Erasure coding for fault oblivious linear system solvers

David F. Gleich, Ananth Grama, and Yao Zhu
Purdue University, Department of Computer Science

Abstract

Dealing with hardware and software faults is an important problem as parallel and distributed systems scale to millions of processing cores and wide area networks. Traditional methods for dealing with faults include checkpoint-restart, active replicas, and deterministic replay. Each of these techniques has associated resource overheads and constraints. In this paper, we propose an alternate approach to dealing with faults, based on input augmentation. This approach, which is an algorithmic analog of erasure coded storage, applies a minimally modified algorithm on the augmented input to produce an augmented output. The execution of such an algorithm proceeds completely oblivious to faults in the system. In the event of one or more faults, the real solution is recovered using a rapid reconstruction method from the augmented output. We demonstrate this approach on the problem of solving sparse linear systems using a conjugate gradient solver. We present input augmentation and output recovery techniques. Through detailed experiments, we show that our approach can be made oblivious to a large number of faults with low computational overhead. Specifically, we demonstrate cases where a single fault can be corrected with less than 10% overhead in time, and even in extreme cases (fault rates of 20%), our approach is able to compute a solution with reasonable overhead. These results represent a significant improvement over the state of the art.

1 Introduction

The next generation of parallel and distributed systems are projected to scale to millions of processing cores and beyond. Distributed systems already span wide geographic areas. In these regimes, hardware and software faults present major challenges for scalable execution of programs. In this paper, we consider the solution of an $n \times n$ nonsingular linear system

$$\mathbf{A}\mathbf{x} = \mathbf{b} \tag{1}$$

in an environment with faults.

Existing fault tolerant computations can be broadly classified as algorithmic or system-supported. Algorithmic fault tolerance mechanisms alter the algorithm to render it robust to faults. This requires deep algorithmic and analytical insight to quantify fault tolerance properties and associated overheads. It can also change the algorithmic cost and complexity significantly. System supported fault tolerance schemes include checkpoint-restart, active replicas, and deterministic replay. Checkpoint-restart schemes involve the overheads of consistent checkpointing and I/O, particularly for ultra-scale platforms, where I/O capacity and bandwidth are both at a premium relative to the compute capability. Furthermore, the asynchronous nature of many highly scalable algorithms makes it costly to identify consistent checkpoints and to perform associated rollbacks. Variants of these schemes include in-memory checkpointing, use of persistent storage (flash memory), and application-specified checkpoints.

Active replicas, commonly used in mission-critical real-time applications, execute multiple replicas of each task. Failures are detected and replicas are managed to support real-time constraints. While the runtime characteristics of these schemes can be controlled (e.g., through worst-case runtime estimation), resource overheads of such schemes are high, since tolerating a k-process failure among p processes requires $(k+1)p$ active processes. This cost is significant – as an example, tolerating 10 faults in an ensemble of 1000 cores (a 1% error rate) requires 11,000 processes! Yet other systems such as MapReduce use the concept of deterministic replay. In this model, computation proceeds in steps with checkpoints at the end of each step. Processes are monitored within the steps, and in the event of a failure, the computation associated with the failed process is replayed at an alternate node. While this model has been successfully applied to a large set of wide-area distributed applications, it has the drawbacks of staged execution and increased job makespan, particularly when the number of faults is large. Furthermore, checkpointing to persistent storage (typically a distributed file system) can add significant overhead, particularly, when steps are small.

Our contribution in this paper is the design of a new type of method for solving (1) that we call a fault oblivious algorithm. The essence of the idea is that we develop an algorithm that allows us to recover the correct solution to the problem even if some of the computational units involved in the solution process fail during the execution of the algorithm. More specifically, given system (1), we design an augmented system:

$$\tilde{\mathbf{A}}\tilde{\mathbf{x}} = \tilde{\mathbf{b}} \tag{2}$$

together with a solution strategy such that using the solution strategy to solve (2) in an environment with faults would have the following properties:

1. *Deterministic finite termination.* The solution process terminates in finite steps. When it terminates, it indicates one and only one of the two cases: (i) it fails to solve (2) to a specified precision; (ii) an approximate solution $\tilde{\mathbf{x}}$ to (2) within the specified precision has been found.

2. *Recoverability of the intended solution.* In case (ii) above, we are able to recover the intended solution $\mathbf{x}$ to (1) from $\tilde{\mathbf{x}}$ through an inexpensive computation.

As discussed above, a variety of existing fault tolerant techniques such as check-pointing and replication give us finite termination and trivially satisfy recoverability. Thus, our goal in designing the augmented system is that the computational complexity of a solution process with deterministic finite termination is bounded by the computational complexity of available fault-tolerant techniques. In other words – we want augmented systems that enable us to solve a given problem in an environment with faults more quickly than existing techniques.

Our design for the augmented system (2) is a linear coding of the input system (1). For this reason, we refer to (2) as the *encoded system*, and the input system (1) the *raw system*. There are three major components of our approach that we discuss in the next three sections.

- The fault model – Section 2. We need to define the types of the faults and their semantics that our coded linear solver can handle.

- The encoding scheme – Section 3. We add a set of rows and columns to the matrix to render the new linear system singular, but consistent. It remains this way for up to k component-wise faults in the solution vector $\tilde{\mathbf{x}}$.

- The solution process and recovery scheme – Section 4. Our solution process is to run a conjugate gradient algorithm. We show that when this algorithm terminates, it does so at a consistent solution of the encoded system. We then describe how to recover the true solution $\mathbf{x}$ in light of the encoding.

Because of the close relationship between the encoding scheme, solution process, and recovery scheme, we adopt a co-design approach for these three tasks.

For simplicity, in this manuscript, we restrict ourselves to the case where $\mathbf{A}$ is symmetric positive definite (SPD). We report experiments on the efficiency of our encoding in Section 6. Our main findings are that the encoded system takes minimal additional work to solve in the presence of a fault. In the presence of a substantial number of faults (20% of components failing), it takes 5 times the number of iterations of a linear solver. However, note that each iteration of this solver requires no additional overhead to achieve fault tolerance. Thus, this is a substantial savings compared with a checkpoint-restart system.

2 Fault model

We view the execution of the algorithm in two stages – the setup phase and the execution phase. The setup phase consists of the input augmentation step of the algorithm. During this phase, we assume that a small amount of reliable work can be done. In the presence of faults, this can be achieved using more expensive fault tolerance techniques such as replicated execution or deterministic replay. Since this step is a very small fraction of the overall computation (less than 1% for typical systems), the overhead is not significant. Please note that in current systems, the entire execution is performed in this reliable model. Thus, we may assume that we have this ability, although we wish to limit our use of these expensive schemes to minimize performance overhead.

The execution phase of the algorithm corresponds to the solve over the augmented system. In this phase, we assume an ensemble of message passing processes executing the solver. During this phase of execution, we assume fail-stop failures; i.e., in the event of a fault, a process halts. No further messages are received from this process by any of the other processes. Indeed there are other fault models as well, ranging from transient (soft) faults to Byzantine behavior. Soft faults manifest themselves in the form of erroneous data. This data, when incorporated into data at other processes, can lead to cascading error in programs.

Our proposed method can be extended to these other fault classes using existing fault detection schemes. In such schemes, messages are signed with a checksum, allowing us to detect on-the-wire errors. Asserts in the program, corresponding to predicates whose violation signifies an error can be used to detect soft errors in processes. If either of these soft efforts are detected, the receiving process simply drops messages, thus emulating a fail-stop error. When a soft error is detected at a process, the process is killed, once again resulting in a fail stop failure. Asserts work similarly when Byzantine failures are detected. Thus, a combination of tolerance to fail-stop failures with fault detection techniques allows us to deal with a broad set of faults.

Please note that there are other system software related issues associated with the proposed fault oblivious paradigm. Specifically, in many APIs a single process failure can cause the entire program to crash. In yet other scenarios, a crashed process can cause group communication operations (reductions, broadcasts, etc.) to block. These kinds of program behavior would not allow leveraging of our proposed schemes. We assume program behavior in which faulty processes simply drop out of the ensemble, while the rest continue. This paper does not address the design of such a fault oblivious API. Rather it algorithmically establishes the feasibility and superior performance of the erasure coded computation scheme for linear solvers.

3 Encoding scheme

Let $\mathbf{x}^*$ be the true solution of the linear system $\mathbf{Ax} = \mathbf{b}$. Let $k \leq n$ be the number of allowed faults during its execution. Let $\mathbf{E} \in \mathbb{R}^{n \times k}$ be an encoding matrix that we'll specify completely shortly.

We design the augmented matrix $\tilde{\mathbf{A}} \in \mathbb{R}^{(n+k)\times(n+k)}$ as follows:

$$\tilde{\mathbf{A}} = \begin{bmatrix} \mathbf{A} & \mathbf{AE} \\ \mathbf{E}^T\mathbf{A} & \mathbf{E}^T\mathbf{AE} \end{bmatrix}. \tag{3}$$

We choose the encoding of $\mathbf{x}^*$ to be an embedding into $\mathbb{R}^{n+k}$, i.e.,

$$\tilde{\mathbf{x}}^* = \begin{bmatrix} \mathbf{x}^* \\ 0 \end{bmatrix} \tag{4}$$

Accordingly, the encoding of $\mathbf{b}$ is given by

$$\tilde{\mathbf{b}} = \tilde{\mathbf{A}}\tilde{\mathbf{x}}^* = \begin{bmatrix} \mathbf{b} \\ \mathbf{E}^T\mathbf{b} \end{bmatrix} \tag{5}$$

3.1 Basic properties

We now establish a few properties of these systems in terms of their rank, a characterization of the solutions, and the semi-definiteness of $\tilde{\mathbf{A}}$.

PROPOSITION 1 *A null space basis of* $\tilde{\mathbf{A}}$ *is* $\begin{bmatrix} \mathbf{E} \\ -\mathbf{I}_k \end{bmatrix}$.

Proof. From the design of $\tilde{\mathbf{A}}$ in (3) and $\mathbf{A}$ being SPD, we have $\text{rank}(\tilde{\mathbf{A}}) = n$. Thus the null space has dimension k. Then, by inspection, $\tilde{\mathbf{A}} \left[\begin{smallmatrix} \mathbf{E} \\ -\mathbf{I}_k \end{smallmatrix}\right] = 0$ and $\left[\begin{smallmatrix} \mathbf{E} \\ -\mathbf{I}_k \end{smallmatrix}\right]$ has column rank k. ∎

As a corollary of the above proposition, we have the following proposition regarding the non-ambiguity of the solution encoding (4).

PROPOSITION 2 *Let* $\left[\begin{smallmatrix} \mathbf{y} \\ \mathbf{z} \end{smallmatrix}\right]$ *be any solution of* (2) *where* $\mathbf{y} \in \mathbb{R}^n$*. Once* $\mathbf{z} \in \mathbb{R}^k$ *is specified, then the components of* $\mathbf{y}$ *are uniquely determined. Moreover, if* $\mathbf{z} = 0$*, then* $\mathbf{y} = \mathbf{x}^*$.

Proof. Note that $\left[\begin{smallmatrix} \mathbf{x}^* \\ 0 \end{smallmatrix}\right]$ is a solution to (2). Thus, any solution to (2) can be written as:

$$\begin{bmatrix} \mathbf{y} \\ \mathbf{z} \end{bmatrix} = \begin{bmatrix} \mathbf{x}^* \\ 0 \end{bmatrix} + \begin{bmatrix} \mathbf{E} \\ -\mathbf{I}_k \end{bmatrix} \mathbf{a}$$

for a unique $\mathbf{a} \in \mathbb{R}^k$. Due to the non-zero structure, we have $\mathbf{a} = -\mathbf{z}$. Hence, $\mathbf{y}$ is uniquely determined as $\mathbf{x}^* - \mathbf{Ez}$. The final statement follows from $\mathbf{z} = 0$. ∎

We now prove that $\tilde{\mathbf{A}}$ as given in (3) is symmetric positive semidefinite (SPSD).

PROPOSITION 3 *If* $\mathbf{A}$ *is symmetric positive definite, then* $\tilde{\mathbf{A}}$ *as defined in* (3) *is SPSD.*

Proof. Let the Cholesky factorization of $\mathbf{A}$ be $\mathbf{A} = \mathbf{LL}^T$. Then, by inspection, we have

$$\tilde{\mathbf{A}} = \begin{bmatrix} \mathbf{L} \\ \mathbf{E}^T\mathbf{AL}^{-T} \end{bmatrix} \begin{bmatrix} \mathbf{L} \\ \mathbf{E}^T\mathbf{AL}^{-T} \end{bmatrix}^T.$$

∎

3.2 Solution degeneracies and faults

The matrix $\tilde{\mathbf{A}}$ has rank n, despite having $n+k$ rows and columns. We now show how a specific use of this degeneracy allows us to have fault tolerant solutions to (2). Let $\tilde{\mathbf{x}} \in \mathbb{R}^{(n+k)}$ be the encoded solution. For the same of clarity in presentation, we assume that faults can *only* occur within the components of $\tilde{\mathbf{x}}_{1:n}$, i.e., the redundant components $\tilde{\mathbf{x}}_{(n+1):(n+k)}$ introduced by the encoding cannot be faulty. Please note that this is not a limitation of our scheme. We let the set of faulty indices be $\mathcal{F} \subset [n]$. We constrain the cardinality $|\mathcal{F}| \leq k$. Let $\mathcal{C}$ be the set of non-faulty (or correct) components. Without loss of generality, we consider the system (2) in three components corresponding to the correct, faulty, and redundant components of the solution. This is equivalent to a permutation, after which we have that the solution is:

$$\tilde{\mathbf{x}} = \begin{bmatrix} \mathbf{c} \\ \mathbf{f} \\ \mathbf{r} \end{bmatrix} \begin{matrix} \text{correct} = \tilde{\mathbf{x}}_{\mathcal{C}} \\ \text{faulty} = \tilde{\mathbf{x}}_{\mathcal{F}} \\ \text{redundant.} \end{matrix} \tag{6}$$

The overall permuted system is:

$$\begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} & \mathbf{Z}_1 \\ \mathbf{A}_{12}^T & \mathbf{A}_{22} & \mathbf{Z}_2 \\ \mathbf{Z}_1^T & \mathbf{Z}_2^T & \mathbf{R} \end{bmatrix} \begin{bmatrix} \mathbf{c} \\ \mathbf{f} \\ \mathbf{r} \end{bmatrix} = \begin{bmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \\ \mathbf{E}^T\mathbf{b} \end{bmatrix} \quad \text{where} \begin{cases} \mathbf{Z}_1 & = \mathbf{A}_{11}\mathbf{E}_1 + \mathbf{A}_{12}\mathbf{E}_2 \\ \mathbf{Z}_2 & = \mathbf{A}_{12}^T\mathbf{E}_1 + \mathbf{A}_{22}\mathbf{E}_2 \\ \mathbf{R} & = \mathbf{E}^T\mathbf{A}\mathbf{E} \end{cases}$$

As our solver progresses, the components in $\mathbf{f}$ become "stuck" at some intermediate values as faults occur. We describe the semantics of these faults more formally in the next section (Section 4). The goal of this section is to show that we can recover solutions even when setting $\mathbf{f}$ to some arbitrary value.

Thus, for our erasure coded solvers, we need a condition on the matrix $\mathbf{E}$ such that:

1. There is always a solution to (2) for any $\mathbf{f}$ as long as $|\mathcal{F}| \leq k$ (Proposition 5).
2. Given any solution computed with faulty components ($|\mathcal{F}| \leq k$), we can extract a solution to (2) (Proposition 6).

The condition on the matrix $\mathbf{E}$ that is essential to these results is the Kruskal rank. Recall the definition:

DEFINITION 4 (Kruskal rank [7]) *The Kruskal rank, or k-rank, of a matrix is the largest number k such that every subset of k columns is linearly independent.*

Notice that the condition for a matrix to be of Kruskal rank k is much stronger than being rank k. The following proofs require that the Kruskal rank of $\mathbf{E}^T$ is k in order to handle up to k faults. Some intuition for this requirement is that we need the matrix $\mathbf{E}$ to encode redundancies to any possible faults with a number up to k. Recovering the solution will require us to invert a matrix for the components where the solution was faulty, and hence, we need all possible subsets of k rows of $\mathbf{E}$ to be invertible – giving us the Kruskal rank condition. We now present these two results formally:

PROPOSITION 5 *Let $\mathbf{E}^T \in \mathbb{R}^{k\times n}$ have Kruskal rank k and let $\mathcal{F}$ be an arbitrary subset of $[n]$ with $|\mathcal{F}| \leq k$. Then there exists a solution to (2) with $\tilde{\mathbf{x}}_{\mathcal{F}} = \mathbf{f}$ for any $\mathbf{f}$. When $|\mathcal{F}| = k$, such a solution is unique.*

Proof. Note that any solution of (2) has the form:

$$\begin{bmatrix} \mathbf{x}^* \\ 0 \end{bmatrix} + \begin{bmatrix} \mathbf{E} \\ -\mathbf{I} \end{bmatrix} \mathbf{a}$$

for some $\mathbf{a} \in \mathbb{R}^k$. Let us permute this solution as in (6):

$$\begin{bmatrix} \mathbf{c} \\ \mathbf{f} \\ \mathbf{r} \end{bmatrix} = \begin{bmatrix} \mathbf{x}_1^* \\ \mathbf{x}_2^* \\ 0 \end{bmatrix} + \begin{bmatrix} \mathbf{E}_1 \\ \mathbf{E}_2 \\ -\mathbf{I} \end{bmatrix} \mathbf{a}.$$

It suffices to show that there exists $\mathbf{a}$ such that $\mathbf{f} = \mathbf{x}_2^* + \mathbf{E}_2\mathbf{a}$. Because the rows of $\mathbf{E}_2$ correspond to the faulty components, this is a set of $|\mathcal{F}|$ columns from $\mathbf{E}^T$. These columns are linearly independent by the Kruskal rank condition. Thus, there exists a solution to this underdetermined linear system. If $|\mathcal{F}| = k$, then the system is square and non-singular, so the vector $\mathbf{a}$ is unique. ∎

According to our fault model, as faults occur during an iterative process, the components of $\mathbf{f}$ become stuck (i.e., they are not updated because of lost messages). Thus, the actual system that we solve is what we call a purified system consisting of only non-faulty components:

$$\begin{bmatrix} \mathbf{A}_{11} & \mathbf{Z}_1 \\ \mathbf{Z}_1^T & \mathbf{R} \end{bmatrix} \begin{bmatrix} \mathbf{c} \\ \mathbf{r} \end{bmatrix} = \begin{bmatrix} \mathbf{b}_1 \\ \mathbf{E}^T\mathbf{b} \end{bmatrix} - \begin{bmatrix} \mathbf{A}_{12} \\ \mathbf{Z}_2^T \end{bmatrix} \mathbf{f}. \tag{7}$$

By Proposition 5, if the encoding matrix $\mathbf{E}^T$ has Kruskal rank k, there exists a solution to the purified subsystem (7) from a solution to (2) with $\mathbf{x}_\mathcal{F} = \mathbf{f}$ fixed. We now ask the reverse question. Suppose $\left[\begin{smallmatrix} \mathbf{c} \\ \mathbf{r} \end{smallmatrix}\right]$ is any solution to (7), will $\left[\begin{smallmatrix} \mathbf{c} \\ \mathbf{f} \\ \mathbf{r} \end{smallmatrix}\right]$ be a solution to (2) (under the permutation)? The following proposition shows the answer is yes. The reason we need this proof is that there are many possible solutions to the purified subsystem. We need to establish that all solutions to (7) with $\mathbf{f}$ fixed will lend us a full solution to (2).

PROPOSITION 6 *Let $\mathbf{E}^T \in \mathbb{R}^{k \times n}$ have Kruskal rank k. Let $\left[\begin{smallmatrix} \mathbf{c} \\ \mathbf{r} \end{smallmatrix}\right]$ be any solution to the purified system* (7)*, where $|\mathcal{F}| \leq k$. Then $\left[\begin{smallmatrix} \mathbf{c} \\ \mathbf{f} \\ \mathbf{r} \end{smallmatrix}\right]$ is a solution to* (2)*.*

Proof. This proof is equivalent to checking whether the following equation is satisfied by the purified solution:

$$\begin{bmatrix} \mathbf{A}_{12}^T & \mathbf{Z}_2 \end{bmatrix} \begin{bmatrix} \mathbf{c} \\ \mathbf{r} \end{bmatrix} = \mathbf{b}_2 - \mathbf{A}_{22}\mathbf{f}. \tag{8}$$

We establish this fact algebraically from the solution of the purified system. First note that $\mathbf{E}_2^T$ is a k-by-$|\mathcal{F}|$ matrix with full column rank, and thus it has a left-inverse $(\mathbf{E}_2^T)^\dagger = (\mathbf{E}_2\mathbf{E}_2^T)^{-1}\mathbf{E}_2$. Now consider the two equations in the purified system:

$$\mathbf{A}_{11}\mathbf{c} + \mathbf{Z}_1\mathbf{r} = \mathbf{b}_1 - \mathbf{A}_{12}\mathbf{f} \tag{9}$$

$$\mathbf{Z}_1^T\mathbf{c} + \mathbf{R}\mathbf{r} = \mathbf{E}^T\mathbf{b} - \mathbf{Z}_2^T\mathbf{f}. \tag{10}$$

The result of (10) $- \mathbf{E}_1^T$(9) is:

$$\mathbf{E}_2^T\mathbf{A}_{12}^T\mathbf{c} + \mathbf{E}_2^T\mathbf{Z}_2\mathbf{r} = \mathbf{E}_2^T\mathbf{b}_2 - \mathbf{E}_2^T\mathbf{A}_{22}\mathbf{f}.$$

To complete the proof, we premultiply this equation by left inverse $(\mathbf{E}_2^T)^\dagger$. ∎

Algorithm 1

```
1: Let $\mathbf{x}_0$ be the initial guess and $\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0$, $\beta_0 = 0$.
2: for $t = 0, 1, \ldots$ until convergence do
3:     $\beta_t = (\mathbf{r}_t, \mathbf{r}_t)/(\mathbf{r}_{t-1}, \mathbf{r}_{t-1})$
4:     $\mathbf{p}_t = \mathbf{r}_t + \beta_t \mathbf{p}_{t-1}$
5:     $\mathbf{q}_t = \mathbf{A}\mathbf{p}_t$
6:     $\alpha_t = (\mathbf{r}_t, \mathbf{r}_t)/(\mathbf{q}_t, \mathbf{p}_t)$
7:     $\mathbf{x}_{t+1} = \mathbf{x}_t + \alpha_t \mathbf{p}_t$
8:     $\mathbf{r}_{t+1} = \mathbf{r}_t - \alpha_t \mathbf{q}_t$
9: end for
```

4 The solution process and recovery scheme

Proposition 3 establishes $\tilde{\mathbf{A}}$ being SPSD when $\mathbf{A}$ is SPD. Thus we can apply the conjugate gradient (CG) method to solve a singular but consistent linear system (2) [1]. For the $n+k$ eigenvalues of $\tilde{\mathbf{A}}$, we have $0 = \widetilde{\lambda}_1 = \cdots = \widetilde{\lambda}_k < \widetilde{\lambda}_{k+1} \leq \cdots \leq \widetilde{\lambda}_{n+k}$. Let the eigenvalues of $\mathbf{A}$ be $0 < \lambda_1 \leq \cdots \leq \lambda_n$. Because of the interlacing property, we have $\lambda_1 \leq \widetilde{\lambda}_{k+1}$ and $\lambda_n \leq \widetilde{\lambda}_{n+k}$. The effective condition number of $\tilde{\mathbf{A}}$ is defined as $\kappa_e(\tilde{\mathbf{A}}) = \frac{\widetilde{\lambda}_{n+k}}{\widetilde{\lambda}_{k+1}}$. Specifically, we use the following two-term recurrence form of CG [9].

We consider the setting when Algorithm 1 is executed in a distributed environment. For the encoded system (2), this means the encoded matrix $\tilde{\mathbf{A}}$ and the encoded vectors are distributed among multiple processes by rows. Let the index set associated with process i be $\mathcal{I}_i$, then $[n+k] = \bigcup_i \mathcal{I}_i$. According to our fault model, the operations of Algorithm 1 affected by faults in a distributed environment are the aggregation operations — inner products and the matrix-vector multiplication $\mathbf{A}\mathbf{p}_t$. Thus our erasure-coded CG can be defined by specifying the semantics of these two aggregation operations under faults. At the t-th iteration of erasure-coded CG, let the set of failed processes be $\mathcal{P}_t$. Then the set of faulty indices is $\mathcal{F}_t = \bigcup_{i \in \mathcal{P}_t} \mathcal{I}_i$. We assume that each viable process can detect the breakdown of its neighbor processes. Based on this assumption, we specify the semantics of the two aggregation operations as follows

- Inner products $(\mathbf{r}_t, \mathbf{r}_t)$ and $(\mathbf{q}_t, \mathbf{p}_t)$. The viable processes carry out the all-reduce operation by skipping the faulty components $\mathcal{F}_t$ in the vectors.

$$\begin{aligned} (\mathbf{r}_t, \mathbf{r}_t) &= \big((\mathbf{r}_t)_{[n+k]\backslash\mathcal{F}_t}, (\mathbf{r}_t)_{[n+k]\backslash\mathcal{F}_t}\big) \\ (\mathbf{q}_t, \mathbf{p}_t) &= \big((\mathbf{q}_t)_{[n+k]\backslash\mathcal{F}_t}, (\mathbf{p}_t)_{[n+k]\backslash\mathcal{F}_t}\big) \end{aligned} \tag{11}$$

  Furthermore, we require no reuse of aggregation operation results. This means that when computing α_t, $(\mathbf{r}_t, \mathbf{r}_t)$ is recomputed simultaneously with $(\mathbf{q}_t, \mathbf{p}_t)$. Similarly when computing β_t, $(\mathbf{r}_{t-1}, \mathbf{r}_{t-1})$ is recomputed simultaneously with $(\mathbf{r}_t, \mathbf{r}_t)$. For this purpose, we have to maintain both $\mathbf{r}_t$ and $\mathbf{r}_{t-1}$.

- Matrix-vector multiplication $\mathbf{q}_t = \mathbf{A}\mathbf{p}_t$. A viable process carries out its local aggregation operation for computing $\mathbf{A}_{\mathcal{I}_i,:}\mathbf{p}_t$ by skipping the faulty components $\mathcal{F}_t$ in $\mathbf{p}_t$.

$$\mathbf{A}_{\mathcal{I}_i,:}\mathbf{p}_t = \mathbf{A}_{\mathcal{I}_i,[n+k]\backslash\mathcal{F}_t}(\mathbf{p}_t)_{[n+k]\backslash\mathcal{F}_t}$$

Given the above semantics on aggregation operations, the erasure-coded CG on $\tilde{\mathbf{x}}$ can be effectively considered as an iterative solve process on the subsystem defined on $\tilde{\mathbf{x}}_{[n+k]\backslash\mathcal{F}_t}$, as given in (7). Note that the RHS of the purified subsystem (7) depends on $\mathbf{f}$, the snapshot value of $\tilde{\mathbf{x}}_{\mathcal{F}_t}$. For this reason, we require each viable process to cache the freshest snapshot values it have received from its neighbor processes. Another technical issue we need to consider when faults happen is the update to the search direction $\mathbf{p}_t$. In fault-free CG with exact arithmetic, we have $(\mathbf{r}_t, \mathbf{p}_{t-1}) = 0$. However, given the semantics of inner product as defined in (11), the orthogonality of $\mathbf{r}_t$ and $\mathbf{p}_{t-1}$ will generally not hold. We truncate the update $\mathbf{p}_t = \mathbf{r}_t + \beta_t \mathbf{p}_{t-1}$ to be

$$\mathbf{p}_t = \mathbf{r}_t$$

whenever new faults occur before the next update to $\mathbf{p}_t$.

Now we consider the recovery of the solution to the raw system (1). Suppose the erasure-coded CG converges on the encoded system (2) after T iterations. Let the encoding matrix $\mathbf{E}^T \in \mathbb{R}^{k \times n}$ have Kruskal rank k. Let $\mathcal{F} \in [n]$ be the set of all faulty indices upon convergence such that $|\mathcal{F}| \leq k$. In erasure-coded CG, the snapshot value $\mathbf{f}$ of the faulty components $\tilde{\mathbf{x}}_{\mathcal{F}}$ are cached on the viable processes. Because of the semantics of aggregation operations, the erasure-coded CG solves the purified subsystem defined in (7). Let the returned approximate solution be $\left[\begin{smallmatrix}\mathbf{c}\\ \mathbf{r}\end{smallmatrix}\right]$. According to Proposition 6, then

$$\tilde{\mathbf{x}} = \begin{bmatrix} \mathbf{c} \\ \mathbf{f} \\ \mathbf{r} \end{bmatrix}.$$

is a solution to the encoded system (2). Then, by Proposition 1, we can recover the intended solution to the raw system (1) through the equation

$$\begin{bmatrix} \mathbf{x}^* \\ 0 \end{bmatrix} = \tilde{\mathbf{x}} + \begin{bmatrix} \mathbf{E} \\ -\mathbf{I}_k \end{bmatrix} \mathbf{r}. \tag{12}$$

And by Proposition 2, the recovery equation (12) is non-ambiguous.

5 Related work

The paper [2] also considers the problem of designing a fault-tolerant linear solver. They model the faults by encapsulating all fault prone operations as an unreliable preconditioning operator and then build the fault-tolerant linear solver within the framework of flexible Krylov subspace methods, like flexible GMRES [11]. The flexible outer iteration is required to be completely reliable. In contrast, our erasure-coded approach does not require any part of the distributed environment to be reliable. Another salient difference between [2] and our approach is on the kinds of tolerable faults. In [2] the targeted faults are value errors resulted from numerical operations, while our erasure-coded approach targets more general fail-stop failures (subsystem breakdowns) in a distributed environment. It's not hard to understand that the approach in [2] can not recover from such faults because of the permanent data loss when subsystems breakdown in a distributed environment.

The fault-tolerant linear solver in [2] can be considered as a natural application of the theory of inexact and flexible Krylov subspace methods [5, 12, 13]. For inexact Krylov subspace methods, the magnitude of perturbations need to be controlled to maintain the convergence, although the perturbations can happen anywhere in the result of a numerical operation. In contrast, our erasure-coded approach does not presume any bound on the perturbations; but it does require the perturbations to be sparse in the affected components. However, the theory of inexact and flexible Krylov subspace methods can still provide the mathematical framework and tools to guide the design and analysis of the erasure-coded linear solvers.

Table 1: Test matrices

Matrix	n	nnz	Type
Ltridiag500	500	$1,498$	1D model problem
mhdb416	416	$2,312$	electromagnetics problem
nos3	960	$15,844$	structural problem

6 Experimental results

In this section, we report on experimental results with varying degrees of faults and associated overhead. The main purpose of these experiments is to demonstrate the feasibility of the idea of an erasure-coded linear solver. Through these experiments, we intend to identify critical research problems to be solved in order to realize the idea of an erasure-coded linear solver in a distributed setting.

6.1 Experiment design

In our experiments we uses three SPD test matrices, with their sizes (n), number of nonzeros (nnz), and types shown in Table 1. `Ltridiag500` is a 500×500 1D model matrix with the stencil $\begin{bmatrix} -1 & 2 & -1 \end{bmatrix}$. `mhdb416` and `nos3` are from University of Florida Sparse Matrix Collection [4]. We implemented a MATLAB code to simulate the erasure-code CG described in Section 4. In our current simulation, we inject fault components only at the end of a CG iteration. Furthermore, we assume all the faults happen at the same time. More extensive and thorough simulations that allow faults happen anywhere in a CG iteration and temporally interspersed are relegated to future work. For a given k number of tolerable faults, we design the $n \times k$ encoding matrix $\mathbf{E}$ as a random Gaussian matrix scaled by $\frac{1}{\sqrt{n}}$, i.e.,

$$\mathbf{E} = \frac{1}{\sqrt{n}}\bar{\mathbf{E}}, \quad \bar{\mathbf{E}}_{ij} \sim \mathcal{N}(0,1), \quad i = 1,\ldots,n,\ j = 1,\ldots,k$$

This matrix has Kruskal rank k with high probability.

6.2 Experimental Results

For each test matrix in Table 1, we run an erasure-coded CG simulation code for $k = 0, 1, 20\%n$ number of faults. The RHS $\mathbf{b}$ of the raw system (1) are synthesized using random solution vectors $\mathbf{x}$ in $(0,1)^n$. We set the number of maximum CG iterations to be $10n$, and monitor the convergence using the stopping criterion $\|\mathbf{r}_t\|_2 \le 10^{-10}$. For each value of k, the k fault components are randomly selected from $[n]$. These fault components are injected simultaneously at the end of one randomly selected CG iteration that is no larger than $0.25n$. We refer to this CG iteration as the fault point. In Figures 1 - 3, we plot the residual norm vs. CG iteration for the three test matrices; and from left to right for $k = 0, 1, 20\%n$. The fault point is marked by the red cross.

To evaluate the quality of the recovered solution $\mathbf{x}^*$, we compute the relative residual achieved by $\mathbf{x}^*$ on the raw system (1):

$$\frac{\|\mathbf{b} - \mathbf{A}\mathbf{x}^*\|_2}{\|\mathbf{b}\|_2}.$$

We summarize the number of iterations and relative residuals on the raw system for three test matrices in Tables 2–4. We observe that on `Ltridiag500` and `nos3`, when there is only 1 fault, the erasure-coded CG can recover a solution with almost the same quality as if there is no fault. And the

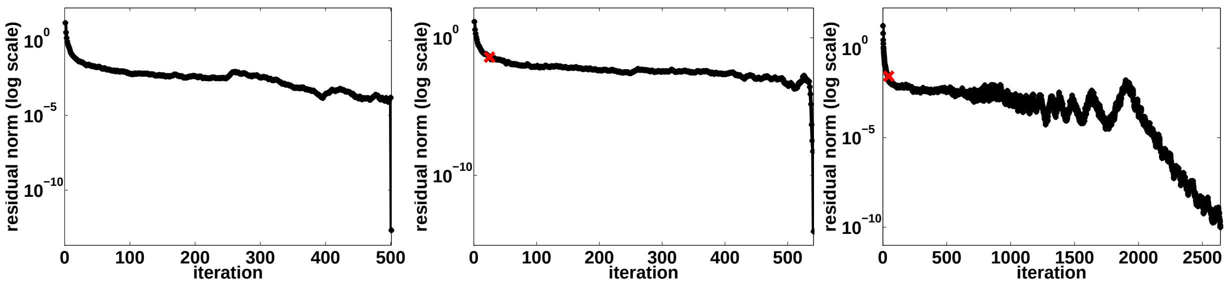

Figure 1: `Ltridiag500` residual norm. From left to right $k = 0, 1, 20\%n$. The fault point is marked by the red cross.

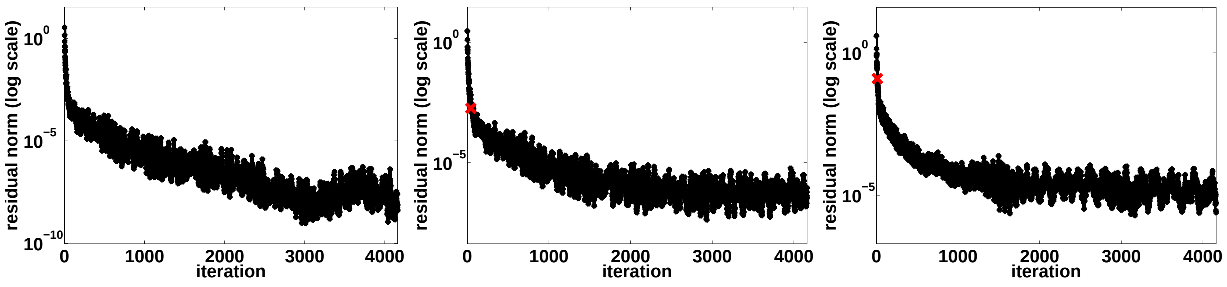

Figure 2: `mhdb416` residual norm. From left to right $k = 0, 1, 20\%n$. The fault point is marked by the red cross.

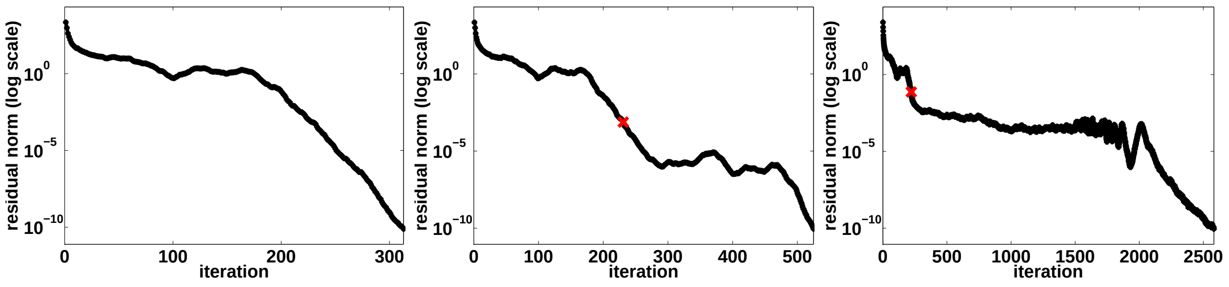

Figure 3: `nos3` residual norm. From left to right $k = 0, 1, 20\%n$. The fault point is marked by the red cross.

number of iterations are also comparable. When there are 20% fault components, the erasure-coded CG can still recover an approximate solution with good quality, although the number of iterations increase substantially. In contrast to `Ltridiag500` and `nos3`, on `mhdb416`, the erasure-coded CG reaches the maximum number of iterations (i.e., $10n$) for all the three values of k. Comparing the solution quality of $k = 0$ to those of $k = 1, 20\%n$ in Table 3 indicates more iterations are required for the latter two cases.

7 Future work

The current manuscript describes and documents our preliminary investigation along the idea of erasure-coded linear solvers. To further develop the theory and techniques for practically realizing this idea, we plan to explore the following directions.

- More thorough simulation and prototype tests. Our current experimental simulation assumes the faults can only occur at the end of a CG iteration, as well as being simultaneously. We

Table 2: Results on `Ltridiag500`

k	Iteration number	Relative residual on raw system
0	500	1.39×10^{-14}
1	540	3.76×10^{-15}
$20\%n$	$2,640$	3.72×10^{-11}

Table 3: Results on `mhdb416`

k	Iteration number	Relative residual on raw system
0	$4,160$	1.19×10^{-9}
1	$4,160$	1.47×10^{-5}
$20\%n$	$4,160$	2.09×10^{-6}

plan to design and test under a more realistic simulation, where the faults could happen anywhere during a CG iteration, as well as being temporally interspersed.

- Structure and sparse random projection design of the encoding matrix. Our current design of the encoding matrix $\mathbf{E}$ uses the random Gaussian matrix, which is very dense. To enhance both the encoding and recovery efficiency, we plan to adopt structure and sparse random projections [3, 6, 8], which allow fast matrix-vector multiplications.

- Improve the convergence of erasure-coded CG. Our current erasure-coded CG adopts the truncation strategy when new faults occur. Our preliminary experiments indicate that such a strategy may result in slow and wavy convergence when there is a large number of faults. We plan to adapt the flexible CG technique [10] to our erasure-coded CG solver in order to improve its convergence speed.

References

[1] S. F. Ashby, T. A. Manteuffel, and P. E. Saylor. A taxonomy for conjugate gradient methods. *SIAM J. Numer. Anal.*, 27(6):1542–1568, 1990.

[2] P. G. Bridges, K. B. Ferreira, M. A. Heroux, and M. Hoemmen. Fault-tolerant linear solvers via selective reliability. *CoRR*, abs/1206.1390, 2012.

[3] K. L. Clarkson, P. Drineas, M. Magdon-Ismail, M. W. Mahoney, X. Meng, and D. P. Woodruff. The fast Cauchy transform and faster robust linear regression. In Sanjeev Khanna, editor, *SODA*, pages 466–477. SIAM, 2013.

[4] T. A. Davis and Y. Hu. The university of florida sparse matrix collection. *ACM Trans. Math. Softw.*, 38(1):1, 2011.

[5] J. Eshof and G. Sleijpen. Inexact Krylov subspace methods for linear systems. *SIAM J. Matrix Anal. Appl.*, 26:125–153, 2004.

Table 4: Results on `nos3`

k	Iteration number	Relative residual on raw system
0	312	3.51×10^{-14}
1	524	4.09×10^{-14}
$20\%n$	$2,581$	1.91×10^{-13}

[6] S. Foucart and H. Rauhut. *A Mathematical Introduction to Compressive Sensing*. Birkhauser Basel, 2013.

[7] Joseph B. Kruskal. Three-way arrays: rank and uniqueness of trilinear decompositions, with application to arithmetic complexity and statistics. *Linear Algebra and its Applications*, 18(2):95–138, December 1977.

[8] X. Meng and M. W. Mahoney. Low-distortion subspace embeddings in input-sparsity time and applications to robust linear regression. In *Proceedings of the Forty-fifth Annual ACM Symposium on Theory of Computing*, STOC '13, pages 91–100. ACM, 2013.

[9] Gerard Meurant. *The Lanczos and Conjugate Gradient Algorithms: From Theory to Finite Precision Computations (Software, Environments, and Tools)*. SIAM, 2006.

[10] Y. Notay. Flexible conjugate gradients. *SIAM J. Sci. Comput.*, 22(4):1444–1460, 2000.

[11] Y. Saad. A flexible inner-outer preconditioned GMRES altorithm. *SIAM J. Sci. Comput.*, 14(2), 1993.

[12] V. Simoncini and D. B. Szyld. Flexible inner-outer Krylov subspace methods. *SIAM J. Numer. Anal.*, 40(6):2219–2239, 2003.

[13] V. Simoncini and D. B. Szyld. Theory of inexact Krylov subspace methods and applications to scientific computing. *SIAM J. Sci. Comput.*, 25(2):454–477, 2003.