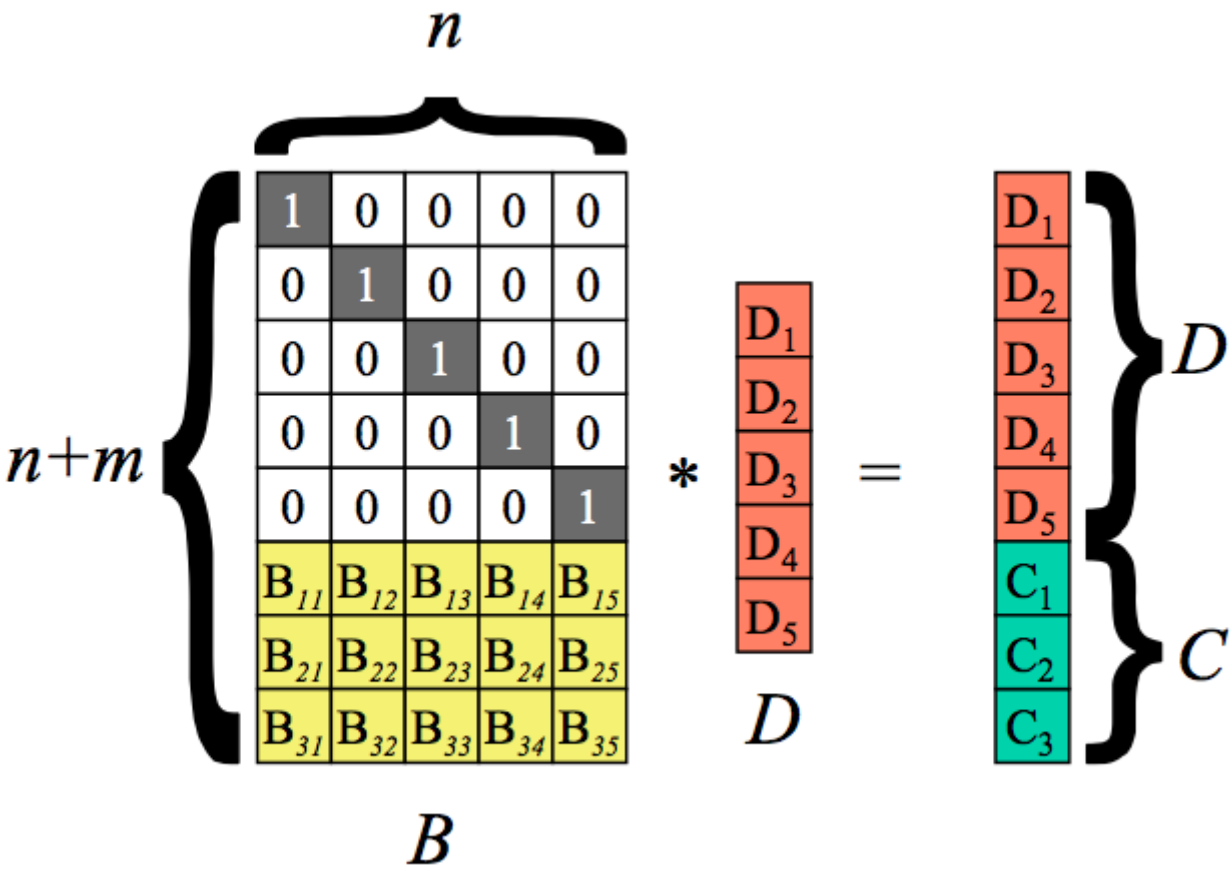


纠删码Erasure Coding （分布式存储系统）

副本策略和纠删码是存储领域常见的两种数据冗余技术。相比于副本策略，纠删码具有更高的磁盘利用率。 Reed-Solomon码是一种常见的纠删码。
Erasure Code是一种编码技术，它可以将n份原始数据，增加m份数据，并能通过n+m份中的任意n份数据，还原为原始数据。即如果有任意小于等于m份的数据失效，仍然能通过剩下的数据还原出来。 纠删码技术在分布式存储 系统中的应用主要有三类，阵列纠删码（Array Code：RAID5、RAID6等）、RS(Reed-Solomon)里德-所罗门类纠删码和LDPC(LowDensity Parity Check Code)低密度奇偶校验纠删码。 LDPC码目前主要用于通信、视频和音频编码等领域。

多副本策略即将数据存储多个副本（一般是三副本，比如HDFS），当某个副本丢失时，可以通过其他副本复制回来。三副本的磁盘利用率为1/3。

纠删码技术主要是通过纠删码算法将原始的数据进行编码得到冗余，并将数据和冗余一并存储起来，以达到容错的目的。其基本思想是将n块原始的数据元素通过一定的计算，得到m块冗余元素（校验块）。对于这n+m块的元素，当其中任意的m块元素出错（包括原始数据和冗余数据）时，均可以通过对应的重构算法恢复出原来的n块数据。生成校验的过程被成为编码（encoding），恢复丢失数据块的过程被称为解码（decoding）。磁盘利用率为n/(n+m)。基于纠删码的方法与多副本方法相比具有冗余度低、磁盘利用率高等优点。

两种冗余技术对比如下表：

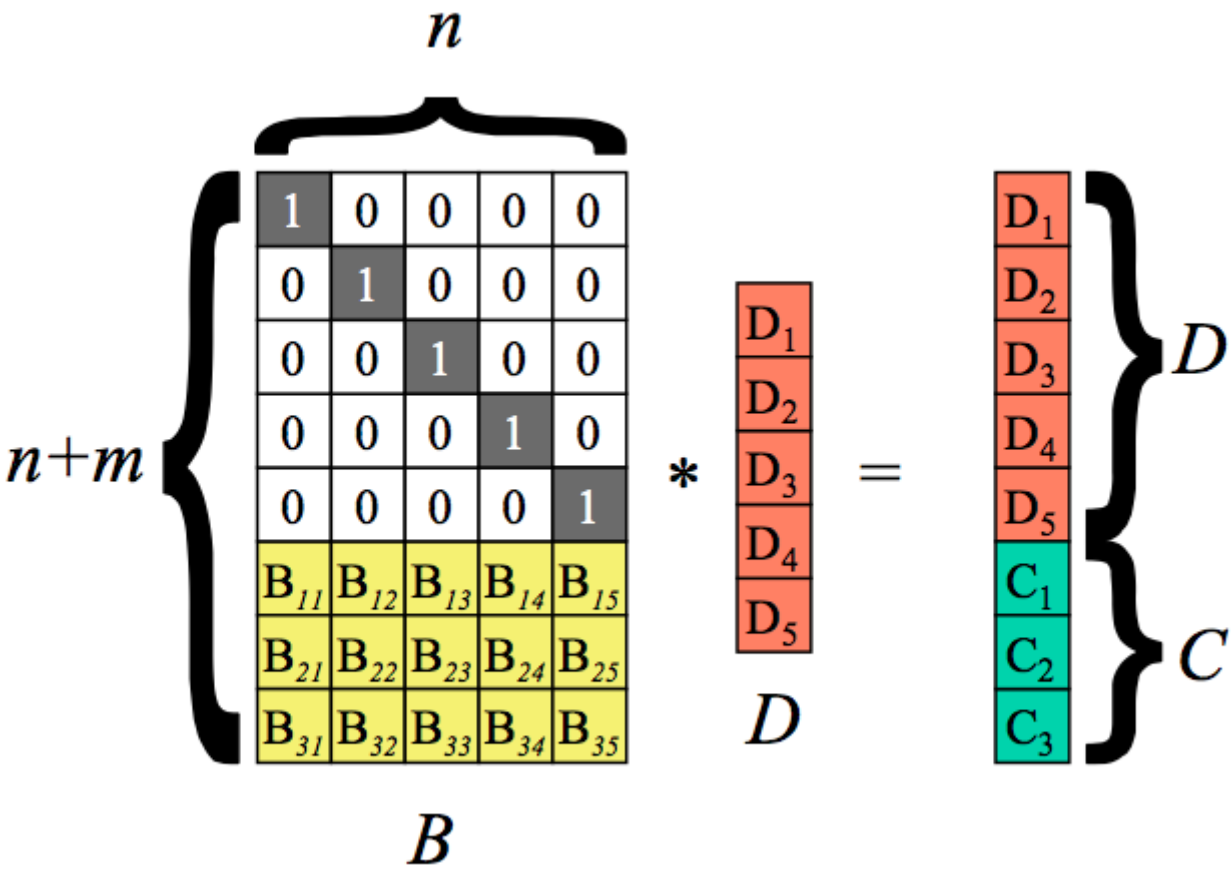
两种技术	磁盘利用率	计算开销	网络消耗	恢复效率
多副本(3副本)	1/3	几乎没有	较低	较高
纠删码(n+m)	n/(n+m)	高	较高	较低

Reed-Solomon(RS)码

Reed-Solomon（RS）码是存储系统较为常用的一种纠删码，它有两个参数n和m，记为RS(n,m)。n代表原始数据块个数。m代表校验块个数。接下来介绍RS码的原理。

RS码原理

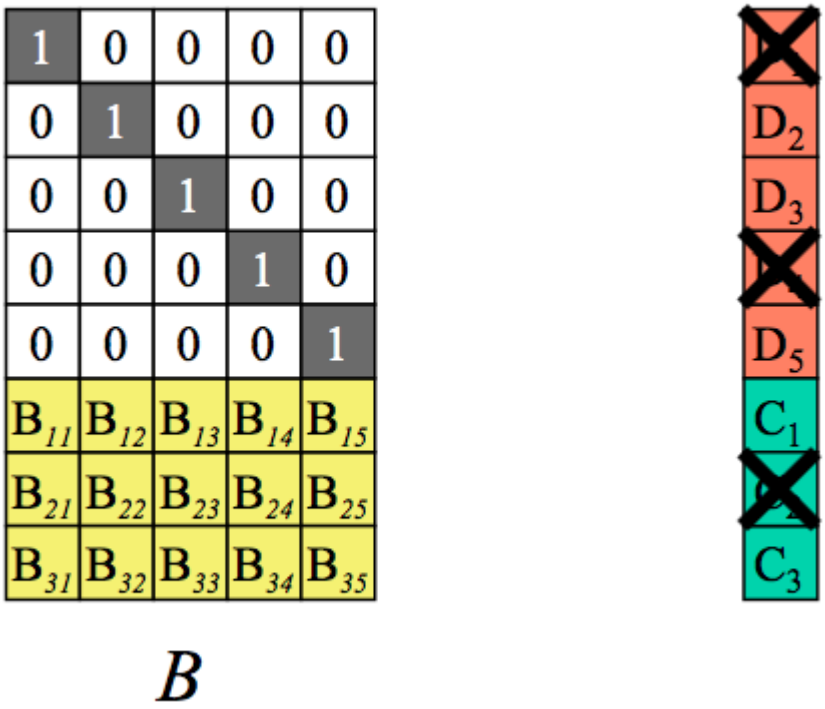
以n=5，m=3为例。即5个原始数据块，乘上一个(n+m)*n的矩阵，然后得出一个(n+m)*1的矩阵。根据矩阵特点可以得知结果矩阵中前面5个值与原来的5个数据块的值相等，而最后3个则是计算出来的校验块。



image

以上过程为编码过程。D是原始数据块，得到的C为校验块。

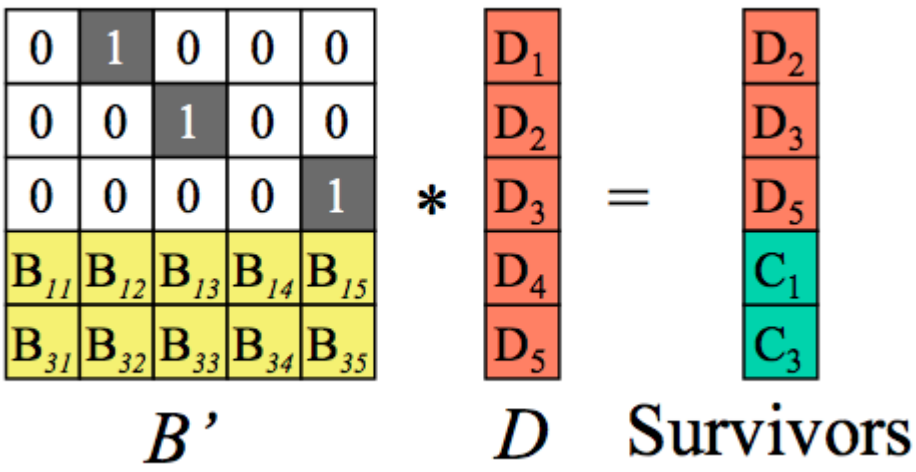
假设丢失了m块数据。如下：



image

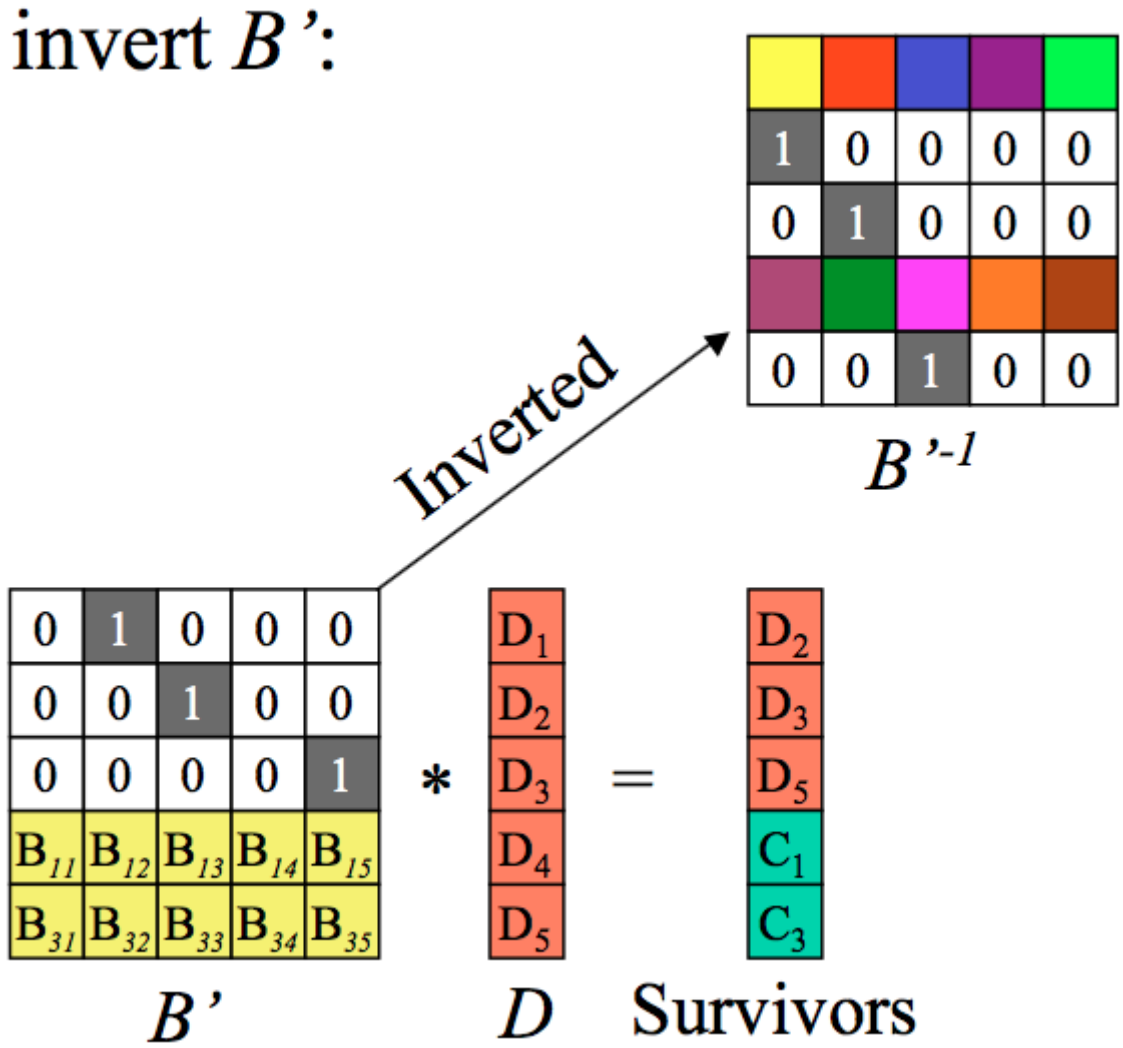
那我们如何从剩余的n个数据块（注意，这里剩余的n块可能包含几个原始数据块+几个校验块）恢复出来原始的n个数据块呢，就需要通过下面的decoding（解码）过程来实现。

第一步：从编码矩阵中删去丢失数据块和丢失编码块对应行。 将删掉m个块的(n+m)*1个矩阵变形为n*1矩阵，同时B矩阵也需要删掉对应的m个行得出一个B'的变形矩阵，这个B'就是n*n矩阵。如下：假设D1、D4、C2丢失，我们得到如下B’矩阵及等式。



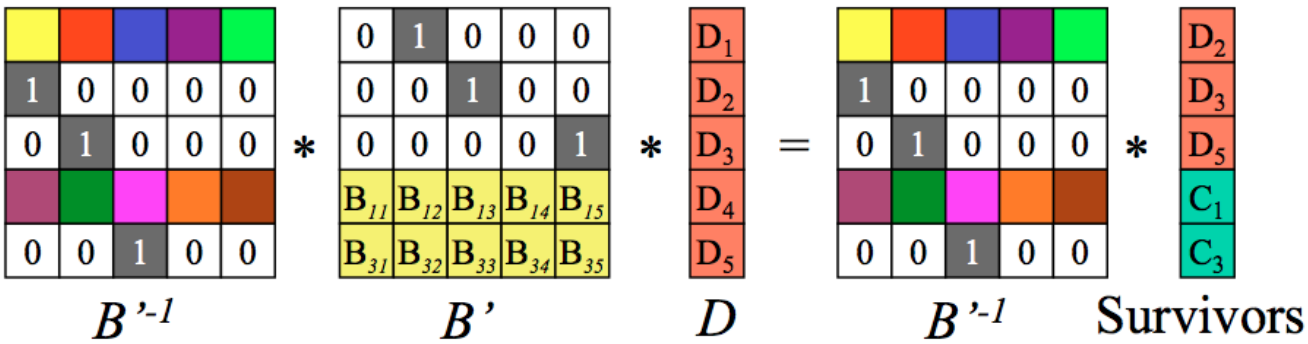
image

第二步：求出B’的逆矩阵。



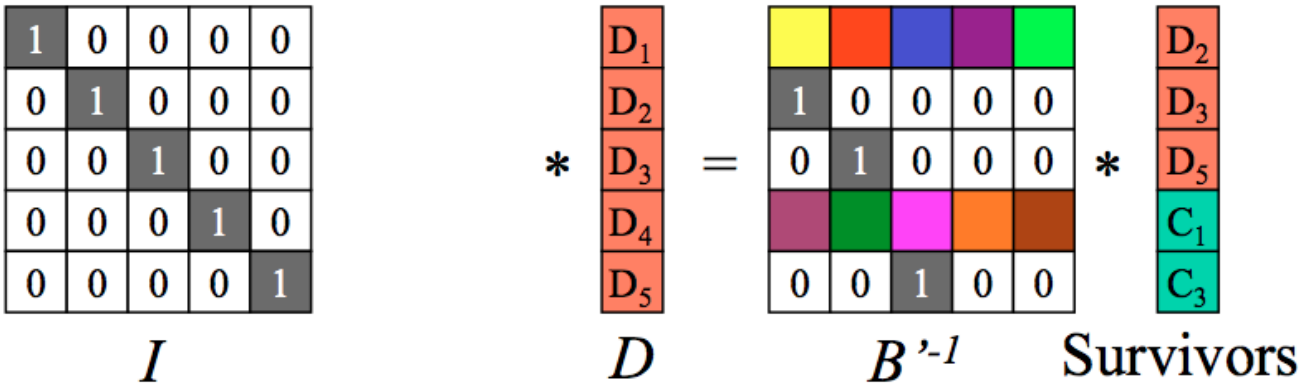
image

第三步：等式两边分别乘上B'的逆矩阵。



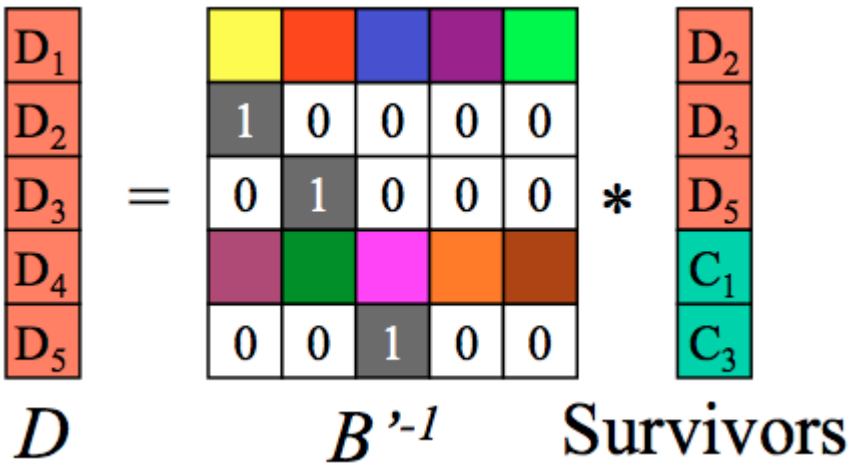
image

B'和它的逆矩阵相乘得到单位矩阵I，如下：



image

左边只剩下原始数据矩阵D：



image

至此完成解码过程。

注：图中黄色部分为范德蒙矩阵。至于如何生成B矩阵，以及如何求B'的逆矩阵，请查看其他相关文献，这里不再赘述。

RS的特点

- 低冗余度，高磁盘利用率。
- 数据恢复代价高。 丢失数据块或者编码块时， RS需要读取n个数据块和校验块才能恢复数据， 数据恢复效率也在一定程度上制约了RS的可靠性。
- 数据更新代价高。 数据更新相当于重新编码， 代价很高， 因此常常针对只读数据，或者冷数据。

工程实践中，一般对于热数据还是会使用多副本策略来冗余，冷数据使用纠删码。

纠删码引擎

ISA-L

纠删码作为 ISA-L 库所提供的功能之一，其性能应该是目前业界最佳。需要注意的是 Intel 采用的性能测试方法与学术界常用的方式略有出路，其将数据块与冗余块的尺寸之和除以耗时作为速度，而一般的方法是不包含冗余块的。另外，ISA-L 未对 vandermonde 矩阵做特殊处理，而是直接拼接单位矩阵作为其编码矩阵，因此在某些参数下会出现编码矩阵线性相关的问题。好在 ISA-L 提供了cauchy 矩阵作为第二方案。

ISA-L 之所以速度快，一方面是由于 Intel 谙熟汇编优化之道，其次是因为它将整体矩阵运算搬迁到汇编中进行。但这导致了汇编代码的急剧膨胀，令人望而生畏。

Jerasure2.0

不同于 ISA-L 直接使用汇编代码，Jerasure2.0 使用 C 语言封装后的指令，这样代码更加的友好。另外 Jerasure2.0 不仅仅支持 GF(2^8) 有限域的计算，其还可以进行 GF(2^4) - GF(2^128) 之间的有限域。并且除了 RS 码，还提供了 Cauchy Reed-Solomon code （CRS 码）等其他编码方法的支持。它在工业应用之外，其学术价值也非常高。目前其是使用最为广泛的编码库之一。目前 Jerasure2.0 并不支持 AVX 加速，尽管如此，不过在仅使用 SSE 的情况下，Jerasure2.0 依然提供了非常高的性能表现。不过主要作者之一 James S. Plank 教授转了研究方向，另外一位作者 Greenan 博士早已加入工业界。因此后续的维护将是个比较大的问题。

klauspost 的 ReedSolomon

klauspost 利用 Golang 的汇编支持，友好地使用了 SIMD 技术，此款引擎的 SIMD 加速部分是目前我看到的实现中最为简洁的，矩阵运算的部分逻辑被移到了外层高级语言中，加上 Golang 自带的汇编支持，使得汇编代码阅读起来更佳的友好。不过 Go 并没有集成所有指令，部分指令不得不利用 YASM 等汇编编译器将指令编译成字节序列写入汇编文件中。一方面导致了指令的完全不可读，另外一方面这部分代码的语法风格是 Intel 而非 Golang 汇编的 AT&T 风格，平添了迷惑。这款引擎比较明显的缺陷有两点：1.对于较大的数据块，编码速度会有巨大的下滑；2.修复速度明显慢于编码速度。

Hadoop 3.0 开始支持 纠删码(EC)存储。
Swift现在支持纠删码(EC)存储策略类型。
这样部署人员、以极少的RAW容量达到极高的可用性，如同在副本存储中一样。然而，EC需要更多的CPU和网络资源，所以并不适合所有应用场景。EC非常适合在一个独立的区域内极少访问的、大容量数据。

Swift纠删码的实现对于用户是透明的。对于副本存储和纠删码存储的类型，在API上没有任何区别。

为了支持纠删码，Swift现在需要依赖PyECLib和liberasurecode。liberasurecode是一个可插件式的库，允许在你选择的库中实现EC算法。

纠删码优/劣势

优势

纠删码技术作为一门数据保护技术，自然有许多的优势，首先可以解决的就是目前分布式系统，云计算中采用副本来防止数据的丢失。副本机制确实可以解决数据丢失的问题，但是翻倍的数据存储空间也必然要被消耗，这一点却是非常致命的。EC技术的运用就可以直接解决这个问题。

劣势

EC技术的优势确实明显，但是他的使用也是需要一些代价的，一旦数据需要恢复，他会造成2大资源的消耗:

```
1、网络带宽的消耗，因为数据恢复需要去读其他的数据块和校验块
2、进行编码，解码计算需要消耗CPU资源
```

就是既耗网络又耗CPU

总结

最好的选择是用于冷数据集群，有下面2点原因可以支持这种选择

```
1、冷数据集群往往有大量的长期没有被访问的数据，体量确实很大，采用EC技术，可以大大减少副本数
2、冷数据集群基本稳定，耗资源量少，所以一旦进行数据恢复，将不会对集群造成大的影响
```

出于上述2种原因，冷数据集群无非是一个很好的选择。

扩展阅读:

HDFS EC：将纠删码技术融入HDFS

Erasure Code - EC纠删码原理

原文链接:
纠删码（Erasure Code）浅析
https://blog.csdn.net/runningtortoises/article/details/51567417