软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2017,28(8):1940−1951 [doi: 10.13328/j.cnki.jos.005204]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

基于简单再生码的带宽感知的分布式存储节点修复优化*

丁 尚[1], 童 鑫[1], 陈 艳[2], 叶保留[1]

[1](计算机软件新技术国家重点实验室(南京大学),江苏 南京 210023)

[2](上海市电力公司,上海 200122)

通讯作者: 叶保留, E-mail: yebl@nju.edu.cn

摘 要: 分布式存储系统为了保证可靠性,会采用一定的存储冗余策略,如多副本策略、纠删码策略.纠删码相对于副本具有存储开销小的优点,但节点修复网络开销大.针对修复网络开销优化,业界提出再生码和以简单再生码为代表的局部可修复码,显著降低了修复网络开销.然而,现有的基于编码的分布式容错存储方案大都假设节点处于星型逻辑网络结构中,忽略了实际的物理网络拓扑结构和带宽信息.为了实现拓扑感知的容错存储优化,相关研究在纠删码和再生码修复过程中结合网络链路带宽能力,建立树型修复路径,进一步提高了修复效率.但是,由于编码和修复过程的差异性,上述工作并不适合于简单再生码修复.针对该问题,结合实际物理网络拓扑结构,将链路带宽能力引入到简单再生码的修复过程中,对带宽感知的简单再生码修复优化技术开展研究.建立了带宽感知节点修复时延模型,提出了基于最优瓶颈路径和最优修复树的并行修复树构建算法,并通过实验对算法性能进行了评估.实验结果表明,与星型修复方式相比,该算法有效地降低了节点修复时延,提高了修复效率.

关键词: 分布式存储;简单再生码;网络拓扑;节点修复

中图法分类号: TP306

中文引用格式: 丁尚,童鑫,陈艳,叶保留.基于简单再生码的带宽感知的分布式存储节点修复优化.软件学报,2017,28(8):1940−1951. http://www.jos.org.cn/1000-9825/5204.htm

英文引用格式: Ding S, Tong X, Chen Y, Ye BL. Bandwidth-Aware node repair optimization for distributed storage system based on simple regenerating code. Ruan Jian Xue Bao/Journal of Software, 2017,28(8):1940−1951 (in Chinese). http://www.jos.org.cn/1000-9825/5204.htm

Bandwidth-Aware Node Repair Optimization for Distributed Storage System Based on Simple Regenerating Code

DING Shang[1], TONG Xin[1], CHEN Yan[2], YE Bao-Liu[1]

[1](National Key Laboratory for Novel Software Technology (Nanjing University), Nanjing 210023, China)

[2](State Grid Shanghai Municipal Electric Power Company, Shanghai 200122, China)

Abstract: In order to ensure the data reliability, fault-tolerant distributed storage systems usually apply some data redundant strategies such as the multi-replicas strategy and the erasure coding strategy. Compared with the multi-replicas strategy, the erasure coding strategy has an advantage of storing less data while costs much more network traffic when repairing a failed node. To reduce the repair network cost, the regenerating code and locally repairable codes were proposed, and simple regenerating code is a representative of the locally repairable codes. However, most of the current fault-tolerant distributed storage methods based on coding strategy assume that the storage nodes are located in the star shaped logic network, ignoring the physical network topology and link bandwidth capacity. So with applying the physical network topology into the repair process of erasure code and regenerating code, some related researches propose methods of building tree shaped repair paths to get further more efficiency for repairing a failed node. But because of the difference in encoding and

* 基金项目: 国家自然科学基金(61373014); 国家电网科技项目(521104170019)
Foundation item: National Natural Science Foundation of China (61373014); R&D Program of State Grid (521104170019)
收稿时间: 2016-08-18; 修改时间: 2016-09-21, 2016-11-11; 采用时间: 2016-12-01; jos 在线出版时间: 2017-01-12
CNKI 网络优先出版: 2017-01-12 10:36:02, http://www.cnki.net/kcms/detail/11.2560.TP.20170112.1036.006.html

repairing process, those methods are not fit for the repair of simple regenerating code. To resolve the problem, this paper introduces the link bandwidth capacity into the repair process of simple regenerating code based on the physical network topology. It builds a bandwidth-aware node repair analysis model, and proposes an algorithm to build parallel repair trees based on the optimal bottleneck path and optimal repair tree methods. Experiments are also designed to evaluate the performance of the algorithm. The result shows that compared with the method based on star shaped network topology, the proposed algorithm reduces the latency of repair process effectively.

Key words: distributed storage; simple regenerating code; network topology; node repair

近年来,随着云计算和大数据以及电子商务、社交网络、视频分享等应用的兴起,面向大规模数据的分布式存储系统(如 Google File System[1])得到越来越多的重视.分布式存储系统大多搭建在大量廉价的商用机器集群上,然而受限于集群中的存储节点性能影响,节点易于失效.因此,如何建立高效的容错存储机制、提升数据存储的可靠性,成为相关技术研究的热点问题.

早期的分布式存储系统大多采用基于多副本的冗余策略,将数据块的多个副本放置在不同节点上.当某个副本失效时,系统从另外的副本进行简单的拷贝即可恢复.多副本策略简单易用,但副本的冗余存储开销巨大.针对该问题,业界提出纠删码策略.典型的纠删码具有 Maximum Distance Separable 性质[2].*MDS*(*n*,*k*)性质,是将原始文件切分成 *k* 个原始数据块,对其编码,形成 *n* 个编码块,编码块存储到 *n* 个不同节点.当需要恢复原始数据时,从任意 *k* 个节点上获取 *k* 个编码块,对其进行解码,即可恢复出原始数据.相对于多副本,在同等容错能力下,纠删码的存储开销要低很多.但在修复失效数据时,纠删码需要从 *k* 个节点上传输 *k* 个数据块.与多副本策略只传输 1 个数据块相比,纠删码策略增加了网络开销.

为了缓解网络开销,Dimakis 等人[3]首次将网络编码引入分布式存储系统,提出了再生码.再生码中,每个节点存储由 *a* 个编码段组成的编码块.修复时,首先在提供恢复数据的节点上对编码段进行线性组合,得到由 *b*(*b* 远小于 *a*)个修复段组成的修复块,然后将修复块传输给新生节点.随后,新生节点对收集到的修复块进行解码,从而得到失效的编码块.虽然连接节点个数大于 *k*,但再生码无需下载整个编码块,因此,再生码从根本上减少了修复传输数据量.

为了减少纠删码修复时的数据传输开销,学术界还提出了基于分组的局部可修复码.在此策略中,原始文件切分的数据块被平均划分成多个互不重叠的小组,小组内使用 MDS 编码生成局部的冗余块.另外,所有数据块使用 MDS 编码生成全局的冗余块.当有少量数据块丢失时,可以利用小组内的冗余块进行修复,显著减少了数据下载量;只有当较多数据块丢失导致小组内冗余块不足以完成修复时,才使用全局冗余块来进行修复.基于分组的局部可修复码通过增加少量局部冗余块降低修复数据量,从而缓解网络传输开销.

简单再生码 [4]是一种简单却高效的局部可修复码.在(*n*,*k*,*f*)简单再生码中,每个丢失数据的节点通过从最多 2*f* 个特定节点上获取数据块来修复(*f* 可以远小于 *k*),因而简单再生码具有修复参与节点数目少、传输数据量小的优点,且保持了 *MDS*(*n*,*k*)性质,即可以保证利用 *n* 个存储节点中的任意 *k* 个节点上存储的数据块,能够恢复出原始数据.

现有基于网络编码的分布式容错存储方案大都假设节点处于星型逻辑网络结构中,参与节点通过直接链路向丢失节点传递所需的数据块.然而在实际的分布式存储系统中,数据节点往往按照一定的拓扑结构进行分布(如图 1 所示),如 FAT-tree[5],DCell[6]等,并且不同数据节点之间的网络带宽往往也不同.为了实现拓扑感知的容错存储优化,Li 等人[7]在再生码修复过程结合网络链路带宽能力,在中间节点上进行部分解码操作,设计出面向再生码数据块的树型修复模型.Sun 等人[8]在基于纠删码的分布式存储系统中,利用类似的树型修复路径来并行修复多个失效节点.其提出的树型并发修复方法能够提高多节点修复时的网络利用率,减少多节点修复所需的修复时延.

现有基于纠删码与再生码的拓扑感知的优化编码修复算法都假设每个存储节点上存储一个编码块.节点失效后,利用所有修复参与节点构建一条树型修复路径,即可完成失效节点的修复.但该假设在简单再生码修复中不再成立.在简单再生码修复中,每个节点存储多个数据块,节点失效后,其上所有丢失的数据块都需要被恢

复.与此同时,修复一个丢失数据块所需的参与节点固定,而非任意节点都可提供修复所需的数据块.

针对上述问题,本文结合实际物理网络拓扑结构,将链路带宽能力引入到简单再生码的修复过程中,对带宽感知的简单再生码修复优化技术开展了研究.本文首先分析了可行性,建立了节点修复时延评估模型,在此基础上提出了基于最优瓶颈路径和最优修复树的并行修复树构建算法,并对算法的有效性进行了理论分析.本文还通过实验对该算法性能进行了评估.实验结果表明,与星型修复方式相比,本文提出的算法有效地降低了节点修复时延,提高了修复效率.

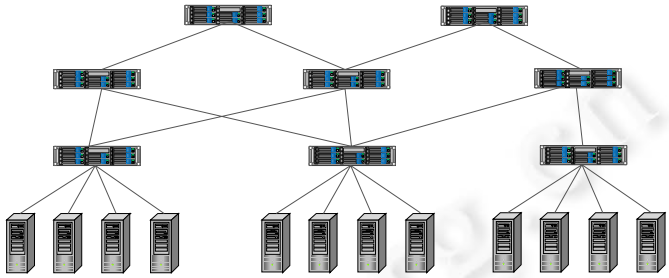


Fig.1 Network in real world

图 1 实际网络模型

本文第 1 节简单介绍必要的简单再生码编码和修复基础知识,并通过实例分析网络拓扑结构对简单再生码修复性能的影响.第 2 节介绍简单再生码中利用网络拓扑信息构建并行修复树问题的定义和形式化.第 3 节介绍基于最优瓶颈路径和最优修复树的并行修复树构建算法.第 4 节设计实验对比分析本文提出算法和星型直接修复算法的性能差别.第 5 节对全文加以概括总结,并对进一步研究提出展望.

1 基于简单再生码的存储节点修复问题描述

1.1 (*n*,*k*,*f*)简单再生码的数据编码与修复过程

基于简单再生码的数据编码过程如下.

1) 将待写入平均分割成 f 个部分.
2) 对 f 个部分分别用 $MDS(n,k)$纠删码进行编码,得到 f 组数据块,每组由 n 个编码后的数据块 $x_1^{(i)}, x_2^{(i)},..., x_n^{(i)}$ $(1\le i\le f)$组成.
3) 将 f 组中下标相同的数据块进行异或运算,得到一组校验数据块$[s_1,s_2,s_3,\ldots,s_{n-1},s_n]$,其中, $s_j = x_j^{(1)} \oplus x_j^{(2)} \oplus ... \oplus x_j^{(n)}$ $(1\le j\le f)$.
4) 将步骤 2)和步骤 3)得到的 f+1 组数据共$(f+1)\times n$ 块数据块,按照一定的规则放置到编号从 1 到 n 的存储节点中.放置规则要求:每个节点上存放 f+1 个数据块,放置到同一节点上的数据块不能属于同一组,即下标不能重复,如图 2 所示为放置情况.这里,⊕代表加法,并对 n 取模.

node 1	node 2	...	node $n-1$	node n
$x_1^{(1)}$	$x_2^{(1)}$	...	$x_{n-1}^{(1)}$	$x_n^{(1)}$
$x_2^{(2)}$	$x_3^{(2)}$	...	$x_n^{(2)}$	$x_1^{(2)}$
$x_3^{(3)}$	$x_4^{(3)}$	...	$x_1^{(3)}$	$x_2^{(3)}$
⋮	⋮	⋮	⋮	⋮
$x_f^{(f)}$	$x_{f\oplus 1}^{(f)}$	...	$x_{f\oplus(n-2)}^{(f)}$	$x_{f\oplus(n-1)}^{(f)}$
$s_{f\oplus 1}$	$s_{f\oplus 2}$	...	$s_{f\oplus(n-1)}$	$s_{f\oplus n}$

Fig.2 Blocks distribution in simple regenerating code

图 2 简单再生码各个节点所存数据块图

假设 n 个存储节点中的第 i 个节点失效,其上存储的数据块 $x_i^{(1)},x_{i\oplus 1}^{(2)},\ldots,x_{i\oplus(f-1)}^{(f)},s_{i\oplus f}$ 发生了丢失.以 $x_i^{(1)}$ 的修复过程为例.

从节点 i–1 上获取数据块 $x_i^{(2)}$,从节点 i–2 上获取数据块 $x_i^{(3)}$,…,从节点 i–(f–1)上获取数据块 $x_i^{(f)}$,从节点 i–f 上获取数据块 s_i.经由异或运算得到 $x_i^{(1)}$.

从数据块 $x_i^{(1)}$ 的修复过程可以看出,每个数据块的修复需要固定的 f 个参与节点提供固定的数据块.在失效节点上的所有 f+1 个数据块的修复过程中,每个数据块修复所需的 f 个参与节点集合会有重叠,并且修复节点上所有丢失数据块所需的参与节点数目为 $\min(2f,n-1)$.

1.2 网络拓扑结构对简单再生码修复性能影响的分析

研究网络拓扑结构对简单再生码修复性能的影响时,数据恢复与数据服务的冲突是需要分析的内容.因此,考虑到数据服务所产生的数据流量,本文中使用的链路带宽指减去数据服务产生的数据流量而剩余的可用链路带宽,而不是链路的完全带宽.

图 3 表示物理网络拓扑结构,假设节点 i 上的数据块全部发生了丢失,修复时,节点 i–2,i–1,i+1,i+2 要向节点 i 传递相应的数据块以进行数据块恢复运算,假设每个数据块大小为 m.

图 4 给出了传统星型逻辑网络直接修复过程,简称星型直接修复.数据块通过直接链路传递的情况下,如图 4 中实线带箭头链路所示.

1) 数据块 s_i 通过 10KB/s 的链路从节点 i–2 上发送到节点 i 上,耗时 m/10;
2) 数据块 $y_i,s_{i\oplus 1}$ 通过 40KB/s 的链路从节点 i–1 上发送到节点 i 上,耗时 2m/40;
3) 数据块 $x_{i\oplus 1},y_{i\oplus 2}$ 通过 50KB/s 的链路从节点 i+1 上发送到节点 i 上,耗时 2m/50;
4) 数据块 $x_{i\oplus 2}$ 通过 30KB/s 的链路从节点 i+2 上发送到节点 i 上,耗时 m/30.

只有当所有所需数据块都发送到节点 i 上时,丢失的数据块才能够被恢复.因此,节点 i 的修复时延为

$$\max\{m/10,2m/40,2m/50,m/30\}=m/10.$$

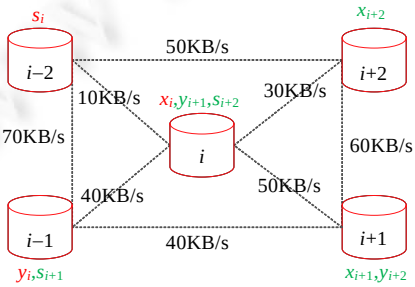


Fig.3 Physical network topology

图 3 物理网络拓扑结构

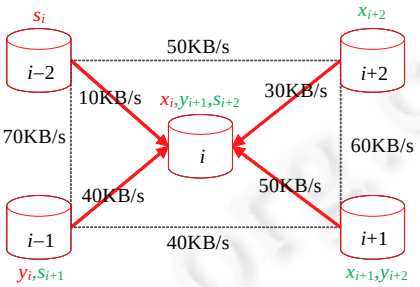


Fig.4 Star topology repair

图 4 星型直接修复

如图 3 所示,在实际物理网络中,节点之间的链路带宽往往是不同的.因此,与前面提到的再生码树形修复[7]和并行纠删码树形修复[8]类似,在数据块传递的过程中,可以为特定数据块选择合适的传输路径来避免那些带宽较低的链路,从而减少传输时延.在中间节点上,用接收到的部分数据包与中间节点上的相应数据,立刻进行数据恢复运算,然后转发出去.由于简单再生码中数据块恢复运算主要是异或运算,运算时间占传输时间比重很小,可以忽略.因此在此流水线式恢复模式下,一个数据块传输的时延主要取决于数据块传输链路上的最小带宽.多个数据块的修复并行进行,因此,一个数据节点的修复时延取决于修复最慢的数据块.

图 5 中考虑到实际物理网络拓扑结构和带宽信息,利用中间节点的编解码能力,为简单再生码设计的多个丢失数据块树型修复路径,简称并行树型修复.图 5 中数据块传递的情况描述如下.

1) 如实线带箭头路径所示,数据块 s_i 通过 70KB/s 的链路传输到节点 i–1,与其上的数据块 y_i 进行异或运

算得到数据块 x_i,数据块 x_i 通过 40KB/s 的链路传输到节点 i,耗时 $m/40$.

2) 如虚线带箭头路径所示,数据块 $s_{i\oplus1}$ 通过 40KB/s 的链路传输到节点 $i+1$,与其上的数据块 $x_{i\oplus1}$ 进行异或运算得到数据块 $y_{i\oplus1}$,数据块 $y_{i\oplus1}$ 通过 50KB/s 的链路传输到节点 i,耗时 $m/40$.

3) 如点划线带箭头路径所示,数据块 $y_{i\oplus2}$ 通过 60KB/s 的链路传输到节点 $i+2$,与其上的数据块 $x_{i\oplus2}$ 进行异或运算得到数据块 $s_{i\oplus2}$,数据块 $s_{i\oplus2}$ 通过 30KB/s 的链路传输到节点 i,耗时 $m/30$.

这 3 条传输链路中的数据传输相互独立,故节点 i 的修复耗时为 $\max\{m/40,m/40,m/30\}=m/30$.

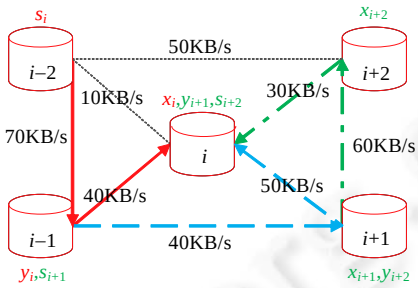


Fig.5 Tree topology repair

图 5 树型并行修复

从以上分析可知:并行树型修复时延 $m/30$,比星型直接传输路径的时延 $m/10$ 要小很多.可见,不同逻辑网络拓扑结构对简单再生码修复性能有明显差异.在简单再生码的单节点修复过程中,通过合理地利用实际网络拓扑和链路带宽信息,使用中间节点的编码运算能力,构建针对待修复数据块的树型修复路径,可以有效降低修复时延.

2 带宽感知的简单再生码修复问题建模

2.1 带宽感知的简单再生码修复模型

本文构建(n,k,f)简单再生码系统网络模型时,以实际系统中拓扑结构为基础进行了合理的简化,构建出完全图网络模型.当实际系统中物理链路不存在时,完全图中对应的边权重为 0 即可.而为了进一步简化繁琐的分析过程,突出本文算法的核心,模型假定该图模型为无向图,即两节点之间只附属 1 个带宽权重.

在(n,k,f)简单再生码系统中,一个节点失效时,修复所需要的参与节点数目最多为 $2f$ 个,因此,修复网络中总共涉及 $2f+1$ 个存储节点:一个为丢失了全部数据块的失效节点,其他为修复参与节点.此 $2f+1$ 个节点之间通过网络链路连接成完全图 $G=(N,E)$,其中 N 为节点集合,满足$|N|=2f+1$;E 为链路集合,满足$|E|=f(2f+1)$,且 E 中的每条链路都关联一个带宽 $C(e),e\in E$.

每个丢失数据块的修复过程对应于一棵修复树.失效节点为该修复树的根节点,修复树包含数据块修复所要求的 f 个参与节点.一个失效节点上共有 $f+1$ 个丢失数据块被独立地修复,因此,修复一个失效节点共有 $f+1$ 棵独立的修复树.修复树集合记作 F,$|F|=f+1$.F 中的第 i 棵树 $F_i(1\le i\le f+1)$对应于失效节点上的第 i 个丢失数据块.

在一棵修复树的数据块恢复过程中,修复树中的子节点向父节点发送恢复所需的数据块.父节点在接收到子节点们发来数据块的部分字节后,用自身存储的数据块中对应部分的字节与其做异或运算,然后将得到的数据转发给自身的父节点,直到根节点.最终,根节点恢复出数据块.

在一棵修复树所涉及的网络链路中,带宽最小的链路为该修复树中的数据块修复瓶颈链路,在前述流水线式数据恢复过程中,该链路限制着数据块的整体修复时间.对于修复树 $F_i(1\le i\le f+1)$,其中的瓶颈链路的带宽为 $w_i(1\le i\le f+1)$,则整个修复树中的数据块修复时间为 $t_i=m/w_i(1\le i\le f+1)$,其中,m 为单个数据块的大小.给定单个失效节点,修复过程中的 $f+1$ 棵修复树,节点的修复时间 T 为 $f+1$ 棵修复树中最长的修复时间,即 $T=\max t_i(1\le i\le f+1)$.问题在于,给定由一个失效节点及其修复参与节点组成的网络,规划 $f+1$ 棵修复树,使得失效节点的修复时

间最小,即 min$T$,等价于 min$\max t_i(1\le i\le f+1)$,等价于 min$\max w_i/m(1\le i\le f+1)$,等价于 max$\min w_i(1\le i\le f+1)$,即:规划出 f+1 棵以失效节点为根的修复树,使这些修复树中的最小瓶颈链路的带宽最大.

2.2 带宽感知的简单再生码修复问题形式化

输入:(n,k,f)简单再生码中的单个失效节点和 2f 个修复参与节点组成的网络图 G=(N,E).

目标:构建 f+1 棵以失效节点为根的修复树,每棵修复树 $F_i(1\le i\le f+1)$中的瓶颈链路带宽为 $w_i(1\le i\le f+1)$,要求这 f+1 修复树中的最小瓶颈链路带宽最大,即 max$\min w_k(1\le k\le f+1)$.

约束:

$$Y_i^k=\begin{cases}1, & N(i)\in F_k\\ 0, & N(i)\notin F_k\end{cases}(1\le i\le 2f+1,1\le k\le f+1) \tag{1}$$

$$X_{i,j}^k=\begin{cases}1, & (i,j)\in F_k\\ 0, & (i,j)\notin F_k\end{cases}(1\le i\le 2f+1,1\le j\le 2f+1,1\le k\le f+1) \tag{2}$$

$$Y_i^k\ge X_{i,j}^k(1\le i\le 2f+1,1\le j\le 2f+1,1\le k\le f+1) \tag{3}$$

$$\sum_{i=1}^{2f+1}Y_i^k=\sum_{i=1}^{2f+1}\sum_{j=1}^{2f+1}X_{i,j}^k+1(1\le k\le f+1) \tag{4}$$

$$\sum_{k=1}^{f+1}X_{i,j}^k\times w_k\le C(i,j)(1\le i\le 2f+1,1\le j\le 2f+1) \tag{5}$$

约束条件(1)表示修复树 $F_k(1\le k\le f+1)$是否经过网络图中的节点 i:若 $F_k(1\le k\le f+1)$经过节点 i, Y_i^k=1;否则, Y_i^k=0.

约束条件(2)表示修复树 $F_k(1\le k\le f+1)$是否通过节点 i 与节点 j 之间的链路:若 $F_k(1\le k\le f+1)$通过节点 i 与节点 j 间的链路, $X_{i,j}^k$=1;否则, $X_{i,j}^k$=0.

约束条件(3)基于这样的观察:若节点 i 与节点 j 之间的链路在修复树 $F_k(1\le k\le f+1)$通过,则节点 i 与节点 j 一定出现在修复树 $F_k(1\le k\le f+1)$中.

约束条件(4)基于这样的观察:一个图是一棵树,当且仅当图中节点的数量比边的数量大 1.这样的约束条件保证构造出来的数据块修复路径可以构成一棵修复树.

约束条件(5)表示通过一条网络链路的所有修复树的修复带宽开销不超过该链路的总带宽.

3 带宽感知的简单再生码修复算法

3.1 算法设计思路

基于最优瓶颈路径和最优修复树构建针对所有待修复数据块的修复树,使得每块丢失数据修复时尽可能地利用带宽资源充足的链路.通过降低每个丢失数据块的恢复时延来达到降低节点修复时延的目的.

给定一个(n,k,f)简单再生码的节点修复网络,首先构建从失效节点到其他节点的最优瓶颈路径,基于此构建出 f+1 棵对应于失效节点上 f+1 块丢失数据块的最优修复树.在这 f+1 棵构建好的最优修复树中,按照一定的策略选择第 1 棵合适的修复树.将选定的修复树中瓶颈链路的带宽分配给该修复树,用作对应数据块的修复,并从选定的修复树通过的链路中减去分配出去的带宽.然后,在剩余的网络图上构建 f 棵对应于失效节点上 f 个待修复数据块,再从中按照一定的策略选择出第 2 棵合适的修复树.将选择的修复树的瓶颈链路的带宽分配它,并从其通过的链路中减去分配出去的带宽.这样的过程一直重复,直到分配出 f+1 棵修复树,完成对应 f+1 个丢失数据块的修复.

下面按照算法使用的先后顺序依次介绍最优瓶颈路径、最优修复树和并行修复树构造算法.

3.2 最优瓶颈路径

定义 **1**. 给定一个目标节点 t,所有以源节点为起点、t 为终点的简单路径的集合为 P_t,P_t 中瓶颈带宽最大的

那条简单路径为从根节点到 t 的最优瓶颈路径[9],即

$$\max_{p\in P_t}\min_{e\in p}\{C(e)\} \tag{6}$$

最优瓶颈路径的求解,可以使用 Dijkstra 算法[10]的一个修改版本,如算法 1 所示.

算法 **1**.

输入:图 G,图 G 中每条边 e 有一个带宽权重 $C(e)(C(e)\geq 0)$,源节点 s.

输出:s 到其他节点的最优瓶颈路径.

初始化:$D=\{s\}$,定义从源节点 s 到节点 x 的最优瓶颈路径的瓶颈带宽为 $widthto(x)$,节点 s 到其自身的瓶颈带宽 $widthto(s)$为∞.

步骤:

1) 若所有元素已加入集合 D,结束;否则,对于 D 中所有节点的每一个未加入到 D 中的邻居节点 x,利用下面的公式收缩从源节点 s 到节点 x 的瓶颈路径的瓶颈带宽:

$$widthto(x)=\max_{e=(v,x);v\in D}\{\min\{widthto(v),C(e)\}\}.$$

2) 将步骤 1)中得到的具有最大瓶颈带宽的节点 x 加入到集合 D 中.原本在集合 D 中的 x 的前驱节点 v 即为从源节点 s 到 x 的最优瓶颈路径的前驱节点.执行步骤 1).

定理 **1**. 通过算法 1,可以得到从源节点到各个节点的最优瓶颈路径.

证明:证明算法 1 的正确性,只需证明不变量:“被加入到集合 D 中的节点都已经找到从节点 s 出发的最优瓶颈路径”成立.

设从节点 s 到节点 v 的最优瓶颈路径的可用带宽为$\delta(s,v)$,则不变量等价于:对于 $v\in D$,有 $widthto(v)=\delta(s,v)$.

初始化:$D=\{s\}$,由于 s 到自身的可用带宽为∞,$widthto(s)=\delta(s,s)=\infty$成立.

保持:希望说明在每一轮迭代中,对加入到集合 D 中的顶点 u,都有 $widthto(u)=\delta(s,u)$.利用反证法,假设 u 为加入集合 D 中的第 1 个满足 $widthto(u)\neq\delta(s,u)$的顶点.由于 $widthto(u)\neq\delta(s,u)$,可得知 $u\neq s$.因为从 u 到 s 至少存在 1 条路径,那么从 s 到 u 存在一条最优瓶颈路径 p.在 u 加入 D 之前,路径 p 连接着 D 中的点 s 与 $N–D$ 中的点 u.设顶点 y 为路径 p 第 1 个属于 $N–D$ 的顶点,并设 $x\in D$ 为 y 的前驱节点.p 被分割为

$$s\overset{p_1}{\Rightarrow}x\rightarrow y\overset{p_2}{\Rightarrow}u\ (\text{路径 } p_1 \text{ 和 } p_2 \text{ 可能没有边}).$$

因为在 s 到 u 的最优瓶颈路径上,y 在 u 之前出现,由最优瓶颈带宽的性质可知,$widthto(y)\geq\delta(s,u)$.可知,$widthto(y)\geq\delta(s,u)\geq widthto(u)$.而选择 u 加入 D 时,顶点 u 与顶点 y 都在 $N–D$ 中,可知 $widthto(u)\geq widthto(y)$.由此可得到 $widthto(u)=\delta(s,u)$.这与当初节点 u 的选择相矛盾.所以当 u 被加入集合 D 时,$widthto(u)=\delta(s,u)$.

终止:算法结束后,所有节点都被加入集合 D 中,对于所有顶点 $u\in D$,$widthto(u)=\delta(s,u)$. □

3.3 最优修复树

在(n,k,f)简单再生码的节点修复网络中,一棵最优修复树以失效节点为根,其描述的是失效节点上一个丢失数据块的修复过程所需要的数据块的传递路径.

定义 **2**. 一个丢失数据块修复过程所需的所有参与节点集为 V,定义 T_V 为所有以丢失节点为根,包含 V 中所有节点的修复树集合,则最优修复树为 T_V 中瓶颈链路带宽最大的那棵修复树.

构建一棵最优修复树的算法如算法 2 所示.

算法 **2**.

输入:图 G,图 G 中每条边 e 有一个带宽权重 $C(e)(C(e)\geq 0)$,源节点 s,参与节点集合 V.

输出:最优修复树 T_V^*.

步骤:

1) 使用算法 1,生成从根节点到 V 中每一个节点的最优瓶颈路径.

2) 合并步骤 1)中生成的所有最优瓶颈路径,合并过程如下:

- 由源节点 s 出发到节点 i 的最优瓶颈路径 p_i:$s,n_1,n_2,\ldots,n_k,\ldots,i$;

- 由源节点 s 出发到节点 j 的最优瓶颈路径 p_j:$s,m_1,m_2,\ldots,m_t,\ldots,j$;
- 假设两条路径在各自的点 n_k 与 m_t 发生交叉,即 $n_k=m_t,n_{k-1}!=m_{t-1}$,则合并之后的树型结构:选择 $s,n_1,n_2,\ldots,n_k$ 或 $s,m_1,m_2,\ldots,m_t$ 中可用带宽较大的链路作为公共链路,p_i,p_j 上 n_k 与 m_t 之后的链路作为子链路,分别链接到公共链路之后.

定理 **2**. 由算法 2 产生的树 T_V^* 是以源节点为根,包含 V 中所有节点的最优修复树.

证明:定义以源节点为根,生成节点集合 V 中所有节点的生成树为 T_V.易知 $T_V^*\in T_V$.则 T_V^* 中的瓶颈链路的带宽一定不大于 T_V 中所有修复树中瓶颈链路带宽最大的那棵修复树的瓶颈链路的带宽,即

$$\min_{e\in T_V^*}\{C(e)\}\le\max_{T\in T_V}\min_{e\in T}\{C(e)\} \tag{7}$$

设 T_V^* 中的瓶颈链路带宽小于 T_V 中所有修复树中瓶颈链路带宽最大的那棵修复树的瓶颈链路的带宽,即

$$\min_{e\in T_V^*}\{C(e)\}<\max_{T\in T_V}\min_{e\in T}\{C(e)\} \tag{8}$$

这意味着 T_V^* 不是以源节点为根、生成 V 中所有节点的最优修复树.

那么,在 T_V 中存在一棵修复树 T_s,使得 T_V^* 瓶颈链路小于 T_s 的瓶颈链路,即

$$\min_{e\in T_V^*}\{C(e)\}<\min_{e\in T_S}\{C(e)\} \tag{9}$$

假设 e^*为 T_V^* 中的瓶颈链路,即

$$C(e^*)=\min_{e\in T_V^*}\{C(e)\} \tag{10}$$

在节点集合 V 中存在一个节点 t,在 T_V^* 中由根节点出发到达节点 t 的路径$^*tp$ 经过瓶颈链路 e^*;同时,在 T_s 中,也存在一条从根节点出发到达 t 的路径 q.

由公式(9)可知:

$$e^*=\min_{e\in p_t^*}\{C(e)\}=\min_{e\in T_V^*}\{C(e)\}<\min_{e\in T_S}\{C(e)\}\le\min_{e\in q}\{C(e)\} \tag{11}$$

即路径 q 上的瓶颈链路大于 pt^*上的瓶颈链路.而 q 和 pt^*都是从根节点到节点 t 的路径,这与“pt^*是由源节点出发、到达节点 t 的最优瓶颈路径”这一性质相冲突.该性质是由 T_V^* 的定义所决定的.

因此,公式(7)中的小于号不成立,即

$$\min_{e\in T_V^*}\{C(e)\}=\max_{T\in T_V}\min_{e\in T}\{C(e)\} \tag{12}$$

即 T_V^* 为从根节点出发、生成 V 中所有节点的最优修复树. □

3.4 并行修复树构造算法

在给出了针对失效节点上的单个丢失数据块的最优修复树构造算法后,可以进行针对失效节点上所有丢失数据块的修复路径的构造,如算法 3 所示.

算法 **3**.

输入:(n,k,f)简单再生码中的一个失效节点和 $2f$ 个修复参与节点组成的网络图 $G=(N,E),N=2f+1,E=f(2f+1)$.图中每条边上的带宽 $C(e),e\in E$;丢失节点中待修复数据块下标集合 L;每个待修复数据块修复过程所需的参与节点集合信息 $V_i(1\le i\le f+1)$,每个参与节点集合内包含 f 个元素,分别代表单个数据块修复所需的 f 个参与节点;失效节点下标 s.

输出:失效节点上 $f+1$ 个数据块并行修复的修复树路径.

初始化:L 中初始化为失效节点中所有丢失数据块的下标.

步骤:

1) 若 L 中无元素,则结束;否则,利用算法 1 和 L 中元素对应的待修复数据块所需的参与节点集合信息 V_i,运用算法 2,得到针对 L 中每个待修复数据块的最优修复树 OT_i.
2) 将步骤 1)中得到的$|L|$个最优修复树按照一定的规则排序,得到优先级最高的 OT_s.

3) 为步骤 2)中得到的 OT_s 分配相应的瓶颈链路带宽 B,用于修复 OT_s 对应的数据块.对于在 OT_s 中出现的链路,即 $e\in OT_s\cap G, C(e)=C(e)-B$.从 L 中删除 OT_s 对应的数据块的下标.执行步骤 1).

排序策略

假设一棵最优修复树 OT 有 d 条边,每条边所在的链路都有一个可用带宽值.在对所有边按照边所在链路的可用带宽升序排列后,得到 d 条边的一个排列 $b_1\le b_2\le b_3\le\ldots\le b_d$.由定义可知,$b_1$ 为该修复树的带宽瓶颈链路.对多棵最优修复树按 b_1 的升序排序.在对构造好的多棵最优修复树进行排序的策略主要基于:

1) 在算法的一次迭代中,每一个待修复的数据块都对应一棵在当前网络带宽情况下的最优修复树,每一棵最优修复树都有着它的瓶颈带宽.
2) 针对一个特定待修复文件块的最优修复树的瓶颈带宽,随着网络链路的带宽资源在算法的早期迭代轮次中被分配给其他丢失数据块的最优修复树,会变得越来越小.这是由最优修复树的定义性质与其构造算法决定的.
3) 在相互独立的数据块的修复过程并行化的情况下,一个失效节点的整体修复时间,是由节点上所有丢失数据块的修复时间中最长的修复时间决定的.
4) 同一网络带宽情况下的两个最优修复树,其中一棵修复树涉及链路的数量比另一棵修复树涉及链路的数量少,且根据带宽对链路升序排序之后,涉及链路较少的最优修复树的链路带宽序列为涉及链路交大的最优修复树的链路带宽序列的前缀,则说明在相同的链路可用带宽开销下,涉及链路数目较小的最优修复树对网络整体带宽的开销更小一些.

给定两棵对应不同待修复文件块的修复树 OT_1 和 OT_2,假设两棵树分别经过网络中的 n 和 m 条链路,则 $OT_1<OT_2$ 当且仅当存在 $k(k\le n,k\le m)$,使得 $b_1^{OT_1}==b_1^{OT_2},b_2^{OT_1}==b_2^{OT_2},\ldots,b_{k-1}^{OT_1}==b_{k-1}^{OT_2},b_k^{OT_1}==b_k^{OT_2},\ldots$;或 $n<m$ 且 $b_1^{OT_1}==b_1^{OT_2},b_2^{OT_1}==b_2^{OT_2},\ldots,b_n^{OT_1}==b_n^{OT_2}$.

4 实验与分析

在实验中,我们主要考察简单再生码中的两种节点修复方式:不考虑网络链路拓扑与带宽信息的星型直接修复方式与本文提出的利用网络链路拓扑与带宽信息的并行树型修复方式.

实验主要考察修复过程的性能指标为修复时延:一个失效节点从修复过程开始到所有丢失数据块都得到恢复所需时延.

在由失效节点和参与节点组成的网络中,运行星型直接修复算法与并行树型修复算法.网络中的链路带宽分布遵守 PlanetLab 网络中的链路带宽分布[11],见表 1.编码后的数据块每个大小设置为 10Mb.

Table 1 End-to-End available bandwidth distribution

表 **1** 端到端可用带宽分布

Capacity(C) (Mb/s)	Number of paths	Percentage (%)
$C<20$	6 733	30.8
$20\le C<50$	1 910	8.74
$50\le C<80$	1 303	5.96
$80\le C<120$	11 744	53.72
$120\le C<200$	139	0.64
$200\le C<500$	21	0.096
$500\le C$	11	0.05

分别在参数为(5,3,2),(10,8,3),(17,13,7)的简单再生码节点修复网络中进行树型修复算法与直接修复算法的对比.每组对比进行 20 次.在每次的对比过程中,对网络中的链路按照所述分布分配带宽,对比两种算法在相同网络状况下的修复时延.

对比分析图 6~图 8 的实验结果,可以得出以下结论.

1) 在相同编码参数的节点修复网络中,并行树型修复方法的修复时延明显要小于星型直接修复方法的修

复时延.

由于在相同的网络链路状况下,针对特定的丢失数据块,并行树型修复方法会采用贪心策略为修复所需的数据块规划传输带宽尽可能大的路径,降低丢失数据块修复过程中的传输时延.

星型直接修复方法中,修复参与节点通过直接链路向待修复节点传输所有参与数据块.在这种方式下,传输时延被直接链路所决定,并且链路上传输的数据块数量也随着节点上参与数据块数量不同而变化.当直接链路的可用带宽资源不够时,会明显增加节点的修复时延.因为在直接修复方法中,只有当所有参与节点都将其上存储的数据块传输给待修复节点时,待修复节点才可以恢复出所有丢失数据数据块.

2) 在相同编码参数的节点修复网络中,随着分配给链路的带宽变化,星型直接修复方法的修复时延波动表现得非常大,而并行树型修复方法的修复时延表现得相对更加稳定.

由于在并行树型修复方法中,每一步构建多棵最优修复树,从中按照所述策略选择一棵最优修复树作为特定数据块的修复路径.由最优修复树的定义与构建算法可知,每次选取的修复树都是当前网络状况中针对特定的丢失数据块的最优修复路径.在这种修复路径构建策略下,当直接链路的带宽较小时,直接链路会被排除在修复路径之外.而在星型直接修复方法中,直接链路的带宽直接限制节点的修复时延.

实验中构建的节点修复网络中的链路带宽按照所述分布进行分配,直接链路的带宽值有着很大的随机性.表现在实验结果中即为:在相同参数下的节点修复网络中,随着节点间链路带宽分配的不同,星型直接修复方法中的修复时延波动很大.

相对来说,由于在并行树型修复方法中待修复数据块的修复路径构建过程会规避带宽较小的链路,这使得链路带宽分布的随机性对数据块的修复路径瓶颈带宽的影响不是特别大.表现在实验结果中即为:在相同参数下的节点修复网络中,随着节点间链路带宽分配的不同,并行树型修复方法中的修复时延波动与星型直接修复方法中的修复时延波动相比较小.

3) 随着编码参数的增大,星型直接修复方下的修复时延与并行树型修复方法的修复时延都逐渐增大.

待修复节点的修复过程在所有丢失数据块都被恢复时结束.随着编码参数的增大,待修复节点修复过程所需的参与节点数目也增加.修复网络中,节点之间的链路带宽的分配符合同样的分布.修复过程中的参与节点数据越多,修复时涉及到的网络链路越多,修复路径的瓶颈带宽就越小.

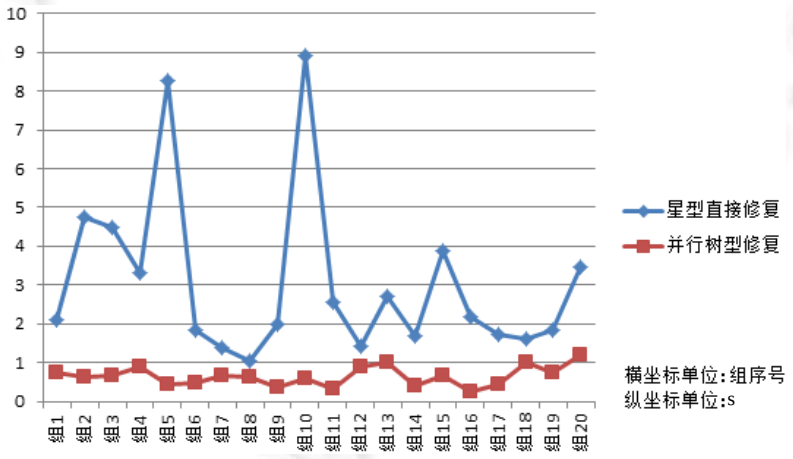


Fig.6 (5,3,2) SRC tree-based repairing compared with direct repairing

图 6 (5,3,2)简单再生码树型修复与直接修复对比

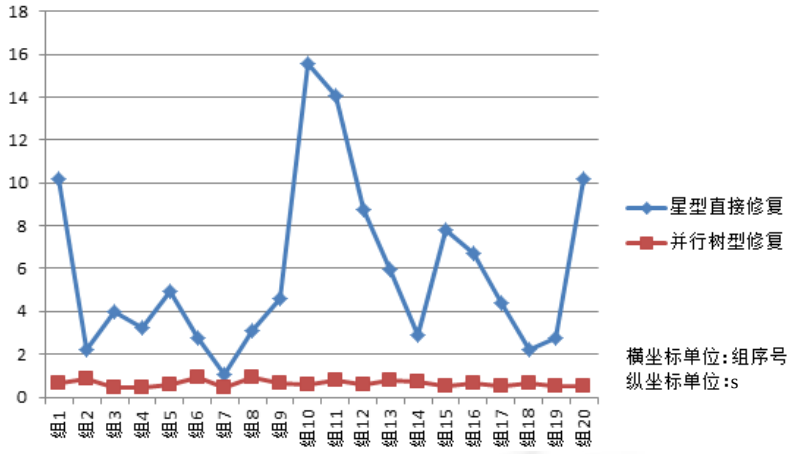


Fig.7 (10,8,3) SRC tree-based repairing compared with direct repairing

图 7 (10,8,3)简单再生码树型修复与直接修复对比

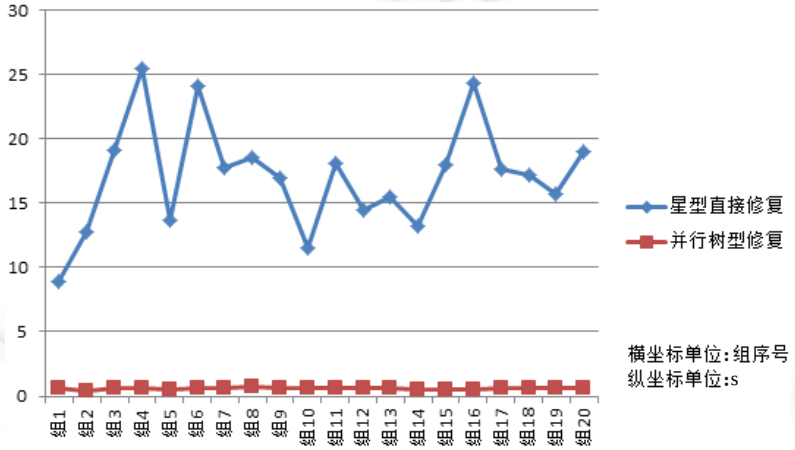


Fig.8 (17,13,7) SRC tree-based repairing compared with direct repairing

图 8 (17,13,7)简单再生码树型修复与直接修复对比

5 总结与展望

分布式存储系统为保证可靠性会采用一定的存储冗余策略,如多副本策略、编码策略.在编码策略中,简单再生码策略不但具有节点修复时参与节点数目少、磁盘 I/O 负载小的特性,而且保持了类 MDS 性质.然而,当前编码方法大都假设存储节点处于星型逻辑网络中,不能充分利用实际网络拓扑的信息.为此,有相关研究将物理网络和带宽信息考虑到纠删码和再生码修复过程中,进一步提高了修复效率.但由于编码和修复过程的不同,这些工作都不适合于简单再生码修复.本文针对此问题展开研究,详细介绍了利用带宽感知来优化简单再生码修复的问题分析与模型建立,针对修复失效节点上的所有数据块,设计了基于最优瓶颈路径和最优修复树的并行修复树算法.算法通过修复每个丢失数据块时尽可能利用带宽资源充足的链路来降低每个数据块的恢复时延,进而降低整个节点修复时延.本文还设计了实验来验证本文所提出的算法.实验结果表明:在相同编码参数的简单再生码节点修复网络中,该算法的修复时延比星型直接修复时延要小得多;并且随着网络链路带宽的不断变化,该算法的修复时延要比星型直接修复时延更加稳定.

本文提出的模型,假设在带宽相对稳定的情况下完成基于带宽感知的数据修复,减少数据块的恢复时延.广

域网中特定网络拓扑负载和带宽变化较大的情况,是值得进一步研究的扩展点.另外,本文着重研究系统中只出现 1 个故障节点的情况.若发生多节点故障,为不同故障节点构建修复树时,将会产生相互影响.多个故障节点情况也是值得进一步研究的工作.

References:

[1] Ghemawat S, Gobioff H, Leung ST. The Google file system. ACM SIGOPS Operating Systems Review, 2003,37(5):29–43. [doi: 10.1145/945445.945450]

[2] MacWilliams FJ, Sloane NJA. The Theory of Error Correcting Codes. Elsevier, 1977. 33–34.

[3] Dimakis AG, Godfrey PB, Wu Y, Wainwright MJ, Ramchandran K. Network coding for distributed storage systems. IEEE Trans. on Information Theory, 2010,56(9):4539–4551. [doi: 10.1109/TIT.2010.2054295]

[4] Papailiopoulos DS, Luo J, Dimakis AG, Huang C. Simple regenerating codes: Network coding for cloud storage. In: Proc. of the IEEE INFOCOM 2012. IEEE, 2012. 2801–2805. [doi: 10.1109/INFCOM.2012.6195703]

[5] Al-Fares M, Loukissas A, Vahdat A. A scalable, commodity data center network architecture. ACM SIGCOMM Computer Communication Review, 2008,38(4):63–74. [doi: 10.1145/1402958.1402967]

[6] Guo C, Wu H, Tan K, Shi L, Zhang Y, Lu S. Dcell: A scalable and fault-tolerant network structure for data centers. ACM SIGCOMM Computer Communication Review, 2008,38(4):75–86. [doi: 10.1145/1402946.1402968]

[7] Li J, Yang S, Wang X, Li B. Tree-Structured data regeneration in distributed storage systems with regenerating codes. In: Proc. of the IEEE INFOCOM 2010. IEEE, 2010. 1–9. [doi: 10.1109/INFCOM.2010.5462122]

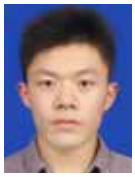
[8] Sun W, Wang Y, Pei X. Tree-Structured parallel regeneration for multiple data losses in distributed storage systems based on erasure codes. China Communications, 2013,10(4):113–125. [doi: 10.1109/CC.2013.6506936]

[9] Hu TC. Letter to the editor-the maximum capacity route problem. Operations Research, 1961,9(6):898–900. [doi: 10.1287/opre.9.6.898]

[10] Dijkstra EW. A note on two problems in connexion with graphs. Numerische Mathematik, 1959,1(1):269–271. [doi: 10.1007/BF01386390]

[11] Lee SJ, Sharma P, Banerjee S, Basu S, Fonseca R. Measuring bandwidth between planetlab nodes. In: Proc. of the Int'l Workshop on Passive and Active Network Measurement. Berlin, Heidelberg: Springer-Verlag, 2005. 292–305. [doi: 10.1007/978-3-540-31966-5_23]

丁尚(1992－),男,山东日照人,硕士,主要研究领域为分布式存储.



童鑫(1991－),男,工程师,主要研究领域为分布式存储,Linux 系统开发.

陈艳(1970－),女,高级工程师,主要研究领域为企业信息化.

叶保留(1976－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为云计算,无线网络,社交网络.