doi:10.3969/j.issn.1000-436x.2014.04.019

基于再生码的云存储系统——Ustor

柳青，冯丹，李白
（华中科技大学 计算机科学与技术学院，湖北 武汉 430074）

摘 要：当前常使用多个云共同存储数据，以保证用户数据可靠性。为减少存储成本和修复带宽，提出了一种使用功能性修复再生码（FRC）将数据编码为多个数据块，并分布于不同的云中的方法。该方法减少了多个云中单个云发生数据丢失时需要从网络上传输的数据量，并减少了修复成本，已成功地应用于所构建的云存储系统 Ustor 中。实验表明：与不编码比较，冗余编码给系统增加了 5%~10%的响应时间开销，但可保障节点失效；FRC 码编、解码和修复速度与里德-所罗门（Reed-Solomon 或 RS）码基本相当，256 MB 大小文件编码时间差距在 0.5 s 以内；FRC 码修复时与传统的 RS 码相比减少了 25%以上需要下载的数据量。

关键词：云存储；再生码；纠删码；修复带宽
中图分类号：TP393 **文献标识码**：A **文章编号**：1000-436X(2014)04-0166-09

Ustor: cloud storage system based on regenerating codes

LIU Qing, FENG Dan, LI Bai
(School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China)

Abstract: To ensure reliability of user's data, multiple clouds are responsible for storing the data. For purpose of minimizing storage cost and repair bandwidth, a method of utilizing functional regenerating codes(FRC) to encode data to several blocks, which are further distributed to different clouds, was proposed. In this way, amount of data downloaded through the network when data loss happens in a single cloud is cut down, and repair cost is cut down as well. This method was successfully applied in a cloud storage system Ustor. Observations can be drawn from the experimental results: first, erasure coding guarantees data reliability, increasing 5%~10% response time compared with non-coding; second, speed of FRC encode, decode and repair are a little slower than Reed-Solomon(RS) code, and the time gap of encoding 256MB by two codes is less than 0.5 seconds; third, compared with traditional RS codes, FRC reduces more than 25% data amount needed to download during a repair in the experiment.

Key words: cloud storage; regenerating codes; erasure codes; repair bandwidth

1 引言

随着分布式存储系统集群的扩张和云存储的广泛应用，冗余编码逐渐应用于分布式存储系统中保证数据的可靠性，减少了存储容量和存储成本[1]。常用的纠错码有 Reed-Solomon 码（RS 码[2] ）。(n, k)-RS 码是一种最大距离可分（MDS, maximum distance seperable）码，也就是所有数据存储在 n 个节点中，其中，任意 k 个节点的数据可以恢复出原始数据[3]。(n, k)-RS 保证了最多可失效(n−k)节点而原数据不丢失。而分布式存储系统中，单个节点的失效是常态，多个节点的失效不常见。将 RS 码应

收稿日期：2013-09-14；**修回日期**：2013-11-20
基金项目：国家重点基础研究发展计划(“973”计划)基金资助项目(2011CB302301)；国家自然科学基金资助项目(61025008, 61232004)；国家高技术研究发展计划(“863”计划)基金资助项目(2013AA013203)；国家科技支撑计划基金资助项目(2011BAH04B02)；中央高校基本科研业务费基金资助项目(2013TS043)；电子信息产业发展基金资助项目
Foundation Items: The National Basic Research Program of China(973 Program)(2011CB302301); The National Natural Science Foundation of China(61025008, 61232004); The National High Technology Research and Development Program of China (863 Program)(2013AA013203); The National Key Technology R&D Program of China(2011BAH04B02); The Fundamental Research Funds for the Central Universities(2013TS043); Electronic Development Fund of Information Industry Ministry

用于分布式存储系统中有几点值得注意：1) RS 码修复一个节点的数据所需要的修复带宽远大于该节点数据的数据量，从信息角度来说，传输过量数据用于修复少量数据是一种浪费；2) RS 码修复一个存储节点和多个存储节点所需要下载数据量是一样的。从本质上看导致以上 2 点的原因是：RS 码的修复重建了原始数据并重新对丢失的数据进行编码，所以每次修复无论修复数据量的大小，都必须从网络上传输相当于原始数据量大小的数据。云存储系统中存储了海量数据，如何减少因为数据丢失而产生的修复数据量是云存储系统需要面对的关键问题之一。

云存储服务提供了开放的存储接口，用户只需要付费即可使用相应的存储和带宽服务。基于现有的多个商业云存储服务，搭建了统一云存储系统 Ustor[4]，并采用功能性修复再生码（FRC）保证数据冗余的同时减少修复带宽。实验表明，与传统的 RS 码相比，采用 FRC 能够减少修复单个云存储系统中丢失数据的修复带宽，节省开支，且对系统响应时间影响较小。

2　相关编码技术

2.1　冗余编码

Ustor 实现了 RAID 码、RS 码和再生码等冗余编码，它们都是线性码，即使用线性组合的方式计算得到编码后的冗余数据。本节将介绍 Ustor 如何使用这些冗余编码将文件编码为冗余数据块。冗余编码包括再生码中的所有计算都是在有限域 GF(2^w) 中进行的，w 为字大小，它决定了有限域的大小和每次运算的位数。w 越小，编解码的速度越快；w 越大，存储系统可以支持更多的存储节点。Ustor 可选的 w 有 8、16、32 和 64，考虑到编、解码速度，w 一般为 8。

生成矩阵（GM, generator matrix）定义了如何将原始数据块编码为冗余数据块，RS 码的生成矩阵是一个 n 行 k 列矩阵，将 k 块原始数据块编码为 n 块冗余数据块。如果对应的编码是系统码（比如 RAID），编码后包含了原始数据，则生成矩阵中包含一个 $k\times k$ 大小的单位矩阵和$(n-k)\times k$ 的冗余矩阵，单位矩阵对应的是原始数据块，冗余矩阵对应的是冗余数据块。非系统码没有单位矩阵，整个生成矩阵都是冗余矩阵，因此编码后只有冗余数据块。行向量中项对应的是由原始数据块的组合系数，每个冗余数据块是原始数据块线性组合得到的。生成矩阵的每一个行向量对应一个冗余数据块，它们关系可以表示为

$$E_i = \sum_{j=1}^{k} \boldsymbol{GM}_{i,j} \times D_j (i = 1, \cdots, n)$$

其中，E_i 和 D_j 分别表示第 i 块编码数据块和第 j 块数据块，$\boldsymbol{GM}_{i,j}$ 代表生成矩阵 $\boldsymbol{GM}$ 第 i 行、第 j 列项的值。RS 码保持着 MDS 性质。MDS 性质指的是，编码后 n 个云存储服务中任意 k 个可以恢复出原始数据，这要求 RS 码生成矩阵中 n 行向量中任意 k 行都是行满秩的。

2.2　修复带宽和再生码

如果将每个云存储服务看做独立的存储节点，修复指的是当该存储节点数据丢失时，从其他助手存储节点下载一定量的数据用于恢复丢失的数据。如果恢复出来的数据和原来丢失的数据一样，则这样的修复是精确修复；但如果恢复出的数据和丢失的数据不同，而只是保持了 MDS 性质，本文称之为功能性修复。精确修复恢复出来的数据和丢失的数据完全一致，因此编码数据块 MDS 性质得以保持。RAID 码和 RS 码都是精确修复，每个存储节点保存一块冗余数据块，修复一个冗余数据块时，从其余任意 k 个助手存储节点中下载冗余数据块可以精确修复出丢失的冗余数据块。

修复带宽指的是节点发生失效导致数据丢失时，需要从网络中其他节点下载的数据量。在分布式存储系统中，单个节点失效已经是常态，减少修复带宽能够更快地修复丢失的数据[5]，并减少对其他正常数据使用的影响，因为较大的修复带宽会占据更多宝贵的网络资源和磁盘 I/O 资源。微软 Azure 云存储系统[6] 使用局部重建码（LRC, local reconstruction codes）对固定大小的数据块进行冗余编码，通过减少每次编码关联的节点个数减少了修复带宽。NCFS 是基于云存储系统的一个用户态文件系统，利用最小存储再生码 F-MSR[7]减少了修复带宽。

再生码[8] 是近年提出的一种应用于分布式存储系统的冗余编码。已有理论证明它可以达到存储和修复带宽的最优权衡[9]，显著减少了修复带宽。再生码也是一种 MDS 码，但其并不一定是存储最优的。最小存储再生码[10](MSR, minimum storage regenerating codes)是和 RS 码一样具有最优存储效率，最小带宽存储再生码（MBR,

minimum bandwidth regenerating codes）不是存储最优的，它在每个存储节点存储了更多的数据，但在修复单个节点失效数据时具有最高的修复效率，其在网络上传输的修复数据量和该节点丢失的数据量相同。

再生码或者通过增加每个云存储服务中存储的数据量，或者增加每次修复时助手存储节点的数量(d)，来减少修复带宽。不同于 RS 码，再生码在每个云存储服务中保存 α 块冗余数据块，每次修复时是从 $d(d\geqslant k)$个助手存储节点中下载共计 $d\times\beta$ 个冗余数据块，每个节点下载 $\beta(\alpha>\beta)$块。每个再生码由一个六元组参数确定$(n, k, \alpha, \beta, d, B)$，最后一个参数 B 是原始数据块的个数。根据 Dimakis 推导出的再生码理论限制，以上参数应满足

$$B\leqslant\sum_{i=0}^{k-1}\mathrm{MIN}(\alpha,(d-i)\beta)$$

相应地，再生码的生成矩阵 $\boldsymbol{GM}$ 有 $n\alpha$ 行 B 列，$n\alpha$ 个行向量对应着 $n\alpha$ 个冗余数据块。如果将生成矩阵 $\boldsymbol{GM}$ 对应 n 个节点按行分为 n 个子矩阵 $\boldsymbol{GM}_i(i=1, \cdots, n)$，每个子矩阵对应着一个云存储服务，且 $\boldsymbol{GM}=[GM_1, GM_2,\cdots,GM_n]^{\mathrm{T}}$。$\boldsymbol{GM}_i$ 是第 i 个云存储服务中冗余数据块的生成矩阵，即云存储服务 i 中的 α 个冗余数据块（$E_{(i-1)\alpha+1},\cdots,\mathrm{E}_{i\alpha}$）等于子矩阵 $\boldsymbol{GM}_i$ 和原始数据块$(D_1, D_2,\cdots, D_B)$的乘积，即

$$\begin{bmatrix}E_{(i-1)\alpha+1}\\ \vdots\\ E_{i\alpha}\end{bmatrix}=\boldsymbol{GM}_i\times\begin{bmatrix}D_1\\ \vdots\\ D_B\end{bmatrix}$$

再生码也是 MDS 码，也必须满足 MDS 性质。那么 n 个云存储中任意 k 个中的所有冗余数据块可以恢复出原始数据，同理 n 个子矩阵 $\boldsymbol{GM}_i$ 中任意 k 个子矩阵组合成的矩阵秩等于 B，则 MDS 性质满足。

3 Ustor 系统结构

3.1 Ustor 简介

统一云存储系统（Ustor）基于成熟的商业云存储服务，并对用户提供理论上无限的存储资源。Ustor 向下通过统一的接口对多个云存储服务提供商的存储资源进行管理，向上对用户提供简单的上传、下载接口。多个云存储服务对用户是透明的，用户可以像使用本地磁盘一样使用多个云存储服务的存储资源。Ustor 由 Ustor 服务器和多个云存储服务组成的混合云（cloud of clouds）组成。Ustor 服务器并不存储用户的任何数据，其职责有：1) 云网关，缓存用户上传和下载的数据，并将其上传到云存储服务或者返回给用户；2) 冗余编码，对用户上传的数据进行冗余编码，并对下载的编码数据块解码为原始数据；3) Web 服务器，Ustor 提供一个简单易用的 Web 接口使用户可以通过浏览器上传和下载文件。

3.2 Ustor 系统架构

如图 1 所示，Ustor 服务器和多个云存储服务组成了 Ustor 云存储系统。Ustor 支持一次写入、多次读取的访问模式。用户可以通过广域网连接至 Ustor 服务器，下载或上传文件。Ustor 服务器将负责用户数据缓存和数据编码，也通过广域网连接不同地域上、异构的云存储系统。这些云存储系统在地域上分布广泛，国内外都有。这些云存储提供的认证方法不同，访问接口不相同，但都提供一种类似于 REST 的访问接口，于是 Ustor 虚拟化出统一的接口代替云存储服务商提供异构的、差异化云存储接口，实现编码数据块在云存储系统的存与取。多个云存储服务共同负责存储数据文件和元数据文件。其中，数据文件通过冗余编码的方式保障可靠性，元数据文件通过副本的方法保障可靠性。当某个云存储因为网络不可访问，通过访问其他可用的云存储服务，保证数据的可用性。当某个云存储中冗余数据块发生丢失时，通过修复过程生成新的冗余数据块保证数据的可靠性。

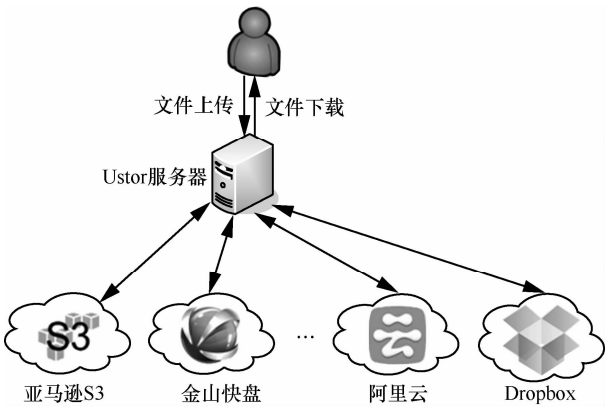


图 1 Ustor 云存储系统结构

Ustor 服务器主要由数据编码模块、元数据管理和数据分布模块组成。模块层次和组成如图 2 所示。数据编码模块负责将用户上传的文件进行冗余编码。可以采用 RS 码或 RAID 码，也可以使用功能性修复再生码(FRC, functional regenerating codes)。

这些冗余码共同的基础是一个编码库（coding library），其负责将读入内存的文件编码为多个冗余数据块。编码库中使用的部分数据结构来自于线性代数库（linear algebra library），比如生成矩阵 ***GM*** 和生成矩阵每一行的行向量。生成矩阵 ***GM*** 决定了如何将原始文件编码为冗余数据块，RS 码使用的是范德蒙德矩阵，FRC 码使用随机矩阵。编码过程中生成矩阵和数据块的计算不同于通常的算术运算，而是基于有限域。伽罗瓦有限域运算库（galois arithmetic library）提供了有限域上基本的算术运算，是编码库的基础。整个编码模块就像一个黑盒子（erasure coding box），输入的是将原始文件分割为 k 块等大小的原始数据块，输出的是编码后的 n 个冗余数据块。编码过程中的编码方法和编码参数将作为元数据信息由元数据管理模块记录于元数据文件中。

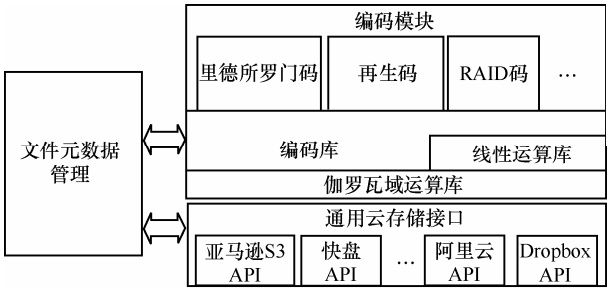


图 2　Ustor 服务器模块

LibCloud 是 Apache 的项目之一，它消除不同云服务提供商的差异性服务，抽象出了云存储服务的统一接口。Ustor 系统将其作为数据分布层的基础，它的作用相当于 Ustor 的 I/O 驱动模块（不同的云存储类似于不同的存储设备）。Ustor 写入数据时，所有编码后的数据块都是通过其写入到具体的云存储中；读取数据或者修复数据块时，它从指定的云存储中读取指定的数据块，并返回给编码层用于解码或修复。系统在原有 LibCloud 的基础上做了 2 点改进：1) 原有 LibCloud 兼容 Amazon S3 等国外云存储，不支持 Dropbox 和国内的云存储，本系统整合了 Dropbox、阿里云存储和快盘云存储，并补充了各自的身份认证方法；2) 修改 LibCloud 对上层的接口，使其能够兼容编码层。每一个云存储对应着一个存储节点，为这些存储节点编号，并记录下每个云存储与编码的一一对应关系。元数据管理模块将在数据块写入时，记录下编码后数据块所上传的云存储编号等元数据信息，并在读取数据块时，利用元数据信息中编码数据块所在的存储节点编号找到相应的云存储。

如之前两段所言，数据进行冗余编码和冗余数据块分布的过程中会产生一些元数据信息，这些上传文件的元数据由文件元数据模块管理。对于用户上传的每个文件都有对应一个元数据文件保存这些全局和冗余数据块的元数据。在编码时，元数据模块记录下编码前文件大小、所采用的编码方法、所使用的生成矩阵和计算有限域的大小等编码信息，并记录下每个冗余数据块所对应存储节点的编号。分布层中 LibCloud 按照云存储的编号和每个冗余数据块对应的编号来上传冗余数据块。在解码或者修复某个节点的数据块时，也需要这些冗余数据块的分布信息和编码信息进行解码和修复。因此 Ustor 为保证元数据文件的高可靠性，在保存了该文件冗余数据块的所有云存储上都保留了一份元数据文件副本。在修复时冗余数据块会发生改变时，Ustor 将同步更新所有元数据文件以保证数据的一致性。每个元数据文件大小在 1 KB 左右，带来额外的存储、网络开销很小，可以忽略不计。下面给出了一个包含全局（global）元数据信息和一个冗余数据块（chunk00）的部分元数据信息的例子。全局元数据信息包含的是原始文件信息、编码相关参数和所使用的云存储编号，冗余数据块元数据信息记录了某个冗余数据块所在云存储编号等信息。

包含全局元数据信息和一个冗余数据块的部分元数据信息：

[global]

coding_method = FRC　　#编码方法功能性修复再生码 FRC

filename = /home/root/test/testfile.txt　#原始文件名

row = 8　　#生成矩阵行数

col = 4　　#生成矩阵列数

alpha = 2　　#参数 α，代表每个云存储存储的冗余数据块数目

paritycoeff = 34 155 19 47 77 245 122 181 127 94 224 42 152 46 138 188 88 251 241 83 238 92 90 36 31 200 19 99 15 156 75 21

#生成矩阵每一项的值

file_size = 31127　#原始文件长度（byte）

word_size = 8　　#字大小，即有限域 GF(2^w) 中 w 的大小

total_node = 4 #所有云存储的个数

nodeids = 0 1 2 3 #4 个云存储的 id

[chunk00] #共计 *n* 个冗余数据块，第一个冗余数据块的信息

healthy = 1 #冗余数据块是否正常

chunkname = testfile.txt_00 #冗余数据块名称为“原文件名_编号”

nodeid = 0 #冗余数据块所在云存储的 id

chunkid = 0 #冗余数据块 id，对应生成该块的生成矩阵的行数

…

4 基于再生码的云存储系统

Ustor 将用户上传的文件编码为冗余数据块，分别保存至多个云存储服务中。当用户下载文件时，从任意指定个数量的云存储服务中下载冗余数据块，解码出原始数据并将文件返回给用户。当冗余数据块发生损坏或者丢失时，从其他云存储服务下载一定数量的冗余数据块再生该冗余数据块。

4.1 文件上传

文件的上传指的是用户将文件 F 保存至 Ustor 云存储系统，包括以下步骤：1) 编码模块（erasure coding box）将用户上传的文件等分为 B 块原始数据块（$F_1, F_2, \cdots, F_B$），不足的位用零填充；2) 编码模块随机生成$(n\alpha)\times B$ 大小、且满足 MDS 性质的生成矩阵 $\boldsymbol{GM}$，如不满足，重新生成新的生成矩阵直至该性质满足；3) 编码模块使用 $\boldsymbol{GM}$ 将 B 块原始数据块编码为 $n\alpha$ 块冗余数据块($E_1, \cdots, E_\alpha, \cdots, E_{n\alpha}$)；4) LibCloud 模块将 $n\alpha$ 块冗余数据块上传到 n 个可用的云存储服务中；5) LibCloud 模块将元数据文件 F_{meta} 上传至上一步保存冗余数据块的 n 个云存储服务中，每个保存一份副本。

4.2 文件下载

文件的下载指的是用户从 Ustor 下载指定文件 F。包括以下步骤：1) LibCloud 从任意一个云存储服务中下载该文件的元数据文件 F_{meta}，获得用于编码的生成矩阵 $\boldsymbol{GM}$ 等元数据信息；2) 从 n 个云存储服务中选定 k 个，并从相应的 k 个子矩阵中选取 B 个不相关行向量构成 $B\times B$ 大小的正方矩阵，根据 MDS 性质这样的 B 个行向量总是可以找到的。这个正方矩阵的逆矩阵为解码矩阵（DM, decode matrix）；3) LibCloud 从上一步骤中选定的 k 个云存储服务中下载正方矩阵中行向量一一对应的冗余数据块，共计 B 块（记作 $E_{d,1}$，$E_{d,2},\cdots,E_{d,B}$）；4) 计算解码矩阵 $\boldsymbol{DM}$ 和下载的 B 块冗余数据块（$E_{d,1}$，$E_{d,2},\cdots,E_{d,B}$）的乘积，得到 B 块原始数据块（$D_1, D_2, \cdots, D_B$）。合并原始数据块，利用元数据中文件长度截取得到文件 F 返回给用户。

4.3 数据修复

具有 MDS 性质的编码最多允许 $n-k$ 个云存储服务同时丢失数据，但这里仅讨论一个云存储服务丢失其存储的冗余数据块的情况（$n=d+1$）。不妨设发生冗余数据块丢失的云存储服务编号为 i，那么该云存储中冗余数据块（$E_{(i-1)\alpha+1},\cdots, E_{i\alpha}$）丢失，且生成矩阵 $\boldsymbol{GM}$ 中的生成子矩阵 $\boldsymbol{GM}_i$ 也不再可用。数据修复过程包括：生成子矩阵 $\boldsymbol{GM}_i$ 的修复和冗余数据块的修复。生成子矩阵修复将生成新的 $\boldsymbol{GM}_i'$代替不可用的 $\boldsymbol{GM}_i$，使生成矩阵的 MDS 性质继续保持，修复前的生成矩阵记作 $\boldsymbol{GM}$，修复后的生成矩阵记作 $\boldsymbol{GM}'$。冗余数据块的修复将真正地修复冗余数据块，而上一步生成子矩阵的成功修复保证了将要生成的冗余数据块也是满足 MDS 性质的。

修复 $\boldsymbol{GM}_i$ 的思路是从 d 个参与修复的子矩阵（$\boldsymbol{GM}_1,\cdots,\boldsymbol{GM}_{i-1}, \boldsymbol{GM}_{i+1},\cdots,\boldsymbol{GM}_n$）中各找到 β 个行向量，将这 $d\beta$ 个行向量线性组合为具有 α 个行向量的新子矩阵 $\boldsymbol{GM}_i'$，用其代替原来失效的 $\boldsymbol{GM}_i$，使新的生成矩阵 $\boldsymbol{GM}'$满足 MDS 性质。修复流程如图 3 所示。具体步骤如下：1) LibCloud 从任意一个云存储服务中下载文件的元数据文件 F_{meta}，获得用于编码的生成矩阵 $\boldsymbol{GM}$ 和 n、k、α、β、d、B 等参数；2) 除去失效的子矩阵 $\boldsymbol{GM}_i$，从剩余的 $d=n-1$ 个子矩阵中随机选取 $d\beta$ 个行向量组成一个候选矩阵（CM, candidate matrix），每个子矩阵随机选取 β 个行向量；3) 随机生成一个 $\alpha\times d\beta$ 大小的修复矩阵（RM, repair matrix），修复矩阵 $\boldsymbol{RM}$ 的作用是将候选矩阵 $\boldsymbol{CM}$ 线性组合为新的子矩阵 $\boldsymbol{GM}_i'$；4) 新的子矩阵 $\boldsymbol{GM}_i'$为修复矩阵 $\boldsymbol{RM}$ 和候选矩阵的乘积，即 $\boldsymbol{GM}_i'_{\alpha\times B}=\boldsymbol{RM}_{\alpha\times d\beta}\times\boldsymbol{CM}_{d\beta\times B}$。5)将原先 $\boldsymbol{GM}$ 中子矩阵 $\boldsymbol{GM}_i$ 替换为 $\boldsymbol{GM}_i'$，如果替换后的新生成矩阵 $\boldsymbol{GM}'$的 MDS 性质得以保持，则子矩阵修复成功；如果 MDS 性质不保持，则返回至第 2)步进行下一轮修复尝试，并重新选取候选矩阵和修复矩阵生成新的生成子矩阵 $\boldsymbol{GM}_i'$，直至它使得新生成 $\boldsymbol{GM}'$的 MDS 性质满足。

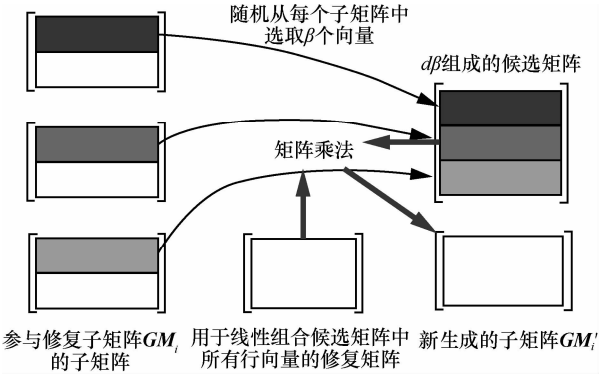


图 3　其他生成子矩阵参与修复生成子矩阵 $\boldsymbol{GM}_i$

以上循环的修复过程可能会遇到一种困境：无论修复尝试多少次，从 d 个子矩阵中都无法找到一个候选矩阵 $\boldsymbol{CM}$，使得由它线性组合得来的子矩阵 $\boldsymbol{GM}_i'$都无法使得 $\boldsymbol{GM}'$满足 MDS 性质。因此设置一个阈值（比如 1 000 次），当尝试修复了超过了这个阈值，则从第 2)步 d 个子矩阵中多选择一些行向量以辅助修复。实际修复中超过阈值的修复很少发生，可以忽略不计。需要说明的是生成子矩阵的修复是在 Ustor 服务器上进行的，只需要对生成矩阵进行修复计算，每一次修复尝试也不需要从云端下载任何数据，所以这个过程非常快。只有冗余数据块的修复才真正地从云端下载冗余数据块。

一旦生成子矩阵 $\boldsymbol{GM}_i$的修复成功，LibCloud 从 d 个云存储服务中下载 $d\beta$ 个冗余数据块，每个冗余数据块对应着候选矩阵 $\boldsymbol{CM}$ 中的每一行向量，也就是说这些数据块都是通过候选矩阵 $\boldsymbol{CM}$ 编码得来的。计算修复矩阵 $\boldsymbol{RM}$ 和这 $d\beta$ 个冗余数据块的乘积可以得到新的 α 个冗余数据块，这 α 个冗余数据块和其他节点的冗余数据块是满足 MDS 性质的。这是因为修复子矩阵和修复冗余数据块是对应的，它们之间有关系：

$$\boldsymbol{GM}_i' = \boldsymbol{RM} \times \boldsymbol{CM}$$

$$[E'_{(i-1)\alpha+1}, \cdots, E'_{i\alpha}]^{\mathrm{T}} = \boldsymbol{RM} \times \boldsymbol{CM} \times [D_1, D_2, \cdots, D_B]^{\mathrm{T}}$$

$$= \boldsymbol{GM}_i' \times [D_1, D_2, \cdots, D_B]^{\mathrm{T}}$$

因此，只要包含修复得到新生成子矩阵 $\boldsymbol{GM}_i'$的生成矩阵 $\boldsymbol{GM}'$是满足 MDS 性质的，那么新生成的冗余数据块总是满足 MDS 性质的。接着 LibCloud 将这 α 个新生成冗余数据块上传到编号为 i 的云存储服务中，将它们作为新的冗余数据块（$E'_{(i-1)\alpha+1}$,…, $E'_{i\alpha}$）。最后生成新的元数据文件并更新所有云存储服务中的元数据文件 $\mathrm{F}_{\mathrm{meta}}$，修复完毕。

5　性能评价

本文从 2 个方面评测基于再生码的 Ustor 云存储系统性能：分析 Ustor 所使用的云存储服务的价格和速度；分析系统响应时间，分析不同编码对数据存储的影响。实验环境：一台 Ustor 服务器 CPU 为 Intel Xeon E5620 2.40 GHz，32 GB 内存，并通过 100 Mbit/s 专线宽带连接 Internet。

5.1　Ustor 的云存储服务分析

Ustor 云存储系统现已采用国内的阿里云（AliYun）、金山快盘（KuaiPan）和国外的 Amazon S3 和 Dropbox 等云存储服务。Ustor 服务器通过广域网连接着以上 4 个云存储服务。表 1 给出了在 2013 年 6 月这 4 个云存储服务提供商的计费。所有的云存储服务都主要以存储计费和下载计费作为主要盈利方式。虽然所有云存储服务上传带宽都是免费的，但下载数据的费用却都比较高，大于相同数据量数据存储一个月的费用。

每个云存储服务不仅价格不同，所提供的上传、下载带宽也不同。图 4 是 Ustor 从单个云存储服务和多个云存储服务上传或下载 100 MB 数据的速度与响应时间。如果 Ustor 连接的是单个云存储服务，上传速度最快的是阿里云存储，最慢的是金山快盘；下载速度最快的是金山快盘，最慢的是 Dropbox。相比单个云存储服务，多个云存储服务组成的混合云相应时间更加稳定，速度也更快，这是因为利用了多个云存储服务的网络带宽，抹平了单个云带宽的“短板”。因此 Ustor 的存储带宽对应的是图 3 中 4 个云存储服务一起使用时的上传下

表 1　　**4 个云存储服务计费比较**

云存储服务	免费空间	请求计费	存储计费	上传	下载
阿里云（AliYun）	无	0.001 元/1 000 个	0.6 元/GB/月	免费	0.75 元/GB
金山快盘（KuaiPan）	15 GB	0.01 元/1 000 个	0.42 元/GB/月	免费	0.52 元/GB
亚马逊 S3	5 GB	0.005 美元/1 000 个	0.095 美元/GB/月	免费	0.12 美元/GB
Dropbox	2 ~18 GB	—	100 GB 每月 9.99 美元，其余免费	—	—

载带宽，这 4 个云存储服务组成的混合云也是本文下一小节测试响应时间的基础。

5.2 Ustor 响应时间分析

测试 Ustor 使用 FRC 对响应时间的影响，图 5 给出了 Ustor 使用再生码编码和不使用编码时，上传和下载不同大小文件的响应时间。编码采用的是(4, 2, 2, 1, 3, 4)-FRC，不编码则将等量的分块数据上传或下载。n=4，所以冗余分块存储在所有 4 个云存储服务中。文件越大，所需要的编码、解码时间越长，其占响应时间就越长，当原始文件大小为 256 MB 时，不编码上传响应时间为 78.8 s，编码上传响应时间为 83.5 s，编码时间占整个上传响应时间 5.6%。测试下载响应时间时，Ustor 仅从下载速度最快的 2 个云存储服务中下载冗余数据块进行解码。256 MB 文件不解码下载和解码下载时间分别为 18.4 s 和 20.5 s，解码占响应时间的 10.2%。如果文件小于 256 MB 时，编、解码占响应时间更少。

接下来使用 100 MB 文件测试 Ustor 上传、下载和修复 3 个操作中响应时间的组成。本文一共测试了 4 组编码，包括容 2 个云存储服务失效的(4, 2, 2, 1, 3, 4)-FRC（简写为(4, 2)-FRC）和(4, 2)-RS 与容单个云存储服务失效的(4, 3, 2, 1, 5)-FRC（简写为(4, 3)-FRC）和(4, 3)-RS。图 6(a)给出了 Ustor 服务器采用 4 种编码方法在上传、下载和修复 3 个操作中响应时间的组成：上传、下载和编码与磁盘读写。不难看出，和编码与磁盘读写所占用的时间相比较，4 种操作中网络传输时间（上传、下载）占了响应时间的绝大部分。进一步分析响应时间的组成，图 6(b)中给出了不包括网络传输的响应时间组成。上传、下载和修复 3 种操作中，修复磁盘读写（灰色）都占了大部分，而编、解码都在 1 s 内完成。因为 FRC 是非系统码，所以其编码、解码时间要长于系统码的 Reed-Solomon 编码，但 2 种编码修复时间相差不大。

图 6 中(4,2)-FRC 和(4,3)-FRC 是功能性修复再生码(4,2,2,1,3,4)-FRC 和(4,3,2,1,3,5)-FRC；(4,2)-RS 和(4,3)-RS 分别表示使用 Reed-Solomon 编码的 RAID6 和 RAID5。

在上面的例子中每个 100 MB 文件被(4,2)-FRC 编码为 8 块 25 MB 冗余块，被(4,2)-RS 编码为 4 块 50 MB 冗余块，每个云存储服务均存储 50 MB 数据。每修复一个失效云存储服务的 50 MB 数据，(4,2)-FRC 需要下载 25 MB×3=75 MB 数据，而(4,2)-RS 却需要下载 50 MB×2=100 MB 数据，(4,2)-FRC 比(4,2)-RS 具有相同的存储效率，却节省了用于下载

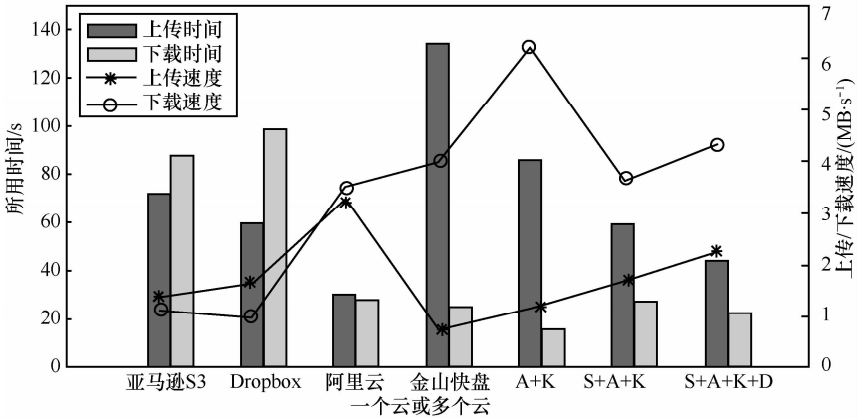


图 4 Ustor 上传/下载 100 MB 文件到单个云存储服务或者多个云存储服务的响应时间和速度
（图中缩写 S/D/A/K 分别表示亚马逊 S3、Dropbox、阿里云和金山快盘）

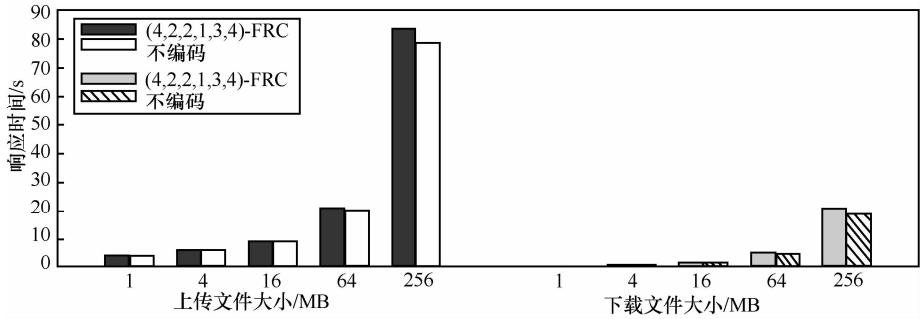
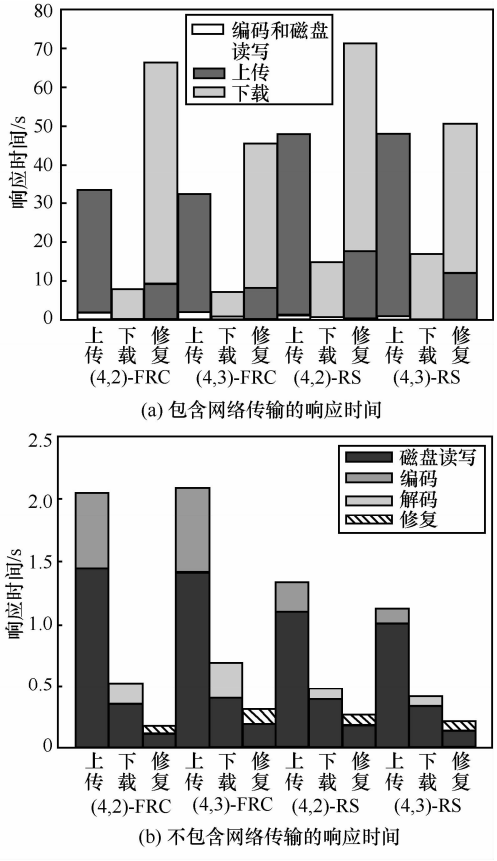


图 5 Ustor 上传/下载 100 MB 文件到多个云存储服务的响应时间和速度

数据块 25%的修复带宽。从上一小节可以知道云存储服务是按照下载计费的，节省了 25%修复带宽意味着节省了 25%的下载成本。同理(4,3)-FRC 比(4,3)-RS 需要额外 16.67%的存储开销，但却节省了 40%的修复带宽。所有情况元数据文件大小远小于冗余数据块大小，成本可忽略。总之，Ustor 在使用 FRC 时，其性能和使用 Reed-Solomon 码接近，但再生码在修复单个云存储服务中丢失的数据时节省了修复带宽，从而节省了下载成本。



(a) 包含网络传输的响应时间

(b) 不包含网络传输的响应时间

图 6　Ustor 采用 FRC 编码和 RS 编码 100 MB 文件的响应时间

图 6 中，(4,2)-FRC 和(4,3)-FRC 是功能性修复再生码(4,2,2,1,3,4)-FRC 和(4,3,2,1,3,5)-FRC；(4,2)-RS 和(4,3)-RS 分别表示使用 Reed-Solomon 编码的 RAID6 和 RAID5。

6　结束语

本文提出并实现了基于再生码的云存储系统 Ustor，其在多个商业云存储服务上使用编码提供可靠性。在单个云存储中数据发生丢失时，相对于传统的 Reed-Solomon 码，功能性修复再生码能够达到存储和修复带宽的折衷，减少数据重建时的修复带宽，即需要从其他云存储中下载的数据量，从而减少成本。从 Ustor 云存储系统的测试来看，使用 FRC 对数据编码带来的响应时间的影响最大在 5%~10%左右，而采用 FRC 或 Reed-Solomon 码对系统的性能影响相当。Ustor 可以通过网址 http://www.ustor.cn/访问。

参考文献：

[1] LI J, LI B. Erasure coding for cloud storage systems: a survey[J]. Tsinghua Science and Technology, 2013, 18(3): 259-272.

[2] PLANK J S. A tutorial on Reed-Solomon coding for fault-tolerance in RAID-like systems[J]. Software: Practice and Experience 1997, 27(9): 995-1012.

[3] PLANK J S, LUO J, SCHUMAN C D. A performance evaluation and examination of open-source erasure coding libraries for storage[A]. Proceedings of the seventh USENIX Conference on File and Storage Technologies (FAST'09)[C]. San Francisco, USA, 2009.253-265.

[4] Ustor[EB/OL]. http://www.ustor.cn/, 2013.

[5] HU Y, YU C M, LI Y K. On the practicality and extensibility of a network-coding-based distributed file system[A]. Proceedings of the Seventh IEEE International Symposium on Network Coding (NetCod'11)[C]. Boston, MA, USA, 2011.1-6.

[6] HUANG C, SIMITCI H, X Y. Erasure coding in windows azure storage[A]. Proceedings of the 25-th USENIX Annual Technical Conference(ATC'12)[C]. Boston, USA, 2012.15-26.

[7] HU Y, CHEN H C, LEE P P. NCCloud: applying network coding for the storage repair in a cloud-of-clouds[A]. Proceedings of The 10th USENIX Conference on File and Storage Technologies (FAST'12)[C]. San Francisco, USA, 2012.265-271.

[8] SHUM K W, HU Y. Functional-repair-by-transfer regenerating codes[A]. Proceedings of the IEEE 20th International Symposium on Information Theory Proceedings (ISIT'12)[C]. Cambridge, MA, USA, 2012.1192-1196.

[9] DIMAKIS A G, GODFREY P B, WU Y. Network coding for distributed storage systems[J]. IEEE Transactions on Information Theory, 2010, 56(9): 4539-4551.

[10] SHAH N B, RASHMI K V, VIJAY K P. Distributed storage codes with repair-by-transfer and nonachievability of interior points on the storage-bandwidth tradeoff[J]. IEEE Transactions on Information Theory, 2012, 59(3): 1837-1852.

作者简介：

柳青（1986-），男，江西九江人，华中科技大学博士生，主要研究方向为存储编码、云存储、分布式存储系统。

冯丹（1970-），女，湖北京山人，华中科技大学教授、博士生导师，主要研究方向为计算机系统结构、大规模网络存储系统、云存储、固态存储技术、容错理论、磁盘阵列体系结构。

李白（1990-），男，湖北仙桃人，华中科技大学硕士生，主要研究方向为云存储。