Erasure Coding Techniques for Faster Distributed Computing and Content Download

Sarah E. Anderson, Alyson Fox, Ann Johnston, Gauri Joshi, Fatemeh Kazemi, Fiona Knoll, Gretchen L. Matthews, Carolyn Mayer, and Emina Soljanin

Abstract Large volumes of data, which are being collected for the purpose of knowledge extraction, have to be reliably and efficiently stored. Furthermore, retrieval and processing of large data files have to be fast and efficient. Recent work has proposed using erasure codes to address both goals. This paper is concerned with two problems that involve coding and are of particular importance in practice. The first problem deals with coded distributed sparse matrix multiplication. It is shown how coded computing methods previously proposed for arbitrary matrices can be modified and efficiently used for sparse matrices that occur more often in applications. The second problem is concerned with service rates in coded storage systems. It is shown how judicious combining of coding and replication can be used to shape service rate regions of distributed storage systems.

Sarah E. Anderson
University of St. Thomas, e-mail: ande1298@stthomas.edu

Alyson Fox
Lawrence Livermore National Laboratory, e-mail: fox33@llnl.gov

Ann Johnston
Penn State University, e-mail: abj5162@psu.edu

Gauri Joshi
Carnegie Mellon University, e-mail: gaurij@andrew.cmu.edu

Fatemeh Kazemi
Texas A&M University, e-mail: fatemeh.kazemi@tamu.edu

Fiona Knoll
University of Cincinnati, e-mail: fiona.knoll@uc.edu

Gretchen L. Matthews
Clemson University, e-mail: gmatthe@clemson.edu

Carolyn Mayer
University of Nebraska - Lincoln, e-mail: cmayer@huskers.unl.edu

Emina Soljanin
Rutgers University, e-mail: emna.soljanin@rutgers.edu

1 Introduction

With the ever increasing amount of data on the cloud, there is a growing need for faster cloud computation. For instance, search engines (Google), data streaming (Netflix), and cloud storage companies (Dropbox) perform vast amounts of computing on the cloud. Companies such as Amazon S3 and Microsoft Azure offer cloud computing as a service, where users can rent machines by the hour to run computation jobs. To reduce latency, cloud computing framework, e.g., MapReduce and Hadoop, employs massive parallelization. Large jobs are divided into hundreds of tasks that can be executed in parallel on different machines.

The critical step in several important algorithms used in optimization and machine learning is the computation of linear transforms of high-dimensional vectors [9]. Existing coded computing methods assume arbitrary matrix structure (see e.g. [13]). Thus, for applications that involve a sparse structure, e.g., graph mining tasks, current erasure methods are wasting processing power on the zero elements. In this paper, we deal with coded distributed sparse matrix multiplication. In Sec. 3, we show how coded computing methods previously proposed for arbitrary matrices can be modified and efficiently used for sparse matrices that occur more often in applications.

Cloud services are implemented on top of a distributed storage layer that acts as a middleware to the applications, and also provides the desired content to the users. Therefore, the performance of a cloud system and the quality of user experience rely on the efficiency of data storage.

Most of the recent work on data access has been concerned with the download latency (see e.g., [7, 19, 21, 22, 30] and references therein). It has recently been recognized, that another important metric that measures the availability of the stored data is the service rate [1, 27]. Maximizing service rate (or the throughput) of a distributed system helps support a large number of simultaneous system users. Rate-optimal strategies are also latency-optimal in high traffic. Thus, maximizing the service rate also reduces the latency experienced by users in particular in highly contending scenarios. In this paper, we are concerned with service rates in storage systems that use redundancy. In Sec. 4, we show how judicious combining of coding and replication can be used to shape service rate regions of distributed storage systems.

The plan of this paper is to state the two emerging problems, provide some examples and preliminary results, and list some open problems and possible future directions. The purpose of the paper is to highlight some usually overlooked aspects of data science, where a number of fundamental math disciplines are important, and many mathematicians could contribute.

2 Linear Codes

Let $\mathbb{F}$ be a field. An $[n,k,d]$-linear code C is a linear subspace of $\mathbb{F}^n$ with dimension k and minimum distance d. The minimum distance d is defined as

$$d := \min\{d(x,y) : x \neq y \in C\},$$

where $d(x,y)$ denotes the Hamming distance between vectors x and y, that is, $d(x,y)$ is the number of differing entries between x and y. The elements of C are called codewords and the Hamming weight of a codeword x, denoted $w_H(x)$, is defined to be $w_H(x) := d(x,0)$.

A generator matrix, denoted G, of code C is a $k \times n$ matrix where the rows form a basis for code C. As a result, each codeword of C can be represented by a linear combination of the rows. That is, $c \in C$ if and only if $c = G^t x$ for some $x \in \mathbb{F}^n$. In this paper, we consider only systematic generator matrices, i.e., the first k columns of G consist of the identity matrix I_k. A code with I_k as a submatrix of its generator matrix G is said to be systematic.

Example 1. Let C be the $[7,4,3]$-code over $\mathbb{Z}_2$ with generator matrix

$$G = \begin{pmatrix} 1&0&0&0&1&1&0 \\ 0&1&0&0&0&1&1 \\ 0&0&1&0&1&1&1 \\ 0&0&0&1&1&0&1 \end{pmatrix}.$$

Then, $C = \{G^t x : x \in \mathbb{Z}_2^4\}$. An example of a codeword is

$$c = G^t \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1&0&0&1&0&1&1 \end{pmatrix}^t.$$

In this paper, we employ error-correcting codes, otherwise known as forward error correction codes. An error-correcting code introduces redundancy to allow for error correction. For instance, a $[2k,k,d]$ code with generator matrix $G = \left[I_k | I_k\right]$, where I_k is the $k \times k$ identity matrix, is a repetition code such that given a k-dimensional vector x, $G^t x$ duplicates the vector, resulting in two stored copies of x. Commonly used error-correcting codes are parity check codes, Hamming codes, Hadamard codes, and Reed-Solomon codes

It is straightforward to see that if $d-1$ or fewer symbols are missing from a codeword of code with minimum distance d, that codeward can be recovered from the remaining symbols. In other words, it is sufficient to have $n-d+1$ symbols of a codeword for its sufficient recovery. We will use this fact for fast coded distributed computing in Sec. 3.

Traditionally, the goal of code design has been to construct codes with large minimum distance d and low redundancy $n-k$. However, in large data storage appli-

cation, one must be able to reconstruct data on a single disk by reading data stored on a small number of other disks; see, for instance, [16]. This property is beneficial when a 1) disk fails and its data is lost and has to be efficiently recovered from the remaining storage and 2) when the demand for some (hot) data is high and cannot be fulfilled by a single server [2, 21, 22]. In Sec. 4, we are concerned with service rates in such systems.

3 Coded Distributed Sparse Matrix Multiplication

The critical step in several important algorithms used in optimization and machine learning is the computation of linear transforms of high-dimensional vectors [9]. In fact, the building blocks of the solutions to various machine learning problems such as regression and classification are linear transforms, which are also used in acquiring and pre-processing the data through Fourier transforms, wavelet transforms, filtering, etc. Thus, fast and reliable computation of linear transforms is necessary for low-latency inference [9]. This task can be easily divided into independent parallel tasks.

System designers have turned to parallel and distributed computing in order to reduce computation time of machine-learning algorithms. However, the bottleneck in distributed computing is the latency in waiting for a small fraction of slow processes called stragglers. The speed of computation is thus reduced as the fusion node must wait on all the outputs from each processor in order to complete the computation. In the majority of distributed systems, a few stragglers have been observed to significantly delay the entire computation [11]. This unpredictable latency in distributed systems is attributed to factors such as network latency, shared resources, maintenance activities, and power limitations. For instance, the execution time of a task on a machine is subject to stochastic variations caused by co-hosting, virtualization and other hardware and network variations [11].

In order to combat the problem of stragglers, cloud computing frameworks like Hadoop [29] employ various straggler detection techniques and reset the task allotted to stragglers. Recent literature [3, 4, 13, 20, 31] proposes introducing redundancy in computations across processors, e.g., using repetition-based strategies or erasure codes. The fusion node can exploit this redundancy by completing the computation using outputs from only a subset of the processors, ignoring the stragglers.

Classical approaches of computing linear transforms across parallel processors, e.g., Block-Striped Decomposition [23], Fox's method [15, 23], and Cannon's method [23], rely on dividing the computational task equally among all available processors without addressing the straggler effect by any redundant computation. The idea of replicating tasks in parallel computing has been recognized by system designers [17] and first adopted at a large scale via the backup tasks in MapReduce [12]. A line of systems work [5, 28] and references therein further developed this idea. Forward error-correction techniques offer an alternative approach to deal with the straggler effect by introducing redundancy in the computational tasks across dif-

ferent processors. By this approach, the fusion node needs to only collect outputs from a subset of all the processors rather than all to successfully complete the computation. In this context, the use of preliminary erasure codes dates back to the ideas of algorithmic fault tolerance [18].

Recently, optimized Repetition and Maximum Distance Separable (MDS) codes have been explored to speed up computations [24, 25, 31]. In [3, 4] the authors studied the impact of coding and replication on the trade off between the "resource usage" and the "execution time" and investigated whether the redundancy should be simple replication or coding, and when it should be introduced. In [13], authors proposed a coding-theory inspired computation technique called Short-Dot for speeding up computing linear transforms of high-dimensional data by distributing it across multiple processing units. Instead of computing long dot products as required in the original linear transforms, the Short-Dot constructs a larger number of redundant and short dot products that can be computed more efficiently at individual processors.

3.1 The Compressed Sparse Row Format for Sparse Matrices

A sparse matrix is a matrix in which the majority of the elements are zero. Sparse matrices are common in many applications such as graph data mining tasks, e.g., graph spectral clustering [26], max-flow min-cut [8], and collaborative recommendation systems [14]. Moreover, differential equations, one of the most import mathematical tools in modeling the physical world, are usually approximated by finite difference or finite volume methods, both of which result in a sparse matrix. These applications often require multiple matrix-vector multiplications and fast computation of sparse linear transforms. A sparse matrix frequently found in applications is the adjacency matrix. The adjacency matrix of a graph can be seen as a stochastic matrix whose nonzeros define a relationship between data points. For instance, in a web graph, the nonzeros of the adjacency matrix denote the probability for a "transition" from one webpage to another. Real-world graphs also tend to be scale-free, meaning that there are many low-degree vertices, i.e., only one or two connections, and very few high-degree vertices, i.e., "hubs" such as Google in a web graph. The rows/columns of the adjacency matrix that represent "hubs" will be densely populated, while most of the rows/columns will only have a few nonzeros. Traditionally, if standard dense-matrix structures are used to represent a sparse matrix, the operations on a sparse matrix will be inefficient as processing power and memory are wasted on the zero entries. Two common sparse data structure are Compressed Sparse Row format (CSR) and Compressed Sparse Column format (CSC), both represent matrix M by three (one-dimensional) arrays and allow for much faster matrix-vector multiplications.

Existing coded computing methods assume arbitrary matrix structure. Thus, for applications that involve a sparse structure, e.g., graph mining tasks, current erasure methods are wasting processing power on the zero elements. Load balancing is also

an issue for existing methods as real-world graphs tend to be scale-free. Thus, a few erasure nodes could potentially be assigned the majority of the work and latency issues are exacerbated. The sparsity structure could be used in conjunction with erasures for speeding up the computation time of linear transforms. In this paper, we address the problem of reducing the time required for computing linear transforms of high-dimensional vectors, leveraging coding techniques and the sparse structure of the matrices in real world applications.

Instead of storing the zeros of the associated matrix in memory, sparse data structures are used to substantially reduce the memory requirement. Compressed Sparse Row (CSR) is one such format that supports efficient memory access and matrix operations. CSR represents the $m \times n$ matrix M with three one-dimensional arrays. Let nnz denote the number of nonzero entries in M. The array $nval \in \mathbb{R}^{nnz}$ will hold the nonzero entries of M in row-major order, i.e., the consecutive non-zero elements of each consecutive row. For each element in $nval$, the array $colval \in \mathbb{R}^{nnz}$ will store the the column index. Let m_i be the number of nonzero entries in row i. Finally, the array $rowptr \in \mathbb{R}^{m+1}$ indicates the number of non-zeros elements in each row and is defined by the following recursive definition. Let $rowptr_1 = 0$, then $rowptr_i = rowptr_{i-1} + m_i$ and finally $rowptr_{m+1} = nnz + 1$ indicating the end of the array.

Example 2. Consider the matrix

$$M = \begin{bmatrix} 3 & -1 & -1 & -1 \\ -1 & 1 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ -1 & 0 & 0 & 1 \end{bmatrix}.$$

To store M in CSR format, three arrays are constructed:

$$nval := [3,-1,-1,-1,-1,1,-1,1,-1,1],$$

which is the ordered by row.

$$colval := [1,2,3,4,1,2,1,3,1,4],$$

which consists of the column index of nonzero elements, e.g., the first positive 1 occurs in the second column, and finally

$$rowptr := [0,4,6,8,11],$$

which designates the pointer to the next row by the number of nonzero entries per row, i.e., the non-zero entries for the second row ($rowptr_2$) begins after 4 entry in $nval$ and $colval$.

An alternative compression format that is less commonly used than CSR is the compressed sparse column (CSC) format. In this paper we focus on matrices stored in the CSR format, and leave the extension to CSC format matrices for future work.

3.2 Erasure Coded Sparse Matrix-Vector Multiplication

Matrices in many modern applications can be massive in size. Even after compressing them using CSR or other formats, performing operations such as matrix-vector multiplication at a single node can be prohibitively slow. Thus, it is necessary to parallelize the computation across multiple processors. Since we need to wait for all the processors to finish computation, even one straggling server can become a bottleneck in completing the matrix-vector multiplication.

We propose a method to use erasure coding to alleviate the problem of stragglers. In particular, before applying CSR to the matrix, we encode the matrix to a larger but still sparse matrix by applying an $[p,k,d]$-linear code, where p is the number of available processors. As a result, we obtain a matrix where any $p-(d-1)$ of the p submatrix-vector multiplications must be returned in order to obtain the desired product Mx. As a result, wait time is reduced.

Let M be an $l \times m$ matrix. We note in applications, M is typically sparse, and in this paper we assume M is sparse and exploit the property of sparseness. Let p be the number of available processors. Let $k \in \mathbb{Z}$ such that $k < p$. We divide the matrix M vertically into k equal dimension sub-matrices. If k does not divide l, then we pad on t additional rows of entries zero, where t is the minimum number such that k divides $l+t$. That is,

$$M = \begin{bmatrix} M_1 \\ M_2 \\ . \\ . \\ M_k \end{bmatrix}.$$

We note that to balance the computation load of the processors, the rows of the matrix M (including the padded zero rows) can be rearranged accordingly.

Let C be a $[p,k,d]$-linear code and let G denote a systematic generator matrix of C. That is, G contains the $k \times k$ identity matrix as a sub-matrix. Applying the generator matrix, we encode the sub-matrices of M to obtain the encoded matrix $\tilde{M} = G^\top M$ (see Example 3). To compute the product Mx, each processor is assigned an encoded submatrix $\tilde{M}_i$ to compute the vector $\tilde{M}_i x$. As C is a $[p,k,d]$-code, it follows that the vector Mx can be obtained from any $p-(d-1)$ of the encoded sub-matrices of $\tilde{M}x$.

Example 3. Let

$$M = \begin{bmatrix} \frac{M_1}{M_2} \\ \hline \frac{M_3}{M_4} \end{bmatrix},$$

where M_1, M_2, M_3, M_4 represent submatrices, and let $p = 7$. Consider the $[7,4,3]$-Hamming Code C from Example 1. Then M encoded is

$$\tilde{M} = G^\top M = \begin{bmatrix} M_1 \\ M_2 \\ M_3 \\ M_4 \\ M_1+M_3+M_4 \\ M_1+M_2+M_3 \\ M_2+M_3+M_4 \end{bmatrix} = \begin{bmatrix} \tilde{M}_1 \\ \tilde{M}_2 \\ \tilde{M}_3 \\ \tilde{M}_4 \\ \tilde{M}_5 \\ \tilde{M}_6 \\ \tilde{M}_7 \end{bmatrix}$$

Processor i is assigned the encoded sub-matrix $\tilde{M}_i$. The fusion node must wait on at most five of the seven processors before computing the end result. More specifically, suppose we have access to the last four sub-matrices of $\tilde{M}x$. Through linear combinations of those matrices, we are able to retrieve M_1x, M_2x, and M_3x. For example,

$$M_1x = M_4x + (M_1x + M_2x + M_3x) - (M_2x + M_3x + M_4x).$$

Once we have the encoded matrix $\tilde{M}$, we store each of the sub-matrices $\tilde{M}_i$ in CSR format.

Example 3 (continued). To store sub-matrices $\tilde{M}_i$, $1 \leq i \leq p$ in CSR format, we construct three arrays as dictated in Section 3.1. Suppose

$$\tilde{M}_1 = \begin{bmatrix} 3 & -1 & -1 & -1 \\ -1 & 1 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ -1 & 0 & 0 & 1 \\ 1 & -1 & 0 & 0 \\ 1 & 0 & 0 & -1 \\ -3 & 1 & 1 & 1 \end{bmatrix}.$$

Then the three arrays constructed are as follows:

$$nval := [3,-1,-1,-1,-1,1,-1,1,-1,1,-1,1,1,-1,1,-1,-3,1,1,1]$$
$$colval := [1,2,3,4,1,2,1,3,1,4,1,2,1,4,1,2,3,4]$$
$$rowptr := [0,4,6,8,10,12,14,17].$$

In general, if M is a $l \times m$ matrix, we choose a code $[p,k,d]$-linear code C, $p \geq k$.

1. Using the generator matrix G of C, compute the encoded matrix $\tilde{M} = G^\top M$.
2. Store sub-matrices $\tilde{M}_i$ in CSR format.
3. Given vector $x \in R^m$, compute $\tilde{M}x$ by assigning sub-matrices $\tilde{M}_i$ among the processors.
4. Given $\gamma = p - (d-1)$ of the computations

$$\tilde{M}_{i_1}x, \ldots, \tilde{M}_{i_\gamma}x$$

return it to the original format $M_1x, M_2x, \ldots, M_kx$, where $\tilde{M}_i$ is a sub-matrix of $\tilde{M}$.

3.3 Advantages of Coded Computing

As discussed earlier, when a computing job is distributed across p processors, waiting for all the processors to finish can significantly slow down the completion of the computing job. In the sparse matrix-vector multiplication problem considered here, the straggling issue is exacerbated by varying sparsity across the rows of the matrix M, due to which some processors may be assigned dense rows and hence take longer than other processors which with get sparse rows.

The coding scheme proposed above efficiently overcomes the issue of stragglers. In our scheme described in Section 3.2, the matrix is split into k sub-matrices, coded using an $[p,k,d]$ code, and then distributed across p processors. We need to wait only for some $p-d+1$ processors to finish multiplying their sub-matrix with the vector. Thus, it can tolerate $d-1$ slow servers.

We now present an analysis of the expected computing time with different values of p, k and d, and the distance of the code to demonstrate the advantage of the proposed scheme. The time taken to multiply r out of the total of l rows of the matrix M to the vector x is modeled by an exponential random variable S_r. Its tail probability distribution $\Pr(S_r > t)$ is

$$\Pr(S_r > t) = e^{-\frac{l\mu}{r}t}, \text{ for } t \geq 0, \tag{1}$$

where μ is the service rate of the exponential distribution.

Theorem 1 *Suppose the matrix M is split horizontally into k sub-matrices. It is then coded using an $[p,k,d]$ code, and the computation is distributed across p processors. Then, the expected time to complete the matrix-vector product is,*

$$\mathbb{E}[T] = \frac{H_p - H_{d-1}}{k\mu}, \tag{2}$$

where H_p is the p^{th} harmonic number,

$$H_p = \sum_{i=1}^{p} \frac{1}{i}. \tag{3}$$

The zero-th harmonic number is defined as $H_0 = 0$.

Proof. The time T to finish the matrix-vector product is equal to the time to wait for $p-d+1$ processors to return their sub-matrix-vector products is the $(p-d+1)^{th}$ order statistic of p i.i.d. random variables $Y_1, Y_2, \ldots Y_p$, where Y_i is the time taken by processor i. By (1), its tail distribution is,

$$\Pr(Y > t) = e^{-k\mu t}, \text{ for } t \geq 0. \tag{4}$$

Thus, the expected value of the total computation time T is,

$$\mathbb{E}[T] = \mathbb{E}\left[Y_{p-d+1:p}\right], \tag{5}$$

$$= \frac{H_p - H_{d-1}}{k\mu}, \tag{6}$$

where $Y_{p-d+1:p}$ denotes the $(p-d+1)^{th}$ order statistic of p i.i.d. random variables $Y_1, Y_2, \ldots Y_p$. Since Y_i are exponential random variables, by the properties of exponential distributions [10], the expected value of $Y_{p-d+1:p}$ can be expressed as a difference of harmonic numbers as shown in (6).

The approximation $H_p \approx \log p$ holds for large p. The uncoded scheme where the matrix is simply split into p parts which are assigned to p processors corresponds to $d = 1$. Thus, $\mathbb{E}[T] = O(\log p)$. Instead if we use coding, the expected computation time is,

$$\mathbb{E}[T] \approx \frac{1}{k\mu} \log\left(\frac{p}{d-1}\right). \tag{7}$$

Thus, if d is large enough, then we can significantly reduce the computation delay.

3.4 Future Directions

There are many interesting open questions for future work on designing codes to speed-up distributed computing. An immediate future direction is to evaluate the proposed coding scheme by running experiments on Amazon S3. We plan to use graph matrices available publicly, for example, through the Stanford Network Analysis Project. Another direction is to explore the use of rate-less fountain codes such as LT codes and Raptor codes for distributed computing, instead of fixed-rate linear codes considered in this paper. Moreover, erasure coding is applicable to several computation problems beyond matrix-vector multiplication, for example, matrix-matrix multiplication, matrix inversion and convolutions. Designing coding techniques for latency reduction as well as error-correction in these problems is an open future direction.

4 Service Capacity Region of a Storage System

Two important considerations for file storage system design are reliability against node failures and the ability to simultaneously serve a large number of users. Often, these considerations are addressed through file replication at multiple nodes. However, recent work suggests coded file storage as a promising alternative.

Suppose files $f_1, \ldots, f_K$ are stored in a system that consists of N nodes. A node is said to by systematic for file f_k if accessing the node gives access to the file. A coded storage system is one in which some nodes are not systematic for any file,

but instead give access to one or more files only when accessed in combination with other nodes of the system.

Assigning each storage system node a label in $1,\ldots,N$, a subset of $[N] := \{1,\ldots N\}$ is called an f_k–repair group if its nodes together give access to file f_k, with no proper subset of its nodes giving access to file f_k. Let $R_{k1},\ldots,R_{k\gamma_k} \subseteq [N]$ denote the minimal collections of nodes that give access to file f_k (that is, each R_{ki} is an f_k–repair group). When a request for file f_k arrives to the storage system, suppose that it is sent with probability α_{ki} to R_{ki}, under the constraint that $\sum_{i=1}^{\gamma_k} \alpha_{ki} = 1$. Here, $\bigcup_{k=1}^{K} \{\alpha_{k1},\ldots,\alpha_{k\gamma_k}\}$ is referred to as the storage system's "splitting strategy."

Suppose that at node $j \in [N]$ the average rate of resolving received requests is μ_j. Further, suppose that the arrival of requests for file f_k to the storage system queue is Poisson with rate λ_k, where λ_k is referred to as the "demand" for file f_k. Whenever demand is such that a node j of the storage system receives requests at a rate in excess of its μ_j, the storage system queue will have a tendency to grow. With this in mind, it is appropriate to call μ_j the service capacity of node j. If, at demand $\lambda = (\lambda_1,\ldots,\lambda_K)$, there exists a splitting strategy under which no storage system node receives requests at a rate in excess of its capacity, then λ is said to be in the service capacity region of the storage system. More formally, letting

$$\delta_k(i,j) := \begin{cases} 1, & \text{if node } j \text{ is in } R_{ki}, \\ 0, & \text{else,} \end{cases}$$

a storage system's service capacity region $\mathscr{S}$ is the set of all $\lambda \in \mathbb{R}^K_{\geq 0}$ such that there exists a splitting strategy with

$$\sum_{k\in[K]} \sum_{i\in[\gamma_k]} \alpha_{ki}\delta_k(i,j)\,\lambda_k \leq \mu_j, \quad \text{for all } j \in [N]. \tag{8}$$

A given storage system's serviceable demand intensity and its robustness to demand variability can be studied via an analysis of its service capacity region.

4.1 Recent work

If a given splitting strategy makes demand $\lambda = (\lambda_1,\ldots,\lambda_K)$ serviceable, then that same splitting strategy is also sufficient to serve any demand λ' that satisfies $\lambda'_k \leq \lambda_k$ for all $k \in [K]$. In this way, and noting that the service capacity region is a subset of $\mathbb{R}^K_{\geq 0}$, a storage system's service capacity region is fully described by its upper boundary $L(\cdot)$ on the storage system's $(K-1)$–dimensional service domain

$$\mathscr{S}^{(K-1)} := \left\{(\lambda_1,\ldots,\lambda_{K-1}) \in \mathbb{R}^{K-1}_{\geq 0} : (\lambda_1,\ldots,\lambda_{K-1},0) \in \mathscr{S}\right\}. \tag{9}$$

Here,

$$L(\lambda_1,\ldots,\lambda_{K-1}) := \max_{\left\{\lambda_K, \bigcup_{k=1}^{K}\{\alpha_{k1},\ldots,\alpha_{k\gamma_k}\}\right\}} \lambda_K, \quad \text{subject to}$$

$$\text{for all } k \in [K], \quad \lambda_k \geq 0, \quad \sum_{i=1}^{\gamma_k} \alpha_{ki} = 1, \quad \text{and}$$

$$\text{for all } i \in [\gamma_k], \quad \alpha_{ki} \geq 0; \quad \text{and,}$$

$$\text{for all } j \in [N],$$

$$\sum_{k\in[K]} \sum_{i\in[\gamma_k]} \alpha_{ki}\delta_k(i,j)\,\lambda_k \leq \mu_j. \tag{10}$$

In this way, the problem of identifying a storage system's service capacity region can be viewed as one of optimization. The service capacity region was introduced in [27] and considered in [1] and [6].

In [1], we consider (N,K) systematic MDS coded systems, which are those in which each file is stored on a single systematic node and may be accessed either via that node or by accessing any K of the $N-1$ other nodes. To address this situation, we introduce the Water-filling Algorithm and show that is optimal, and we show that the service capacity region is given by

$$\sum_{i=1}^{K} \left(\min(\lambda_i,\mu) + K(\lambda_i-\mu)^+\right) \leq N\mu$$

for codes of rate $\frac{K}{N} \leq \frac{1}{2}$. Here, we use the notation $x^+ := \max(0,x)$.

Water-filling Algorithm: Given arrival rates $\lambda_1,\lambda_2,\ldots,\lambda_K$ for the K files, assign requests to the N nodes as follows:

- Let γ_i be the load on node i. Assign file requests to their respective systematic nodes until these nodes are saturated. Set $\gamma_i = \min(\lambda_i,\mu)$ for $i = 1,\ldots,K$.
- Each of the remaining $\lambda_{coded} = \sum_{i=1}^{K}(\lambda_i-\mu)^+$ can be served by any K unsaturated servers.
- While $\lambda_{coded} > 0$ and $\min_i \gamma_i < \mu$ do the following: Find the K least-loaded servers in the system, that is, the K servers with minimum γ_i's. From λ_{coded}, send an infinitesimally small rate $\varepsilon > 0$ to each of these K servers. Decrement λ_{coded} by ε, and increment the corresponding K γ_i's by ε.

In the same paper, we also consider storage systems for 2 files. Here, we let λ_a^* denote the maximum demand for a that can be supported by a given storage system. We show that the service capacity region s bounded by $\lambda_a = 0, \lambda_b = 0, \lambda_a = \min\{(A+C)\mu, (A+\frac{B}{2}+\frac{C}{2})\mu\}$, and

$$L(\lambda_a) = \begin{cases} (B+C)\mu & \text{if } A > C \text{ and} \\ & 0 \le \lambda_a \le (A-C)\mu \\ -\frac{1}{2}\lambda_a + (\frac{A}{2}+B+\frac{C}{2})\mu & \text{if } A > C \text{ and} \\ & (A-C)\mu < \lambda_a \le A\mu \\ -\frac{1}{2}\lambda_a + (\frac{A}{2}+B+\frac{C}{2})\mu & \text{if } A \le C \text{ and} \\ & 0 \le \lambda_a \le A\mu \\ -\lambda_a + (A+B+\frac{C}{2})\mu & \text{if } A\mu < \lambda_a \le (A+\frac{C}{2})\mu \\ -2\lambda_a + (2A+B+C)\mu & \text{if } B > C \text{ and} \\ & (A+\frac{C}{2})\mu < \lambda_a \le A+C \\ -2\lambda_a + (2A+B+C)\mu & \text{if } B \le C \text{ and } (A+\frac{C}{2})\mu \\ & < \lambda_a \le (A+\frac{B}{2}+\frac{C}{2})\mu, \end{cases}$$

where there are A systematic nodes for file a, B systematic nodes for file b, and C coded nodes.

In [6], we consider the 3-file case as well as systems which have an MDS core, meaning storage systems for K files whose coded nodes satisfy the following three conditions:

1. Each K–subset of coded nodes forms an f_k–repair group for every $k \in [K]$.
2. No subset of $k < K$ coded nodes forms an f_k–repair group, for any $k \in [K]$.
3. With addition of systematic nodes for any n distinct files (naturally, $n < K$) every $(K-n)$–subset of coded nodes from the core completes these systematic nodes to form an f_k–repair group for every $k \in [K]$.

We assume uniform node capacities $\mu = \mu_1 = \cdots = \mu_N$. We see that in a system with all C coded nodes the service capacity region is bounded by $\lambda_1 = 0, \ldots, \lambda_K = 0$ and

$$L(\lambda_1, \ldots, \lambda_{K-1}) = \frac{C}{K}\mu - \sum_{i=1}^{K-1} \lambda_i$$

if $C > K-1$; whereas if there are $C \le K-1$ coded nodes, then the service capacity region is the point $(0, \ldots, 0)$. In addition, we provide an algorithm for maximizing λ_k for the general K-file case. The 3-file case is studied as well.

In the next section, we provide a detailed example in the 2-file case in order to share some intuition behind the bounds.

4.2 Advantages of Coded Storage Systems

Coded storage systems have larger service capacity regions compared to uncoded storage systems, which allows them to better handle requests when there are skews in demand. For example, consider the differences between the following two storage systems for 2 files with a total of 14 nodes.

Example: Assume we have a uncoded storage system for 2 files with 7 systematic nodes for file f_1 and 7 systematic nodes for f_2. Then the maximum achievable λ_2 is $L(\lambda_1) = 7\mu$ for $0 \leq \lambda_1 \leq 7\mu$. This region is shown in Figure 1.

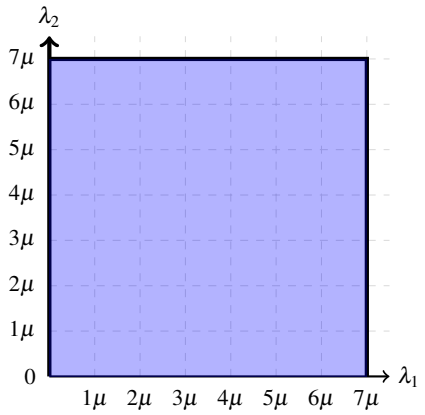

Fig. 1 Service capacity region for an uncoded storage system with 7 systematic nodes for file f_1 and 7 systematic nodes for file f_2.

Example: Assume we have a coded storage system for 2 files with an MDS core with 4 systematic nodes for file f_1, 4 systematic nodes for file f_2, and 6 coded nodes. That is, any repair group consisting of two coded nodes or a coded node and a systematic file node can recover either file.

If $0 \leq \lambda_1 \leq 4\mu$, then we may satisfy all the demand for file f_1 by sending $\frac{\lambda_1}{4}$ demand to each systematic file f_1 node. Note, each systematic file f_1 node has a remaining capacity of $\mu - \frac{\lambda_1}{4}$. We can now satisfy demand for file f_2. Recall, in addition to the systematic nodes for file f_2, any two coded nodes or any coded node with a systematic file f_1 node can be used to recover file f_2. There are 24 repair groups with a coded node and a systematic file f_1 node. We can satisfy $4\left(\mu - \frac{\lambda_1}{4}\right)$ demand for file f_2 by sending demand equally to these repair groups. The systematic file f_1 nodes have zero capacity remaining while the coded nodes have $\mu - \frac{4}{6}\left(\mu - \frac{\lambda_1}{4}\right)$ capacity remaining. We can use the remaining coded node capacity by creating repair groups with two coded nodes and sending demand equally to each of these repair groups, which will satisfy $\frac{6}{2}\left(\mu - 4\left(\mu - \frac{\lambda_1}{4}\right)\right)$ demand for file f_2. The systematic file f_2 nodes can also satisfy 4μ demand for file f_2. Thus, when $0 \leq \lambda_1 \leq 4\mu$, the maximum achievable λ_2 is

$$4\left(\mu - \frac{\lambda_1}{4}\right) + 3\left(\mu - \frac{2}{3}\left(\mu - \frac{\lambda_1}{4}\right)\right) + 4\mu = 9\mu - \frac{1}{2}\lambda_1.$$

If $4\mu < \lambda_1 \leq 7\mu$, then we may satisfy 4μ demand for file f_1 by using all of the systematic file f_1 nodes; however, we must also use coded nodes to satisfy demand for file f_1. As before, since any two coded nodes can be used to recover file f_1 or f_2, we can use repair groups that consist of two unique coded nodes to satisfy a total of 3μ demand. Thus, by sending $\lambda_1 - 4\mu$ demand equally to each of these repair groups, we have $3\mu - (\lambda_1 - 4\mu) = 7\mu - \lambda_1$ demand remaining to utilize in serving demand for file f_2. In addition, the systematic file f_2 nodes may be used, so the maximum achievable λ_2 is $11\mu - \lambda_1$.

If $7\mu < \lambda_1 \leq 9\mu$, then again we may satisfy 4μ demand for file f_1 by using all of the systematic file f_1 nodes. In this case, we need to also use coded nodes as well as systematic file f_2 nodes to satisfy the remaining demand for file f_1. First, consider repair groups consisting of one coded node and one systematic file f_2 node. We can satisfy $2(\lambda_1 - 7\mu)$ demand for file f_1 by sending demand equally to each of these repair groups. The remaining capacity of each coded node is $\mu - \frac{2}{6}(\lambda_1 - 7\mu)$, and the remaining capacity of each systematic file f_2 node is $\mu - \frac{2}{4}(\lambda_1 - 7\mu)$. We may satisfy the remaining file f_1 demand by sending it all equally to all the repair groups consisting of two coded nodes, which can handle $\frac{6}{2}\left(\mu - \frac{2}{6}(\lambda_1 - 7\mu)\right)$ demand. This satisfies all demand for file f_1, since

$$4\mu + 2(\lambda_1 - 7\mu) + 3\left(\mu - \frac{2}{6}(\lambda_1 - 7\mu)\right) = \lambda_1.$$

The remaining capacity of systematic file f_2 nodes may be used to satisfy demand for f_2. Hence, the the maximum achievable λ_2 is $4\left(\mu - \frac{2}{4}(\lambda_1 - 7\mu)\right) = 18\mu - 2\lambda_1$.

Therefore,

$$L(\lambda_1) = \begin{cases} 9\mu - \frac{1}{2}\lambda_1, & \text{if } 0 \leq \lambda_1 \leq 4\mu \\ 11\mu - \lambda_1, & \text{if } 4\mu < \lambda_1 \leq 7\mu \\ 18\mu - 2\lambda_1, & \text{if } 7\mu < \lambda_1 \leq 9\mu. \end{cases}$$

This storage system's service capacity region is shown in Figure 2.

We can note that in the uncoded storage system the maximum amount of demand that can be handled for either file is 7μ; however, the coded storage system can better satisfy demand when one file is more popular, up to 9μ.

This idea of systematically utilizing repair groups to satisfy demand was generalized in Algorithm 1 in [6] for MDS core coded storage systems with K files.

4.3 Future Directions

There are still many open questions regarding coded storage systems.

Question 1. What is the service capacity region for an MDS core coded storage systems with K files?

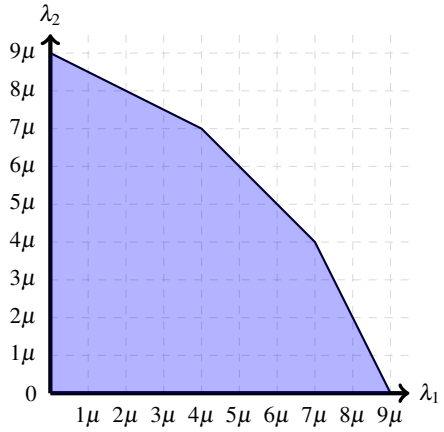

Fig. 2 Service capacity region for a coded storage system with 6 coded nodes in an MDS core, 4 systematic nodes for f_1 and 4 systematic nodes for file f_2.

MDS core coded storage systems with $K = 2$ and $K = 3$ files were addressed in [1] and [6], respectively. Additionally, a bound was given for storage capacity regions of systems with MDS coded cores for K files, in cases where the storage system consists of one systematic node for each file and $N - K$ coded nodes. However, an explicit $L(\cdot)$ function for the service capacity region for general MDS coded core storage systems for K files has yet to be given.

Question 2. What is the service capacity region of a coded storage system when each user may simultaneously request either a single file or multiple files?

Example: As shown in [1], the service capacity region of the system that consists of one systematic file f_1 node, one systematic file f_2 node, and one coded node has

$$L(\lambda_1) = \begin{cases} -\frac{1}{2}\lambda_1 + 2\mu & \text{if } 0 \leq \lambda_1 \leq \mu \\ -\lambda_1 + \frac{5}{2}\mu & \text{if } \mu < \lambda_1 \leq \frac{3}{2}\mu \\ -2\lambda_1 + 4\mu & \text{if } \frac{3}{2}\mu < \lambda_1 \leq 2\mu \end{cases}.$$

If file f_1 requests and file f_2 requests are sent separately, note that the point $(\frac{3}{2}\mu, \frac{3}{2}\mu)$ is outside the service capacity region. However, $\frac{3}{2}\mu$ requests for both files may be served if both files are retrieved together. This can be done by sending $\frac{\mu}{2}$ requests to each of the following three repair groups: **(i)** the systematic f_1 node and the coded node, **(ii)** the systematic f_2 node and the coded node, and **(iii)** both systematic nodes.

In a coded storage system in which users may request files either separately or together, we are interested in the optimal allocation of coded and systematic nodes.

Question 3. What is the service capacity region of a coded storage system under different coding schemes?

We have considered an MDS core coding scheme in which any $N \leq K$ coded nodes along with systematic nodes for $K - N$ distinct files can recover any file. How the service capacity changes as this condition is relaxed remains an open question. Locally recoverable codes, which allow for file recovery using small repair groups, may be of particular interest in this setting.

Acknowledgements Part of this research is based upon work supported by the National Science Foundation under Grant No. DMS-1439786 while the authors were in residence at the Institute for Computational and Experimental Research in Mathematics in Providence, RI, during the *Women in Data Science and Mathematics Research Collaboration Workshop* at ICERM in July 2017.

References

[1] Aktas M, Anderson SE, Johnston A, Joshi G, Kadhe S, Matthews GL, Mayer C, Soljanin E (2017) On the service capacity of accessing erasure coded content. In: Proceedings of the Allerton Conference on Communication, Control and Computing

[2] Aktas MF, Najm E, Soljanin E (2017) Simplex queues for hot-data download. In: Proceedings of the SIGMETRICS/International Conference on Measurement and Modeling of Computer Systems, ACM

[3] Aktas MF, Peng P, Soljanin E (2017) Effective straggler mitigation: Which clones should attack and when? arXiv preprint arXiv:171000748

[4] Aktas MF, Peng P, Soljanin E (2017) Straggler mitigation by delayed relaunch of tasks. arXiv preprint arXiv:171000414

[5] Ananthanarayanan G, Ghodsi A, Shenker S, Stoica I (2013) Effective straggler mitigation: Attack of the clones. In: NSDI, vol 13, pp 185–198

[6] Anderson SE, Johnston A, Joshi G, Matthews GL, Mayer C, Soljanin E (2018) On the service capacity region of accessing erasure coded content. preprint

[7] Chen S, Sun Y, Kozat UC, Huang L, Sinha P, Liang G, Liu X, Shroff NB (2014) When queueing meets coding: Optimal-latency data retrieving scheme in storage clouds. In: IEEE Conf. on Computer Communications (INFOCOM), pp 1042–1050

[8] Christiano P, Kelner JA, Madry A, Spielman DA, Teng SH (2011) Electrical flows, laplacian systems, and faster approximation of maximum flow in undirected graphs. In: Proceedings of the Forty-third Annual ACM Symposium on Theory of Computing, ACM, New York, NY, USA, STOC '11, pp 273–282, DOI 10.1145/1993636.1993674, URL http://doi.acm.org/10.1145/1993636.1993674

[9] Dally W (2015) High-performance hardware for machine learning. NIPS Tutorial

[10] David HA, Nagaraja HN (2003) Order statistics. John Wiley, Hoboken, N.J.

[11] Dean J, Barroso LA (2013) The tail at scale. Communications of the ACM 56(2):74–80

[12] Dean J, Ghemawat S (2008) Mapreduce: simplified data processing on large clusters. Communications of the ACM 51(1):107–113
[13] Dutta S, Cadambe V, Grover P (2016) Short-dot: Computing large linear transforms distributedly using coded short dot products. In: Advances In Neural Information Processing Systems, pp 2100–2108
[14] Fouss F, Pirotte A, m Renders J, Saerens M (2007) Random-walk computation of similarities between nodes of a graph with application to collaborative recommendation. IEEE Transactions on Knowledge and Data Engineering 19(3):355–369
[15] Fox GC, Otto SW, Hey AJ (1987) Matrix algorithms on a hypercube i: Matrix multiplication. Parallel computing 4(1):17–31
[16] Gao S, Knoll F, Manganiello F, Matthews G (2017) Codes for distributed storage from 3-regular graphs. Discrete Applied Mathematics 229:82 – 89
[17] Ghare GD, Leutenegger ST (2004) Improving speedup and response times by replicating parallel programs on a snow. In: JSSPP, Springer, pp 264–287
[18] Huang KH, et al (1984) Algorithm-based fault tolerance for matrix operations. IEEE transactions on computers 100(6):518–528
[19] Joshi G, Liu Y, Soljanin E (2012) Coding for fast content download. In: Communication, Control, and Computing (Allerton), 2012 50th Annual Allerton Conference on, pp 326–333
[20] Joshi G, Soljanin E, Wornell G (2017) Efficient redundancy techniques for latency reduction in cloud systems. ACM Transactions on Modeling and Performance Evaluation of Computing Systems 2(12)
[21] Kadhe S, Soljanin E, Sprintson A (2015) Analyzing the download time of availability codes. In: Information Theory (ISIT), 2015 IEEE International Symposium on, IEEE, pp 1467–1471
[22] Kadhe S, Soljanin E, Sprintson A (2015) When do the availability codes make the stored data more available? In: Communication, Control, and Computing (Allerton), 2015 53rd Annual Allerton Conference on, IEEE, pp 956–963
[23] Kumar V, Grama A, Gupta A, Karypis G (1994) Introduction to parallel computing: design and analysis of algorithms, vol 400. Benjamin/Cummings Redwood City
[24] Lee K, Lam M, Pedarsani R, Papailiopoulos D, Ramchandran K (2016) Speeding up distributed machine learning using codes. In: Information Theory (ISIT), 2016 IEEE International Symposium on, IEEE, pp 1143–1147
[25] Li S, Maddah-Ali MA, Avestimehr AS (2016) A unified coding framework for distributed computing with straggling servers. In: Globecom Workshops (GC Wkshps), 2016 IEEE, IEEE, pp 1–6
[26] von Luxburg U (2007) A tutorial on spectral clustering. Statistics and Computing 17(4):395–416, DOI 10.1007/s11222-007-9033-z, URL https://doi.org/10.1007/s11222-007-9033-z
[27] Noori M, Soljanin E, Ardakani M (2016) On storage allocation for maximum service rate in distributed storage systems. In: 2016 IEEE International Symposium on Information Theory (ISIT), pp 240–244

[28] Ousterhout K, Wendell P, Zaharia M, Stoica I (2013) Sparrow: distributed, low latency scheduling. In: Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles, ACM, pp 69–84

[29] Shvachko K, Kuang H, Radia S, Chansler R (2010) The hadoop distributed file system. In: Mass storage systems and technologies (MSST), 2010 IEEE 26th symposium on, IEEE, pp 1–10

[30] Tandon R, Mohajer S (2014) New bounds for distributed storage systems with secure repair. In: Allerton Conf. on Communication, Control, and Computing, pp 431–436

[31] Wang D, Joshi G, Wornell G (2015) Using straggler replication to reduce latency in large-scale parallel computing. ACM SIGMETRICS Performance Evaluation Review 43(3):7–11