CaCo: An Efficient Cauchy Coding Approach for Cloud Storage Systems

Guangyan Zhang, Guiyong Wu, Shupeng Wang, Jiwu Shu, *Member, IEEE*, Weimin Zheng, and Keqin Li, *Fellow, IEEE*

Abstract—Users of cloud storage usually assign different redundancy configurations (i.e., (k, m, w)) of erasure codes, depending on the desired balance between performance and fault tolerance. Our study finds that with very low probability, one coding scheme chosen by rules of thumb, for a given redundancy configuration, performs best. In this paper, we propose CaCo, an efficient Cauchy coding approach for data storage in the cloud. First, CaCo uses Cauchy matrix heuristics to produce a matrix set. Second, for each matrix in this set, CaCo uses XOR schedule heuristics to generate a series of schedules. Finally, CaCo selects the shortest one from all the produced schedules. In such a way, CaCo has the ability to identify an optimal coding scheme, within the capability of the current state of the art, for an arbitrary given redundancy configuration. By leverage of CaCo's nature of ease to parallelize, we boost significantly the performance of the selection process with abundant computational resources in the cloud. We implement CaCo in the Hadoop distributed file system and evaluate its performance by comparing with "Hadoop-EC" developed by Microsoft research. Our experimental results indicate that CaCo can obtain an optimal coding scheme within acceptable time. Furthermore, CaCo outperforms Hadoop-EC by 26.68-40.18 percent in the encoding time and by 38.4-52.83 percent in the decoding time simultaneously.

Index Terms—Cloud storage, fault tolerance, Reed-Solomon codes, Cauchy matrix, XOR scheduling

1 Introduction

Cloud storage is built up of numerous inexpensive and unreliable components, which leads to a decrease in the overall mean time between failures (MTBF). As storage systems grow in scale and are deployed over wider networks, component failures have been more common, and requirements for fault tolerance have been further increased. So, the failure protection offered by the standard RAID levels has been no longer sufficient in many cases, and storage designers are considering how to tolerate larger numbers of failures [1], [2]. For example, Google's cloud storage [3], Windows Azure Storage [4], OceanStore [5], DiskReduce [6], HAIL [7], and others [8] all tolerate at least three failures.

To tolerate more failures than RAID, many storage systems employ Reed-Solomon (RS) codes for fault-tolerance [9], [10]. Reed-Solomon coding has been around for decades, and has a sound theoretical basis. As an erasure code, Reed-Solomon code is widely used in the field of data storage. Given k data blocks and a positive integer m, Reed-Solomon codes can encode the content of data blocks into m coding blocks, so that the storage system is resilient to any m disk failures. Reed-Solomon codes operate on binary words of data, and each word is composed of w bits, where $2^w \geq k + m$. In the rest of this paper, we denote a combination of k, m, and w by a redundancy configuration (k, m, w), where k data blocks are encoded into m coding blocks, with the coding unit of w-bit words. Reed-Solomon codes [9] treat each word as a number between 0 and $2^w - 1$, and employ Galois Field arithmetic over GF(2^w). In GF(2^w), addition is performed by bitwise exclusive-or (XOR), while multiplication is more complicated, typically implemented with look-ups to logarithm tables. Thus, Reed-Solomon codes are considered expensive.

Cauchy Reed-Solomon (CRS) codes improve Reed-Solomon codes by using neat projection to convert Galois Field multiplications into XOR operations [11]. Currently, CRS codes represent the best performing general purpose erasure codes for storage systems [12]. In addition, CRS coding operates on entire strips across multiple storage devices instead of operating on single words. In particular, strips are partitioned into w packets, and these packets may be large. Fig. 1 illustrates a typical architecture for a cloud storage system with data coding. The redundancy configuration of the system is $k = 4$ and $m = 2$. With CRS codes, k data blocks are encoded into m coding blocks. In such a way, the system can tolerate any m disk failures without data loss. Note that those k data blocks and m coding blocks should be stored on different data nodes. Otherwise, the failure of one node may lead to multiple faults in the same group of $n = k + m$ blocks.

A matrix-vector product, $AX = B$, is central to CRS codes. All elements of A, X, and B are bits. Here, A is defined as a Cauchy matrix, X stands for data strips, and B stands for strips of coding information. Since the act of CRS coding involves only XOR operations, the number of XORs required

- *G. Zhang, G. Wu, J. Shu, and W. Zheng are with the Department of Computer Science and Technology, Tsinghua University, Beijing, China 100084.*
  E-mail: {gyzh, shujw, zwm-dcs}@tsinghua.edu.cn, wugy10@gmail.com.
- *S. Wang is with the Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100029, China.*
  E-mail: wangshupeng@iie.ac.cn.
- *K. Li is with the Department of Computer Science, State University of New York, New Paltz, New York 12561. E-mail: lik@newpaltz.edu.*

Manuscript received 25 Feb. 2014; revised 23 Mar. 2015; accepted 27 Apr. 2015. Date of publication 30 Apr. 2015; date of current version 15 Jan. 2016.
Recommended for acceptance by Z. Tari.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TC.2015.2428701

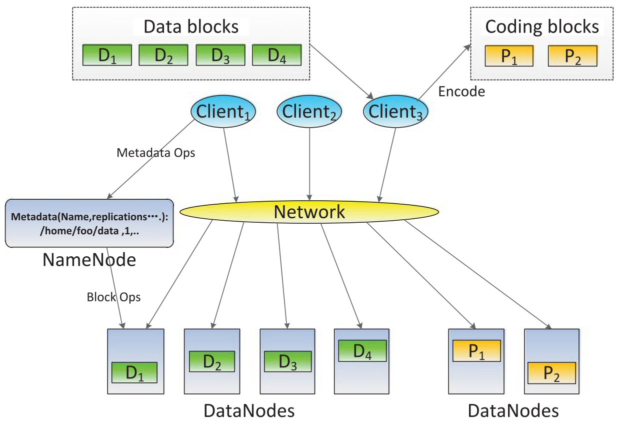


Fig. 1. A distributed architecture for a cloud storage system with data coding, where $k = 4$, and $m = 2$.

by a CRS code impinges directly upon the performance of encoding or decoding. While there are other factors that impact performance, especially cache behavior and device latency, reducing XORs is a reliable and effective way to improve the performance of a code. For example, nearly all special-purpose erasure codes, from RAID-6 codes (e.g., EVENODD [13], RDP [14], X [15], and P [16] codes) to codes for larger systems (e.g., STAR [17], T [18], and WEAVER [19] codes), aim at minimizing XOR operations at their core. The goal of this paper is to find a best-performing coding scheme that performs data coding with the fewest XOR operations.

1.1 The Current State of the Art

Many efforts have been devoted to achieve this goal. Initially, people discover that the density of a Cauchy matrix dictates the number of XORs [20]. For this reason, an amount of work has sought to design codes with low density [12], [20], [21]. Moreover, some lower bounds have been derived on the density of MDS Cauchy matrices. In the current state of the art, the only way to discover lowest-density Cauchy matrices is to enumerate all the matrices and select the best one. Given a redundancy configuration (k, m, w), the number of matrices is $\binom{2^w}{k+m}\binom{k+m}{k}$, which is clearly exponential in k and m. Therefore, the enumeration method for the optimal matrix makes sense only for some small cases.

When scheduling of XOR operations is introduced, the density of the matrix does not have a direct effect on the number of XORs [22]. Huang et al. addressed the issue of identifying and exploiting common sums in the XOR equations [23]. They conjectured that deriving an optimal schedule based on common sums is NP-Complete, and gave a heuristic based on Edmonds maximum matching algorithm [24]. Plank et al. attacked this open problem, deriving two new heuristics called Uber-CHRS and X-Sets to schedule encoding and decoding bit-matrices with reduced XOR operations [25].

For data of different importance, one might provide different degrees of redundancy. In other words, users of cloud storage usually assign different redundancy configurations (i.e., (k, m, w)) of CRS codes, depending on the desired balance between performance and fault tolerance. It is still an open problem how to derive a Cauchy matrix and one of its schedules so as to obtain the fewest XOR operations for a given redundancy configuration.

1.2 The Contribution of This Paper

The contribution of this paper is two-fold. First, through a number of experiments and numerical analyses, we get some beneficial observations as follows (See Section 6.1 for more details).

- Given a redundancy configuration (k, m, w), the shortest XOR schedule that one can get with a different Cauchy matrix has an obviously different size.
- For a given redundancy configuration, there is a large gap in the coding performance when using different *coding schemes*.[1]
- None of existing coding schemes performs best for all redundancy configurations (k, m, w).
- For a given redundancy configuration, it is with very low probability that one coding scheme chosen by rules of thumb can achieve the best coding performance.

Second, based on the preceding observations, we propose *CaCo*, an efficient *Cauchy Coding* approach for cloud storage systems. CaCo uses Cauchy matrix heuristics to produce a matrix set. Then, for each matrix in this set, CaCo uses XOR schedule heuristics to generate a series of schedules, and selects the shortest one from them. In this way, each matrix is attached with a locally optimal schedule. Finally, CaCo selects the globally optimal schedule from all the locally optimal schedules. This globally optimal schedule and its corresponding matrix will be stored and then used during data encoding and decoding. Incorporating all existing matrix and schedule heuristics, CaCo has the ability to identify an optimal coding scheme, within the capability of the current state of the art, for an arbitrary given redundancy configuration.

By leverage of CaCo's nature of ease to parallelize, we boost significantly the performance of the selection process with abundant computational resources in the cloud. First, for different Cauchy matrix heuristics, the calculations of every matrix and the corresponding schedules have no data dependency. Therefore, the computational tasks of CaCo are easy to partition and parallelize. Second, there are enough computational resources available in the cloud for the parallel deployment of CaCo. Therefore, we can perform CaCo in parallel and accelerate the selection of coding schemes.

Since data encoding only follows the schedules determined, we can execute CaCo in advance to get and store the optimal schedule before encoding. When writing data, we directly use the corresponding schedule for encoding. Then we store the generated erasure codes instead of writing multiple replication to achieve the data redundancy in the cloud storage system.

We implement CaCo in the Hadoop distributed file system (HDFS) and evaluate its performance by comparing with "Hadoop- EC" developed by Microsoft research [26]. Our experimental results indicate that CaCo can obtain an optimal coding scheme for an arbitrary redundancy configuration within an acceptable time. Furthermore, CaCo outperforms "Hadoop-EC" by 26.68-40.18 percent in the

1. To make our description brief, we use the term "coding scheme" to denote a combination of a matrix heuristic and a schedule heuristic in the rest of this paper.

encoding time and by 38.4-52.83 percent in the decoding time simultaneously.

1.3 Paper Organization

The rest of this paper is organized as follows. Section 2 briefly reviews research background and the related work. In Section 3, an overview of the CaCo approach is given. In Section 4, we describe how CaCo accelerates selection with parallel computing. In Section 5, we show an implementation of CaCo in a cloud storage. In Section 6, we present the evaluation results on a real system. Finally some conclusions and comments on future work are given in Section 7.

2 Background and Related Work

In this section, we first give a general overview of *Cauchy Reed-Solomon Coding*. Then, we provide a description in brief of the research and related work on generating Cauchy matrices and encoding with schedules. From these descriptions, we can make clearer the motivation of our work.

2.1 Cauchy Reed-Solomon Coding

Reed-Solomon codes [9] are based on a finite field, often called *Galois field*. When encoding data using RS codes, to implement a Galois filed arithmetic operation (addition or multiplication) requires many computations, so the performance is often unsatisfactory. CRS [11] codes modify RS codes and give two improvements. First, CRS codes use a Cauchy matrix instead of a Vandermonde matrix [27]. Second, CRS codes convert Galois field multiplications into XOR operations.

The key to CRS codes is construction of Cauchy matrices, and we can achieve that in the following way. Given a redundancy configuration (k, m, w) where $k + m \leq 2^w$, let $X = \{x_1, \ldots, x_m\}$, $Y = \{y_1, \ldots, y_k\}$, and $X \cap Y = \emptyset$, so that each x_i and y_j is a distinct element of $GF(2^w)$. Then we calculate the Cauchy matrix in element (i, j) using $1/(x_i + y_j)$ (the addition and division are defined over Galois field) [11], [22]. Since the elements of GF(2^w) are the integers from zero to $2^w - 1$, each element e can be represented by a w-bit column vector, $V(e)$, using the primitive polynomial over Galois Field [10]. Furthermore, each element e of GF(2^w) can be converted to a $(w \times w)$ binary matrix, $M(e)$, whose ith $(i = 1, \ldots, w)$ column is equal to the column vector $V(e2^{i-1})$ [28]. Thus according to the value of w, we can transform the Cauchy matrix into a $(mw \times kw)$ binary matrix, denoted as A.

We divide every data block X and erasure codes block B into w trips. In this way, when there exists "1" in every row of A, we can do XOR operations on the corresponding data in X, to obtain the elements of B. As Fig. 2 shows, the erasure codes require 11 XOR operations.

In a storage system using erasure codes, data are encoded to obtain data redundancy when data are written. Therefore, to improve the overall performance of a system, we should reduce the cost of erasure coding, i.e., the number of XOR operations. There are two encoding strategies to achieve this goal.

- *Encoding data using Cauchy matrices directly.* As Fig. 2 shows, the density of a Cauchy matrix decides the number of XOR operations and the encoding performance.
- *Encoding data using schedule.* The sequence of XOR operations in the schedule decides the encoding performance.

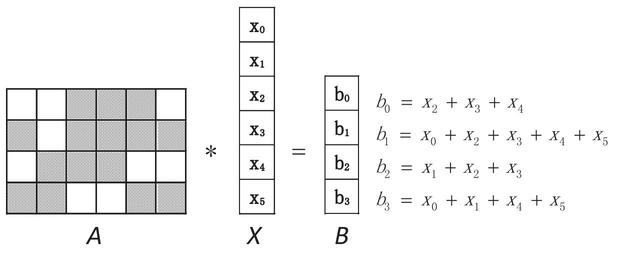


Fig. 2. Erasure coding using Cauchy Reed-Solomon codes.

2.2 Constructing Cauchy Matrices

While using a binary Cauchy matrix for CRS coding, the number of XORs depends on the number of ones in the matrix. So, in order to get better performance, the number of ones in the binary Cauchy matrix should be as few as possible. To discover an optimal matrix from numerous Cauchy matrices, the simplest way is to enumerate them. Given a redundancy configuration (k, m, w), we can construct $\binom{2^w}{k+m}\binom{k+m}{k}$ Cauchy matrices, each of which can be used for encoding. Therefore, the *enumeration* method is applicable only when the values of k, m, and w are small. Otherwise, the running time for enumerating all the matrices will be unacceptable, because the number of Cauchy matrices is a combinatorial problem. Some heuristics such as *Original* [11], *Optimizing Cauchy* [20] and *Cauchy Good* [29] can generate a good matrix which contains fewer ones for larger w, but it may not be the optimal one.

To construct a Cauchy matrix, called $GC(k, m, w)$, the *Optimizing Cauchy* heuristic first constructs a $(2^w \times 2^w)$ matrix, denoted as $ONES(w)$, whose element (i, j) represents the number of ones in the binary matrix $M(1/(i + j))$. Fig. 3a shows the matrix $ONES(3)$. Then we select two disjoint sets $X = \{x_1, \ldots, x_m\}$ and $Y = \{y_1, \ldots, y_k\}$ from $\{0, 1, \ldots, 2^w - 1\}$, as follows.

- When $k = m$ and k is a power of two, for $k > 2$, $GC(k, k, w)$ contains the elements of $GC(k/2, k/2, w)$, and $GC(2, 2, w)$ always contains the column set $Y = \{1, 2\}$. For example, $GC(4, 4, 3)$ is shown in Fig. 3b.
- When $k = m$ and k is not a power of two, we define $GC(k, k, w)$ by constructing $GC(k', k', w)$ first, where $k' > k$ and k' is a power of two. Then we delete redundant rows and columns alternately until we get a $k \times k$ matrix. As Fig. 3c shows, $GC(3, 3, 3)$ is defined by deleting one row and one column from $GC(4, 4, 3)$.
- When $k \neq m$, we define $GC(k, m, w)$ by constructing $GC(min(k, m), min(k, m), w)$ first, and then add some rows or columns correspondingly. As Fig. 3d shows, we construct $GC(4, 3, 3)$ by adding one column to $GC(3, 3, 3)$.

With the *Cauchy Good* heuristic, we first construct a Cauchy matrix called GM. Then divide (defined over Galois field)

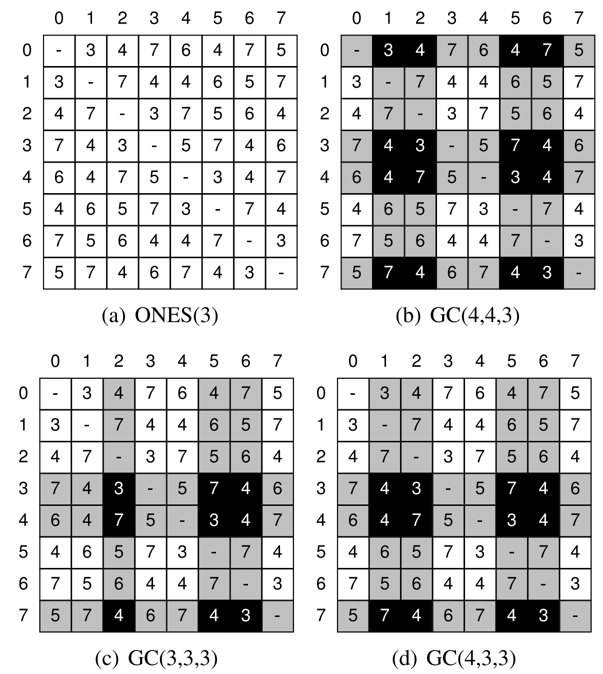


Fig. 3. Example of producing Cauchy matrices with the optimizing Cauchy heuristic [20].

every element of GM such as in column j by $GM_{0,j}$, such that GM is updated and the elements of row 0 are all "1". In the rest of the rows, such as row i, we count the number of ones, recorded as N. Then we divide the elements of row i by $GM_{i,j}$, and respectively count the number of ones, denoted as N_j ($j \in [0, k-1]$). Finally, select the minimum from $\{N, N_0, \ldots, N_{k-1}\}$ and do the operations that generate it. Thus we succeed in constructing a matrix using *Cauchy Good* heuristic.

The two heuristics above can produce a binary matrix which contains fewer ones; however, it may not be the optimal one in the numerous Cauchy matrices. The research on how to reduce the number of XOR operations in the process of erasure coding has revealed that the number of ones in a Cauchy matrix has lower bounds [4]. Therefore, only by reducing the density of the Cauchy matrix, it is difficult to improve the encoding performance greatly.

2.3 Encoding with Scheduling

While performing data encoding with a given Cauchy matrix, we can use intermediate results to reduce duplication of calculations. Therefore, strategically scheduling these XOR operations can bring into CPU savings, since fewer XORs are performed. Fig. 4 shows an example of simple scheduling. With the schedule S of the matrix A determined, it just requires six XOR operations to generate all erasure codes elements, less than 11 times when encoding using Cauchy matrices directly. The idea of scheduling makes the times of XOR operations break the limit of the number of ones in Cauchy matrix [30], thereby reducing the times of XOR operations and accelerating subsequent erasure coding.

There are some scheduling heuristics for a Cauchy matrix as follows.

- *CSHR* [31]. It uses the erasure codes previously generated to calculate subsequent erasure coding elements to avoid repeated computations.

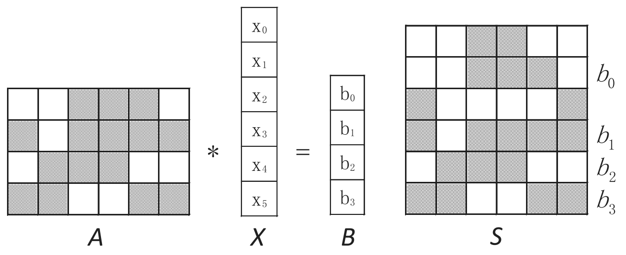


Fig. 4. Erasure coding with an optimal schedule.

- *Uber-CSHR* [32]. It is an improvement on CSHR. It uses not only erasure coding elements, but also the intermediate results generated to accelerate subsequent computations.
- *X-Sets* [32]. It identifies and makes use of the common XOR operations while generating erasure codes to avoid repeated computations. We calculate a certain common XOR operation and store the result, such that the subsequent computations can use it directly. There are many ways to select exactly which common XOR operation first into schedule, such as MW, MW-SS, MW-Matching, MW^2, Uber-XSet, and Subex.

It is still an open problem to derive a schedule from a Cauchy matrix that minimizes the number of XOR operations. Current algorithms for seeking schedule, such as CSHR, Uber-CSHR, and X-Sets, are heuristics, which could not guarantee to get the optimal solution. Huang et al. conjectured that deriving an optimal schedule based on common sums is an NP-Complete problem [23].

2.4 Observations and Motivation of Our Work

Cloud systems always use different redundancy configurations (i.e., (k, m, w)), depending on the desired balance between performance and fault tolerance. Through the preceding discussions and a number of experiments and analyses, we get some observations as follows.

- For different combinations of matrix and schedule, there is a large gap in the number of XOR operations.
- No one combination performs the best for all redundancy configurations.
- With the current state of the art, from the $\binom{2^w}{k+m}\binom{k+m}{k}$ Cauchy matrices, there is no method discovered to determine which one can produce the best schedule.
- Giving a Cauchy matrix, different schedules generated by various heuristics lead to a great disparity on coding performance.
- For a given redundancy configuration, it is with very low probability that one coding scheme chosen by rules of thumb performs the best.

In view of the problems above, it is necessary to discover an efficient coding approach for a cloud storage system. And this approach is desired to be able to identify the optimal coding scheme in the current state of the art, for an arbitrary given redundancy configuration.

3 CaCo Overview

Given a redundancy configuration (k, m, w), our goal is to find a Cauchy matrix, whose schedule is desired to be the

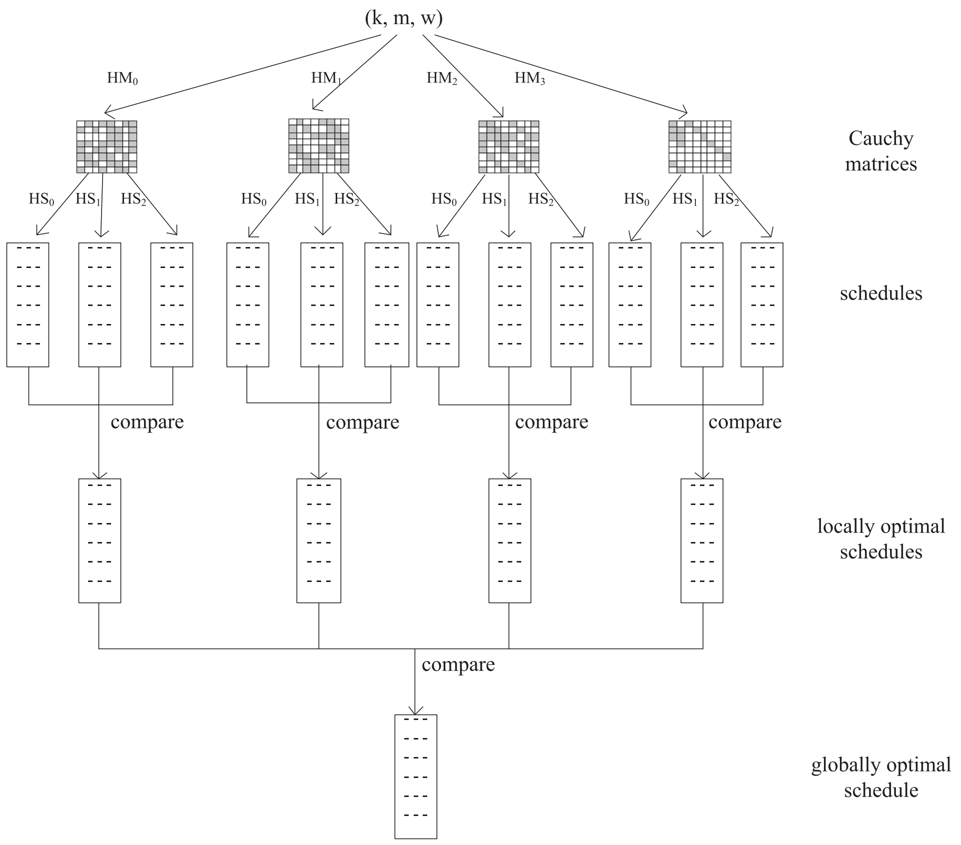


Fig. 5. A graphical representation of the overall framework.

shortest. In this paper, we propose CaCo, a coding approach that incorporates all existing matrix and schedule heuristics, and therefore is able to discover an optimal solution for data coding in a cloud storage system, within the capability of the current state of the art.

3.1 Selecting a Matched Coding Scheme

The key idea of CaCo is to select a Cauchy matrix, and one of its schedules whose size is desired to be minimized, for a redundancy configuration (k, m, w). As shown in Fig. 5, CaCo consists of four steps as follows.

- *Generating Cauchy matrices.* CaCo uses p different heuristics to produce a set of Cauchy matrices, denoted as $S_m = \{m_0, m_1, \ldots, m_{p-1}\}$.
- *Constructing schedules for each matrix.* For each matrix $m_i (0 \leq i < p)$ in the set S_m generated in the first step, CaCo uses q various heuristics to obtain a set of schedules, denoted as $S_{s,i} = \{s_{i,0}, s_{i,1}, \ldots, s_{i,q-1}\}$.
- *Selecting the locally optimal schedule for each matrix.* For each matrix $m_i (0 \leq i < p)$ in the set S_m, CaCo selects the shortest schedule from the set $S_{s,i}$, denoted as s_i. After this step, we get a set of matrices and their shortest schedules, denoted as $S = \{(m_0, s_0), (m_1, s_1), \ldots, (m_{p-1}, s_{p-1})\}$.
- *Selecting the globally optimal solution.* Finally, CaCo selects the shortest schedule $s_j (0 \leq j < p)$ from the set $\{s_0, s_1, \ldots, s_{p-1}\}$. Thus, (m_j, s_j) is the globally optimal solution to obtain the best encoding performance.

3.1.1 Generating Cauchy Matrices

Selecting the best one from Cauchy matrices using the enumeration method is a combinatorial problem. Given a redundancy configuration $(10, 6, 8)$, the magnitude of the matrices to be constructed can be up to 10^{29}, and it is unrealistic to enumerate them. We can not even determine which one of the matrices will produce better schedules. In the CaCo approach, we choose only a certain number of them for scheduling.

Considering the performance, we tend to choose the binary Cauchy matrices with fewer ones. We use some heuristics introduced in Section 2.2, such as *Original*, *Cauchy Good* and *Optimizing Cauchy*, to define a set of Cauchy matrices, namely, $S_m = \{m_0, m_1, \ldots, m_{p-1}\}$. To increase the diversity of the Cauchy matrices, we design a *Greedy* heuristic.

Our greedy heuristic to generate a light Cauchy matrix is described as follows.

- *Constructing the matrix ONES.* First, CaCo constructs a matrix named *ONES*, whose element (i, j) is defined as the number of ones contained in the binary matrix $M(1/(i + j))$.
- *Choosing the minimal element.* Second, CaCo chooses the minimal element from the matrix *ONES*. Supposing the element is (x_1, y_1), we initialize X to be $\{x_1\}$, and Y to be $\{y_1\}$.
- *Determining the set Y.* Besides the element (x_1, y_1), CaCo chooses the top $k - 1$ minimums from row x_1. Then, CaCo adds the corresponding $k - 1$ column numbers to Y, and we have $Y = \{y_1, y_2, \ldots, y_k\}$. Fig. 6c shows the results.

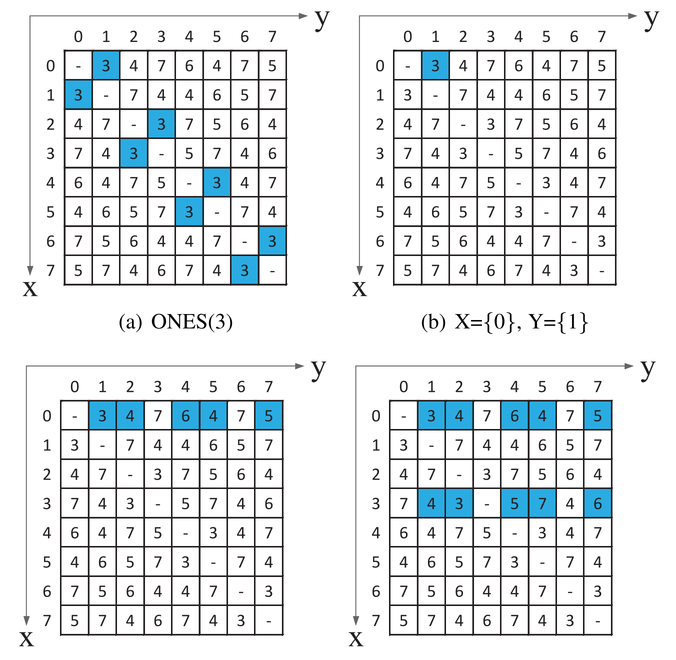

(a) ONES(3) (b) X={0}, Y={1} (c) X={0}, Y={1,2,4,5,7} (d) X={0,3}, Y={1,2,4,5,7}

Fig. 6. Example of generating Cauchy matrices using Greedy heuristic, with $k = 5$, $m = 2$, $w = 3$.

- *Determining the set X.* For each row r $(r \neq x_1)$ in the matrix *ONES*, CaCo calculates $C_r = \sum_{y \in Y}(r, y)$. Then, it chooses the top $m - 1$ minimums from the set $\{C_r, r \neq x_1\}$, adds the corresponding $m - 1$ row numbers to X, and finally gets the set $X = \{x_1, x_2, \ldots, x_m\}$, as shown in Fig. 6d.

3.1.2 Constructing Schedules for Each Matrix

For each matrix $m_i (0 \leq i < p)$ in the set S_m, we pass the parameters including k, m, w and pointer of the matrix to the function $do\_schedule(int\ k, int\ m, int\ w, int*\ \ matrix)$ to perform q heuristics in the function, such as Uber-CSHR, X-Sets, and so on. In this manner, we get a set of schedules, denoted as $S_{s,i} = \{s_{0,i}, s_{1,i}, \ldots, s_{q-1,i}\}$. If there appears a good heuristic for scheduling at a later date, we can add it to the function $do\_schedule$.

3.1.3 Selecting Locally Optimal Schedule for Each Matrix

For each matrix $m_i (0 \leq i < p)$ in the set S_m, we select the shortest schedule from the set $S_{s,i}$, denoted as s_i, so that we get a set of matrices and their shortest schedules, denoted as $S = \{(m_0, s_0), (m_1, s_1), \ldots, (m_{p-1}, s_{p-1})\}$.

For m_i in the set S_m, we can encode data in an order of XORs given by s_i. In this way, the times of XOR operations no longer have direct relationship with the density of the matrix. Therefore, scheduling excludes the influence of the lower limit of the number of ones in the matrix, so the performance improves significantly.

3.1.4 Selecting the Globally Optimal Solution

From the collection of combinations of Cauchy matrix and schedule, namely $\{(m_0, s_0), (m_1, s_1), \ldots, (m_{p-1}, s_{p-1})\}$, we choose the combinations with the shortest schedule. On this basis, for better performance, we tend to select the one containing the fewest ones in the matrix to be (m_{best}, s_{best}). Once selected, s_{best} can be used for encoding data.

3.1.5 Remark

The design of CaCo not only proves efficacious in the improvement of performance, but also has good scalability. First, CaCo incorporates all existing matrix and schedule heuristics so that it is able to discovery a comparable solution. Compared to the enumeration algorithm, CaCo has much lower complexity. Second, if there are new heuristics for generating Cauchy matrices or scheduling to be derived in the future, we can easily add them to CaCo. If some other matrices are later found better for scheduling, we can add them to the set S_m.

3.2 Encoding and Decoding with a Selected Scheme

Each redundancy configuration (k, m, w) is attached with a combination of (m_{best}, s_{best}), so we can select and store them once and for all. Every time we encode data, we just fetch the s_{best} from the memory at a minimum cost. Thus we do not need to spend much time on the computation of scheduling any more, and the efficiency of data encoding will be greatly improved.

If any m data blocks or coding blocks fail, we can recover the original data by decoding the remaining valid blocks. First, according to the chosen matrix m_{best} for data encoding and the specific situation of data corruption in the same group of $k + m$ blocks, we can generate a matrix m_d for decoding. Then we discover the optimal schedule for m_d using the same approach mentioned in Sections 3.1.2 and 3.1.3. We denote the schedule as s_d, and use it for decoding.

3.3 Discussions

In the framework, when we select the Cauchy matrix, we only consider the cost of data encoding, without considering the decoding cost. In fact, in most cases, the influence of decoding efficiency on the overall performance is negligible. Every time we write data into the storage, we need to encode data and obtain the erasure codes. On the other hand, only when an error occurs, we decode the erasure codes to recover the original data. In an actual storage system, the data error rarely occurs, so we should place more weight on the efficiency of encoding.

In some cases of high error rate of data, the failure of data occurs more frequently, and the frequency of decoding increases. To obtain higher accuracy, we take into account the cost of both encoding and decoding when we select a coding scheme. Assuming that each matrix generated or its inverse matrix has a score determined by the cost of scheduling and the number of ones in the matrix, respectively denoted as $Score_e$ and $Score_d$, we give each of them a weight to measure their importance, such as α and $1 - \alpha$. Then we can get a new evaluation formula for selecting matrix, namely

$$Score = \alpha * Score_e + (1 - \alpha) * Score_d.$$

The value of α depends on the error rate of data in an actual system. If the error rate is low enough, we can set the value of α to be 1.

In Section 3.1, while selecting the optimal coding scheme, we only consider the cost of data encoding, without

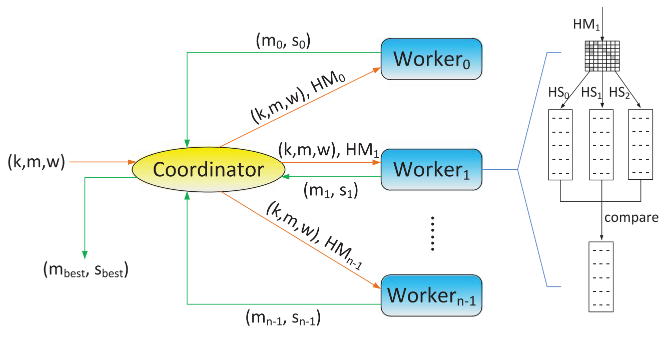


Fig. 7. A schematic diagram of the framework executing in distributed environments.

considering the decoding cost. To obtain higher accuracy, we take into account the cost of both encoding and decoding. For each matrix generated or its inverse matrix, we define a score by the cost of scheduling and the number of ones in the binary Cauchy matrix, respectively denoted as $Score_e$ and $Score_d$. We give each of them a weight to measure their importance, i.e., α and $1 - \alpha$. Then we can get a new evaluation formula for selecting matrix, namely

$$Score = \alpha * Score_e + (1 - \alpha) * Score_d.$$

The value of α depends on the error rate of data in an actual system.

Every time we write data into the storage, we need to encode data and obtain the erasure codes. Only when an error occurs, we decode the erasure codes to recover the original data. In an actual storage system, the data error rarely occurs, so we should place more weight on the efficiency of encoding. If the error rate is low enough, we can set α as 1. On the other hand, in some cases of high error rate, data decoding occurs more frequently, so the influence of decoding efficiency on the overall performance will be greater, and the value of α should be decreased.

4 Accelerating Selection with Parallel Computing

In order to accelerate the selection of coding schemes, we perform CaCo in parallel by distributing the computational tasks to individual nodes in a cluster. For different heuristics, the calculations of Cauchy matrices and schedules are completely independent. Therefore, it would be feasible and effective to distribute the tasks of generating matrices and schedules to different processors.

Fig. 7 summarizes the mechanism of performing the framework over a distributed system.

1) *The Coordinator dispatches computational tasks.* The Coordinator receives the parameters passed in, such as the redundancy configuration (k, m, w), and the number of Workers. Then the Coordinator sends some parameters to Workers, such as (k, m, w), *HM* (heuristic for generating a Cauchy matrix), and the other parameters required. In this procedure, we should concern about load balance among Workers.
2) *Workers execute heuristics for generating matrices and schedules.* Workers receive the message from the Coordinator, and obtain a Cauchy matrix using the heuristic *HM*. Then Workers schedule for the matrix with multiple heuristics, and select the locally optimal schedule respectively, and finally send the selected combination of (m_i, s_i) to the Coordinator.
3) *The Coordinator assembles results from Workers and selects the optimum.* The Coordinator collects the combinations that Workers send back, and from them, by comparison of the size of the schedules and the density of the binary Cauchy matrices, selects the optimal combination to be (m_{best}, s_{best}).

In most cases, CaCo is applied to the cloud storage system with multiple nodes in the cluster. So there are readily available machines that we can use for the deployment of distributed environments.

5 Implementation in HDFS

This section mainly describes how we implement CaCo's selection of the optimal coding scheme for a given redundancy configuration (k, m, w), and how we use the selected coding scheme for data encoding in the Hadoop distributed file system.

5.1 Parallel Deployment of CaCo in MPI

We deploy CaCo in MPICH with the release version 3.0.4 [33], so that the selection of the best coding scheme can be accelerated in parallel over the distributed system. MPICH is a high-performance and widely portable implementation of the Message Passing Interface (MPI) standard.

Our distributed environment is built up with multiple nodes, containing one node as *Coordinator* for dispatching computational tasks and assembling results, and other nodes as *Workers* for generating matrices and schedules.

For every redundancy configuration (k, m, w), the Coordinator generates and sends different identifiers of matrix heuristics to Workers via message passing function *MPI_Send*. According to the given heuristic identifier, each Worker generates a Cauchy matrix and then selects a locally best schedule. Implementation of *Optimizing Cauchy* heuristic references PLANK's work about CRS Coding [20], and the *Original* matrix is defined by BLOMER [11], and *Cauchy Good* heuristic is a part of the Jerasure library [29], and the schedule heuristics reference the open source programs of Uber-CSHR and X-Sets [32], and we design and implement the *Greedy* heuristic to increase the diversity of Cauchy matrices.

Once a Worker completes its computational task, it immediately reports to the Coordinator via message passing function *MPI_Send* and requests another task. In this way, our cluster achieves load balancing and makes the computational resources well function.

While all the computational tasks are completed, the Coordinator assembles all the generated combinations and discovers the global optimum. Finally, we store the selected combination for subsequent data coding.

5.2 Employing CaCo in HDFS

To achieve data redundancy, HDFS requires triple replication. In CaCo, we use erasure codes instead of triple replication of data to guarantee the fault tolerance of the system. To implement CaCo, we should modify the architecture of

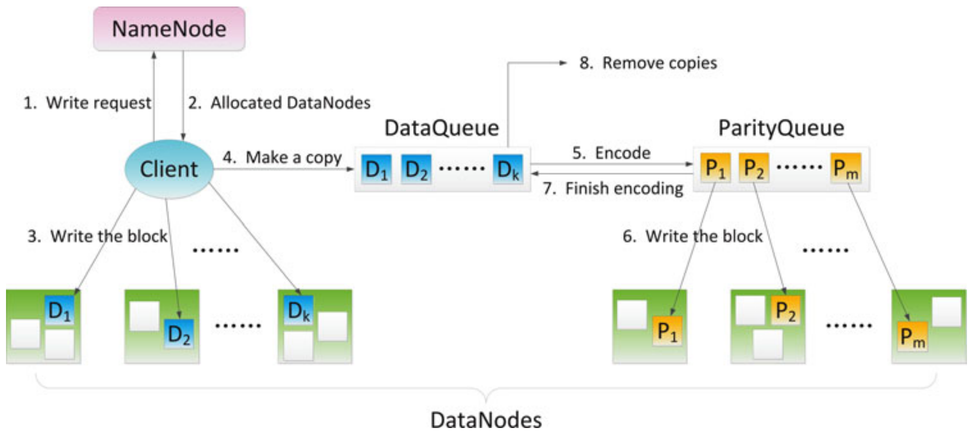


Fig. 8. Data flow of the write operation with CaCo.

HDFS to certain extent. For example, the illustration of write operation with CaCo is shown in Fig. 8, and we can conclude the procedure to several steps as follows.

Algorithm 1: Write Operation with CaCo

```
1 The Client sends a write request to the NameNode.
2 The NameNode allocates some DataNodes to the Client.
3 Write the data blocks into DataNodes.
4 Make a copy of data and put it into DataQueue.
5 Encode data with the schedule selected by CaCo.
6 Write the coding blocks into DataNodes.
7 Data encoding finishes.
8 Remove the copies of data from DataQueue.
```

With erasure coding, we reduce the number of data copies from three to one. So, when a block is written, we only allocate one DataNode to the Client.

After writing the data blocks to the allocated DataNodes, we store a copy of data into DataQueue to be encoded. While there are k data blocks ready in DataQueue, we encode them with the schedule selected by CaCo. After encoding, we get m coding blocks, and add them to ParityQueue. Then, we allocate some DataNodes for the coding blocks. Note that those k data blocks and m coding blocks should be stored on different data nodes. Otherwise, the failure of one node may lead to multiple faults in the same group of $n = k + m$ blocks.

Finally, we remove the copies of data that have been encoded, and thus the write operation with CaCo finishes.

In the read operation with CaCo, if there is not data corruption in the requested blocks, the Client directly retrieves the data. When failures of some nodes occur, we need to decode the remaining valid blocks to rebuild the original data, using the optimal schedule generated by CaCo. Then, we supply the rebuilt data to the Client's read request.

6 Performance Evaluation

This section mainly presents results of a comprehensive experimental evaluation comparing CaCo with the existing data coding approaches. This performance study analyzes their performance in terms of data encoding time and data decoding time.

6.1 Evaluation Methodology

To evaluate CaCo's performance in every aspect, we do a set of experiments and analyses. First, while generating the combinations of matrix and schedule in every heuristic, we count the size of the generated schedules and the number of ones in the Cauchy matrices, to prove that CaCo can identify the optimal combination. Second, we run the framework with various redundancy configurations and accelerate it in MPI, to testify the feasibility of parallelizing in CaCo. Finally, we implement CaCo in HDFS and evaluate its performance of data encoding and decoding by comparing with Hadoop-EC.

The test-bed used in these experiments consists of two nodes, as described bellow. The first node, installed with Linux kernel 2.6.35-22-generic, is equipped with Intel Xeon E5-2620 2.00 GHz six-core processor and 32 GB of memory. The other node, installed with Linux kernel 2.6.32-33-generic, is equipped with Intel Xeon E5606 2.13 GHz quad-core processor and 8 GB of memory. And they both use EX4 file system.

6.2 Effectiveness of the Generated Schedules

The purpose of our first set of experiments is to quantitatively characterize the advantages of CaCo through a comparison with all other coding schemes. One of them is a combination of a Cauchy matrix heuristic and an XOR schedule heuristic. The four Cauchy matrix heuristics used in the experiments are *Cauchy Good*, *Optimizing Cauchy*, *Original*, and *Greedy*. The eight XOR schedule heuristics used in the experiments are *Uber*, *MW*, *MW-SS*, *MW-Smallest-Sum*, *MW-SQ*, *Uber-XSet*, *MW-Matching*, and *Subex*. There are 32 combinations of Cauchy matrix heuristics and XOR schedule heuristics. In other words, we compare the coding performance between the CaCo approach with 32 other coding schemes. The number of XOR operations required by an erasure code has a direct relationship to the performance of encoding or decoding. In this set of experiments, we use the number of XOR operations to represent the coding performance.

First, we explore the coding performance of CaCo and 32 other coding schemes with the redundancy configuration of $(k, m, w) = (7, 3, 4)$. As shown in Table 1, the Cauchy matrix generated with the *Cauchy Good* heuristic has 130 ones. We schedule this matrix with each of the eight XOR schedule

TABLE 1
The Effect of Different Cauchy Matrix and XOR Schedule for the Redundancy Configuration $(k, m, w) = (7, 3, 4)$

Schedule	Cauchy Good		Optimizing Cauchy		Original		Greedy	
	ONEs	XORs	ONEs	XORs	ONEs	XORs	ONEs	XORs
Uber		86		88		**101**	150	89
MW		74		77		88	145	77
MW-SS		74		77		86	145	74
MW-Smallest-Sum	130	74	143	77	187	88	145	77
MW-SQ		72		75		87	146	74
Uber-XSet		72		76		86	145	72
MW-Matching		**70**		75		85	146	74
Subex		73		77		86	146	72

heuristics. The number of XORs per schedule varies from 70 to 86. So, the size of the local optimal schedule is 70. The Cauchy matrix generated with the *Optimizing Cauchy* heuristic has 143 ones. We schedule this matrix with each of the eight XOR schedule heuristics. The number of XORs per schedule varies from 75 to 88. So, the size of the local optimal schedule is 75. The Cauchy matrix generated with the *Original* heuristic has 187 ones. We schedule this matrix with each of the eight XOR schedule heuristics. The number of XORs per schedule varies from 85 to 101. So, the size of the local optimal schedule is 85. With the *Greedy* heuristic, the size of each of the Cauchy matrices fluctuate between 145 and 150. The number of XORs per schedule varies from 72 to 89. So, the size of the local optimal schedule is 72. Finally, CaCo has a Cauchy matrix with the size of 130 ones, and an XOR schedule with the size of 70.

Second, we examine the coding performance of CaCo with the redundancy configuration of $(k, m, w) = (8, 4, 4)$. As shown in Table 2, as to the Cauchy matrix generated with the *Cauchy Good* heuristic, the number of XORs per schedule varies from 111 to 126. So, the size of the local optimal schedule is 111. For the Cauchy matrix generated with the *Optimizing Cauchy* heuristic, the number of XORs per schedule varies from 115 to 144. So, the size of the local optimal schedule is 115. For the Cauchy matrix generated with the *Original* heuristic, the number of XORs per schedule varies from 126 to 143. So, the size of the local optimal schedule is 126. With the *Greedy* heuristic, the number of XORs per schedule varies from 109 to 136. So, the size of the local optimal schedule is 109. Finally, CaCo has a Cauchy matrix with the size of 232 ones, and an XOR schedule with the size of 109.

Third, we examine the coding performance of CaCo with the redundancy configuration of $(k, m, w) = (10, 5, 5)$. As shown in Table 3, as to the Cauchy matrix generated with the *Cauchy Good* heuristic, the number of XORs per schedule varies from 241 to 324. So, the size of the local optimal schedule is 241. For the Cauchy matrix generated with the *Optimizing Cauchy* heuristic, the number of XORs per schedule varies from 246 to 341. So, the size of the local optimal schedule is 246. For the Cauchy matrix generated with the *Original* heuristic, the number of XORs per schedule varies from 267 to 345. So, the size of the local optimal schedule is 267. With the *Greedy* heuristic, the number of XORs per schedule varies from 253 to 333. So, the size of the local optimal schedule is 253. Finally, CaCo has a Cauchy matrix with the size of 495 ones, and an XOR schedule with the size of 241.

For completeness, we also conducted some additional experiments with eleven other redundancy configurations (k, m, w). Table 4 summarizes a list of the best combinations of a Cauchy matrix heuristic and an XOR schedule heuristic, selected by CaCo, for different redundancy configurations.

Some conclusions that one may draw from the experimental results are as follows.

- Given a Cauchy matrix, changing the XOR schedule heuristic affects the coding performance significantly. This observation is consistent with the results from Plank et al. [25].
- Given a redundancy configuration (k, m, w), the locally optimal schedule with a different Cauchy matrix heuristic has an obviously different size. To the best of our knowledge, this observation is not shown in any other literature.

TABLE 2
The Effect of Different Cauchy Matrix and XOR Schedule for the Redundancy Configuration $(k, m, w) = (8, 4, 4)$

Schedule	Cauchy Good		Optimizing Cauchy		Original		Greedy	
	ONEs	XORs	ONEs	XORs	ONEs	XORs	ONEs	XORs
Uber		126		**144**		143	236	136
MW		114		115		130	232	113
MW-SS		111		121		134	232	110
MW-Smallest-Sum	213	114	232	115	292	130	232	113
MW-SQ		115		119		126	232	**109**
Uber-XSet		112		116		129	232	113
MW-Matching		112		117		131	232	113
Subex		114		121		134	236	115

TABLE 3
The Effect of Different Cauchy Matrix and XOR Schedule for the Redundancy Configuration $(k, m, w) = (10, 5, 5)$

Schedule	Cauchy Good		Optimizing Cauchy		Original		Greedy	
	ONEs	XORs	ONEs	XORs	ONEs	XORs	ONEs	XORs
Uber	495	324	504	341	642	$\overline{\mathbf{345}}$	539	333
MW		252		252		272	533	260
MW-SS		252		254		278	533	261
MW-Smallest-Sum		252		252		272	533	260
MW-SQ		$\underline{\mathbf{241}}$		246		267	533	253
Uber-XSet		252		262		275	539	259
MW-Matching		242		254		272	533	257
Subex		243		254		275	544	261

- For a given redundancy configuration, there is a large gap in the coding performance when using different combinations of a Cauchy matrix heuristic and an XOR schedule heuristic.
- None of existing coding schemes performs best for all redundancy configurations (k, m, w).
- CaCo always performs best for all redundancy configurations (k, m, w).
- When setting an existing coding scheme by rules of thumb, there is a probability of 3.13 percent to reach the coding performance of CaCo.

In order to measure the performance gap between the coding scheme generated by CaCo and that by the enumeration approach, we have enumerated all Cauchy matrices, and performed all the scheduling heuristics, used in the CaCo approach, for each matrix. Then, we compare the coding scheme selected by CaCo with the optimal coding scheme derived by the enumeration approach.

For the redundancy configuration $(k, m, w) = (7, 3, 4)$, the enumeration method provides a schedule of size 66, while CaCo finds out a schedule of size 70. For the redundancy configuration $(8, 4, 4)$, the enumeration method provides a schedule of size 101, while CaCo finds out a schedule of size 109. Although the enumeration method can find out a bit better coding scheme than CaCo, the running time is unacceptable for a relatively large redundancy configuration. For example, given a redundancy configuration $(10, 5, 5)$, the magnitude of the matrices to be constructed can be up to 10^{12}, and it is unrealistic to enumerate them. We can conclude that compared with the enumeration approach, CaCo can identify a good enough coding scheme with much lower complexity.

TABLE 4
The Best Combination of Cauchy Matrix and XOR Schedule for Different Redundancy Configurations

k, m, w	matrix	schedule	XORs
4, 4, 3	Cauchy Good	Uber-XSet	31
4, 3, 4	Optimizing Cauchy	MW	36
5, 3, 4	Optimizing Cauchy	MW	49
6, 3, 4	Cauchy Good	Uber-XSet	57
7, 3, 4	Cauchy Good	MW-Matching	70
13, 3, 4	Cauchy Good	MW-SQ	137
4, 4, 4	Optimizing Cauchy	MW-SQ	42
5, 4, 4	Optimizing Cauchy	MW-SQ	60
6, 4, 4	Cauchy Good	MW-SQ	76
7, 4, 4	Greedy	MW	94
8, 4, 4	Greedy	MW-SQ	109
9, 4, 4	Cauchy Good	Subex	123
10, 4, 4	Cauchy Good	Uber-XSet	134
11, 4, 4	Cauchy Good	MW-SQ	153
5, 5, 5	Optimizing Cauchy	Uber-XSet	113
10, 5, 5	Cauchy Good	MW-SQ	241

6.3 Running Time of CaCo

In the preceding set of tests, we focus primarily on the effectiveness of the generated schedules. In this set of experiments, we examine the running time of CaCo.

We vary the redundancy configuration (k, m, w) by stabilizing m and w on 4 and increasing k from 6 to 12. For a given redundancy configuration, we run CaCo for three times and collect the average running time. In Fig. 9, we compare the running time of CaCo when one or four Workers are used.

First, as the figure shows, the running time of CaCo takes on an upward trend as k increases along the x-axis. When one Worker is used to compute all the selection tasks, the running time increases from 43.26 to 529.68 s. When four Workers are used to share the computing burden, the running time increases from 16.46 to 169.24 s. In other words, the longest running time is 2.82 minutes. Second, for a given redundancy configuration, the running time of CaCo decreases significantly when four Workers are used. The speedup ratio varies between 2.63 and 3.15.

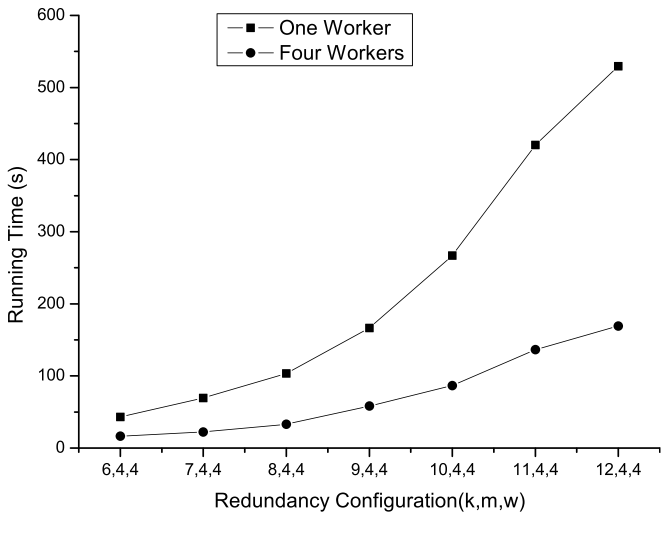


Fig. 9. Running time of CaCo with different redundancy configurations.

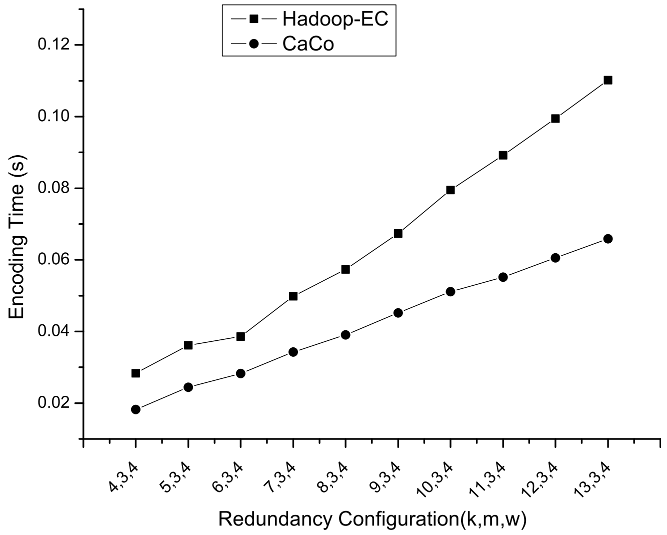


Fig. 10. Performance comparison of data encoding between CaCo and Hadoop-EC.

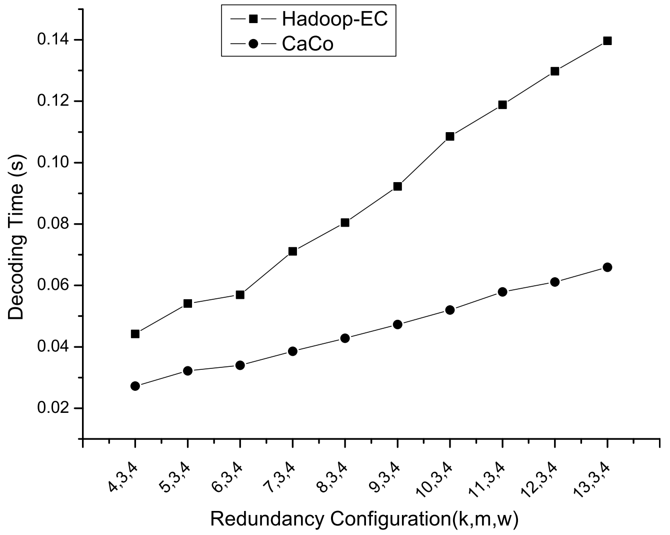


Fig. 11. Performance comparison of data decoding between CaCo and Hadoop-EC when one data disk fails.

With those redundancy configurations used in this set of experiments, the running time of CaCo are absolutely acceptable. Although the computation complexity of CaCo rises along with the increase of k, m, or w in the redundancy configuration (k, m, w), we can use better parallelism to reduce the running time of CaCo for two reasons. First, the problem of selecting the best coding scheme is easy to be parallelized. Second, there are enough computational resources in a modern cloud storage system available for running CaCo. On the other hand, we view running time as secondary to the effectiveness of the schedules produced. The reason is that real implementations will very likely pre-compute and store the matrices and schedules rather than compute them on the fly when encoding or decoding data.

6.4 Coding Performance in a Cloud System

Finally, we evaluate the performance of performing data encoding and decoding in the cloud storage system with the Cauchy matrices and XOR schedules generated by CaCo. In order to quantitatively characterize the advantages of CaCo, we make a comparison between CaCo and Hadoop-EC [26]. Hadoop-EC is a Hadoop library with erasure codec functionalities, developed by Microsoft research. Hadoop-EC uses the “Cauchy Good” heuristic in the Jerasure library [29] for erasure calculations.

Data encoding. We vary the redundancy configuration (k, m, w) by stabilizing m on 3 and w on 4, and increasing k from 4 to 13. For a given redundancy configuration, we encode data to produce parity data with CaCo and Hadoop-EC, and collect the encoding time. In Fig. 10, we compare CaCo and Hadoop-EC in the encoding time.

First, as the figure shows, the encoding times of CaCo and Hadoop-EC take on an upward trend as k increases along the x-axis. When Hadoop-EC is used in the cloud system, the encoding time increases from 28.35 to 110.19 ms. When CaCo is used in the cloud system, the encoding time increases from 18.24 to 65.92 ms. The reason behind this phenomenon is that a larger k means more data blocks, and further more XOR operations, involved in a data encoding operation.

Second, for a given redundancy configuration, the encoding time of CaCo is significantly smaller than that of Hadoop-EC. The percentage of improvement varies between 26.68 and 40.18 percent.

Data decoding. We vary the redundancy configuration (k, m, w) by stabilizing m on 3 and w on 4, and increasing k from 4 to 13. For a given redundancy configuration, we decode data to reproduce lost data with CaCo and Hadoop-EC, and collect the decoding time. With these redundancy configurations, simultaneous failures of at most three disks can be tolerated. Computation complexities are different when recovering from different situations of data corruption in the same group of $k + m$ blocks.

We first examine the performance of CaCo when recovering from a single drive failure. This is the most common failure event and thus should be an interesting case study. Fig. 11 plots the decoding times using CaCo and Hadoop-EC when recovering from one data disk failure. First, the decoding times of CaCo and Hadoop-EC take on an upward trend as k increases along the x-axis. When Hadoop-EC is used in the cloud system, the decoding time increases from 44.23 ms to 139.71 ms. When CaCo is used in the cloud system, the decoding time increases from 27.24 to 65.9 ms. The reason behind this phenomenon is that a larger k means more data blocks, and further more XOR operations, involved in a data decoding operation. Second, for a given redundancy configuration, the decoding time of CaCo is significantly smaller than that of Hadoop-EC. The percentage of improvement varies between 38.4 and 52.83 percent.

We then compare CaCo and Hadoop-EC in the decoding time when recovering from failures of three data disks. As shown in Fig. 12, the decoding times of CaCo and Hadoop-EC take on an upward trend as k increases along the x-axis. When Hadoop-EC is used in the cloud system, the decoding time increases from 51.22 to 165.03 ms. When CaCo is used in the cloud system, the decoding time increases from 39.44 to 112.59 ms. Furthermore, for a given redundancy configuration, the decoding time of CaCo is significantly smaller than that of Hadoop-EC.

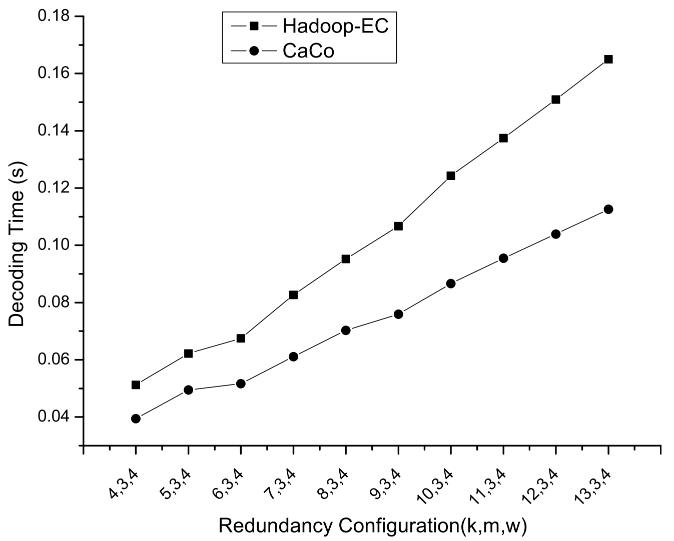


Fig. 12. Performance comparison of data decoding between CaCo and Hadoop-EC when three data disks fail.

The percentage of improvement varies between 20.56 and 31.78 percent.

In Fig. 10, we compare CaCo and Hadoop-EC in the decoding time when recovering from failures of two data disks and one coding disk. First, the decoding times of CaCo and Hadoop-EC take on an upward trend as k increases along the x-axis. When Hadoop-EC is used in the cloud system, the decoding time increases from 33.68 to 121.04 ms. When CaCo is used in the cloud system, the decoding time increases from 22.07 to 70.03 ms. Second, for a given redundancy configuration, the decoding time of CaCo is significantly smaller than that of Hadoop-EC. The percentage of improvement varies between 30.38 and 42.14 percent.

7 Conclusions and Future Work

Cloud storage systems always use different redundancy configurations (i.e., (k, m, w)), depending on the desired balance between performance and fault tolerance. For different combinations of matrices and schedules, there is a large gap in the number of XOR operations, and no single combination performs best for all redundancy configurations.

In this paper, we propose CaCo, a new approach that incorporates all existing matrix and schedule heuristics, and thus is able to identify an optimal coding scheme within the capability of the current state of the art for a given redundancy configuration. The selection process of CaCo has an acceptable complexity and can be accelerated by parallel computing. It should also be noticed that the selection process is once for all. The experimental results demonstrate that CaCo outperforms the "Hadoop-EC" approach by 26.68-40.18 percent in encoding time and by 38.4-52.83 percent in decoding time simultaneously.

Finally, we readily acknowledge that reducing XORs is not the only way to improve the performance of an erasure code. Other code properties, like the amount of data required for recovery and degraded reads [34], [35], [36], may limit performance more than the CPU overhead. We look forward to addressing these challenges in the future.

Acknowledgments

The authors are grateful to Xinkui Qu for providing helpful comments and assistance with our experimentation. This work was supported by the National Natural Science Foundation of China under Grants 61170008, 61272055, 61433008, and U1435216, the National Grand Fundamental Research 973 Program of China under Grant No. 2014CB340402, and the National High Technology Research and Development Program of China under Grant 2013AA01A210.

References

[1] L. N. Bairavasundaram, A. C. Arpaci-Dusseau, R. H. Arpaci-Dusseau, G. R. Goodson, and B. Schroeder, "An analysis of data corruption in the storage stack," *Trans. Storage*, vol. 4, pp. 8:1–8:28, Nov. 2008.

[2] J. L. Hafner, V. Deenadhayalan, W. Belluomini, and K. Rao, "Undetected disk errors in raid arrays," *IBM J. Res. Develop.*, vol. 52, pp. 413–425, Jul. 2008.

[3] D. Ford, F. Labelle, F. I. Popovici, M. Stokely, V.-A. Truong, L. Barroso, C. Grimes, and S. Quinlan, "Availability in globally distributed storage systems," in *Proc. 9th USENIX Symp. Oper. Syst. Des. Implementation*, 2010, pp. 61–74.

[4] B. Calder, J. Wang, A. Ogus, N. Nilakantan, A. Skjolsvold, S. McKelvie, Y. Xu, S. Srivastav, J. Wu, H. Simitci, J. Haridas, C. Uddaraju, H. Khatri, A. Edwards, V. Bedekar, S. Mainali, R. Abbasi, A. Agarwal, M. F. u. Haq, M. I. u. Haq, D. Bhardwaj, S. Dayanand, A. Adusumilli, M. McNett, S. Sankaran, K. Manivannan, and L. Rigas, "Windows Azure storage: A highly available cloud storage service with strong consistency," in *Proc. 23rd ACM Symp. Oper. Syst. Principles*, New York, NY, USA, 2011, pp. 143–157.

[5] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao, "Oceanstore: An architecture for global-scale persistent storage," *SIGPLAN Notices*, vol. 35, pp. 190–201, Nov. 2000.

[6] B. Fan, W. Tantisiriroj, L. Xiao, and G. Gibson, "Diskreduce: Raid for data-intensive scalable computing," in *Proc. 4th Annu. Workshop Petascale Data Storage*, New York, NY, USA, 2009, pp. 6–10.

[7] K. D. Bowers, A. Juels, and A. Oprea, "Hail: A high-availability and integrity layer for cloud storage," in *Proc. 16th ACM Conf. Comput. Commun. Security*, New York, NY, USA, 2009, pp. 187–198.

[8] G. Xu, F. Wang, H. Zhang, and J. Li, "Redundant data composition of peers in p2p streaming systems using Cauchy Reed-Solomon codes," in *Proc. 6th Int. Conf. Fuzzy Syst. Knowl. Discovery-Volume 2*, Piscataway, NJ, USA, 2009, pp. 499–503.

[9] J. S. Plank, "A tutorial on Reed-Solomon coding for fault-tolerance in raid-like systems," *Softw. Pract. Experience*, vol. 27, pp. 995–1012, Sept. 1997.

[10] I. S. Reed and G. Solomon, "Polynomial codes over certain finite fields," *J. Soc. Ind. Appl. Math.*, vol. 8, no. 2, pp. 300–304, 1960.

[11] J. Blömer, M. Kalfane, M. Karpinski, R. Karp, M. Luby, and D. Zuckerman, "An xor-based erasure-resilient coding scheme," International Computer Science Institute, Berkeley, California, USA, 1995.

[12] J. S. Plank, J. Luo, C. D. Schuman, L. Xu, and Z. Wilcox-O'Hearn, "A performance evaluation and examination of open-source erasure coding libraries for storage," in *Proc. 7th Conf. File Storage Technol.*, Berkeley, CA, USA, 2009, pp. 253–265.

[13] M. Blaum, J. Brady, J. Bruck, and J. Menon, "Evenodd: An efficient scheme for tolerating double disk failures in raid architectures," *IEEE Trans. Comput.*, vol. 44, no. 2, pp. 192–202, Feb. 1995.

[14] P. Corbett, B. English, A. Goel, T. Grcanac, S. Kleiman, J. Leong, and S. Sankar, "Row-diagonal parity for double disk failure correction," in *Proc. 3rd USENIX Conf. File Storage Technol.*, Berkeley, CA, USA, 2004, pp. 1–1.

[15] L. Xu and J. Bruck, "X-code: MDS array codes with optimal encoding," *IEEE Trans. Inform. Theory*, vol. 45, no. 1, pp. 272–276, Jan. 1999.

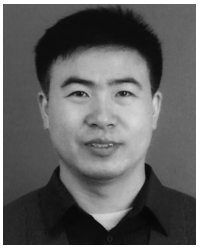
[16] C. Jin, H. Jiang, D. Feng, and L. Tian, "P-code: A new raid-6 code with optimal properties," in *Proc. 23rd Int. Conf. Supercomput.*, New York, NY, USA, 2009, pp. 360–369.

[17] C. Huang and L. Xu, "Star: An efficient coding scheme for correcting triple storage node failures," in *Proc. 4th Conf. USENIX Conf. File Storage Technol.-Volume 4*, Berkeley, CA, USA, 2005, pp. 15–15.
[18] S. Lin, G. Wang, D. Stones, J. Liu, and X. Liu, "T-code: 3-erasure longest lowest-density mds codes," *IEEE J. Sel. Areas Commun.*, vol. 28, no. 2, pp. 289–296, Feb. 2010.
[19] J. L. Hafner, "Weaver codes: Highly fault tolerant erasure codes for storage systems," in *Proc. 4th Conf. USENIX Conf. File Storage Technol.-Volume 4*, Berkeley, CA, USA, 2005, pp. 16–16.
[20] J. S. Plank and L. Xu, "Optimizing Cauchy Reed-Solomon codes for fault-tolerant network storage applications," in *Proc. 5th IEEE Int. Symp. Netw. Comput. Appl.*, Washington, DC, USA, 2006, pp. 173–180.
[21] M. Blaum and R. M. Roth, "On lowest density mds codes," *IEEE Trans. Inform. Theory*, vol. 45, no. 1, pp. 46–59, Jan. 1999.
[22] J. S. Plank, "XOR's, lower bounds and MDS codes for storage," Univ. of Tennessee, TN, USA, Tech. Rep. CS-11-672, May 2011.
[23] C. Huang, J. Li, and M. Chen, "On optimizing xor-based codes for fault-tolerant storage applications," in *Proc. IEEE Inform. Theory Workshop*, 2007, pp. 218–223.
[24] J. Edmonds, "Paths, trees, and flowers," *Can. J. Math.*, vol. 17, no. 3, pp. 449–467, 1965.
[25] J. S. Plank, C. D. Schuman, and B. D. Robison, "Heuristics for optimizing matrix-based erasure codes for fault-tolerant storage systems," in *Proc. 42Nd Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw.*, Washington, DC, USA, 2012, pp. 1–12.
[26] Z. Zhang, A. Deshpande, X. Ma, E. Thereska, and D. Narayanan, "Does erasure coding have a role to play in my data center?" Tech. Rep. MSR-TR-2010-52, Microsoft Research, Cambridge, England, May 2010.
[27] J. S. Plank and Y. Ding, "Note: Correction to the 1997 tutorial on Reed-Solomon coding," *Softw. Pract. Exp.*, vol. 35, pp. 189–194, Feb. 2005.
[28] X. Li, Q. Zheng, H. Qian, D. Zheng, and J. Li, "Toward optimizing cauchy matrix for Cauchy Reed-Solomon code," *Comm. Lett.*, vol. 13, pp. 603–605, Aug. 2009.
[29] J. S. Plank, S. Simmerman, and C. D. Schuman, "Jerasure: A library in C/C++ facilitating erasure coding for storage applications-Version 1.2," Univ. of Tennessee, TN, USA, Tech. Rep. CS-08-627, Aug. 2008.
[30] J. Luo, L. Xu, and J. S. Plank, "An efficient xor-scheduling algorithm for erasure codes encoding," in *Proc. IEEE/IFIP Int. Conf. Dependable Syst. Netw.*, 2009, pp. 504–513.
[31] J. L. Hafner, V. Deenadhayalan, K. K. Rao, and J. A. Tomlin, "Matrix methods for lost data reconstruction in erasure codes," in *Proc. 4th Conf. USENIX Conf. File Storage Technol.-Volume 4*, Berkeley, CA, USA, 2005, pp. 14–14.
[32] J. S. Plank, "Uber-CSHR and X-Sets: C++ programs for optimizing matrix-based erasure codes for fault-tolerant storage systems," Univ. of Tennessee, TN, USA, Tech. Rep. CS-10-665, Dec. 2010.
[33] W. Gropp, E. Lusk, N. Doss, and A. Skjellum, "A high-performance, portable implementation of the MPI message passing interface standard," *Parallel Comput.*, vol. 22, pp. 789–828, Sep. 1996.
[34] O. Khan, R. Burns, J. Plank, W. Pierce, and C. Huang, "Rethinking erasure codes for cloud file systems: Minimizing I/O for recovery and degraded reads," in *Proc. 10TH USENIX Conf. File Storage Technol.*, 2012, pp. 251–264.
[35] K. M. Greenan, X. Li, and J. J. Wylie, "Flat xor-based erasure codes in storage systems: Constructions, efficient recovery, and tradeoffs," in *Proc. IEEE 26th Symp. Mass Storage Syst. Technol.*, Washington, DC, USA, 2010, pp. 1–14.
[36] L. Xiang, Y. Xu, J. C. Lui, and Q. Chang, "Optimal recovery of single disk failure in RDP code storage systems," *SIGMETRICS Perform. Eval. Rev.*, vol. 38, pp. 119–130, Jun. 2010.

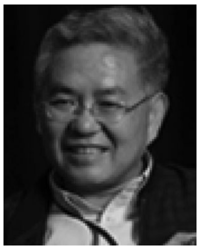
Guangyan Zhang received the bachelor's and master's degrees in computer science from Jilin University, in 2000 and 2003, respectively, and the doctor's degree in computer science and technology from Tsinghua University in 2008. He is now an associate professor in the Department of Computer Science and Technology at Tsinghua University. His current research interests include big data computing, network storage, and distributed systems. He is a professional member of the ACM.

Guiyong Wu received the bachelor's degree in computer science and technology from Tsinghua University in 2014 and is working towards the master degree in the Department of Computer Science and Technology at Tsinghua University. His current research interests include network storage and distributed systems.

Shupeng Wang received the master's and doctor's degree from the Harbin Institute of Technology, China, in 2004 and 2007, respectively. He is an associate research fellow in the Institute of Information Engineering, Chinese Academy of Sciences (CAS), China. His research interests include big data management and analytics, network storage.



Jiwu Shu received the PhD degree in computer science from Nanjing University in 1998, and finished the postdoctoral position research at Tsinghua University in 2000. He is a professor in the Department of Computer Science and Technology at Tsinghua University. His current research interests include storage security and reliability, non-volatile memory based storage systems, and parallel and distributed computing. Since then, he has been teaching at Tsinghua University. He is a member of the IEEE.



Weimin Zheng received the master's degree from Tsinghua University in 1982. He is now a professor in the Department of Computer Science and Technology at Tsinghua University. His research covers distributed computing, compiler techniques, and network storage.

Keqin Li is a SUNY distinguished professor of computer science. His current research interests include parallel computing and high-performance computing, distributed computing, energy-efficient computing and communication, heterogeneous computing systems, cloud computing, big data computing, CPU-GPU hybrid and cooperative computing, multicore computing, storage and file systems, wireless communication networks, sensor networks, peer-to-peer file sharing systems, mobile computing, service computing, Internet of things, and cyber-physical systems. He has published more than 350 journal articles, book chapters, and refereed conference papers, and has received several best paper awards. He is currently or has served on the editorial boards of *IEEE Transactions on Parallel and Distributed Systems*, *IEEE Transactions on Computers*, *IEEE Transactions on Cloud Computing*, *Journal of Parallel and Distributed Computing*. He is a fellow of the IEEE.