计算机研究与发展
Journal of Computer Research and Development
DOI:10.7544/issn1000-1239.2014.20121095
51(8):1671-1680,2014

分布式存储中精确修复最小带宽再生码的性能研究

卫东升　李　钧　王　新
(智能信息处理上海市重点实验室(复旦大学计算机科学技术学院)　上海　201203)
(12210240069@fudan.edu.cn)

Performance Study of Exact Minimum Bandwidth Regenerating Codes in Distributed Storage

Wei Dongsheng, Li Jun, and Wang Xin
(*Shanghai Key Laboratory of Intelligent Information Processing* (*School of Computer Science*, *Fudan University*), *Shanghai* 201203)

Abstract　Distributed storage systems need to introduce redundancy to ensure data reliability against node failures. To repair failed nodes, a significant amount of bandwidth is consumed. Regenerating codes are able to achieve the optimal tradeoff between the storage overhead and the repair bandwidth overhead. Based on the current situation that bandwidth resources are more precious than computing resources in distributed storage systems, exact minimum bandwidth regenerating (E-MBR) codes, which can be implemented by a product-matrix construction, enjoy the advantages of regenerating codes as well as systematic codes, and have no restrictions for all construction parameters, making themselves a promising candidate towards the application in distributed storage systems. However, the performance overhead of distributed storage systems based on this coding scheme has not been investigated and analyzed. This paper gives a formal description of coding operations, which can be categorized into three distinct phrases: uploading, downloading and repairing. We hereby analyze the impact of the CPU utilization, the file size, the buffer size and the Galois field size to the computing rates in the three distinct phrases above. We find that distributed storage systems based on E-MBR codes are able to achieve a high computing throughput if we configure the construction parameters of E-MBR codes appropriately.

Key words　distributed storage; regenerating codes; network coding; product-matrix; performance study

摘　要　分布式存储系统为保证数据可靠性,需要对数据进行冗余存储来应对由于节点失效所带来的数据不可靠性.基于矩阵积构造的精确修复最小带宽再生码除了能够显著降低系统的存储冗余,而且编码的构造参数之间没有约束限制,还能够显著降低修复带宽的开销,具有广阔的应用前景.然而,基于此编码方案所设计的分布式存储系统的性能开销并没有得到充分的研究和分析.针对该编码在分布式存储系统中数据上传、修复、下载3个阶段,分别比较CPU使用率、文件大小、缓冲区大小以及有限域大小对上述3个阶段中运算速度的影响,发现通过对相关参数进行合理配置,可以使得基于相应编码方案的分布式存储系统能够获得良好的运行性能.

收稿日期:2012-11-29;**修回日期**:2013-05-02
基金项目:国家自然科学基金项目(61171074);国家"八六三"高技术研究发展计划基金项目(2009AA01A348);教育部新世纪优秀人才支持计划基金项目(NCET-11-0113)
通信作者:王　新(xinw@fudan.edu.cn)

关键词 分布式存储;再生码;网络编码;矩阵积;性能研究

中图法分类号 TP302.8

分布式存储系统通过使用大量由廉价商用硬件组成的存储节点提供大规模存储服务. 在分布式存储系统中,一旦有节点因硬件故障等原因导致节点失效,为保证系统的可靠性,需要及时在失效节点以外的某个新的节点上进行数据修复,来维持一个合理的冗余度. 传统的冗余方式包括副本和纠删码方式. 副本方式直接在存储系统中的多个节点上分别存放一个或多个原文件的副本数据,其存储开销非常大;纠删码方式是将原文件划分为 k 块,对其进行编码得到 n 块($n>k$),若该纠删码满足最大距离可分(maximum distance separable, MDS)性质,则其中任意 k 块可以解码得到原数据. 纠删码方式虽能显著降低冗余存储的开销,但在进行修复过程时,需要解码得到原文件后,再重新编码生成丢失的数据块,整个过程需要消耗大量的网络带宽. 近年来,Dimakis 等人[1]提出的再生码(regenerating codes)能够获得修复带宽和存储冗余之间的最优权衡关系,并且其权衡曲线上的一个极端点——最小带宽点(minimum bandwidth regeneration point, MBR point),通过连接 $d(k\leqslant d<n)$ 个存储节点并从每个节点上下载部分经过编码后的数据片(数据块中的一部分数据)进行数据修复,理论上能够将修复过程中的带宽开销降至最低.

再生码的构造和修复过程仅能保证修复后的数据仍然满足 MDS 性质,即所谓的功能修复(functional repair),其经过修复后的数据同失效数据不一致,却仍能保证整个系统维持 MDS 性质. 但在分布式存储系统中,为了加快数据访问的速度,编码后的冗余数据要求满足系统码(systematic codes)特性,即原数据可以通过某些节点无须解码直接访问. 系统码性质要求修复过程满足精确修复(exact repair),即修复后的数据同失效数据精确一致. 另外,虽然功能修复较精确修复实现简单,但它本身存在一些重要缺陷:例如无法保证编码的系统码特性、每次修复时都要更新解码规则等,因此精确修复较功能修复更具有实际意义. 文献[2]利用矩阵积的方法给出了精确修复最小带宽再生码(E-MBR codes)的设计思路,该方法所设计的再生码对于参数 n,k,d 之间无需任何限制,这对于实际分布式存储系统具有很好的适用性.

文献[2]从理论上给出了利用矩阵积的方法来设计构造精确修复最小带宽再生码,但对于实际中如何利用该编码方案实现对文件的分布式存储没有给出实现过程. 此外,不同的相关参数(CPU 利用率、文件大小等)对于分布式存储系统的性能开销影响较大,而文献[2]也没有对此进行分析. 因此,本文将聚焦对该编码方案在分布式存储系统中所涉及到的数据上传、数据下载、数据修复过程中资源开销的分析,并通过实验比较了不同 CPU 利用率、不同文件大小和缓冲区大小,以及不同有限域大小对以上 3 个阶段中运算速度的影响,发现了通过对上述参数进行合理的配置,可以使得基于该编码方案的分布式存储系统可以获得良好的运行性能.

1 相关工作

Dimakis 等人[1]通过在分布式存储系统的修复过程中引入信息流图(information flow graph)这一概念,将分布式系统的修复模型转化为一个组播网络模型,从而利用网络编码[3]技术,实现了对修复带宽的优化,并且获得了在满足一定可靠性要求的前提下,修复带宽和存储开销之间的最优权衡(optimal tradeoff)关系,称满足这种最优权衡关系的编码为再生码. 关于再生码的综述文献可以参考文献[4]. 关于存储系统中纠删码的研究综述可以参考文献[5].

Duminuco 等人[6]随后针对功能修复的再生码,利用随机线性编码(random linear codes)[7]的方法对其进行实现,通过将分布式存储系统的整个生命周期分为 3 个阶段:数据插入、数据维持以及数据重构,分别分析这 3 个阶段中存储、通信以及计算量的开销. 发现了在再生码中通过增加节点的冗余数据量,可以有效改善修复过程的通信带宽开销. 然而,该工作仅仅分析了功能修复再生码的性能,对于精确修复再生码的实际资源开销情况没有涉及.

Hu 等人[8]设计实现了基于网络编码的实际可扩展的分布式文件系统 NCFS,该系统在实际分布式网络环境中实现了再生码,且对于各个存储节点之间无需相互协作的要求. 但该系统将所有的运算量集中在一个专门进行运算处理的代理节点中,这使得整个系统的实际性能将受限于代理节点的性能配置,而对于整个分布式存储系统中的其他节点的计算资源没有进行合理的利用,因此该模型具有一定

的局限性.在随后的工作[9]中,Hu 等人设计了运用于云存储服务的数据修复系统 NCCloud,然而该系统也仅仅实现了功能修复最小存储再生码,并未对其他编码,尤其是精确修复再生码进行研究.

Arnoult[10]重点分析了精确修复最小带宽再生码在文献[8]中所设计系统中的实际性能开销情况,但由于其所分析的精确修复最小带宽再生码适用于 $d=n-1$ 的情况,且存储节点同样不具备计算能力,所有的计算量都集中在同一代理节点中,因而他所得到的资源开销结果对分析实际分布式存储系统的资源开销情况同样具有局限性.

Rashmi 等人[2]从理论上给出了利用矩阵积的方法来设计构造精确修复最小带宽再生码,该方法所设计的精确修复最小带宽再生码对于参数 n,k,d 之间无需任何限制,这对于实际带宽资源较计算资源相对稀缺的分布式存储系统,具有很好的适用性.但该工作对如何利用该编码方案来实现实际文件的分布式存储以及对实际分布式存储系统的资源开销和性能没有进行分析,因此本文将重点进行对于这两方面的研究和分析.

2 背景介绍

对于分布式存储系统,大致包含 3 个相对独立的阶段:数据上传(data uploading)过程是将原文件通过编码上传存储至各个存储节点;数据下载(data downloading)过程进行对原文件的下载恢复工作;而数据修复(data repairing)完成对于失效节点中数据的修复,以保证整个系统数据的持续可靠性.同时,针对以上 3 个阶段中所具体涉及到的资源开销,又可以对其节点进行分类:在数据上传过程时,定义完成对原文件进行编码以及上传操作的节点为编码节点(encoding node);在数据下载时,定义解码节点(decoding node)是完成对数据的接收并进行相应文件恢复工作的节点;在数据修复时,定义新节点(newcomer node)是替换失效节点并精确修复出失效数据的那个节点;而参与节点(participant node)是那些对本地数据进行线性运算,并传输给新节点以便于新节点能够精确修复失效数据的那些节点.

文献[2]提出利用矩阵积的方法来构造精确修复最小带宽再生码,其编码和解码运算在有限域 $GF(q)$ 上进行.设原文件由 B 个 $GF(q)$ 上的符号元素(symbol)构成,其所包含的数据集用 $\{u_i\}_{i=1}^{B}$ 表示.n 个存储节点分别存储 α 个符号元素,且任意 $k(k<n)$ 个节点上保存的数据能恢复原文件.当有节点失效时,新节点连接剩余节点中的任意 $d(k\leqslant d\leqslant n-1)$ 个参与节点,并从每个参与节点上下载 β 个符号元素,从而精确恢复失效数据.此过程中,新节点总共需要下载 βd 个符号元素.这里,把参数[n,k,d]称为 1 级参数(primary parameter),把[α,β,B]称为 2 级参数(secondary parameter),由下文可以看出,1 级参数可以唯一确定 2 级参数.此外定义大小为 $d\times d$ 的消息矩阵(message matrix)为 $\boldsymbol{M}$,大小为 $n\times d$ 的编码矩阵(encoding matrix)为 $\boldsymbol{\Psi}$,以及码字矩阵(code matrix)为 $\boldsymbol{C}$,大小也为 $n\times d$,满足有 $\boldsymbol{C}=\boldsymbol{\Psi M}$.

首先考虑 $\beta=1$ 的情况,即修复过程中,新节点从每个参与节点下载一个符号元素.利用下文所给出的矩阵积方法,可以实现对于原文件的数据上传、数据下载已经数据修复工作.对于 $\beta\neq1$ 的情况,假设 $\beta'=\delta\beta(\delta\in\mathbb{N})$,有 $\alpha'=\delta\alpha,B'=\delta B$,即可以认为原文件的大小为 δB 个元素.通过将 δB 等分为 δ 组,然后只要每一组再分别使用同样的方法,可以将 $\beta\neq1$ 转化为 $\beta=1$ 来实现.

2.1 数据上传

由文献[2]可知,最小带宽再生码(minimum bandwidth regenerating codes, MBR codes)中 2 级参数可以表示为

$$\begin{aligned}\alpha&=d,\\ \beta&=1,\\ B&=\mathrm{C}_{k+1}^{2}+k(d-k).\end{aligned}\tag{1}$$

令 $\boldsymbol{S}$ 是一个 $k\times k$ 的矩阵,它的上三角部分(包含对角线上的元素)共 C_{k+1}^{2} 个元素,由消息符号元素集合 $\{u_i\}_{i=1}^{B}$ 的前 C_{k+1}^{2} 个元素从左往右从上往下依次填充,剩下的 C_{k}^{2} 个下三角部分的元素通过构造 $\boldsymbol{S}$ 为一个对称矩阵来进行填充.令 $\boldsymbol{T}$ 是一个 $k\times(d-k)$ 矩阵,其元素为消息符号元素集合 $\{u_i\}_{i=1}^{B}$ 中剩下的 $k\times(d-k)$ 个元素从左往右从上往下依次填充.最后令消息矩阵 $\boldsymbol{M}=\begin{bmatrix}\boldsymbol{S} & \boldsymbol{T}\\ \boldsymbol{T}^{\mathrm{T}} & \boldsymbol{0}\end{bmatrix}$.显然,$\boldsymbol{M}$ 也是一个对称矩阵.

令编码矩阵为 $\boldsymbol{\Psi}=[\boldsymbol{\Phi}\quad\boldsymbol{\Delta}]$,其中 $\boldsymbol{\Phi}$ 和 $\boldsymbol{\Delta}$ 分别为 $n\times k$ 与 $n\times(d-k)$ 矩阵,且满足以下条件:

1) $\boldsymbol{\Psi}$ 中任意 d 行线性无关;

2) $\boldsymbol{\Phi}$ 中任意 k 行线性无关.

这里,为了保证编码后的码字具有系统码性质,对编码矩阵 $\boldsymbol{\Psi}$ 稍作调整,让 $\boldsymbol{\Psi}$ 变换为如下形式:

$$\boldsymbol{\Psi}=\begin{bmatrix}\boldsymbol{I}_k & \boldsymbol{0}\\ \tilde{\boldsymbol{\Phi}} & \tilde{\boldsymbol{\Delta}}\end{bmatrix},\tag{2}$$

其中 $\boldsymbol{I}_k$ 为 $k\times k$ 单位矩阵,$\boldsymbol{0}$ 为 $k\times(d-k)$零矩阵,$\tilde{\boldsymbol{\Phi}}$ 和 $\tilde{\boldsymbol{\Delta}}$ 分别为$(n-k)\times k$ 与$(n-k)\times(d-k)$矩阵,且满足$[\tilde{\boldsymbol{\Phi}}\quad\tilde{\boldsymbol{\Delta}}]$矩阵中所有的子矩阵都为满秩矩阵.显然该矩阵不但能够同时满足上述 2 个条件,而且从下面式(3)可以看出,所产生的码字矩阵具有系统码特性.因此,下文中所涉及的编码矩阵皆为该类型.

根据以上分析,则码字矩阵 $\boldsymbol{C}$ 可以表示为

$$\boldsymbol{C}=\boldsymbol{\Psi}\boldsymbol{M}=\begin{bmatrix}\boldsymbol{I}_k & \boldsymbol{0}\\ \tilde{\boldsymbol{\Phi}} & \tilde{\boldsymbol{\Delta}}\end{bmatrix}\begin{bmatrix}\boldsymbol{S} & \boldsymbol{T}\\ \boldsymbol{T}^{\mathrm{t}} & \boldsymbol{0}\end{bmatrix}=\begin{bmatrix}\boldsymbol{S} & \boldsymbol{T}\\ \tilde{\boldsymbol{\Phi}}\boldsymbol{S}+\tilde{\boldsymbol{\Delta}}\boldsymbol{T}^{\mathrm{t}} & \tilde{\boldsymbol{\Phi}}\boldsymbol{T}\end{bmatrix},\tag{3}$$

这样第 $i(i=1,\cdots,n)$个存储节点保存了码字矩阵 $\boldsymbol{C}$ 中第 i 行的 d 个符号元素.

2.2 数据下载

由式(3)可知,节点 $i(i=1,\cdots,k)$存储的码字没有编码,因此通过下载节点集合$\{i\mid i=1,\cdots,k\}$上所有的数据,得到消息矩阵 $\boldsymbol{M}$ 的前 k 行,即$[\boldsymbol{S}\quad\boldsymbol{T}]$,由于 $\boldsymbol{S}$ 和 $\boldsymbol{T}$ 在构造时包含了原文件符号元素集合的所有元素$\{u_i\}_{i=1}^{B}$,因此通过从中抽取相应的元素,即能恢复出原文件 B.

2.3 数据修复

定义编码矩阵 $\boldsymbol{\Psi}$ 的第 i 行 $\underline{\boldsymbol{\phi}}_i^{\mathrm{T}}$ 为节点 i 的编码向量(encoding vector).假设节点 $f(f=1,\cdots,n)$失效,则修复工作就是将失效节点上的数据 $\underline{\boldsymbol{\phi}}_f^{\mathrm{T}}\boldsymbol{M}$ 在新节点上精确修复出来.新节点连接剩余 $n-1$ 个节点中的任意 d 作为本次修复过程的参与节点,假设参与节点集合为$\{h_j\mid j=1,\cdots,d\}$.每个参与节点自身先分别进行矩阵乘积运算 $\underline{\boldsymbol{\phi}}_{h_j}^{\mathrm{T}}\boldsymbol{M}\underline{\boldsymbol{\phi}}_f$,随后将其计算所产生的结果(一个符号元素)传送给新节点,则新节点接收到的数据矩阵为 $\boldsymbol{\Psi}_{\mathrm{repair}}\boldsymbol{M}\underline{\boldsymbol{\phi}}_f$,其中

$$\boldsymbol{\Psi}_{\mathrm{repair}}=[\underline{\boldsymbol{\phi}}_{h_1}^{\mathrm{T}}\quad\underline{\boldsymbol{\phi}}_{h_2}^{\mathrm{T}}\quad\cdots\quad\underline{\boldsymbol{\phi}}_{h_d}^{\mathrm{T}}]^{\mathrm{T}},\tag{4}$$

由 2.1 节可知 $\boldsymbol{\Psi}_{\mathrm{repair}}$可逆,将接收到的数据矩阵左乘 $\boldsymbol{\Psi}_{\mathrm{repair}}^{-1}$,得到 $\boldsymbol{M}\underline{\boldsymbol{\phi}}_f$.又由 $\boldsymbol{M}$ 具有对称性,则有$(\boldsymbol{M}\underline{\boldsymbol{\phi}}_f)^{\mathrm{T}}=\underline{\boldsymbol{\phi}}_f^{\mathrm{T}}\boldsymbol{M}$,从而失效数据得到精确修复.

3 算法设计

在第 2 节中,所有的运算都是在有限域 $GF(q)$上进行.在实际具体操作过程中,一般令 $q=2^w$,即有限域为 $GF(2^w)$,这样可以加快运算速度[6].因此,当所有的元素都取自有限域 $GF(2^w)$上,每一个元素表示 w 个位所组成的二进制数据序列,这样 1 次编码能够完成处理的数据量为 Bw 个位.更进一步,根据第 2 节的结论的推广,可以处理任意 $\delta Bw(\delta\in\mathbb{N})$个位的文件.

假设原文件大小为$|size|$个位,当系统接收到编码操作指令后,从磁盘中找到相应的文件,然后将其读入到内存中进行编码.如果原文件非常大,为了减小内存开销,需要把文件划分成多块,分批次地而非一次性地读入到内存中进行操作.因此在编码操作开始之前,需要在内存中开辟一定大小的数据缓冲区.假设缓冲区大小为$|buffer|$个位,那么实际中真正读入内存的参与运算的数据量为$|actual_buffer|=\delta Bw\leqslant|buffer|$,满足 $\delta>0$ 且$(\delta+1)Bw>|buffer|$,即需要保证读入内存的数据量为 Bw 的倍数并且刚好不能超过$|buffer|$.由于一般情况下 B 与 w 都远小于$|buffer|$,因此可以近似认为$|actual_buffer|\approx|buffer|$.

然而容易发现,过大的$|buffer|$值会增加内存开销,甚至会对内存中其他进程产生影响,但过小的$|buffer|$值会增加各阶段的磁盘读写开销,降低整个系统的性能,因此需要对$|buffer|$大小进行合理的估计.如果$|actual_buffer|<|size|$且$|size|$不能整除$|actual_buffer|$,则需要将原文件进行适当扩充(一般扩充在文件末尾),使得每一轮进入内存进行运算的数据量都保持为$|actual_buffer|$个位,假设扩充后的文件大小为$|padding_size|>|size|$,满足$|padding_size|=num|actual_buffer|$,$num\in\mathbb{N}$.另一方面,如果$|actual_buffer|>|size|$,那么需要做的是将原文件从$|size|$个位填充至$|actual_buffer|$个位(即这里$|padding_size|=|actual_buffer|$),且只需进行 $num=1$ 轮即可完成所有数据的上传过程.在后一种情况即$|actual_buffer|>|size|$情况下,易知$|actual_buffer|\approx|buffer|$,因此如果所设定的$|buffer|$远大于$|size|$,那么就需要对原文件进行较多的数据扩充才能参与存储,这样严重增加了整个系统的存储开销.因此,一般情况下,尤其是对于大文件(上百 MB 甚至上 GB 的文件),$|buffer|$的取值一方面不能超过原文件大小,另一方面也要保证其内存开销尽量较少.通过这两方面的综合限制所确定的$|buffer|$值,能够保证$|padding_size|\approx|size|$.

根据以上分析,要对原文件进行基于精确修复最小带宽再生码的分布式存储,就需要对其进行相应的预处理操作,包括对于有限域的选择、$|buffer|$值的设置、原文件的扩充、1 级参数 n,k,d 的配置等.

对于数据上传过程,每一轮读取参与编码运算的$|actual_buffer|$个位大小的数据量,每一轮中的每一个码字大小为$|slice|=|actual_buffer|/B$,经过$num$轮运算操作后,每个存储节点共存储$d$块数据块,每块数据块由$num$片大小为$|slice|$的数据片组成,即$|block|=num\times|slice|$.

算法 1. 数据上传算法.

输入:原始文件;

输出:编码后的编码块.

① $0\Rightarrow counter$;

② 将原始文件保存在编码节点的磁盘上;

③ repeat

④ $counter+1\Rightarrow counter$;

⑤ 编码节点从原文件中顺序读取$|actual_buffer|$位进入内存;

⑥ 生成$n\times d$的编码矩阵$\boldsymbol{C}$;

⑦ 并行地将$\boldsymbol{C}$中第$i(i=1,2\cdots,n)$行向量传递给第i个存储节点上;

⑧ if $counter>1$ then

⑨ 每个存储节点将接收到的d片数据片写到本地磁盘上,写入的位置为对应的上一次接收到的d片数据片之后;

⑩ else

⑪ 每个存储节点将接收到的d片数据片写到本地磁盘上;

⑫ end if

⑬ until $counter\geqslant num$.

对于数据下载过程,解码节点首先从k个保存系统码码字的节点下载所有数据块.然后每一轮从获得的kd数据块中顺次读取kd片数据片并在完成解码操作生成数据片大小为$|slice|'=B|slice|$,共进行num轮.最后截取前$|size|$个位的数据即为恢复出的原文件.

算法 2. 数据下载算法.

输入:编码后的编码块;

输出:原始文件.

① $0\Rightarrow counter$;

② 解码节点连接k个保存系统码码字数据的节点,下载其上所保存的数据块;

③ repeat

④ $counter+1\Rightarrow counter$;

⑤ 解码节点从收到的$k\times d$个数据块中分别顺次读取$|slice|$大小的数据片;

⑥ 解码节点进行解码操作,获得一片大小为$|slice|'=B|slice|$的数据片;

⑦ if $counter>1$ then

⑧ 解码节点将获得数据片写到磁盘中,其位置为上一次获得的数据片之后;

⑨ else

⑩ 解码节点将获得数据块片写到本地磁盘中;

⑪ end if

⑫ until $counter\geqslant num$;

⑬ 截取获得的数据文件前$|size|$位数据为解码恢复出的原文件.

对于数据修复过程,每一轮中每个参与节点首先进行本地运算生成一片$|slice|$大小的数据片后发送给新节点,新节点收到d个参与节点发送来的d片数据片后解码修复运算,生成d片数据片后保存本地磁盘中,两者进行num轮协作后,精确修复出失效的数据.

算法 3. 数据修复算法.

输入:失效一个块后的编码块集合;

输出:修复好的编码块.

① $0\Rightarrow counter$;

② 每个参与节点将各自的编码向量独立发送给新节点;

③ 新节点收集满d个编码向量后,进行对该些编码向量所组成矩阵的取逆运算;

④ repeat

⑤ $counter+1\Rightarrow counter$;

⑥ 每个参与节点从本地磁盘的d块数据块中依次读取$|slice|$大小的数据片;

⑦ 每个参与节点进行矩阵积运算后,生成一片$|slice|$大小的数据片,同时将生成的数据片各自独立发送给新节点;

⑧ 新节点收集到d个参与节点所发来的数据片之后进行矩阵积运算,获得d片数片;

⑨ if $counter>1$ then

⑩ 新节点将解码修复得到的d片数据片元素写到本地磁盘上,写入位置为对应的上一次获得的d片数据片之后;

⑪ else

⑫ 新节点将解码修复获得的d片数据片元素写到本地磁盘上;

⑬ end if

⑭ until $counter\geqslant num$.

算法 1~3 分别表示分布式存储系统中数据上传、

数据下载、数据修复3个阶段的算法流程. 在数据上传之前,将扩充后的文件分成 num 份,每一轮的操作相互独立,而数据下载和数据修复过程也都由 num 轮相互独立的操作实现完成. $counter$ 记录循环次数,当其累加至 num 时运行完成. 这里需要指出的是,当 $counter=1$ 时,在数据下载和数据修复2个阶段中,只需要将生成的数据片写在本地磁盘中即可;而当 $counter=1$ 时,将上述2个阶段中生成的数据片写在上一次对应的数据片之后,这样可以避免存储在节点上的文件块过于零散,加快以后磁盘读取速度.

4 资源开销估计

本节将分析分布式存储系统3个阶段中相应的资源开销. 表1为所获得的数据上传、数据下载、数据修复过程所涉及到的4类节点中的计算开销和磁盘读写开销情况. 这里,计算开销分为运算数据量 $DATA$ 和有限域运算次数 $GALOIS$ 两部分来讨论,而磁盘读写开销分为数据吞吐量 $THROUGHPUT$、磁盘读操作次数 IO_{read} 以及磁盘写操作次数 IO_{write} 3部分来讨论.

Table 1 The Resource Costs Corresponding to the Different Phrases in Distributed Storage

表1 分布式存储中不同阶段所对应的资源开销

Resource Costs		Data Uploading	Data Downloading	Data Repairing	
Overhead	Operations Cost	Encoding Node	Decoding Node	Participant Node	Encoding Node
Computing Overhead	Computation Data/b	$\frac{nd\lvert size\rvert}{B}$	$\frac{kd\lvert size\rvert}{B}$	$\frac{d\lvert size\rvert}{B}$	$\frac{d\lvert size\rvert}{B}$
	Galois Field Operations	$\frac{5d^2(n-k)}{wB}\lvert size\rvert$	0	$\frac{5d}{wB}\lvert size\rvert$	$\frac{5d^2}{wB}\lvert size\rvert$
Disk I/O Overhead	Data Throughput/b	$\lvert size\rvert$	$2\lvert size\rvert$	$\frac{d}{B}\lvert size\rvert$	$\frac{d}{B}\lvert size\rvert$
	Input Operations	$\frac{\lvert size\rvert}{\lvert buffer\rvert}$	$kd\frac{\lvert size\rvert}{\lvert buffer\rvert}$	$d\frac{\lvert size\rvert}{\lvert buffer\rvert}$	0
	Output Operations	0	$\frac{\lvert size\rvert}{\lvert buffer\rvert}$	0	$d\frac{\lvert size\rvert}{\lvert buffer\rvert}$

4.1 计算开销

4.1.1 运算数据量

对于数据上传过程,编码节点总共进行了 num 轮编码操作,每一轮生成 nd 片 $\lvert slice\rvert$ 大小的数据片,因此

$$DATA^{\text{upload}}=nd\lvert slice\rvert num\approx\frac{nd\lvert size\rvert}{B}. \quad (5)$$

对于数据下载过程,解码节点总共进行 num 轮,每一轮读取 kd 片 $\lvert slice\rvert$ 大小的数据片,因此

$$DATA^{\text{download}}\approx\frac{kd\lvert size\rvert}{B}. \quad (6)$$

对于数据修复过程,参与运算的有两类节点:参与节点和新节点. 对于参与节点每一轮参与的数据量为 d 片 $\lvert slice\rvert$ 大小数据片,共进行 num 轮,因此

$$DATA^{\text{repiar}}_{\text{participant}}\approx\frac{d\lvert size\rvert}{B}. \quad (7)$$

对于新节点,每一轮从 d 个参与节点中共收到 d 片 $\lvert slice\rvert$ 大小的数据片,共有 num 轮,因此

$$DATA^{\text{repair}}_{\text{newcomer}}\approx\frac{d\lvert size\rvert}{B}. \quad (8)$$

4.1.2 有限域运算

由于有限域上 $GF(2^w)$ 元素(除0外)集合为 $\{\sigma^0,\sigma^1,\cdots,\sigma^{q-2}\}$,这里,$\sigma$ 为本原多项式的根,$q=2^w$ 且 $\sigma^{q-1}=1$,因此有限域上的运算根据如下规则进行设计实现的:

1) 1次加法和减法运算对应的是1次XOR运算操作.

2) 乘法运算可以通过查找(lookup)正反对数表完成的. 例如计算 ab,根据有限域上的性质,假设 $a=\sigma^i$,$b=\sigma^j$,从而 $ab=\sigma^{i+j}$,通过查找正对数表分别获得 i 与 j 的值,然后相加得到 $i+j$,最后通过查找反对数表获得结果. 可以看出,1次乘法运算需要3次查表和1次加法,共4次运算操作. 除法运算也基本类似,这里不在赘述.

在分析有限域运算时,把1次的查表操作或1次XOR运算称为1次运算操作. 重点考察各个阶段中有限域的运算操作次数.

对于数据上传过程,每一轮参与编码的消息矩阵由 $d\times d$ 个元素构成,每个元素表示一个长度为 δ

的数值序列,每个数值都由 w 个位组成.由于要求编码后的码字矩阵具有与系统码特性,因此实际编码运算过程中参与有限域运算的编码矩阵大小(不考虑系统码的编码向量)为$(n-k)\times d$,即 $\boldsymbol{\Psi}$ 的最后 $n-k$ 行.当两者进行有限域 $GF(2^w)$ 上的矩阵积运算时,每一轮分别进行了 $d^2(n-k)\delta$ 次乘法与加法运算,总共进行了 num 轮完成对整个数据的编码.由于 1 次乘法运算共进行 4 次运算操作,1 次加法运算共进行了 1 次运算操作,因此数据上传过程中有限域操作次数为

$$GALOIS^{\text{upload}}=5d^2\delta(n-k)num\approx\frac{5d^2(n-k)}{wB}|size|. \tag{9}$$

对于数据下载过程,由于具有系统码特性,因此解码过程不涉及有限域运算

$$GALOIS^{\text{download}}=0. \tag{10}$$

对于数据修复过程中的参与节点,类似于数据上传过程中有限域操作次数的计算思路,容易得到:

$$GALOIS^{\text{repair}}_{\text{participant}}\approx\frac{5d}{wB}|size|. \tag{11}$$

对于数据修复过程中的新节点,对其分析时稍有差别.因为当新节点首先接收到的 d 个编码向量后,需要对这个 $d\times d$ 矩阵进行在 $GF(2^w)$ 上的矩阵取逆(inversion)的运算,根据文献[6],可以知道该取逆运算的涉及到 d^3 次有限域乘法和加法,因此取逆运算需要 $5d^3$ 次运算操作.当完成矩阵取逆运算后,再进行矩阵积操作,这里的分析思路同参与节点.从而,新节点上的有限域运算次数为

$$GALOIS^{\text{repair}}_{\text{newcomer}}\approx5d^2\left(d+\frac{|size|}{wB}\right)\approx\frac{5d^2}{wB}|size|. \tag{12}$$

可以发现,由于一般情况下 d 远小于$\frac{|size|}{wB}$,其结果基本不受取逆运算的影响,对此将该运算所带来的影响基本可以忽略.

4.2 磁盘读写开销

分布式存储系统中 3 个阶段都会涉及到磁盘的读写操作.其中磁盘读写操作次数的多少不仅直接影响到整个系统的运行性能(例如各阶段运行速度),而且由于每次读写操作都会涉及磁道、磁臂的机械运转,因此较多磁盘读写操作势必增大节点出错甚至失效的概率.另外磁盘的数据吞吐量大小也是直接影响各阶段运行性能的一个重要参数,这里同样会对其进行分析.

对于数据上传过程中,编码节点将原文件$|size|$填充分块,每次读取$|actual_buffer|$大小的数据量进行编码运算,将运算后获得的编码片传递给存储节点,共读取 num 次,因此读操作次数为

$$IO^{\text{upload}}_{\text{read}}=num\approx\frac{|size|}{|buffer|}. \tag{13}$$

在编码节点上由于不涉及写操作,因此:

$$IO^{\text{upload}}_{\text{write}}=0. \tag{14}$$

容易看出,读操作过程中总共读取$|actual_buffer|$ num 个位数据,因此编码节点的磁盘吞吐量为

$$THROUGHTPUT^{\text{upload}}\approx|size|. \tag{15}$$

在数据下载过程中,解码节点每次从收到的 kd 个数据块中分别读取$|slice|$大小的数据片进行数据解码恢复操作,共进行 num 轮,因此总计有 $num\times kd$ 次读操作;在每轮操作之后,将恢复获得的数据片写入磁盘,总计有 num 次.则数据下载过程磁盘读写操作次数为

$$IO^{\text{download}}_{\text{read}}\approx kd\,\frac{|size|}{|buffer|}, \tag{16}$$

$$IO^{\text{download}}_{\text{write}}=num\approx\frac{|size|}{|buffer|}. \tag{17}$$

而磁盘吞吐量由于涉及读和写 2 部分,因此为

$$THROUGHPUT^{\text{download}}\approx2|size|. \tag{18}$$

对于数据修复过程中的参与节点,每一轮从本地的 d 块数据块中分别读取$|slice|$大小的数据片进行运算,将运算获得一片$|slice|$大小的数据片发送给新节点,总共进行 num 轮.因此,每个参与节点读写操作次数以及磁盘吞吐量为

$$IO^{\text{repair}}_{\text{participant\_read}}=d\times num\approx d\,\frac{|size|}{|buffer|}, \tag{19}$$

$$IO^{\text{repair}}_{\text{participant\_write}}=0, \tag{20}$$

$$THROUGHPUT^{\text{repair}}_{\text{participant}}=\frac{d|size|}{B}. \tag{21}$$

对于新节点,每一轮将接收到的 d 片$|slice|$大小的数据片进行运算,随后将生成的 d 片数据片写到磁盘中,共进行 num 次写操作,则新节点上磁盘读操作次数和磁盘吞吐量为

$$IO^{\text{repair}}_{\text{newcomer\_read}}=0, \tag{22}$$

$$IO^{\text{repair}}_{\text{newcomer\_write}}=d\times num\approx d\,\frac{|size|}{|buffer|}, \tag{23}$$

$$THROUGHPUT^{\text{repair}}_{\text{newcomer}}=\frac{d|size|}{B}. \tag{24}$$

5 实验及结果

本节根据第 2 节的矩阵积方法和第 3 节的算法思路,用 C 语言分别实现数据上传、数据下载、数据

修复 3 个阶段,并在 Intel Pentium Dual CPU E2180 的 PC 机上进行实验,其 CPU 主频 2.00 GHz,内存大小 2 GB,磁盘大小 320 GB,磁盘转速为 7 200 rpm,操作系统为 Ubuntu 10.10. 分析不同 CPU 利用率、不同原文件大小、不同缓冲区大小以及不同有限域大小这 4 个参数对于 3 个阶段中 4 类节点运算速度的影响,所有实验过程都进行 10 次取平均值. 这里的运算速度为各类节点中运算数据量与运行时间的比值,其中运算数据量由表 1 中可以直接得出,而运行时间则不仅仅和有限域运算次数有关,也和磁盘读写开销密切相关. 一般的,有限域运算次数越大,磁盘读写次数越大,磁盘吞吐量越大,那么运行时间就会越长. 通过分析以上相关参数对于节点运行速度的影响,从多个方面分析限制整个分布式存储系统性能的瓶颈所在.

固定 1 级参数的取值为 $[n,k,d]=[10,5,7]$,即表示整个分布式存储系统共有 10 个节点,连接任意 5 个节点即可恢复原文件. 当有节点失效时,新节点连接剩余节点中的任意 7 个能够精确恢复出原失效节点上的数据.

5.1 CPU 利用率

固定文件大小 $|size|=30$ MB, $|buffer|=100$ KB, $w=8$,即有限域大小为 $GF(2^8)$,每个符号元素占用 1 B,利用软件 cpulimit[11] 来调节 3 个阶段的 CPU 使用率,其范围为[10%,100%],取值间隔为 10%,如图 1 所示:

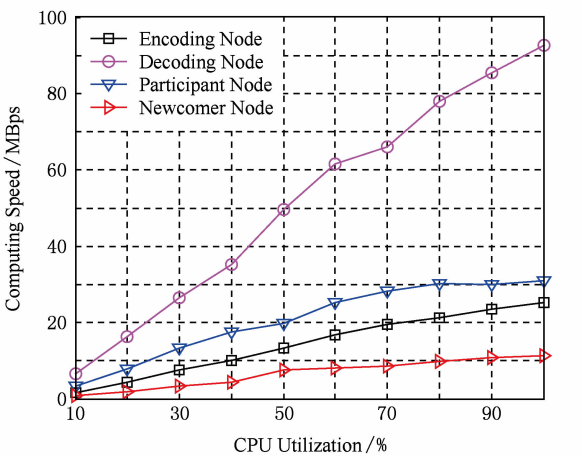


Fig. 1 CPU utilization ratio vs. computing speed.

图 1 CPU 利用率与计算速度

图 1 可知,随着 CPU 利用率的增加,各阶段所涉及的运算速度都有提升. 进一步可以发现,数据下载过程中的编码节点运算速度最快,其原因是下载过程不涉及有限域计算,因此大大降低了运算过程的时间开销. 另外数据修复过程中新节点上的运算速度最慢,根据表 1,新节点中参与运算的数据量最小(为 $d|size|/B$ 位),而有限域运算次数却较大(为 $5d^2|size|/wB$ 次),两者的共同作用导致了其上的运算速度缓慢. 对于数据修复过程中参与节点,其所涉及的运算数据量虽与新节点一致,但由于有限域计算次数少于新节点(为 $5d|size|/wB$ 次),且磁盘读写情况也有所不同(写操作相对于读操作缓慢),因此,其运算速度大于新节点的运算速度.

可以看到,对于随着 CPU 利用率的增加,数据上传过程中编码节点运算速度增长较大,近似线性增长,而对于数据修复过程中参与节点和新节点,运算速度增长缓慢,尤其当 CPU 利用率达 60%以后,曲线逐渐趋于平坦. 这一现象为在实际利用节点搭建分布式存储系统提供了很好的设计思路:对于解码节点,可以利用 CPU 配置较高的机器来代替,这样可以提升数据下载过程中用户的访问速度;而对于一般的存储节点(这其中包括了参与节点和新节点),可以利用 CPU 配置较为廉价低端的机器来代替,这样不但可以满足一般的计算需求(运算速度要求不高),而且很好地降低了整个分布式存储系统的设计成本.

5.2 原文件大小

图 2 中,固定 $|buffer|=100$ KB, $w=8$,令 $|size|$ 的取值范围为[10 MB,100 MB],取值间隔为 10 MB.

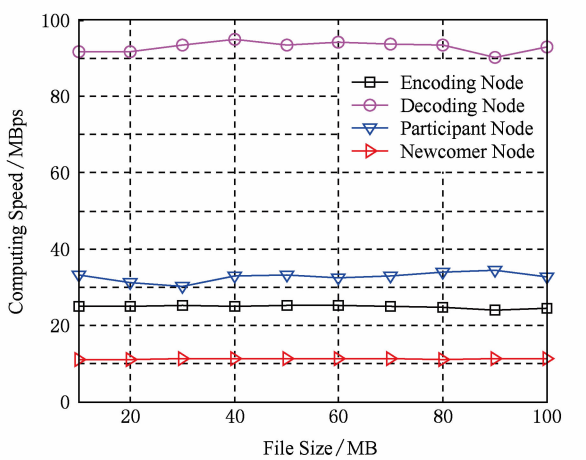


Fig. 2 File size vs. computing speed.

图 2 文件大小与计算速度

可以看出,随着 $|size|$ 的增加,各阶段中对应节点上的运算速度基本保持不变. 这种趋势是显然的,因为在表 1 中,各类节点上的运算数据量与运行时间(受有限域运算次数、磁盘读写次数以及磁盘吞吐量大小影响)都与 $|size|$ 成正比关系,则运算速度与

$|size|$无关.这说明基于矩阵积方法的精确修复最小带宽再生码的分布式系统中各阶段的运算速度与所要编码存储的原文件大小无关,因此该编码方案对于分布式存储系统具有很好的适用性.

此外注意到,相对于编码节点和新节点,解码节点和参与节点上的运算速度具有小幅度的波动(虽然波动不大),其具体原因在于:解码节点的时间开销只与磁盘读写开销有关,而磁盘读写涉及到磁盘的机械运转,运行过程存在一定的不稳定性,因而造成最终的运算速度具有波动现象;对于参与节点,虽然在时间开销中涉及有限域计算,但其有限域计算次数相对较小(为$5d|size|/wB$次,除解码节点外剩余3类节点中最小),因此有限域运算的时间开销对于其运行时间的影响所占的比重相对较小,从而运算速度也有较明显的波动现象.这种由于磁盘机械运转而带来的运算速度波动的现象在其他3个图像中都存在.

5.3 缓冲区大小

图3中,固定文件大小$|size|=30$ MB,$w=8$,$|buffer|$的取值范围从[0 KB,200 KB],取值间隔为5 KB.

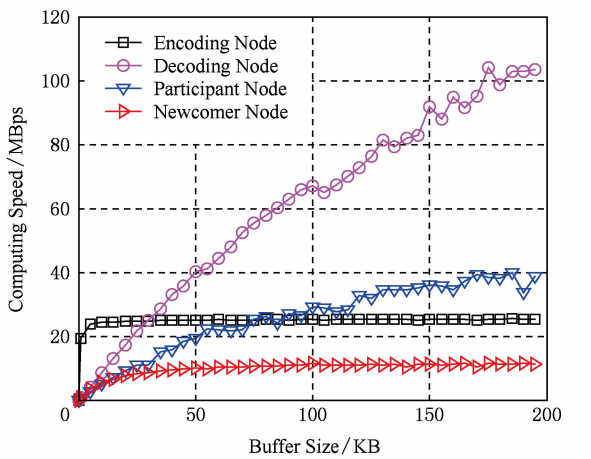


Fig. 3 Buffer size vs. computing speed.

图3 缓冲区大小与计算速度

从表1可以知道,不同$|buffer|$值对于运算速度的影响主要反映在磁盘读写次数的不同.由图3看出,随着$|buffer|$的增大,数据上传过程中编码节点的运算速度迅速上升至某一值后就基本保持不变.数据下载过程中解码节点的运算速度随着$|buffer|$的增大而不断增大.修复过程中的两类节点,随着$|buffer|$增大,参与节点和新节点的运算速度增长较缓慢,而且最终都趋于平坦.

从算法1~3可以看出,整个分布式存储系统的3个阶段中$|buffer|$值需保持一致,这样才能保证整个系统的正常运转.从图3可以看出,这里缓冲区$|buffer|$的设置相对较小(最大为200 KB),因此其内存开销基本不大(相对于2 GB的内存),但其不同的数值却对于解码节点运算速度影响较大.因此,在保证合适的内存开销的前提下,通过增大$|buffer|$的值,能显著提高用户对数据的访问速度.

5.4 有限域大小

图4中,固定文件大小$|size|=30$ MB,$|buffer|=100$ KB.为方便一般机器(例如PC机)进行运算处理,$w=\{8,16,32\}$,使得有限域上元素分别与{1 B,2 B,4 B}相对应.

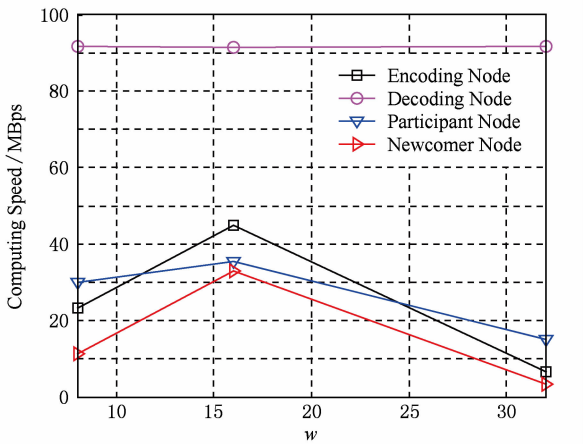


Fig. 4 Galois field size vs. computing speed.

图4 有限域大小与计算速度

可以发现,这里的解码节点的运算速度保持不变,而对于其他3类节点的运算速度当$w=16$时获得最大值.根据表1,解码节点由于不涉及有限域计算,因此不受有限域大小的影响.而对于其他3类节点,随着w变大,相应的有限域次数就会减少;但同时根据4.1.2节的分析可知,w值影响内存中对数表的规模,相应地,随着w值的变大,每一次的查表操作所需的时间开销就会相应增大.两者的相互影响最终反映在对于各类节点的运行时间上,从而呈现出图4的变化趋势.对此,根据图4的提示,在实际设计分布式存储系统时,可以将有限域大小设置为$GF(2^{16})$,这样不但可以方便一般机器进行运算处理(所有元素都为2 B),而且可以有效地提高整个分布式存储系统的运行速度.

6 结束语

本文根据文献[2]中矩阵积的方法实现了精确

修复最小带宽再生码,并就分布式存储系统中所涉及到的数据上传、数据下载、数据修复过程中的计算和磁盘操作开销进行了详细的分析,并通过实验比较了不同 CPU 利用率、不同文件大小和缓冲区大小、以及不同有限域规模对其运算速度的影响,发现了利用矩阵积方法所实现的精确修复最小带宽再生码的分布式存储系统,除了具有构造参数之间无相互制约关系之外,通过对其他相关参数合理的配置,可以使得基于该编码方案所实现的分布式存储系统可以获得一个较好的性能.其主要的最优配置方案:1)根据不同节点的性能需求选择不同 CPU 配置的机器运行;2)尽可能地增大各内存缓冲区大小;3)设置有限域为 $GF(2^{16})$.根据上述要求所搭建的分布式存储系统,不但可以维持数据可靠性和可用性,而且降低数据修复过程的带宽开销,另外可以降低系统成本,且对于不同大小的文件读写具有一致的访问性能.

参 考 文 献

[1] Dimakis A G, Godfrey P G, Wu Y, et al. Network coding for distributed storage systems [J]. IEEE Trans on Information Theory, 2010, 56(9): 4539-4551

[2] Rashmi K, Shah N B, Kumar P V. Optimal exact-regenerating codes for distributed storage at the MSR and MBR points via a product-matrix construction [J]. IEEE Trans on Information Theory, 2011, 57(8): 5227-5239

[3] Ahlswede R, Cai N, Li S Y R, et al. Network information flow [J]. IEEE Trans on Information Theory, 2000, 46(4): 1204-1216

[4] Dimakis A G, Ramchandran K, Wu Y, et al. A survey on network codes for distributed storage [J]. Proceeding of the IEEE, 2011, 99(3): 476-489

[5] Luo Xianghong, Shu Jiwu. Summary of research for erasure code in storage system [J]. Journal of Computer Research and Development, 2012, 49(1): 1-11 (in Chinese)
(罗象宏, 舒继武. 存储系统中的纠删码研究综述[J]. 计算机研究与发展, 2012, 49(1): 1-11)

[6] Duminuco A, Biersack E. A practical study of regenerating codes for peer-to-peer backup systems [C] //Proc of IEEE ICDCS'09. Piscataway, NJ: IEEE, 2009: 376-384

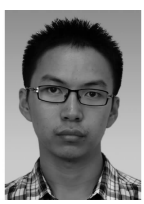
[7] Ho T, Koetter R, Medard M, et al. The benefits of coding over routing in a randomized setting [C] //Proc of IEEE ISIT'03. Piscataway, NJ: IEEE, 2003: 442

[8] Hu Y, Yu C M, Li Y K, et al. Ncfs: On the practicality and extensibility of a network-coding-based distributed file system [C] //Proc of IEEE NetCod'11. Piscataway, NJ: IEEE, 2011: 1-6

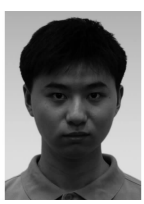
[9] Hu Y, Chen C H, Lee P C, et al. Nccloud: Applying network coding for the storage repair in a cloud-of-clouds [C] //Proc of the 10th USENIX Conf on File and Storage Technologies. Berkeley, CA: USENIX Association, 2012: 21-28

[10] Arnoult E. Study of the feasibility of a distributed storage system based on a minimum bandwidth exact regenerating code [EB/OL]. London: Imperial college London, 2011. [2011-09-01]. http://csi. usc. edu/~ dimakis/StorageWiki/lib/exe/fetch. php? media = wiki: papers: study _ of _ the _ feasibility_of_a_distributed_storage_system_based_on_a_minimum_bandwidth_exact_regenerating_code. pdf

[11] Angelo M. Cpu usage limiter for linux [EB/OL]. (2006-08-09) [2012-05-23]. http://cpulimit. sourceforge. net/



Wei Dongsheng, born in 1989. Master candidate. His main research interests include storage systems and network coding.



Li Jun, born in 1986. PhD candidate. His main research interests include network coding and network storage.

Wang Xin, born in 1973. Professor. Senior member of China Computer Federation. His main research interests include quality of network service, next-generation network architecture, mobile Internet and network coding, etc.