Erasure-Code-推荐码-1-原理篇

张若源
中国科学技术大学
计算机科学与技术
本科生

前言

在分布式存储领域，一直被视为一个不可或缺的关键基础技术 Erasure-Code（EC）。它可以通过冗余数据来实现数据的可靠性和高性能。

在 Ceph 的 EC 推荐文档中提到了，现代分布式存储系统下数据丢失不可避免，而通过冗余数据来实现数据的可靠性，而 EC 则是其中一种比传统多副本方案更优秀的方案。

在 Ceph 的 EC 推荐文档中提到了，现代分布式存储系统下数据丢失不可避免，而通过冗余数据来实现数据的可靠性，而 EC 则是其中一种比传统多副本方案更优秀的方案。

- 第一篇：原理篇
- 第二篇：实现原理篇
- 第三篇：性能测试篇等等

问题

分析完系统的需求，一切都是有一个原因的。

解决了需要解决的问题之后，那接下来的问题就是一般性、实现难、可维护性的三个要素。

- 性能优化实现：实现高性能的编码算法是不可缺少的。不同的 Galois Field 的算法实现是不一样的。
- 实现难：EC 的实现本身并不复杂，但是要实现一个高性能的实现，需要对硬件、编译器、操作系统、内存管理、多线程、网络等等都有深入的理解。
- 可维护性：EC 的实现本身并不复杂，但是要实现一个高性能的实现，需要对硬件、编译器、操作系统、内存管理、多线程、网络等等都有深入的理解。

此外，随着数据量的增长，随着业务的不断迭代和发展，系统要求能够实现数据量的增长和业务需求的变化，这种情况下，EC 的实现本身也要保证足够的可扩展性，才不会出现过时不能用的情况。

而现实情况是随着摩尔定律的失效，现在的一些硬件设备每年性能提升，但是，EC 的实现本身并不能跟上这种发展。

EC的基本原理

EC的实现分为三个步骤：编码、EC 的实现本身的编码、解码 Galois Field 的实现。以及要解决的问题和可能要解决的问题。

第 0 天: 初等GF(2^8)代数基础

编码

EC 有两个参数，编码是指将这两个参数，以及一个原始数据作为输入，输出一个编码数据的过程。

这个编码数据，以及这个原始数据，都是以下原始数据 $d_1, d_2, d_3, \ldots, d_k$ 的线性组合。

- 原始数据：

$$d_1, d_2, d_3, \ldots, d_k$$

- 编码数据：

$$c_1, c_2, c_3, \ldots, c_m$$

- 编码矩阵：

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1k} \\ a_{21} & a_{22} & \cdots & a_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mk} \end{bmatrix}$$

- 编码公式：

$$\begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_m \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1k} \\ a_{21} & a_{22} & \cdots & a_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mk} \end{bmatrix} \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_k \end{bmatrix}$$

- 编码数据：

$$c_1, c_2, c_3, \ldots, c_m$$

- 原始数据：

$$d_1, d_2, d_3, \ldots, d_k$$

- 编码矩阵：

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1k} \\ a_{21} & a_{22} & \cdots & a_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mk} \end{bmatrix}$$

- 编码公式：

$$\begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_m \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1k} \\ a_{21} & a_{22} & \cdots & a_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mk} \end{bmatrix} \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_k \end{bmatrix}$$

解码

EC 的实现本身的编码，一定要保证一个不可缺的关键基础特性 MDS（Maximum Distance Separable）。它可保证任意 k 个编码数据，可以解码出原始数据。

这意味着，任意 k 个编码数据，可以作为输入，通过解码公式，可以解码出原始数据 $d_1, d_2, d_3, \ldots, d_k$。

- 编码数据：

$$c_1, c_2, c_3, \ldots, c_m$$

- 原始数据：

$$d_1, d_2, d_3, \ldots, d_k$$

- 编码矩阵：

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1k} \\ a_{21} & a_{22} & \cdots & a_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mk} \end{bmatrix}$$

- 编码公式：

$$\begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_m \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1k} \\ a_{21} & a_{22} & \cdots & a_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mk} \end{bmatrix} \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_k \end{bmatrix}$$

- 编码数据：

$$c_1, c_2, c_3, \ldots, c_m$$

- 原始数据：

$$d_1, d_2, d_3, \ldots, d_k$$

- 编码矩阵：

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1k} \\ a_{21} & a_{22} & \cdots & a_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mk} \end{bmatrix}$$

- 编码公式：

$$\begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_m \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1k} \\ a_{21} & a_{22} & \cdots & a_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mk} \end{bmatrix} \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_k \end{bmatrix}$$

第 1 天: 初等GF(2^8)代数实现

编码

现在我们来小试牛刀实现编码器。这可能会稍显原始和低效，未来我们要来一步步的优化和提升。

EC 有两个参数：k，m。对应公式中的编码矩阵的行和列。

EC 的实现本身的编码：我们的编码矩阵可以简单的定义一个二维数组，对应公式中的 a_{ij}。

$$a_{ij} = a[i][j]$$

现在，我们来编写编码公式的实现，简单的理解，这一段应该很容易。

```
编码数据的计算公式可以简单的理解为：

    [c1]   [a11 a12 ... a1k]   [d1]
    [c2]   [a21 a22 ... a2k]   [d2]
    [..] = [...  ...  ...  ]   [..]
    [cm]   [am1 am2 ... amk]   [dk]

    其中：
    [c1, c2, ..., cm]：这个数组
    [d1, d2, ..., dk]：这个数组
    [a11, a12, ..., amk]：这个二维数组

    编码数据的计算公式可以简单的理解为：
    c[i] = a[i][1] * d[1] + a[i][2] * d[2] + ... + a[i][k] * d[k]
```

对应的代码实现如下，这个代码应该很容易理解，这个代码的实现，对应公式中的 c_i。

```
// c[i] = a[i][1] * d[1] + a[i][2] * d[2] + ... + a[i][k] * d[k]
// 这个代码应该很容易理解，这个代码的实现，对应公式中的 c[i]
// 编码数据的计算公式可以简单的理解为：
```