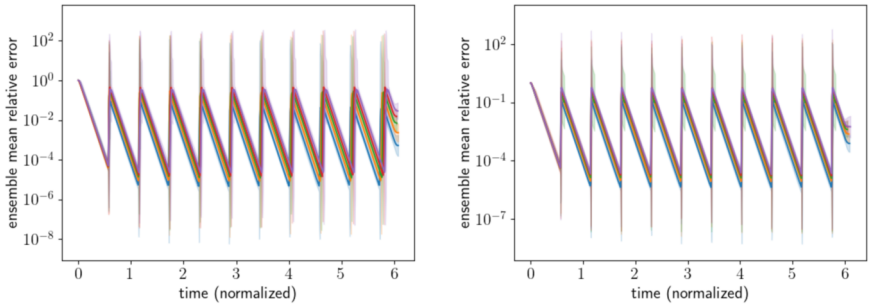

FIG. 10. *Ensemble convergence of ASJ with malevolent corruption with time to failure* $\omega_f = 2$ *s* ($t \approx 0.6$ *in normalized time), various recovery times* ω_r*, and various offset magnitudes* δ *(left: various* ω_r *values and* $\delta = 0.2$*; right:* $\omega_r = 0.02$ *s and various* δ *values). Convergence is not obtained for any of the runs.*

Malevolent data corruption. Our second investigation introduces malevolent manipulation of stored data, as defined in section 2.2. As opposed to the investigation with natural bit flips, here we limit the corruption to double floating point values (i.e., no corruption of signed integer values for ASJ-R). We aim to assess the impact of the recovery time ω_r and mean manipulation offset δ on the convergence of both ASJ and ASJ-R. As described in section 2.2, while agent i is in a degraded state, every element of $\mathbf{x}_i^\kappa$ is manipulated by an additive offset sampled from a normal distribution with a mean of δ and a standard deviation of $\frac{1}{2}\delta$. We introduce corruption to agent $i = 9$ with a time to failure $\omega_f = 2$ s, so the first degraded state occurs before the agents would otherwise start the convergence duration timers without corruption (≈ 2.5 s). We study recovery times ω_r selected from 0.01, 0.02, 0.03, 0.04, and 0.05 s that represent relative uptimes of 99.5%, 99%, 98.5%, 98%, and 97.5%. The offset magnitudes δ are selected from 0.1, 0.2, 0.3, 0.4, and 0.5.

Figure 10 shows the convergence behavior of ASJ for time to failure $\omega_f = 2$ s with various ω_r and fixed $\delta = 0.2$ and with fixed $\omega_r = 0.02$ s (99% uptime) and various δ. All of the ASJ runs fail to converge for the recovery times and offset magnitudes explored. The effect of the corruption is seen around $t = 0.6$, as the solution error is small enough to begin the convergence duration but then rapidly increases due to the corruption on agent 9 that quickly propagates to other agents. The error increases to a peak that coincides with agent 9 returning to a normal state, after which the error does decrease until the next rapid increase, when agent 9 is again degraded.

Increasing either the recovery time ω_r or the offset magnitude δ effectively shifts the overall error evolution upward. While one might technically obtain convergence to the tolerance $\epsilon = 10^{-5}$ by decreasing the convergence duration for the smallest ω_r and δ, the results in Figure 10 show that the ASJ method cannot reliably converge to a given tolerance with malevolent corruption.

In contrast, all of the ASJ-R runs in Figure 11 converge for recovery time $\omega_r = 0.02$ s and various offset magnitudes δ. As with the ASJ results in Figure 10, the solution error does increase with the arrival of the first degraded state in agent 9; however, the values of $\tilde{s}_i(t)$ on neighboring nodes have reached large enough values to reject at least some of the corruption and limit the jump in error. By the time agent 9 enters the second degraded state, the values of $\tilde{s}_i(t)$ on neighboring nodes are such that more corruption is rejected than during the first degraded state, resulting in an even smaller jump in error than the first degraded state. Figure 12 shows similar progressive limiting of solution error increases during degraded states for various recovery times ω_r with offset magnitude $\delta = 0.2$. Note that some of the runs with longer recovery times $\omega_r \geq 0.03$ s do either take longer to converge or fail to converge. Both behaviors are explained by the longer recovery times leading to more corruption occurring during the first degraded state that can be small enough in magnitude to avoid rejection yet large enough to drive up the error on all agents. As we saw in section 2.1, corruption

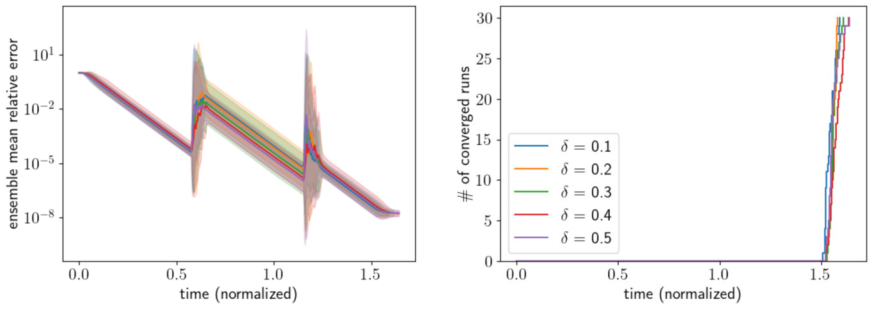

FIG. 11. *Ensemble convergence of ASJ-R with malevolent corruption with time to failure $\omega_f = 2$ s ($t \approx 0.6$ in normalized time), recovery time $\omega_r = 0.02$ s, and various offset magnitudes δ. Convergence is achieved for all runs, resulting in times to solution around 1.5 times longer than without corruption.*

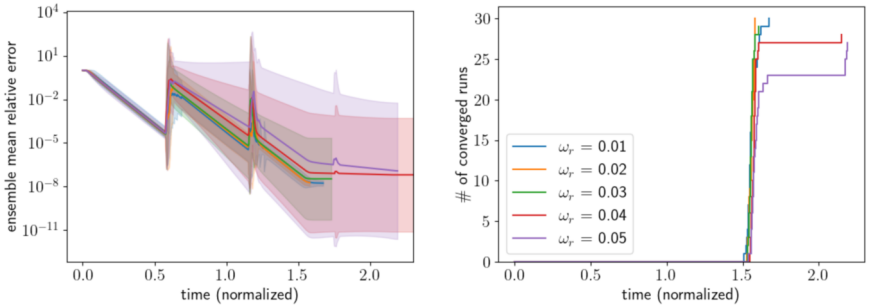

FIG. 12. *Ensemble convergence of ASJ-R with malevolent corruption with time to failure $\omega_f = 2$ s, various recovery times ω_r, and offset magnitude $\delta = 0.2$. Convergence is achieved for all runs with $\omega_r = 0.01$ s and $\omega_r = 0.02$ and almost all runs with larger ω_r, resulting in times to solution around 1.5 times longer than without corruption.*

leads to either a stagnation period that is eventually corrected, leading to delayed convergence, or enough deviation from theory that the rejection criterion rejects all further updates, leading to nonconvergence. That said, the ASJ-R method restores convergence in almost all runs with the time to failure $\omega_f = 2$ s, at least for the recovery times and offset magnitudes selected

4.3. Path-length rejection considerations. Recall that the ASJ-R rejection criterion (3.3) is developed on theory that uses the exact shortest path length $s_i(t)$, which is typically not available to the agents and therefore replaced by an approximation $\tilde{s}_i(t)$. We saw in section 4.2 that whether $\tilde{s}_i(t)$ in (3.3) is sufficiently large to reject significant corruption at a given time t has a profound impact on the convergence of ASJ-R, ranging from a temporary stagnation period that results in a longer time to solution to persistent stagnation that prevents convergence altogether. While a more rigorous study is warranted for future work, the values of $\tilde{s}_i(t)$ as defined in Algorithm 1 are found to consistently underestimate the values of $s_i(t)$ for runs that were anecdotally selected. As such, one might both significantly reduce the ASJ-R time to solution and increase the likelihood of convergence in the presence of corruption with a more accurate approximate shortest path length $\tilde{s}_i(t)$ that reduces or eliminates the stagnation issues in section 4.2.

Another consideration for the practical use of ASJ-R is the dependence of the rejection criterion (3.3) on singular values. For the system sizes considered in section 4.2, the values of $\sigma_{\min}(A) \approx 0.0447$ and $\sigma_{\max}(M) \approx 0.989$ are relatively cheap to compute locally on each agent; however, one might want to apply ASJ-R to large systems or to systems where agents do not have access to all rows of A. As such, the malevolent corruption study with time to failure $\omega_f = 2$ s, recovery time $\omega_r = 0.02$ s, and offset magnitude $\delta = 0.2$ is repeated for ASJ-R but with either $\sigma_{\min}(A)$ or $\sigma_{\max}(M)$ replaced in (3.3) by approximate values. Figure 13 shows the convergence behavior of ASJ-R with $\sigma_{\min}(A)$ replaced by the numerically computed value scaled by one of 10^{-4}, 10^{-2}, 1, 10^2, or 10^4. All the runs converge when the approximated singular value is smaller than the true value, where the smaller singular value results in a looser bound in the rejection criterion. The looser bound means that $\tilde{s}_i(t)$ needs to attain larger values to reject the same data as when the true singular value is used, explaining why the smaller approximated values result in longer times to solution. Increasing the approximated singular values above the true value results in a tighter bound, which likely offsets some of the underestimation in the shortest path length.

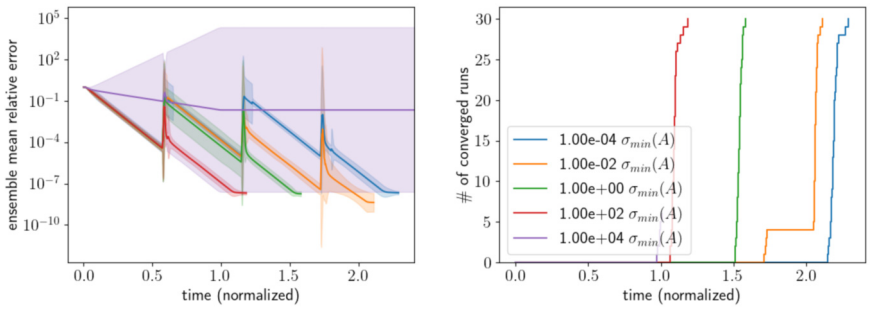

FIG. 13. *Ensemble convergence for ASJ-R with malevolent data corruption using various scaled values of* $\sigma_{\min}(A)$ *in the rejection criterion* (3.3). *Convergence is attained in almost all runs for all scaling factors except* 10^4 *scaling, with times to solution values improving for larger scaling factors.*

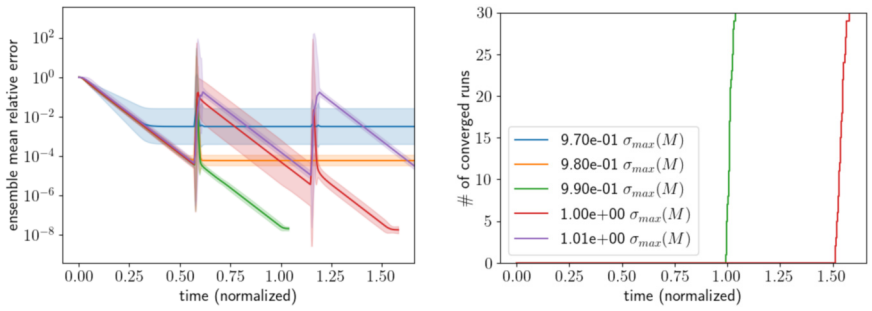

FIG. 14. *Ensemble convergence for ASJ-R with malevolent data corruption using various perturbed values of* $\sigma_{\max}(M)$ *in the rejection criterion* (3.3). *Convergence is only attained for runs with scaling factors* 1 *and* 0.99, *with optimal times to solution for* 0.99 *scaling factor.*

Overall, this results in a shorter time to solution, almost that of ASJ-R without corruption, from the additional rejection for 10^2 and in nonconvergence too much rejection for 10^4. While ASJ-R appears relatively robust to approximation of $\sigma_{\min}(A)$, Figure 14 shows that the method is much less robust to $\sigma_{\max}(M)$ being replaced by the numerically computed value scaled by one of 0.98, 0.99, 1, 1.01, or 1.02. When the scaling value 0.99 is used, the resulting smaller singular value results in a tighter bound in the rejection criterion, leading to a time to solution equal to that of ASJ-R without corruption. All the other scaling values, however, caused the rejection criterion to be too restrictive or too passive to restore convergence. It is worth noting that the results from Figures 13 and 14 both support the hypothesis that a better shortest path length approximation $\tilde{s}_i(t)$ will significantly improve ASJ-R performance in the presence of corruption.

4.4. Margulis–Gabber–Galil expander graph problem. The Poisson benchmark (4.1) results in linear systems of a particular structure; i.e., the matrix A in (2.1) has only two off-diagonal bands. To evaluate whether the resilience to corruption seen in the prior sections extends beyond that particular sparsity pattern, the Margulis–Gabber–Galil (MGG) expander graph is leveraged from the *NetworkX* software library [10] (see https://networkx.org). Specifically, the new linear system is defined by $A = I - G/8$, where $I \in \mathbb{R}^{400\times 400}$ is the identity matrix, $G \in \mathbb{R}^{400\times 400}$ is the MGG expander graph with degree 8, and $\mathbf{b} \in \mathbb{R}^{400}$ is a vector of ones. As before, the linear system is evenly distributed across the agents, i.e., $m_1 = \ldots = m_{16}$. Figure 15 shows that the sparsity pattern of such A results in substantially more connectivity between agents compared to that of the Poisson benchmark.

A convergence duration study is first conducted for the new linear system, as was done in section 4.1, to confirm that 1 s should still be used. The resulting average time to convergence ($\approx$ 2.9 s) for $m = 400$ is used to normalize the time in the new linear system. Next, bit flips in any of the 64 bits of the floating point double are introduced to communicated data with the same probabilities as those used for the Poisson benchmark. Figure 16 shows that the ASJ-R method retains similar resilience to the bit flip corruption for the new linear system as shown for the Poisson benchmark. That said, the increased connectivity in the new linear system does seem to result in longer relative times to solution and a larger likelihood that a given ASJ-R run does not converge, as the probability p of bit flips is increased. This behavior is likely due to how a corrupted broadcasted solution approximation will be received by more neighbors due to the increased connectivity. Malevolent manipulation of

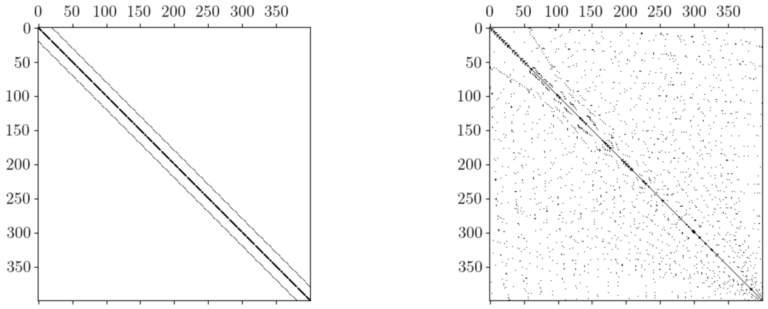

FIG. 15. *Sparsity pattern of matrix A in the Poisson benchmark (left) and in the new linear system that leverages the MGG expander graph (right). The new linear system results in communicated data from each agent being sent to significantly more neighboring agents than the Poisson benchmark, where communicated data are sent to at most two neighboring agents.*

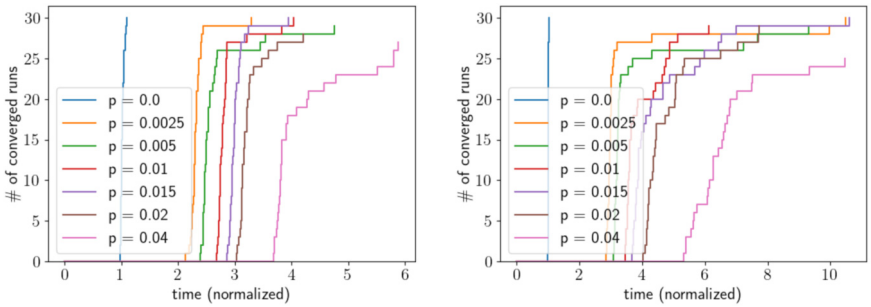

FIG. 16. *Ensemble convergence of ASJ-R for the Poisson benchmark (left) and the MGG system (right) with bit flip corruption probabilities ranging from $p = 0$ to $p = 0.04$, with double floating point flips in any of the 64 bits. The convergence behavior is comparable, with increasing p for the MGG system resulting in longer relative times to solution and a larger likelihood that a given run does not converge.*

stored data on agent $i = 9$ is also explored using the same time to failure ω_f, recovery times ω_r, and offset magnitudes δ as used in the Poisson benchmark. Figure 17 shows that the ASJ-R method retains resilience to the malevolent corruption for the new linear system, with the relative times to solution being slightly faster than those for the Poisson benchmark. Whereas the increased connectivity is a disadvantage for bit flip corruption, the connectivity here provides a faster mechanism to correct the corruption both on agent $i = 9$ and on any other agents that failed to reject the corrupted updates.

4.5. Steady-state advection-diffusion problem. While having the same sparsity pattern as the Poisson benchmark problem (4.1), a finite difference discretization of the following steady-state advection-diffusion problem on the unit square is leveraged to explore the impact of including advection:

$$Pe\left(\frac{\partial u}{\partial x}+\frac{\partial u}{\partial y}\right)-\left(\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}\right)=1, \quad x\in(0,1), y\in(0,1), \tag{4.2}$$
$$u(0,y)=u(1,y)=u(x,0)=u(x,1)=0.$$

Here, Pe denotes the Peclet number. The unit square domain is discretized as with the Poission problem (4.1), and all differential operators are discretized across the points

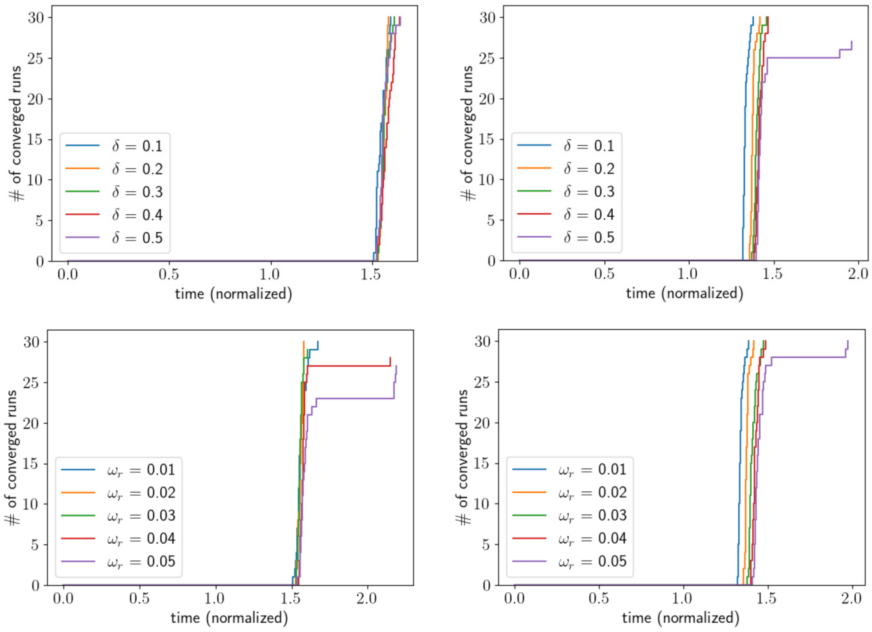

FIG. 17. *Ensemble convergence of ASJ-R in the presence of malevolent corruption for the Poisson benchmark (left column) and the MGG system (right column) with recovery time $\omega_r = 0.02$ s and various offset magnitudes δ (top row) and various recovery times ω_r and offset magnitude $\delta = 0.2$ (bottom row). The convergence behavior is comparable for the two linear systems.*

(x_i, y_j) using centered finite difference; thus, the matrix A in (2.1) is now defined as the following $\ell^2 \times \ell^2$ block tridiagonal matrix:

$$A = \begin{bmatrix} S & a_+ I & & & \\ a_- I & S & a_+ I & & \\ & \ddots & \ddots & \ddots & \\ & & a_- I & S & a_+ I \\ & & & a_- I & S \end{bmatrix}, \text{ where } S = \begin{bmatrix} 4 & a_+ & & & \\ a_- & 4 & a_+ & & \\ & \ddots & \ddots & \ddots & \\ & & a_- & 4 & a_+ \\ & & & a_- & 4 \end{bmatrix}$$

with $a_\pm = \pm Pe\frac{h}{2} - 1.0$. The Peclet number is chosen to be 10 to emphasize advection relative to diffusion. As before, the linear system is evenly distributed across the agents, i.e., $m_1 = \ldots = m_{16}$.

A convergence duration study is first conducted for the new linear system, as was done in section 4.1, to confirm that 1 s should still be used. With the resulting average time to convergence of ≈ 2.0 s for $m = 400$, the convergence duration of 1 s is indeed longer relative to the average time to convergence than with the Poisson benchmark or MGG system. That said, the convergence duration of 1 s still ensures that convergent results are due to the algorithm being able to continue in the presence of corruption instead of the iteration stopping at the right moment between corruption occurrences. As with the Poisson benchmark and MGG system, bit flips in any of the 64 bits of the floating point double are introduced to communicated data. Figure 18 shows that the ASJ-R method retains similar resilience to the bit flip corruption for the new linear system as shown for the Poisson benchmark (note that time is normalized by the average time to convergence of ≈ 2.0 s). That the relative times to solution for the steady-state advection-diffusion problem are longer than those for the Poisson

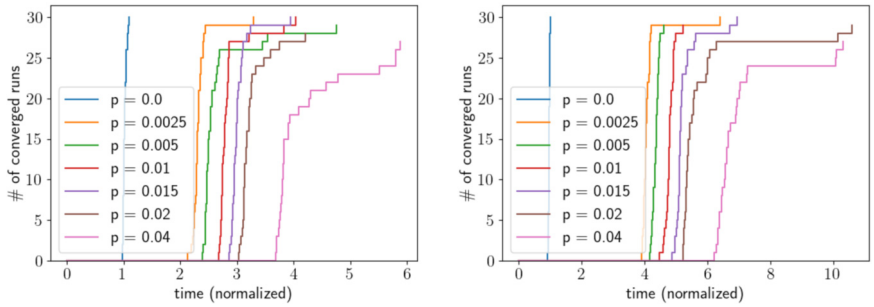

FIG. 18. *Ensemble convergence of ASJ-R for the Poisson benchmark (left) and the steady-state advection-diffusion problem (right) with bit flip corruption probabilities ranging from* $p = 0$ *to* $p = 0.04$*, with double floating point flips in any of the* 64 *bits. The convergence behavior is comparable, with increasing* p *for the steady-state advection-diffusion problem system resulting in longer relative times to solution and a larger likelihood that a given run does not converge.*

benchmark is perhaps partially explained by the shorter time to solution without corruption for the steady-state problem (which is used to obtain the relative values). That said, any corruption missed by the rejection criterion while the approximate shortest path length is not yet sufficiently large is likely to require more time to correct for the steady-state advection-diffusion problem than for the Poisson benchmark: The missed corruption must effectively be advected out of the domain instead of diffused with nearby uncorrupted values. Malevolent manipulation of stored data on agent $i = 9$ is also explored using the same recovery times ω_r and offset magnitudes δ as used in the Poisson benchmark. The time to failure ω_f here is shortened to 1.25 s to account for the shorter time to solution without corruption (≈ 2.0 s). Figure 19 shows that the ASJ-R method retains resilience to the malevolent corruption for the steady-state advection-diffusion problem, with the relative times to solution being longer than for the Poisson benchmark. As with the bit flip corruption, those longer relative times to solution are explained by the nature of relying on advection across the domain to correct missed corruption instead of leveraging diffusion with nearby uncorrupted values as well as the shorter time to solution without corruption for the steady-state advection-diffusion problem that is used to obtain the relative times.

5. Conclusions. We introduced a fault-tolerant ASJ variant that leverages ASJ convergence theory by Hook and Dingle [11] to provide resilience to data corruption. The resulting variant ASJ method (ASJ-R) rejects solution approximations from neighbor nodes if the distance between two successive approximations violates an analytic bound. Because the analytic bound requires the shortest path length, the ASJ-R method includes a shortest path length approximation. Following the work of Anzt, Dongarra, and Quintana-Ortí [2], we studied the resilience of ASJ and ASJ-R to corruption in communicated data due to bit flips in various parts of the IEEE 754 floating point representation. While we observed that both ASJ and ASJ-R reliably converge when the corruption is very small relative to the convergence tolerance, the ASJ-R method generally retains the ability to converge when bit flips occur in locations that cause larger-magnitude corruption. The convergence of the ASJ-R exhibits a stagnation period, a degraded convergence rate, or both depending on the properties of the corruption, resulting in times to solution around 1.5 to 6 times longer than without corruption. By individually studying particular locations for bit flips, we are able to explain both convergence behaviors as well as the lack of convergence

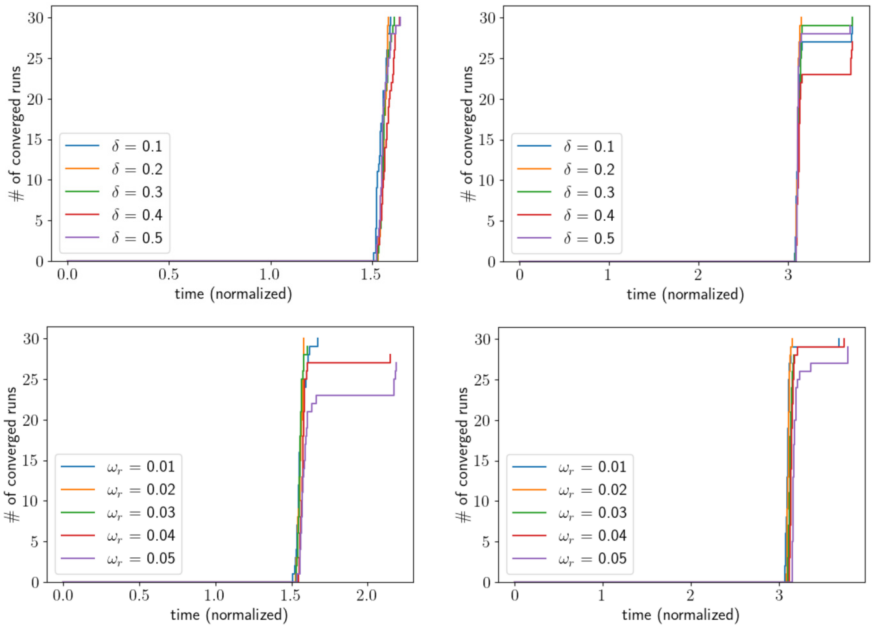

FIG. 19. *Ensemble convergence of ASJ-R in the presence of malevolent corruption for the Poisson benchmark (left column) and the steady-state advection-diffusion problem (right column) with recovery time $\omega_r = 0.02$ s and various offset magnitudes δ (top row) and various recovery times ω_r and offset magnitude $\delta = 0.2$ (bottom row). The convergence behavior is comparable for the two linear systems in that almost all of the runs converge except when the offset magnitude is larger or recovery time is longer.*

by whether the shortest path length increases at a sufficient rate to reject the errors that would otherwise delay or prevent convergence. Stagnation periods occur while the shortest path length increases to a value needing to start rejecting the corruption. A degraded convergence rate occurs when the corruption magnitude is small enough to avoid the rejection criterion (until the shortest path length is large enough) but large enough to slow but not stall the convergence. Nonconvergence occurs when the corruption that avoids the rejection criterion is enough for the solution approximations to deviate substantially from that predicted by convergence theory, resulting in the ASJ-R method rejecting the large updates that would otherwise drive the solution approximations back toward the exact solution.

We also studied the resilience of ASJ and ASJ-R to the corruption of stored data, where the stored values are perturbed by a uniformly distributed amount for periodic windows of time. Whereas ASJ failed to converge in all the scenarios tested, ASJ-R reliably restored convergence in a large majority of those scenarios. As with the bit flip corruption, we observed that the convergence of ASJ-R depends on whether the shortest path length approximation increased at a rate sufficient to prevent the stored data corruption from driving the solution approximations too far from that predicted by the convergence theory on which the rejection criterion is derived. Given the importance of the shortest path approximation and the ASJ-R rejection criterion bound that it appears in, we studied the sensitivity of the method to approximations in the values used for the two singular values in the bound. We found the results to be more sensitive to the maximum singular value of the iteration matrix and less sensitive to the minimum singular value of the linear system matrix, which is promising, as

the latter is typically more difficult to obtain. Skewing of either singular values in the direction that tightened the rejection criterion bound was found to more likely maintain or even improve convergence behavior than skewing that loosened the bound. We also verified that the ASJ-R performance extends to additional linear systems: one constructed using the MGG expander graph for a more dense sparsity pattern and one from a steady-state advection-diffusion problem.

While this work focused on solving a linear system with a Jacobi method in an HPC environment with an empirically determined convergence duration, the key observations should have applicability to other environments and solvers. Edge computing environments will very likely have greater communication latency than the HPC environment used here; however, the objective for the rejection criterion remains to reject the corruption that would cause the solution approximation to deviate too far for the rejection criterion to be useful. While the greater communication latency will result in the shortest path length increasing at a slower rate, the latency will also slow the rate at which the corruption causes the solution approximations to deviate. Thus, one might expect the relative times to solution observed in this work to have some relevance in edge environments. The particular convergence duration empirically determined in this work, however, will likely not have relevance in edge computing environments. Instead, one might empirically determine the appropriate convergence duration for a given environment in the same manner as it was determined here. Better yet, one of the stopping criteria mentioned in [16] or [20, section 3] could be leveraged, e.g., the leader election approach by Bahi et al. [3]. Finally, the particular rejection criterion derived here is indeed unique to the convergence theory for the Jacobi method. That sa,id, the approach of systematically evaluating a dynamic rejection bound by introducing corruption of various magnitudes to broadcasted data can be leveraged by new asynchronous solvers as they are developed.

REFERENCES

[1] IEEE, 754-2019—*IEEE Standard for Floating-point Arithmetic*, *(revision of IEEE* 754-2008*)*, IEEE, New York, 2019, pp. 1–84, https://doi.org/10.1109/IEEESTD.2019.8766229.

[2] H. Anzt, J. Dongarra, and E. S. Quintana-Ortí, *Tuning stationary iterative solvers for fault resilience*, in Proceedings of the Sixth Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems, ScalA '15, Association for Computing Machinery, New York, 2015, https://doi.org/10.1145/2832080.2832081.

[3] J. Bahi, S. Contassot-Vivier, R. Couturier, and F. Vernier, *A decentralized convergence detection algorithm for asynchronous parallel iterative algorithms*, IEEE Trans. Parallel Distrib. Syst., 16 (2005), pp. 4–13, https://doi.org/10.1109/TPDS.2005.2.

[4] R. Canal, C. Hernandez, R. Tornero, A. Cilardo, G. Massari, F. Reghenzani, W. Fornaciari, M. Zapater, D. Atienza, A. Oleksiak, W. Piatek, and J. Abella, *Predictive reliability and fault management in exascale systems: State of the art and perspectives*, ACM Comput. Surv., 53 (2020), https://doi.org/10.1145/3403956.

[5] D. Chazan and W. Miranker, *Chaotic relaxation*, Linear Algebra Appl., 2 (1969), pp. 199–222, https://doi.org/10.1016/0024-3795(69)90028-7.

[6] Z. Chen, *Online-ABFT: An online algorithm based fault tolerance scheme for soft error detection in iterative methods*, in Proceedings of the 18th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPoPP '13, Association for Computing Machinery, New York, 2013, pp. 167–176, https://doi.org/10.1145/2442516.2442533.

[7] L. Erlandson, Z. Atkins, A. Fox, C. J. Vogl, A. Międlar, and C. Ponce, *Resilient s-ACD for asynchronous collaborative solutions of systems of linear equations*, in Proceedings of the 18th Conference on Computer Science and Intelligence Systems, Annals of Computer Science and Information Systems 35, IEEE, New York, 2023, pp. 441–450, https://doi.org/10.15439/2023F8932.

[8] A. Frommer and D. B. Szyld, *On asynchronous iterations*, J. Comput. Appl. Math., 123 (2000), pp. 201–216, https://doi.org/10.1016/S0377-0427(00)00409-X.

[9] G. Guennebaud and B. Jacob, *Eigen v3*, http://eigen.tuxfamily.org, 2010.
[10] A. Hagberg, P. J. Swart, and D. A. Schult, *Exploring network structure, dynamics, and function using NetworkX*, https://www.osti.gov/biblio/960616, 2008.
[11] J. Hook and N. Dingle, *Performance analysis of asynchronous parallel Jacobi*, Numer. Algorithms, 77 (2018), pp. 831–866, https://doi.org/10.1007/s11075-017-0342-9.
[12] Y. Kalyani and R. Collier, *A systematic survey on the role of cloud, fog, and edge computing combination in smart agriculture*, Sensors, 21 (2021), https://doi.org/10.3390/s21175922, https://www.mdpi.com/1424-8220/21/17/5922.
[13] Q. Li, B. Kailkhura, R. Goldhahn, P. Ray, and P. K. Varshney, *Robust decentralized learning using ADMM with unreliable agents*, https://arxiv.org/abs/1710.05241, 2018.
[14] X. Liang, J. Chen, D. Tao, S. Li, P. Wu, H. Li, K. Ouyang, Y. Liu, F. Song, and Z. Chen, *Correcting soft errors online in fast Fourier transform*, in Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC '17, Association for Computing Machinery, New York, 2017, https://doi.org/10.1145/3126908.3126915.
[15] S. Liu, L. Liu, J. Tang, B. Yu, Y. Wang, and W. Shi, *Edge computing for autonomous driving: Opportunities and challenges*, Proc. IEEE, 107 (2019), pp. 1697–1716, https://doi.org/10.1109/JPROC.2019.2915983.
[16] F. Magoulés and G. Gbikpi-Benissan, *Distributed convergence detection based on global residual error under asynchronous iterations*, IEEE Trans. Parallel Distrib. Syst., 29, pp. 819–829, https://doi.org/10.1109/TPDS.2017.2780856.
[17] M. A. Mukwevho and T. Celik, *Toward a smart cloud: A review of fault-tolerance methods in cloud systems*, IEEE Trans. Serv. Comput., 14 (2021), pp. 589–605, https://doi.org/10.1109/TSC.2018.2816644.
[18] A. Olshevsky, I. C. Paschalidis, and A. Spiridonoff, *Fully asynchronous push-sum with growing intercommunication intervals*, in 2018 Annual American Control Conference (ACC), IEEE, New York, 2018, pp. 591–596, https://doi.org/10.23919/ACC.2018.8431414.
[19] C. Ponce, K. Harter, A. Fox, and C. J. Vogl, *Skywing*, https://doi.org/10.11578/dc.20221110.2, 2022.
[20] P. Spiteri, *Parallel asynchronous algorithms: A survey*, Adv. Eng. Softw., 149 (2020), 102896, https://doi.org/10.1016/j.advengsoft.2020.102896, https://www.sciencedirect.com/science/article/pii/S0965997819311378.
[21] G. Wang, G. B. Giannakis, J. Chen, and J. Sun, *Distribution system state estimation: An overview of recent developments*, Front. Inform. Technol. Electron. Eng., 20 (2019), pp. 4–17, https://doi.org/10.1631/FITEE.1800590.
[22] J. Wolfson-Pou and E. Chow, *Convergence models and surprising results for the asynchronous Jacobi method*, in 2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS), IEEE, New York, 2018, pp. 940–949.
[23] P. Wu, C. Ding, L. Chen, F. Gao, T. Davies, C. Karlsson, and Z. Chen, *Fault tolerant matrix-matrix multiplication: Correcting soft errors on-line*, in Proceedings of the Second Workshop on Scalable Algorithms for Large-Scale Systems, ScalA '11, Association for Computing Machinery, New York, 2011, pp. 25–28, https://doi.org/10.1145/2133173.2133185.