2017 IEEE 37th International Conference on Distributed Computing Systems

Distributed Fault Tolerant Linear System Solvers based on Erasure Coding

Xuejiao Kang*, David F. Gleich†, Ahmed Sameh‡ and Ananth Grama§
Department of Computer Science, Purdue University–West Lafayette,USA
{*kang138,†dgleich}@purdue.edu,{‡sameh,§ayg}@cs.purdue.edu

***Abstract*—We present efficient coding schemes and distributed implementations of erasure coded linear system solvers. Erasure coded computations belong to the class of algorithmic fault tolerance schemes. They are based on augmenting an input dataset, executing the algorithm on the augmented dataset, and in the event of a fault, recovering the solution from the corresponding augmented solution. This process can be viewed as the computational analog of erasure coded storage schemes. The proposed technique has a number of important benefits: (i) as the hardware platform scales in size and number of faults, our scheme yields increasing improvement in resource utilization, compared to traditional schemes; (ii) the proposed scheme is easy to code – the core algorithms remain the same; and (iii) the general scheme is flexible – accommodating a range of computation and communication tradeoffs. We present new coding schemes for augmenting the input matrix that satisfy the recovery equations of erasure coding with high probability in the event of random failures. These coding schemes also minimize fill (non-zero elements introduced by the coding block), while being amenable to efficient partitioning across processing nodes. We demonstrate experimentally that our scheme adds minimal overhead for fault tolerance, yields excellent parallel efficiency and scalability, and is robust to different fault arrival models.**

***Keywords*-Fault Tolerance, Erasure Coding, Linear System Solvers, Kruskal rank**

I. Introduction and Motivation

As parallel and distributed platforms scale, fault tolerance becomes an increasingly important problem. Combined with potential constraints on storage capacity and throughput, these problems are rendered particularly challenging, since many traditional solutions such as checkpoint/restart do not straightforwardly address them. In recent efforts, we have introduced the concept of fault-oblivious parallel execution, based on erasure coded computations [1]. Fault oblivious executions suitably augment the input to a parallel program and execute the program on this augmented input in a potentially faulty environment. For a class of faults (fail-stop), the program executes oblivious of the faults (i.e., stopped processes are not restarted), and generates an augmented output. The true output of the program is generated from an inexpensive operation on the augmented output from the faulty execution. In this paper, we present novel coding schemes and a complete fault tolerant distributed sparse linear system solver, and show a number of highly desirable properties of our solver.

In comparison to other algorithmic techniques for fault tolerance, our scheme separates fault tolerance from the algorithm, and enables simple distributed algorithms. We address various issues that arise in coding and executing coded computations in distributed environments, such as the sparsity of the erasure code and their parallel solution using a CG solver, and robustness to a variety of error arrival characteristics. We support claims of superior performance of our solver through a comprehensive experimental study. We specifically demonstrate the following in the context of our distributed erasure coded conjugate gradient solver: (i) the overhead of input augmentation is low – orders of magnitude smaller than corresponding overhead for replicated execution; (ii) the conditioning of the augmented system is comparable to the original system as evidenced by its convergence rates; (iii) the overhead in iterations due to errors is low, as evidenced in comparison of base solver and fault tolerant solver with errors; (iv) the augmented input solver yields excellent parallel performance in the presence of errors; and (v) for three different fault arrival models – uniform, instantaneous, and random, our fault tolerant solver yields excellent performance. All of these results are validated on small to moderate sized systems on up to 32 cores executing an MPI-based solver.

II. Error Correcting Codes, Erasure Coded Storage, and Erasure Coded Computations

We initiate our discussion with a brief review of three related, yet critically distinct concepts – error correcting codes, erasure coded storage, and our proposed concept of erasure coded computations.

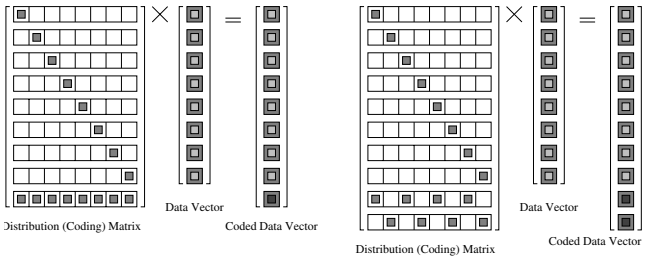


Fig. 1: Basic principles of erasure coding: the data vector is multiplied with a distribution matrix to compute an augmented vector. We can recover from a single element erasure by solving the linear system corresponding to the remaining rows of the distribution vector.

a) Error correcting codes (ECCs) and erasure coded storage: Consider a code that augments k items of data with m

1063-6927/17 $31.00 © 2017 IEEE
DOI 10.1109/ICDCS.2017.261

redundant items, for a total of $n = k+m$ stored items. If such a code is capable of correcting m errors, then the code is space-optimal and is referred to as Maximum Distance Separable (MDS). Figure 1 presents two examples of an MDS systematic code. The code is represented as a distribution (or coding) matrix. This matrix has $k+m$ rows and k columns, with the property that any subset of k rows is linearly independent. The distribution matrix is multiplied by the data vector (of dimension k) to yield a $n = k+m$ item coded vector. If m of the elements of the coded vector are erased, one can recover the data vector by solving the linear system corresponding to the remaining k rows of the distribution matrix.

b) Erasure coded computation: While similar in spirit, our proposed concept of erasure coded computations has several significant differences. First, unlike ECC and erasure coded storage schemes, erasure coded computations aim to *code the results of a computation* in a fault tolerant manner, *as opposed to data*. Second, data structures are not linear bit strings(or blocks), rather, they are sparse matrices. Third, recovery is numerical in nature; i.e., we do not need to perform our computations over a finite field; rather, we can use real arithmetic. This greatly relieves the algorithmic burden of coding/ decoding. Finally, erasure coded computations must deal with failure modes other than erasures. All of these issues pose significant challenges, which form the focus of our recent work [1] and the present paper.

c) Erasure Coded Computations: We illustrate the idea for erasure coded computations in a simple form for a sparse matrix-vector product. We assume that this matrix is partitioned row-wise across processing nodes. In Figure 2, we show an example of such an matrix, partitioned across eight nodes. Consider the first simple case of coding the mat-vec using a parity distribution matrix (Figure 1). We multiply the distribution matrix with the given sparse matrix (Figure 2(b)). This results in an augmented matrix – the first k rows of the matrix remain unchanged because of the identity block of rows in the distribution matrix. The last (augmented) row is the negative sum of all other rows. In a graph view, this corresponds to the addition of a new node (node 9) that is connected to all other nodes in the original graph in a negative sense. The operation is now performed using nine processing nodes (as opposed to the original case of eight nodes). If one of the processes has a fail-stop failure, the corresponding vector element is lost. However, because of the structure on the row-sums, the missing element can be inferred because the sum of the output vector must be zero.

A. Erasure coded linear system solver

We now summarize the work in [1] that describes how to extend these ideas to solving a linear system of equations. This will form the basis for the distributed solver proposed in this paper. Let $\mathbf{x}^*$ be the true solution of the original linear system

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

We set $k \leq n$ to be the allowed number of faults. The encoding matrix $\boldsymbol{E}$ is an n-by-k matrix that should have the property that any subset of k-rows is linearly independent. This condition is equivalent to stating that $\boldsymbol{E}^T$ should have Kruskal rank k.

Given matrices $\boldsymbol{A}$ and $\boldsymbol{E}$, the augmented or encoded system, the solution to the augmented system, and the augmented right hand side, are given by:

$$\tilde{\boldsymbol{A}} = \begin{bmatrix} \boldsymbol{A} & \boldsymbol{AE} \\ \boldsymbol{E}^T\boldsymbol{A} & \boldsymbol{E}^T\boldsymbol{AE} \end{bmatrix} \qquad \tilde{\mathbf{x}}^* = \begin{bmatrix} \mathbf{x}^* \\ 0 \end{bmatrix} \qquad \tilde{\mathbf{b}} = \tilde{\boldsymbol{A}}\tilde{\mathbf{x}}^* = \begin{bmatrix} \mathbf{b} \\ \boldsymbol{E}^T\mathbf{b} \end{bmatrix}$$

With these augmented structures, we solve:

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

Computing the new blocks of this system (such as $\boldsymbol{AE}$) must be done reliably through some type of existing fault tolerance scheme, although this is a small amount of work. When $\boldsymbol{E}^T$ has Kruskal rank k, Gleich et al. [1] show a number of properties that guarantee that solution recovery is possible. In particular, let $\left[\begin{smallmatrix}\mathbf{y}\\ \mathbf{z}\end{smallmatrix}\right]$ be any solution of the augmented system, where $\mathbf{y} \in \mathbb{R}^n$. To recover $\mathbf{x}^*$ in the presence of errors, we need to compute $\mathbf{y} + \boldsymbol{E}\mathbf{z}$ only [1]. This gives us a straightforward recovery algorithm.

B. Benefits and challenges in erasure coded computations

We note several important points relating to this general scheme of fault oblivious computations: (i) tolerating a single fault using traditional replication requires twice as much compute power; tolerating a single fault using deterministic replay increases the makespan of the job by 100%; (ii) our proposed scheme achieves fault tolerance to a single fault by increasing the computation only fractionally, depending on the code used; (iii) the benefits of our scheme are significantly amplified as higher levels of fault tolerance are desired; (iv) controlling fill in augmentation rows and associated communication overheads can be achieved using suitable codes; (v) load balancing considerations can be achieved by distributing the augmentation rows across processes.

a) Deriving Erasure Coded Computation Schemes: The example in Figure 2 illustrated two codes – the first a parity code and the second a block parity code. In erasure coded storage, Vandermonde and Cauchy matrices are often used for generating codes. While space efficient, these codes induce significant communication overheads in our setting. For instance, a Vandermonde matrix induces an augmentation block that is dense. The block parity code illustrated in Figure 2 induces augmentation blocks with $O(n)$ non-zeros (when using n processing nodes), assuming an $O(1)$ non-zeros in each row of the input matrix. While this is more desirable than the $O(r)$ communication and computation of the Vandermonde block (here r is the number of rows in the input matrix), this poses constraints on scaling. We deal with this problem using distribution matrices similar to structured low-density parity codes (LDPC). These codes are described in Section III.

b) Fault models and Erasure Coded Computations: We have, thus far, considered only fail-stop failures; i.e., in the event of a fault, a process halts. Indeed there are other fault models as well, ranging from transient (soft) fault to Byzantine behavior. Soft/transient faults manifest themselves in the form

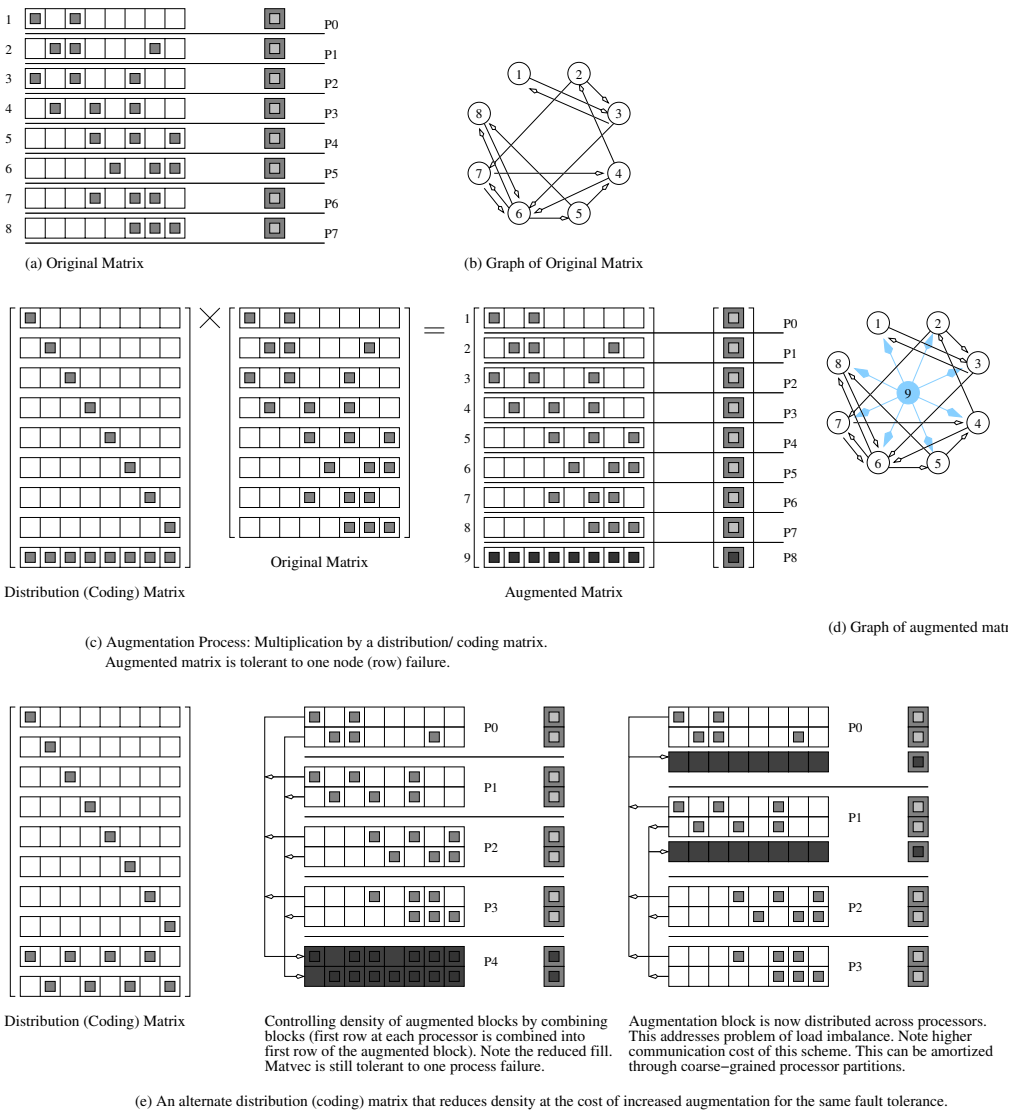


Fig. 2: Illustration of the concept of erasure coded computations: (a) a given sparse matrix and its (sparse) graph representation; (b) the augmentation process multiplies the given sparse matrix with a distribution matrix (also called the coding matrix). This results in a new matrix with one extra row, which is the sum of all other rows; (d) the graph view of the augmented matrix shows a new node added to the graph; (e) using an alternate distribution matrix allows us to control the fill in the augmentation rows. The augmented block now consists of two (sparser) rows. This computation is still resilient to one process failure. The augmentation block can itself be distributed among processors for load balance.

of erroneous data, while Byzantine failures can result in state changes at other processes. Our proposed method can be extended to these other fault classes using an application provided local fault detection scheme. These schemes are presented in the form of asserts (predicates whose violation signifies an error). When such a fault is locally detected, the process is killed – thus emulating a fail stop failure. Asserts work similarly when Byzantine failures are detected. Thus, a combination of tolerance to fail-stop failures, combined with user-specified predicates for local fault detection allow us to deal with broader class of faults.

c) Systems Support for Erasure Computations: There are important issues relating to system support and programming models that are associated with the proposed fault oblivious paradigm. Specifically, a single process failure often causes the entire program to crash. In yet other scenarios, a crashed process can cause group communication operations (reductions, broadcasts, etc.) to block. This program behavior would not allow leveraging of our proposed scheme. Specifically, we assume program behavior in which faulty processes simply drop out of the ensemble, while the rest continue. In this paper, we do not undertake the task of developing such system software infrastructure. Rather, to demonstrate the

feasibility and performance of our solver, we code it entirely using asynchronous non-blocking communication operations. While this is not a direct comparison with their synchronous counterparts, it allows us to demonstrate performance and fault tolerance characteristics of our solver.

III. Coding matrix for parallel implementation

Having established the algebraic basis for our method (Section II-A), we now focus on the problem of developing a suitable coding matrix $\boldsymbol{E}$ designed for working with distributed sparse matrices. This poses two challenges: first, the matrix $\boldsymbol{E}$ must satisfy certain algebraic properties of Kruskal rank in order to utilize the existing theory. Second, it must simultaneously minimize fill in the augmented matrix $\tilde{\boldsymbol{A}}$ for performance, both in terms of operation counts and minimizing communication. In order to achieve these dual objectives, we weaken the requirements on Kruskal rank k.

In the proofs from Gleich et al. [1], having Kruskal rank k is necessary to guarantee that *any possible* subset of components can fail and we can always recover true solution. This is obviously the ideal scenario. However, it is a worst-case analysis. In this paper, we relax the requirement as follows:

Definition A n-by-k matrix $\boldsymbol{E}$ satisfies the recovery-at-random property if a random subset of k rows (selected uniformly with replacement) is rank k with probability $\geq 1 - 1/n^c, c > 1$.

We construct a matrix $\boldsymbol{E}$ by setting p successive elements in each row to non-zero elements. These p non-zero elements are selected in a staggered manner, i.e., the first p elements in row 1, one zero followed by p non-zero elements in row 2, two zeros, followed by p non-zero elements in row 3, and so on. More generally, for the i^{th} row, the $j = (i - 1 + s) \mod k$ elements for s ranging from 1 to p are set to random reals in the range $(0, 1)$. This choice of matrix $\boldsymbol{E}$ satisfies the recovery-at-random property.

Proposition 3.1: Let $\boldsymbol{E}'$ be a submatrix of $\boldsymbol{E}$ formed by selecting any p rows of matrix $\boldsymbol{E}$. The matrix $\boldsymbol{E}'^T$ has rank p (alternately, any p rows of matrix E are linearly independent).

Proof. We consider two cases: (i) all rows of matrix $\boldsymbol{E}'$ have distinct non-zero structure; in this case, all rows of $\boldsymbol{E}'$ are trivially linearly independent; and (ii) all rows of matrix $\boldsymbol{E}'$ have the same non-zero structure; i.e., the non-zeros form a $p \times p$ non-zero block in matrix $\boldsymbol{E}'$. In this case, we know that the $p \times p$ block of randomly generated entries is non-singular with probability 1. Therefore, all rows of matrix $\boldsymbol{E}'$ are linearly independent in this case as well. Other cases where some subset of rows of $\boldsymbol{E}'$ have identical non-zero structure, can be similarly argued to be linearly independent. ■

Proposition 3.1 shows that any submatrix of p rows of matrix $\boldsymbol{E}$ has full rank. We need that any submatrix of k rows of matrix $\boldsymbol{E}$ must have rank k. Clearly, there exist degenerate cases where this is not true – specifically, if we select k rows, each with the same non-zero structure, we end up with a submatrix of rank p, which is less than k. It is important to note that this row-selection process is determined by the location of failures in the system. We show in Theorem 3.2 that such degenerate selections are highly unlikely.

Theorem 3.2: The probability that a random set of k rows of matrix E is linearly dependent is less than $(\frac{e}{p+1})^{p+1}$.

Proof. A necessary and sufficient condition for k rows to be linearly dependent is that some selection of $p + 1$ of these k rows have the same non-zero structure.

Note that there are k distinct non-zero structures in the n rows of matrix E. Furthermore, since rows are uniformly assigned one of these k distinct row structures, the probability that a row has a specific row structure is $1/k$ and the probability that $p + 1$ rows have the same row structure is $(\frac{1}{k})^{p+1}$. Since there are $\binom{k}{p+1}$ ways to select $p + 1$ rows out of the selected block of k rows, the probability that a selected block of k rows is linearly dependent is given by:

$$\binom{k}{p+1} \cdot \left(\frac{1}{k}\right)^{p+1} \leq \left(\frac{e}{p+1}\right)^{p+1}. \qquad \blacksquare$$

As p increases, it is easy to see that this probability rapidly approaches 0. Stated otherwise, matrix $\boldsymbol{E}$ satisfies recovery-at-random for p chosen as a suitable function of n.

We now focus on the problem of selecting the smallest value of p in our coding matrix $\boldsymbol{E}$ to guarantee that we are unlikely to have too many rows with the same non-zero pattern. we investigate the expected maximum number of rows that share the same non-zero structure. If this expected number exceeds p, our recovery-at-random condition fails. Therefore, we must select p greater than this expected maximum number.

Theorem 3.3: The expected maximum number of rows from among k randomly selected rows of matrix $\boldsymbol{E}$ that have same nonzero structure is $O(\frac{\ln k}{\ln \ln k})$.

Proof. Define a random variable M to be the maximum number of rows that have the same non-zero structure when we select k rows uniformly at random from matrix E and $Pr(M = t)$ be the probability that the random variable takes value t. For a given non-zero structure, we fashion each row selection as a Bernoulli trial, with success corresponding to the selection of the specified row structure, and failure otherwise. Since there are k distinct row structures, all with identical probability of selection, the probability of exactly t successes is given by the Bernoulli distribution as $\binom{k}{t}(\frac{1}{k})^t(1 - \frac{1}{k})^{k-t}$. Since there are k different row structures from which we select the one common structure, the probability:

$$Pr(M = t) = \binom{k}{1}\binom{k}{t}\left(\frac{1}{k}\right)^t\left(1 - \frac{1}{k}\right)^{k-t} \leq k\left(\frac{e}{t}\right)^t$$

For $t = c\frac{\ln k}{\ln \ln k}$, where c is some constant, we can show $Pr(M = t) \leq \frac{1}{k^2}$. The expected maximum number of rows sharing a row structure $E(M)$ from k randomly selected rows of matrix E is given by:

$$E(M) = \sum_{t=1}^{k} t \cdot Pr(M = t)$$

$$= \sum_{t=1}^{c\frac{\ln k}{\ln\ln k}} t \cdot Pr(M=t) + \sum_{t=\frac{c\ln k}{\ln\ln k}}^{k} t \cdot Pr(M=t)$$

$$\leq \sum_{t=1}^{\frac{c\ln k}{\ln\ln k}} \frac{c\ln k}{\ln\ln k} \cdot Pr(M=t) + \sum_{t=\frac{c\ln k}{\ln\ln k}}^{k} k \cdot Pr(M=t)$$

$$= \frac{c\ln k}{\ln\ln k} \sum_{t=1}^{\frac{c\ln k}{\ln\ln k}} Pr(M=t) + k \sum_{t=\frac{c\ln k}{\ln\ln k}}^{k} Pr(M=t)$$

$$\leq \frac{c\ln k}{\ln\ln k} + k \cdot \frac{1}{k^{c/2-1}} = O(\frac{\ln k}{\ln\ln k}). \quad \blacksquare$$

Theorems 3.2 and 3.3 provide bounds on values of p for a given value of k. In our experiments, we select $k = 4, 8, 16, 32$, and the number of nonzeros per row $p = \min(k, 5)$. For these selections, our choice of p exceeds the expected number of rows that share the same row structure.

IV. Parallel implementation of erasure coded CG

Gleich et al. show that $\tilde{\boldsymbol{A}}$ is symmetric-positive-semi-definite when $\boldsymbol{A}$ is symmetric positive definite and we can apply CG in this setting when $\tilde{\boldsymbol{A}}$ and $\tilde{\mathbf{b}}$ are consistent [1]. This requires the following two-term recurrence form of CG [2].

Algorithm 1 Fault Oblivious CG with a two-term recurrence. When we notice a fault, we set $\beta_t = 0$ at that iteration.

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

The augmented matrix $\tilde{\boldsymbol{A}}$ and the augmented vectors are distributed among multiple processes by rows. The operations of Algorithm 1 affected by faults in a distributed environment are the aggregation operations – inner products and the matrix-vector multiplication. Let the index set associated with process i be $\mathcal{I}_i$, then $[n+k] = \bigcup_i \mathcal{I}_i$ and the set of faulty indices is $\mathcal{F}_t = \bigcup_{i \in \mathcal{P}_t} \mathcal{I}_i$.

- 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.

$$(\mathbf{r}_t, \mathbf{r}_t) = \left((\mathbf{r}_t)_{[n+k]\setminus\mathcal{F}_t}, (\mathbf{r}_t)_{[n+k]\setminus\mathcal{F}_t}\right)$$
$$(\mathbf{q}_t, \mathbf{p}_t) = \left((\mathbf{q}_t)_{[n+k]\setminus\mathcal{F}_t}, (\mathbf{p}_t)_{[n+k]\setminus\mathcal{F}_t}\right)$$

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

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

Another technical issue we need to consider when faults happen is the update to the search direction $\mathbf{p}_t$. Here, when we observe a fault, we truncate the update $\mathbf{p}_t = \mathbf{r}_t + \beta_t \mathbf{p}_{t-1}$ to be

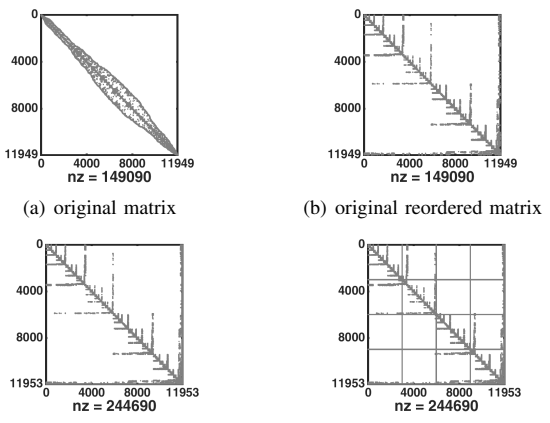
$$\mathbf{p}_t = \mathbf{r}_t.$$

This corresponds to a reset of the Krylov process.

We now consider the recovery of the solution to the raw system (1). Suppose the erasure-coded CG converges on the augmented system (2) after T iterations. Let the returned approximate solution be $\left[\begin{smallmatrix}\mathbf{c}\\ \mathbf{r}\end{smallmatrix}\right]$. We can recover the intended solution to the raw system (1) through the equation

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

a) Partitioning Considerations.: Even with the sparse matrix $\boldsymbol{E}$ described in Section III, the augmentation blocks in $\tilde{\boldsymbol{A}}$ potentially introduce dense blocks if not suitably computed. For this reason, we use a two-step process. In the first step, we order the input matrix (Figure 3(a)) using a traditional matrix/ graph partitioning technique such as Metis(Figure 3(b)). We then use this ordered matrix to compute $\tilde{\boldsymbol{A}}$ (Figure 3(c)). The resulting matrix is then reordered once again to partition $\tilde{\boldsymbol{A}}$ across various nodes in a parallel/ distributed platform (Figure 3(d)). The first step minimizes non-zeros in $\tilde{\boldsymbol{A}}$, and the second partitioning step minimizes communication for the solver applied to $\tilde{\boldsymbol{A}}$.



(a) original matrix (b) original reordered matrix

(c) augmented matrix (d) augment reordered matrix

Fig. 3: Illustration of the process of computing and repartitioning augmented matrix $\tilde{\boldsymbol{A}}$.

V. Experimental Results

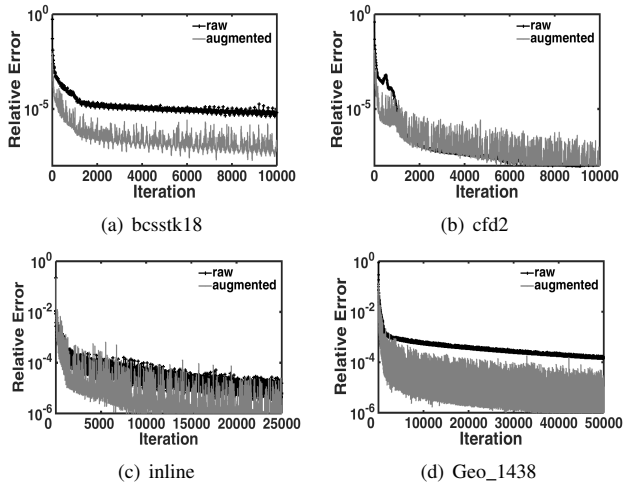
We report comprehensive results from our MPI implementation tested on up to 32 processing cores on four selected matrices.

a) Input Matrices, Augmented Systems, and Execution Parameters: We select four matrices from the University of Florida Matrix Collection – bcsstk18 with $12K$ rows and $149K$ nonzeros, cfd2 with $123K$ rows and $3085K$ nonzeros, inline with $504K$ rows and $36816K$, and geo_1438 with $1438K$ rows and $60236K$ nonzeros.

These matrices are selected based on their sizes, and because they are SPD, i.e., both CG on the raw and augmented system converge on these matrices. All matrices are first reordered using Metis. To construct the augmented system, we first build a suitable encoding matrix E, and use this matrix to generate the coded system. The maximum number of CG iterations for the four matrices are set to 10000, 10000, 25000, and 50000, respectively. The relative error tolerance is set to 10^{-10} for all matrices. Our tests show that CG runs to the maximum number of iterations for the last three matrices.

We test a number of fault models. In our initial set of experiments we use a uniform fault arrival model – faults arrive starting from iteration 1000, and occur every 100 iterations thereafter. Faults are distributed uniformly across processors. Once a fault happens, the search direction of CG is set as the residual from the last iteration. Later in this section, we experiment with alternate fault models and quantify their effect on performance.

A. Convergence of Raw System and Augmented System

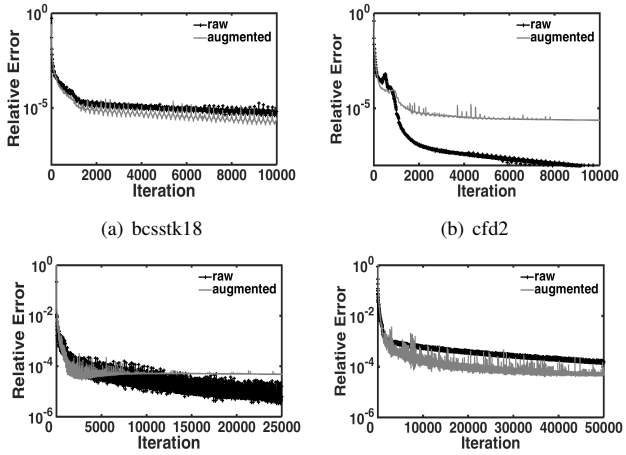


(a) bcsstk18 (b) cfd2

(c) inline (d) Geo_1438

Fig. 4: Convergence of raw and augmented systems for different input matrices.

Our first set of experiments is designed to compare the convergence rates of the raw system and the augmented system. Figure 4 shows convergence of CG for the four matrices for the raw and augmented system. The relative residual for the augmented system is computed with respect to the augmented matrix and not the solution to the raw system. We note from these experiments that input augmentation does not adversely or significantly impact convergence of the base iterative solver. To evaluate the convergence of the solution to the raw system, we execute the solver on the augmented matrix, except, in this case we compute the residual with respect to the raw system. This set of results demonstrates how well the solution converges to the true solution. Figure 5 plots this convergence. For matrices bcsstk18 and inline, we can achieve nearly the same level of relative tolerance as the raw system. For cfd2,



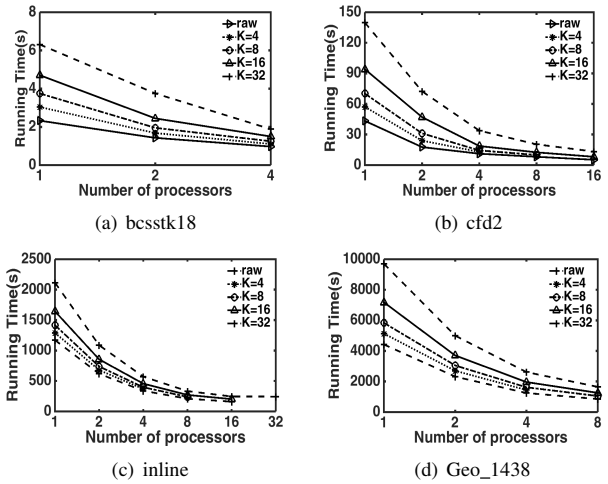
(a) bcsstk18 (b) cfd2

(c) inline (d) Geo_1438

Fig. 5: Convergence of raw and augmented system, with residual computed on the raw system. All results use $k = 4$.

although we do not observe the same relative tolerance as the raw system, we can achieve a relative residual of 10^{-6}. For the largest matrix Geo_1438, we observe better relative tolerance from the augmented system than the raw system. Overall, we observe that the convergence of the solution with respect to the raw system is comparable to the base (error free solver applied to the raw system) method.

B. Parallel Performance of the Solver on Augmented Systems



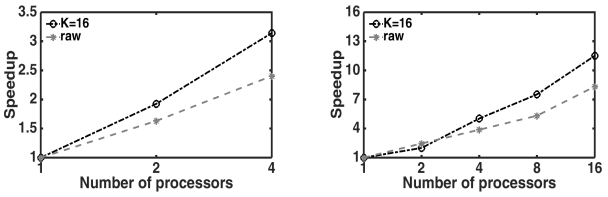
(a) bcsstk18 (b) cfd2

(c) inline (d) Geo_1438

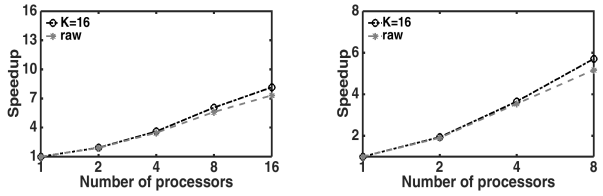
Fig. 6: Parallel performance of augmented systems.

We now investigate the parallel performance of the CG solver on the augmented systems. Figure 6 presents the runtime of the solver on the raw system, augmented system with 4, 8, 16, and 32 faults using different number of processors. We make two observations from these results. First, the runtimes consistently decrease with increasing number of processors for all experiments. Second, we note that increasing the number of

faults increases the parallel runtime. However, this increase in runtime is much lower, compared to the use of active replicas, for instance. Specifically, a tolerance to a single fault would need an active replica for each processor – effectively halving parallel efficiency. This is significantly worse than our results in Figure 6. Figure 7 plots speedups from parallel execution



(a) bcsstk18 (b) cfd2

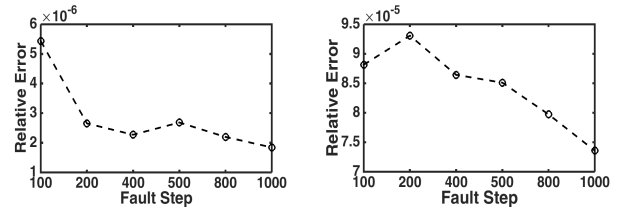


(c) inline (d) Geo_1438

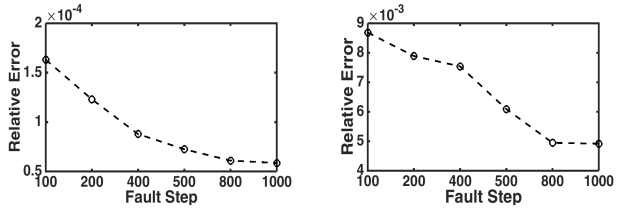
Fig. 7: Speedups of augmented systems.

of the solver on the raw and augmented systems. For the augmented systems, we use $K = 16$ for all experiments. We notice here that in all of the cases, the solver yields good speedups; however, we also notice that the augmented system yields superior speedups compared to the raw system. This is explained by the fact that the augmented system is slightly larger, and therefore, has more computation associated with it. This, combined with similarly low communication overhead yields excellent speedups for the augmented systems.

C. Effect of Fault Rate on Results

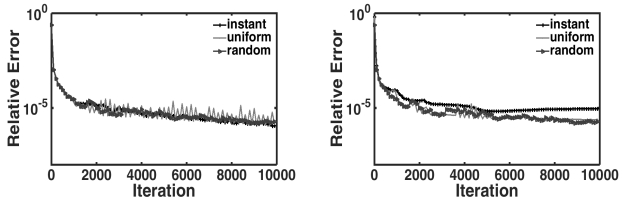


(a) bcsstk18 (b) cfd2

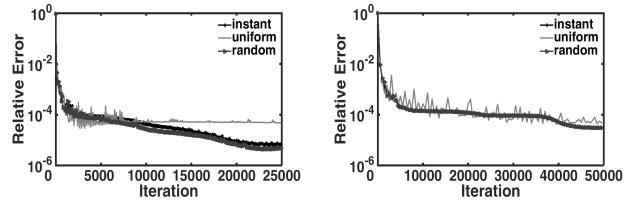


(c) inline (d) Geo_1438

Fig. 8: Effect of fault rate on convergence.



(a) bcsstk18 (b) cfd2



(c) inline (d) Geo_1438

Fig. 9: Convergence of solvers for different fault arrival patterns. In each case here, the augmented matrix uses $k = 4$.

All of our previous results assumed a simple fault arrival model in which faults arrived at a constant rate of one per 100 iterations. In the next set of results, we vary the fault arrival rate and pattern. Figure 8 plots the convergence rate of the solver for varying fault arrival rates. We observe faster convergence when arrival rate is slow, and vice versa. This can be explained by the fact that as faults happen, we set the search direction of CG as the previous residual. Consequently, the higher the fault rate, the more frequently we change search direction and the associated Krylov subspace.

D. Effect of Fault Pattern on Convergence

We now investigate alternate fault arrival patterns and evaluate their impact on convergence of the solver. We specifically experiment with three fault patterns – instantaneous (in which all faults happen at the same time), uniform (faults happen at a uniform rate), and random (in which faults happen at random intervals with a defined mean and variance). Instantaneous faults happen when partitions of machines fail (or multicore nodes fail), resulting in multiple instantaneous failures. Random faults correspond to random single core faults. In each of our experiments, faults are initiated at iteration 1000. Our results in Figure 9 show that the best convergence rate is achieved for the random fault arrival model. The corresponding rate for instantaneous arrivals is close to the random arrival case. These results show that our method is particularly attractive for realistic fault arrival models.

VI. Related Research

The current state of the art in fault tolerance separates into techniques that support fault tolerance through a prescribed computational engine or system, such as Google's MapReduce [3], or those that seek to modify a specific algorithm, though detailed study of its properties, to achieve fault tolerance. Our work can be considered something in between these extremes, where we prescribe a way to modify

the original problem for fault tolerance, and then show how to execute a standard algorithm in a fault-oblivious manner to compute a solution. Commonly used techniques such as checkpoint-restart, active replicas, and deterministic replay are not competitive in the performance regimes we explore. MapReduce, for example, has drawbacks of staged execution and increased job makespan, particularly when the number of faults is large. Furthermore, checkpointing to persistent storage (like a distributed file system) can add significant overhead.

The results most closely related to ours originate in the work of Huang and Abraham [4]. They add a checksum row and column to detect soft errors in a variety of dense matrix computations. This result motivated a line of research [5]–[10] on generalizing the concept, dealing with fail-stop failures, and parallel computation. However, the focus of their work on iterative methods for linear systems involves efficiently emulating a checkpoint restart system, where the efficiently computed checksums are used instead of the checkpoints [10]. Our work simultaneously encodes the entire problem formulation and seeks to ensure erasure coding (rather than checksumming) on the entire end-to-end solution rather than at each step.

Alternative hybrid strategies for solving linear systems of equations with faults include the selective reliability framework, where algorithms are programmed to be tolerant to faults in certain regions of the computations [11]. In the context of linear solves, these methods will prescribe a set of *critical* work that must be done reliably and other sections of *fault tolerant* work that could proceed with a variety of soft errors. For instance, in linear system solvers, residual computations must be done reliably to gauge convergence whereas preconditioning applications can tolerant a variety of faults. This setting then involves flexible Krylov subspace methods, like flexible GMRES [12]–[15]. Generating synthetic but realistic soft errors [16] is a challenge for such methods.

In comparison with the selective reliability work, our erasure-coded approach only requires the encoding and final decoding to be reliable, and these involve a small amount of work. Selective reliability for standard iterative methods usually requires sequences of reliable and fault tolerant sections of code. In comparison with the algorithm checksum work, our ideas explore the use of coding schemes in iterative algorithms in an entirely different manner, where we clearly establish a general framework for these ideas in iterative methods. This involves concepts related to Kruskal rank, as mentioned in previous work on algorithmic fault tolerance [9]. Prior work on fault tolerant iterative methods using coding involves some similar ideas, but in the context of algorithms that restore the state on a per-iteration basis, instead of solution recovery after the entire computation, as in our case.

VII. Conclusion

Fault tolerance is an important problem for scalable parallel and distributed systems. We show how to take recently proposed erasure coding schemes and adapt them to a scenario suitable for distributed computation. This involves creating a new encoding matrix that satisfies the recovery equations for almost all sets of failing components. We show how to partition these matrices and demonstrate their advantage as far as solving systems up to 32 cores in a variety of synthetic fault arrival scenarios.

Our proposed erasure coded computation scheme is general, and can be applied to a number of other problems. Ongoing work in our lab applies these schemes to eigenvalue problems, graph analyses, and other machine learning kernels.

VIII. Acknowledgment

This work was supported in part by DOE award DE-SC0014543.

References

[1] D. F. Gleich, A. Grama, and Y. Zhu, "Erasure coding for fault oblivious linear system solvers," *SIAM J. Sci. Comp.*, To appear.

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

[3] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," in *Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation - Volume 6*, 2004, pp. 10–10.

[4] K.-H. Huang and J. A. Abraham, "Algorithm-based fault tolerance for matrix operations," *IEEE Trans. Comput.*, vol. 33, no. 6, pp. 518–528, Jun. 1984.

[5] Z. Chen and J. Dongarra, "Numerically stable real number codes based on random matrices," in *Computational Science(ICCS)*, 2005, pp. 115–122.

[6] Z. Chen, G. Fagg, E. Gabriel, J. Langou, T. Angskun, G. Bosilca, and J. Dongarra, *Fault tolerant high performance computing by a coding approach*, 2005, pp. 213–223.

[7] Z. Chen and J. Dongarra, "Algorithm-based fault tolerance for fail-stop failures," *IEEE Trans. Parallel Distrib. Syst.*, vol. 19, no. 12, pp. 1628–1641, 2008.

[8] Z. Chen, "Optimal real number codes for fault tolerant matrix operations," in *Proceedings of the ACM/IEEE Conference on High Performance Computing*, 2009.

[9] G. Bosilca, R. Delmas, J. Dongarra, and J. Langou, "Algorithm-based fault tolerance applied to high performance computing," *J. Parallel Distrib. Comput.*, vol. 69, no. 4, pp. 410–416, Apr. 2009.

[10] Z. Chen, "Algorithm-based recovery for iterative methods without checkpointing," in *Proceedings of the 20th ACM International Symposium on High Performance Distributed Computing*, 2011, pp. 73–84.

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

[12] Y. Saad, "A flexible inner-outer preconditioned gmres algorithm," *SIAM J. Sci. Comput.*, vol. 14, no. 2, pp. 461–469, Mar. 1993.

[13] V. Simoncini and D. B. Szyld, "Flexible inner-outer Krylov subspace methods," *SIAM Journal on Numerical Analysis*, vol. 40, pp. 2219–2239, 2003.

[14] J. v. Eshof and G. L. G. Sleijpen, "Inexact krylov subspace methods for linear systems," *SIAM J. Matrix Anal. Appl.*, vol. 26, no. 1, pp. 125–153, Jan. 2004.

[15] V. Simoncini and D. B. Szyld, "Theory of inexact Krylov subspace methods and applications to scientific computing," *SIAM Journal on Scientific Computing*, vol. 25, pp. 454–477, 2003.

[16] J. Elliott, M. Hoemmen, and F. Mueller, "A numerical soft fault model for iterative linear solvers," in *Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing*. ACM, 2015, pp. 271–274.