计算机科学
COMPUTER SCIENCE

# 一种基于纠删码的区块链账本分组存储优化方法

张玉书，何晓彤，肖祥立，朱友文，王良民

引用本文

张玉书，何晓彤，肖祥立，朱友文，王良民. 一种基于纠删码的区块链账本分组存储优化方法[J]. 计算机科学，2023，50(10)：350-361.

ZHANG Yushu, HE Xiaotong, XIAO Xiangli, ZHU Youwen, WANG Liangming. Grouping Storage Optimization Method for Blockchain Ledger Based on Erasure Code [J]. Computer Science, 2023, 50(10): 350-361.

---

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.220800193

# 一种基于纠删码的区块链账本分组存储优化方法

张玉书[1]　何晓彤[1]　肖祥立[1]　朱友文[1]　王良民[2]
1 南京航空航天大学计算机科学与技术学院　南京 211106
2 东南大学网络空间安全学院　南京 211189

**摘　要**　传统区块链系统采用全副本冗余的存储方式,每个节点存储相同的账本,使得区块链的存储负担非常大。目前,相关的区块链存储优化方法能够降低数据存储开销,但仍存在可扩展性差和可用性低的问题。为此,提出了一种基于纠删码的区块链账本分组存储优化方法。该方法引入一种新的区块链节点——分组存储(Grouping Storage,GS)节点,来解决上述问题。区块链账本的主要存储开销位于区块文件中,GS节点采用纠删码对区块文件编码,并以组为单位存储编码后的区块文件,如此,每个组织维持相同的账本,极大地降低了区块链的存储开销且提高了区块链的可用性。针对联盟链的存储扩展,基于GS节点对超级账本文件系统进行改进,重新设计了其存储、恢复和同步区块文件流程,使得本方案能够在实际的区块链架构上工作。最后,理论分析和实验结果表明,所提出的GS节点在存储开销方面取得了显著的进步,且具有较好的可扩展性和可用性。

**关键词**:区块链;存储优化;纠删码;超级账本;分组储存
**中图法分类号**　TP311

## Grouping Storage Optimization Method for Blockchain Ledger Based on Erasure Code

ZHANG Yushu[1],HE Xiaotong[1],XIAO Xiangli[1],ZHU Youwen[1] and WANG Liangming[2]
1 College of Computer Science and Technology,Nanjing University of Aeronautics and Astronautics,Nanjing 211106,China
2 School of Cyber Science and Engineering,Southeast University,Nanjing 211189,China

**Abstract**　The traditional blockchain system adopts the full copy redundant storage mode and each node stores the same ledger, so the blockchain storage burden is very large. At present, the relevant blockchain storage optimization methods can reduce the data storage overhead, but still have the problems of poor scalability and low availability. Thus, this paper proposes a grouping storage optimization method for blockchain ledger based on erasure code. This method introduces a new blockchain node, i. e., grouping storage(GS) node, to solve the above problems. Since the blockchain ledger storage cost is mainly in the block file, the GS node uses the erasure code to encode the block file, and stores the encoded block file in groups. In this way, each organization maintains the same ledger, which greatly reduces the storage overhead of the blockchain and improves the availability of the blockchain. For the storage expansion of the consortium blockchain, this paper improves and expands the file system of hyperledger fabric based on the GS node, and redesigns its process of storing, recovering, and synchronizing block files, which enables the scheme to work on the actual blockchain architecture. Finally, theoretical analysis and experimental results show that the proposed GS node has made significant progress in storage overhead, and performs well in scalability and availability.

**Keywords**　Blockchain, Storage optimization, Erasure code, Hyperledger fabric, Grouping storage

## 1　引言

比特币由中本聪[1]于2008年11月提出,已经成为目前最热门的数字货币之一。区块链作为其底层关键技术随之成为研究热点之一,逐渐走向金融机构、科技企业和医疗部门等领域[2-4]。区块链是一种去中心化的分布式数据库技术,它利用点对点传输、密码学以及共识算法等技术的结合实现去中心化、不可篡改、可追溯、透明等优势。然而,传统的区块链系统采用全副本冗余存储数据,引起了数据库存储开销大和扩展性差的问题。例如,比特币网络每秒仅7笔交易,然而在2022年初,其系统存储规模已经超过了800GB,以太坊则达到了10TB以上的数据存储规模[5]。假设每个节点都存储相同的几个GB的数据,那么整个区块链网络上百个节点就可能造成上百个GB的数据冗余,导致大量的存储空间浪费。区块链网络各节点通过共识算法得到一致的区块,这些区块以区块文件的形式被各个区块链节点存储。图1(a)展示了传统区块链账本的存储方式,其中区块链网络$N$中共有4个全节点$P$,每个$P$都需要存储内容相同的区块文件$b$,意味着

到稿日期:2022-08-22　返修日期:2022-12-04
基金项目:国家重点研发计划(2020YFB1005500)
This work was supported by the National Key Research and Development Program of China(2020YFB1005500).
通信作者:张玉书(yushu@nuaa.edu.cn)

每个 $P$ 维持同一个账本 $L$。随着区块文件数目的不断增大，每个节点的存储开销也不断增加，最终节点及全网的存储负担越来越重。

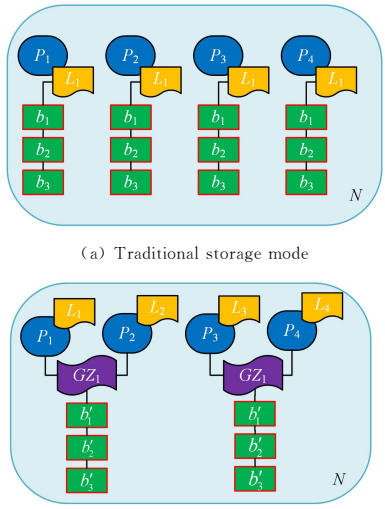
现有的区块存储扩展方案主要有轻节点存储、链下存储和纠删码存储。轻节点只储存区块头而非全部交易数据。然而，通过该方法所节省的存储空间以增加区块链网络的通信成本和全节点的计算成本为代价，因为轻节点必须与全节点交互才可验证交易。此外，大规模轻节点的出现会导致网络不安全。链下存储主要是外部存储原文件本身，而区块链只存储对应文件的地址映射。然而，链下存储网络与区块链网络的安全性并不等效。链下存储方案对区块链本身存储并无优化，只是解决如何辅助大文件上链的问题，对于区块链网络而言依然存在存储负担。纠删码存储主要是将数据编码为编码块，并将编码块分布在系统的几个节点上。然而，Perard 等[6-8]所提出的基于纠删码的存储扩展方法不能直接适用于联盟链环境，可扩展性较差。其次，他们提出的方法只有在节点规模较大时才能够保障区块链的安全。此外，Zhao 等[9]和 Yin 等[10]的方法只存储一份账本，极大降低了区块链的安全性和不可篡改性。进一步，Yin 等[10]的方法对区块进行纠删码编码，对于数百万交易的区块链则会产生数百万个区块，那么就要编码数百万次，这会严重影响区块链性能，导致可用性较差。

为此，本文提出了一种基于纠删码的区块链账本分组存储优化方法，该方法利用一种分组存储（Grouping Storage，GS）节点给区块文件编码并且以组为单位存储区块文件，组内的每个 GS 节点存储区块文件的一部分。其原理简单，如图 1(b)所示，假设图中 $P_1$ 和 $P_2$ 为一个组织，$P_3$ 和 $P_4$ 为另一个组织。*GS* 节点的主要特点是每个组织维持相同的组织账本 $GZ$，其不同于传统区块系统中每个节点维持相同的账本 $L$。本方法可减少每个节点所需的存储空间，鼓励低存储容量的节点参与区块链的存储工作，同时也降低了整个网络的存储空间。假设校验码的存储可忽略不计，GS 方案（见图 1(b)）相比传统节点方案（见图 1(a)）减少了约 50% 的存储空间。利用纠删码和分组方式存储区块文件，在数据丢失的情况下，也可方便区块文件的恢复和同步，从而保证区块文件和区块链的可用性。一个区块文件拥有数百万甚至上千万个区块，因此，对区块文件编码可降低编码频率，降低编码对区块链性能的影响。此外，本文对超级账本[11]（Hyperledger Fabric，简称 Fabric）的区块文件系统进行改进，采用 GS 节点优化了其存储方案，在有效降低节点的存储占用的同时，满足了在联盟链场景下区块文件可扩展性及可用性的需求。本文的主要工作总结如下：

1）针对区块链全副本冗余存储开销大和可用性的问题，提出了一种基于纠删码的分组存储节点。GS 节点以组为单位维护一个完整账本，并采用纠删码技术对区块文件编码，组内的每个节点分担一部分编码区块文件的存储，从而降低存储空间。

2）针对联盟链的存储扩展，运用 GS 节点的原理对超级账本的区块文件系统进行优化。为了均衡每个节点的存储开销，提出了循环移位编码分段存储算法。同时，为使 GS 节点在超级账本中可用，设计了恢复和同步区块文件的过程。

3）分析了 GS 节点的性能，并进行了相关的实验。分析和实验结果表明，本文提出的方法不仅可以降低存储，且具有可扩展性、可用性和可靠性，其网络开销和编译码时间开销对现有的区块链网络不会造成太大负担。




(a) Traditional storage mode

(b) Storage mode using GS nodes

图 1 区块链账本存储方式对比

Fig. 1 Comparison of blockchain ledger storage modes

本文第 2 章介绍区块链存储扩展的相关工作；第 3 章简要介绍 GS 节点原理；第 4 章讲述基于纠删码的区块文件编码和解码方法；第 5 章设计基于 GS 节点的超级账本存储扩展方案，包括区块文件的分组存储、解码恢复和同步更新过程，该方案为 GS 节点的具体实现；第 6 章对 GS 节点进行性能分析；第 7 章展示并分析实验结果；最后总结全文并展望未来。

## 2 相关工作

针对区块链系统中节点全副本冗余存储导致各节点存储开销大以及系统存储扩展性差的问题，本章调查了目前关于区块链存储扩展和降低存储占用方面的研究，并归纳出 3 种解决方案。

### 2.1 轻节点存储

比特币网络中的全节点如果要参与交易，就需要下载数百个 GB 的区块数据，并且数据量随着交易量的增多而不断增长。中本聪在其论文[1]中简要地提出了简单支付验证（Simplified Payment Verification，SPV）的轻节点解决方案。相对于全节点而言，这种轻节点无须下载全部区块数据，只需保留比特币全网的区块头数据（其体积每年以 4MB 增长）。轻节点可通过区块头和交易条件去全网定位要验证的区块默克尔树（Merkel Tree），然后在本地做一次简单的校验即可确认支付的有效性。然而，轻节点无法参与交易验证（与支付验证不同），也就无法参与区块链共识，它只能以“用户客户端”的角色出现在公有区块链网络中。如果所有用户都运行这样的轻节点，整个区块链的可用性将受到威胁，安全性将降低。与之不同，GS 节点不会像轻节点那样影响区块链的共识过程，且随着节点规模的增大，攻击者将会更难篡改区块文件，

因此区块链的安全性也会提高。

### 2.2 链下存储

为了解决区块链系统中数据存储量大的问题，Benet[12]提出了将外部存储网络与区块链网络相结合的概念，提出外部存储原文件本身而区块链只存储对应文件的地址映射的方案，通过外部存储的安全保障设计来保障文件的可用性和可靠性。对于区块链系统，外部存储数据就是链下(off-chain)存储，区块链上存储数据就是链上(on-chain)存储[13]。最近，许多关于链下存储的区块链系统方案都热衷使用 IPFS(InterPlanetary File System，星际文件系统)进行链下存储。IPFS 是一个全球的、点对点分布式文件存储协议，它可以将所有具备相同文件系统的计算机连接起来。传统互联网 HTTP 协议是搜索域名地址，但是 IPFS 是搜索内容地址。IPFS 这种颠覆 HTTP 协议的方法，理论上可以让网络更快、更安全。Zheng 等[14]提出了基于 IPFS 的区块链存储模型，把数据存储到 IPFS 中，然后将对应 IPFS 内容的哈希值存储到区块链上。然而，IPFS 系统中的数据对所有人都是公开的，只要拿到内容哈希值就可以拿取数据，这存在一定的风险。

还有一些方案使用本地存储或云存储[15]作为链下存储方案，但是该链下存储网络与区块链网络的安全性不等效。大文件的存储安全性受外部存储网络的存储方式及网络规模影响，在某些情况下会带来更多的存储负担。同时，链下存储方案对区块链本身存储并无优化，只是解决辅助大文件上链的问题，因此，区块链网络依然存在存储负担。相对而言，本文方法可直接解决链上数据存储量大的问题，从而避免链下存储可能会出现的风险。

### 2.3 纠删码存储

纠删码(Erasure Code，EC)是一种前向错误纠正技术[16](Forward Error Correction，FEC)，主要应用在网络传输中以避免包的丢失，其思想是将数据编码为编码块，并将编码块分布在系统的几个节点上。纠删码技术经过多年的发展，在传统分布式存储系统(如独立磁盘冗余阵列[17]、点对点[18]系统或云存储[19])中得到了广泛的应用，这些系统利用它来提高存储可靠性。相对于副本存储而言，在区块链网络中使用纠删码能够以更小的数据冗余度获得更优秀的数据可靠性、可用性和安全性[20]。

受到纠删码的启发，Perard 等[6]提出了低存储节点，这种节点通过纠删码存储区块链生成的数据。Wilkinson 等[7]使用类似的概念，将编码和加密的数据存储在可以连接到区块链的 P2P 网络中。Trón 等[8]提出将纠删码和区块链智能合约相结合，以在以太坊区块链上提供一个去中心化和冗余的分布式应用。虽然这些方案改善了区块链的存储量，但是它们也仅在比特币、以太坊等方面有相应的融合，仅适合公有链场景，缺乏针对联盟链的方案，可扩展性较差。由于联盟链(以 Fabric 为代表)的系统架构和共识机制与公有链存在较大区别，因此它们的方案不能在联盟链环境直接复用。其次，这些方案只有在节点规模较大时才能够在保障区块链网络安全的同时实现存储扩展。本文提出的优化存储方法适用于联盟链，且区块链节点规模大小对区块链安全不会产生太大影响。

目前国内也有相关研究，Zhao 等[9]提出了基于 RS 纠删码的区块链系统区块文件存储模型。在该模型中，收集足够的编码块就可恢复数据。然而，它所收集的编码块被随机分配到任意节点，这意味着恢复编码块会很耗时。同时，在该模型中，每个编码后的区块文件仅在全网保留一份记录，即保留一份账本，这会导致区块链的可用性大大下降。Yin 等[10]继承拜占庭容错协议的假设，实现了基于纠删码的区块链系统。

该方案在整个区块链网络中也仅存储一份账本，因此也会出现相同的问题。此外，它是对区块进行纠删码编码，对于数百万交易的区块链则会产生数百万个区块，那么就要编码数百万次，这会严重影响区块链性能。本文提出的存储优化方法可解决以上问题，下一章将对本方法所引用的 GS 节点的原理进行简单介绍。

## 3 GS 节点原理

本文提出了一种分组存储节点，它是一种新的区块链存储扩展节点，主要采用纠删码技术对区块文件编码，然后分组存储编码后的区块文件，组内的节点只承担部分区块文件存储开销。下文将简单介绍 GS 节点如何分组存储、恢复区块文件，以及 GS 节点对原有同步区块链网络的影响。

### 3.1 存储区块文件

传统的区块链采用全副本冗余存储数据，而 GS 节点的存储方式与之不同。我们先对整个区块链网络中的所有节点进行分组，每组分别存储区块链的相同账本，而不是像传统区块链中每个节点都存储相同的账本。一个区块账本中有许多区块文件，利用纠删码把区块文件编码成多个编码分段，同组的每个 GS 节点仅存部分编码分段，整合到一起就是一个完整的区块账本。本文选用 QC-LDPC 码作为编码区块文件的纠删码技术，区块文件会先被切分成多个数据分段，经过编码后生成编码分段，其中编码分段包含用来解码恢复的校验分段和原区块文件的数据分段，编码详细过程将在 4.2 节中描述。

图 2 为基于 GS 节点的存储扩展方案实现。假设该区块链网络有 2 个组织 $Org_1$ 和 $Org_2$，每个组织有 3 个 GS 节点，组织中的区块链账本中有多个区块文件。GS 节点将其中一个完整的区块文件划分为 4 个数据分段 $m_1-m_4$。接着将数据分段与生成矩阵 $\boldsymbol{G}_{qc}$ 相乘，得到 6 个编码分段，包括 2 个校验分段 $c_1$ 和 $c_2$ 以及原来的 4 个数据分段 $m_1-m_4$。每个组织保存每个区块文件的编码分段，组内的每个 GS 节点分别保留 2 个编码分段，全网 6 个节点拥有 2 个完整的区块文件。每个编码分段的存储空间为原来区块文件的 1/4，每个节点存储 2 个编码分段，那么每个节点存储空间为原来的 1/2。

### 3.2 恢复区块文件

当将编码分段恢复成区块文件时，GS 节点需要从组内各个节点或者跨组织的各个节点获取足够数量的编码分段，然后利用解码技术恢复区块文件，4.2 节描述了纠删码的解码原理。在丢失部分编码分段的情况下，因为纠删码具备容错能力，所以在组内的节点可通过解码来恢复区块文件。此外，由于每组的账本一致，因此可以跨组织收集解码区块文件，进一步保障区块文件的可用性。图 2 的例子中，在最坏的

情况下，组织 1 中的任意一个节点以及组织 2 的全部节点的数据全部丢失，即丢失全网 2/3 的数据，我们仍然可以恢复所有节点的原始区块文件，保障区块文件不丢失。

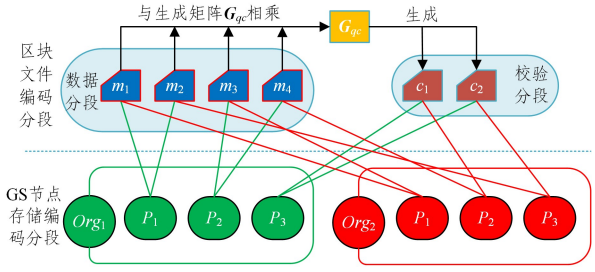



图 2　基于 GS 节点的存储扩展方案

Fig. 2　Storage expansion scheme based on GS nodes

在超级账本中，GS 节点需要恢复区块文件，用户才能查询回溯区块信息。因为状态数据库只有最新状态，仅有链中的原始区块文件才有区块回溯功能，本文在 5.2 节设计了相关的解码恢复过程来解决 GS 节点查询溯源区块信息的问题。

### 3.3　同步区块文件

传统的区块链网络中，某节点同步账本的方式是拷贝其他节点最新的区块链账本，账本包含区块文件和其他数据库信息，那么区块文件也是通过节点间互相拷贝进行同步的。若采用本方法实现存储优化，则以组为单位互相拷贝账本来实现区块文件同步，因为组内的节点分别存储部分的区块文件，每个节点的账本不同，只有由节点账本合成的组织账本相同。因此，需要考虑适合 GS 节点的区块文件同步过程。

结合超级账本场景，本文在 5.3 节实现了同步区块文件来提高 GS 节点的可用性。

## 4　基于纠删码的区块文件编码和解码

为了实现区块文件的编码和解码，本章首先引入 LDPC 码和 QC-LDPC 码的基本概念，然后介绍基于 QC-LDPC 码的区块文件编码和解码原理。

### 4.1　基本概念

早在 1962 年，Gallager[21] 就提出了低密度奇偶校验码（Low-Density Parity-Check，LDPC），它是一种先进的信道编码技术，采用软输出迭代编码的方式，具有逼近香农极限的纠错性能。LDPC 码由稀疏校验矩阵完全确定，校验矩阵形式决定 LDPC 码的解码可并行执行，且减少译码时延。用 $\boldsymbol{H}_{m\times n}$ 表示校验矩阵，$n$ 为码长，$k$ 为信息为个数，$m$ 为校验位个数，$m=n-k$；所有码字 $\boldsymbol{c}$ 满足 $\boldsymbol{cH}^{\mathrm{T}}=0$；由于矩阵大部分元素的数值都为 0，只有少数元素的数值为 1，因此我们称之为稀疏校验矩阵。通过优化 1 的分布可以得到更优的 LDPC 码，一般使用 $R=(n-m)/n$，若其校验矩阵不满秩，且秩为 $r$，那么实际码率为 $R=(n-r)/n$。

例如，一个(6,2,3)LDPC 码的校验矩阵可以表示为：

$$\boldsymbol{H}=\begin{bmatrix}0&1&1&0&1&0\\0&0&1&1&0&1\\1&0&0&0&1&1\\1&1&0&1&0&0\end{bmatrix} \tag{1}$$

其中，6 表示校验矩阵的列数，2 表示列重，3 表示行重。一般使用 Tanner 图表示 LDPC 码，它和校验矩阵一一对应。用 $G=\{(V,E)\}$来描 Tanner 图，其中 $V$ 是节点的集合，$V=V_b\cup V_c$；对于校验矩阵 $\boldsymbol{H}_{m\times n}$，$V_b=(b_1,b_2,\cdots,b_n)$称为比特节点，对应 $\boldsymbol{H}_{m\times n}$ 的列，也对应码字中的比特；$V_c=(c_1,c_2,\cdots,c_m)$是校验节点，对应 $\boldsymbol{H}_{m\times n}$ 的行，也对应校验方程；$E$ 是比特节点和校验节点之间相连边的集合，当 $\boldsymbol{H}_{m\times n}$ 中的 $h_{ij}=1$ 时，节点 $c_i$ 与 $b_j$ 之间有一条边相连。式(1)中校验矩阵对应的 Tanner 图如图 3 所示。

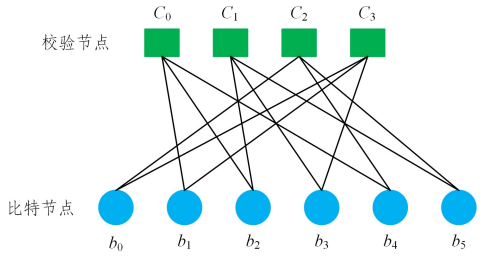



图 3　(10,2,4)LDPC 码的 Tanner 图

Fig. 3　Tanner graph of (10,2,4) LDPC code

作为 LDPC 码的一个重要子类，准循环低密度奇偶校验（Quasi-Cyclic Low-Density Parity-Check，QC-LDPC）[22] 码的校验矩阵具有准循环形式，这种结构特征决定了其编码复杂度较低，在现代通信[23]和存储系统[24]中有重要的应用。QC-LDPC 码不仅可实现快速编码，还具有优越的纠错性能。因此，本文选用其作为编码区块文件的纠删码技术。QC-LDPC 每个循环子矩阵的行重或列重可以大于 1，也可以等于 1。一个简单的 QC-LDPC 码可表示为：

$$\boldsymbol{H}=\begin{bmatrix}\boldsymbol{p}^{L_{11}}&\boldsymbol{p}^{L_{12}}&\cdots&\boldsymbol{p}^{L_{1n}}\boldsymbol{p}^{L_{21}}\\\vdots&\vdots&\ddots&\vdots\\\boldsymbol{p}^{L_{m1}}&\boldsymbol{p}^{L_{m2}}&\cdots&\boldsymbol{p}^{L_{mn}}\end{bmatrix} \tag{2}$$

其中，$\boldsymbol{p}^{Lij}$ 表示单位矩阵 $\mathbf{I}$ 右移 $L_{ij}$ 位得到的循环子矩阵；当 $L_{ij}$ 取∞时，$\boldsymbol{p}^{Lij}$ 代表全零矩阵。由于 QC-LDPC 码具有分块循环特性，因此它可以利用移位寄存器实现线性编码；并且，由于其只需要存储校验矩阵中每一个非零循环子矩阵的位置和循环移位数，因此可以明显减少存储空间。构造 QC-LDPC 码的方法包括有限几何法[25]、均匀完全组设计法[26]和三维立方体网格图法[27]。QC-LPDC 码是 LDPC 码的子类，因此其编码和解码方式与 LDPC 一致，只是校验矩阵和生成矩阵有所不同。此外，度数、环分布以及列重、行重是影响 QC-LDPC 码性能的重要因素。

### 4.2　编码区块文件

设在大小为 $S_B$ 的区块文件 $b$ 中，数据分段 $m$ 的数量为 $k$，校验分段 $c$ 的数量为 $r$，最终存储的编码分段 $b'$ 个数为 $t=k+r$。当编码区块文件 $b$ 时，先将 $b$ 切分成 $k$ 个数据分段 $m$。如果要把 $b$ 切分成固定大小 $S_B/k$，则需要把区块文件用 0 填充到 $S_B$ 大小。我们将一个 QC-LDPC 校验码表示为：

$$\boldsymbol{H}=\begin{bmatrix}\boldsymbol{A}_{1,1}&\boldsymbol{A}_{1,2}&\cdots&\boldsymbol{A}_{1,t}\\\boldsymbol{A}_{1,1}&\boldsymbol{A}_{2,2}&\cdots&\boldsymbol{A}_{2,t}\\\vdots&\vdots&\ddots&\vdots\\\boldsymbol{A}_{r,1}&\boldsymbol{A}_{r,2}&\cdots&\boldsymbol{A}_{r,t}\end{bmatrix} \tag{3}$$

其中，每个循环子矩阵 $\boldsymbol{A}_{i,j}$ 都是 $k$ 阶的。设 $\boldsymbol{H}_{qc}=[\boldsymbol{B}_{r\times k}\mathbf{I}_r]$，假定该矩阵的秩为 $r\times k$，由其得到的矩阵 $\boldsymbol{B}$ 的秩也为 $r\times k$，矩阵 $\boldsymbol{B}$ 为：

$$\boldsymbol{H}=\begin{bmatrix}\boldsymbol{A}_{1,1} & \boldsymbol{A}_{1,2} & \cdots & \boldsymbol{A}_{1,t}\\ \boldsymbol{A}_{1,1} & \boldsymbol{A}_{2,2} & \cdots & \boldsymbol{A}_{2,t}\\ \vdots & \vdots & \ddots & \vdots\\ \boldsymbol{A}_{r,1} & \boldsymbol{A}_{r,2} & \cdots & \boldsymbol{A}_{r,t}\end{bmatrix} \tag{4}$$

然后得到生成矩阵为：

$$\boldsymbol{G}_{qc}=\begin{bmatrix}\boldsymbol{G}_1\\ \boldsymbol{G}_2\\ \vdots\\ \boldsymbol{G}_3\end{bmatrix}=\left[\begin{array}{cccc|cc}\mathbf{I} & \mathbf{O} & \cdots & \mathbf{O} & \boldsymbol{G}_{1,1} & \boldsymbol{G}_{1,2}\\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots\\ \mathbf{O} & \mathbf{O} & \cdots & \mathbf{I} & \boldsymbol{G}_{k,1} & \boldsymbol{G}_{k,2}\end{array}\right] \tag{5}$$

其中，$\boldsymbol{G}_{qc}=[\mathbf{I}_k\boldsymbol{D}_{k\times r}]$，矩阵 $\boldsymbol{D}$ 由矩阵 $\boldsymbol{B}$ 生成，$\boldsymbol{D}=\boldsymbol{B}^{\mathrm{T}}$；$\mathbf{I}_k$ 为 $k$ 阶单位矩阵，$\mathbf{O}$ 是全零矩阵，$\boldsymbol{G}_{i,j}$ 是循环子矩阵；令 $\boldsymbol{g}_{i,j}^{n}$ 表示 $\boldsymbol{G}_{i,j}$ 的第 $n+1$ 行，$0\leqslant n\leqslant k-1$。$\boldsymbol{m}=\boldsymbol{m}_1,\boldsymbol{m}_2,\cdots,\boldsymbol{m}_{(t-r)k}$ 为待编码区块文件 $b$ 的数据分段，可用 $\boldsymbol{m}=\boldsymbol{f}_1,\boldsymbol{f}_2,\cdots,\boldsymbol{f}_{(t-r)}$ 的形式，其中 $\boldsymbol{f}_i=\boldsymbol{m}_{(i-1)k+1},\boldsymbol{m}_{(i-1)k+2},\cdots,\boldsymbol{m}_{ik}$；这样可以生成编码分段 $\boldsymbol{b}'=\boldsymbol{m}\boldsymbol{G}_{qc}$，由于 $\boldsymbol{G}_{qc}$ 是系统循环形式，因此有 $\boldsymbol{b}'=(\boldsymbol{m},\boldsymbol{c})$，其中 $\boldsymbol{c}=\boldsymbol{p}_1,\boldsymbol{p}_2,\cdots,\boldsymbol{p}_r$，$\boldsymbol{p}_j$ 可以表示为：

$$\boldsymbol{p}_j=\boldsymbol{f}_1\boldsymbol{G}_{1,j}+\boldsymbol{f}_2\boldsymbol{G}_{2,j}+\cdots+\boldsymbol{f}_{t-c}\boldsymbol{G}_{t-c,j} \tag{6}$$

其中，

$$\boldsymbol{f}_i\boldsymbol{G}_{i,j}=\boldsymbol{m}_{(i-1)k+1}\boldsymbol{g}_{i,j}^{0},\boldsymbol{m}_{(i-1)k+2}\boldsymbol{g}_{i,j}^{1}+\cdots+\boldsymbol{m}_{ik}\boldsymbol{g}_{i,j}^{(k-1)} \tag{7}$$

$\boldsymbol{p}_j$ 可用移位寄存器和加法寄存器构成的循环移位累加器生成。因此，用简单的移位寄存器进行编码，即可用实现基于生成器的线性编码。

采用固定的生成矩阵 $\boldsymbol{G}_{qc}$，避免每一次编码都重新计算生成矩阵。最后 GS 节点存储区块编码分段 $\boldsymbol{b}'=(\boldsymbol{m},\boldsymbol{c})$，删除原来的区块文件。此外，编码时存储区块文件的哈希值，用于解码后进行校验对比。

### 4.3 编码区块文件

当需要恢复完整的区块文件 $b$，则需要从不同的节点下载最少 $k$ 个编码分段 $\boldsymbol{b}'$，其中包括数据分段 $\boldsymbol{m}=\boldsymbol{f}_1,\boldsymbol{f}_2,\cdots,\boldsymbol{f}_{(t-r)}$ 及校验分段 $\boldsymbol{c}=\boldsymbol{p}_1,\boldsymbol{p}_2,\cdots,\boldsymbol{p}_r$。如果下载 $k$ 个编码分段刚好都属于 $\boldsymbol{m}$，那么就不需要解码，可直接恢复 $b$。如果只下载到部分数据分段 $\boldsymbol{m}$ 和校验分段 $\boldsymbol{c}$，那么需要利用相关解码算法来恢复 $b$。相关解码算法有信息传递解码算法、置信传播解码算法或和积算法，以及在此基础上产生的最小和解码算法和对域 BP 解码算法。它们的原理是利用软信息在比特节点和校验节点之间交互，通过多次迭代，实现更加准确的解码。此外，在解码过程中，需要把生成矩阵计算成校验矩阵，因此，记录对应的校验矩阵 $\boldsymbol{H}_{qc}$，避免每一次解码都需要再次保存生成矩阵。

## 5 基于 GS 节点的存储扩展方案

超级账本为联盟链的典型代表，本章将介绍基于 GS 节点的超级账本存储扩展方案。首先描述区块文件分组存储过程，并提出循环移位编码分段存储算法来优化节点存储；然后介绍节点如何收集恢复区块文件的过程；最后考虑到 GS 节点会影响到区块账本同步问题，提出了一个新的区块文件同步方案。

为了清晰给出区块文件扩展方案，首先形式化描述基于 GS 节点的区块网络下的成员结构和文件存储结构。

**定义 1** 基于 GS 节点的超级账本网络下，该成员结构定义为 $N=(Org,P,L,A,O)$，其中：

1) $Org=\{Org_1,Org_2,\cdots,Org_m,m\geqslant 1\}$ 是区块网络 $N$ 的组织集合，$m$ 为组织编号，一个 $N$ 可对应多个 $Org$，一个 $Org$ 存储一个完整的区块账本。

2) $P=\{P_1,P_2,\cdots,P_n,1\leqslant j\leqslant n\}$ 是组织 $Org$ 内的普通节点(General Peer)集合，$j$ 是组织 $Org$ 内 $P$ 的编号，共有 $n$ 个 $P$，其存储 $Org$ 的部分区块账本。$P$ 分为两种：提交请求节点(Committing Peer)和背书节点(Endorsing Peer)。前者负责提交交易请求和广播区块文件，后者负责模拟计算交易和对交易签名背书。

3) $L=\{L_1,\cdots,L_i,1\leqslant i\leqslant 2\}$ 是组织 $Org$ 内的领导节点(Leader Peer)集合，在组织 $Org$ 内选择 1～2 个 $P$ 设为 $L$ 节点，其负责获取原始区块文件，编码区块文件并分发给 $Org$ 内不同的 $P$，以及收集不同编码分段和解码区块文件，在该方案中起到重要作用。

4) $A=\{A_1,\cdots,A_i,1\leqslant i\leqslant 2\}$ 是组织 $Org$ 内的锚节点(Anchor Peer)集合，一般选择 1～2 个 $P$ 作为锚节点；$A$ 负责跨组织通信，即跨组织传递区块文件编码分段，然后再发送给 $L$ 节点。

5) $O=\{O_1,O_2\cdots,O_i,i\geqslant 0\}$ 是区块网络 $N$ 内的排序节点(Orderer)集合，主要负责按顺序打包交易，再生成区块。

**定义 2** 基于 GS 节点的超级账本网络下，网络账本的存储结构定义为：$Ledger=(GZ,Z,S)$，其中：

1) $GZ=\{Z_1,Z_2,\cdots,Z_j,j\geqslant 1\}$ 是组织账本，每个 $Org$ 维持相同的 $GZ$，包含 $Org$ 内的多个节点账本 $Z_j$。区块文件 $b_i$ 为第 $i$ 个区块文件，其编码后的区块文件也被称为编码分段，可表示为 $b_i'$。$Org$ 内所有节点账本加起来可以生成整个网络 $N$ 的所有编码后的区块文件；$GZ$ 被设计为缓存存储，当某个区块文件使用率低时可以清除，相反，区块文件使用率高时可以缓存存储在 $O$ 中，以此提高查询区块文件的效率。

2) $Z_j=\{S_j^0,S_j^1\cdots,S_j^i,i\geqslant 0,1\leqslant j\leqslant n\}$ 是节点账本，每个 $P$ 都有自己的节点账本 $Z_j$，$j$ 表示组内该 $P$ 的编号；每个 $P$ 都存储每个区块文件的部分编码分段 $S_j^i$，$i$ 表示区块文件编号。

3) $S=\{s_0,s_1\cdots,s_\gamma,1\leqslant\gamma\leqslant k+r\}$ 是部分编码分段集合，每个 $S$ 包含若干个编码分段 $s_\gamma$，其编码分段数量相同或者相差 1，与节点个数 $n$ 有关。一个区块文件的编码分段 $b_i'$ 由多个编码分段 $S$ 组成，即 $b_i'=(S_1^i,S_2^i,\cdots,S_n^i)$。

### 5.1 编码区块文件

原来超级账本的区块文件直接完整地存储在每个节点中，而本文则将区块文件进行切分和编码，然后每个节点分别存储编码后的区块文件的一部分，一个组存储完整的区块文件信息，以此减少区块文件的冗余存储，如图 4 所示。区块文件分组存储方案具体如下：

Step1 获取完整的区块文件

在组织 $Org$ 内的 $L$ 节点同步区块链网络的区块文件 $b_i$，并获取下来。本方案并不是对最新的第 $u$ 个区块文件进行编码，而是选择第 $u-1$ 个区块文件。一个区块文件会存储很多

区块，新的区块不断被添到一个区块文件中，当超过一定的阈值时，才会产生新的区块文件。因此，选择第 $u-1$ 个区块文件编码，可以确保该区块文件不会再记录新的区块且发生变化，从而保证组织的账本是同步一致的。

Step2　区块文件编码

$L$ 节点获取完区块文件后，通过 QC-LDPC 码的编码算法对区块文件 $b_i$ 进行编码，然后生成编码分段 $b_i'$，由 4.2 节可知具体的区块文件编码算法。

Step3　成功存储区块文件

$L$ 节点计算组内的每个 $P$ 对应的编码分段集合 $S$，然后存储到对应的节点账本 $Z$ 中。每个 $Z$ 都不同，它们只存储每个区块文件的部分编码分段。组内的所有 $Z$ 会合成为一个组织账本 $GZ$，每个组织的 $GZ$ 保持一致。$L$ 节点成功存储完相关信息后，会删除掉原来的区块文件 $b_i$。最后整个分组存储区块文件过程结束。

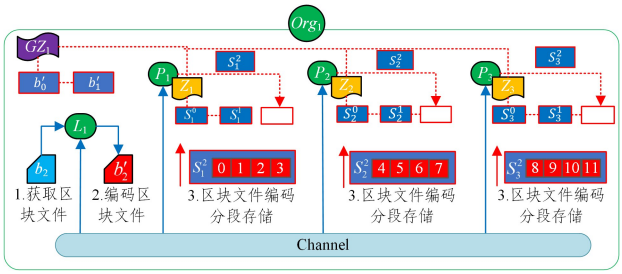



图 4　分组存储区块文件过程

Fig. 4　Process of grouping and storing block files

在保证 $Org$ 内拥有完整的编码分段信息的前提下，如果要计算不同 $P$ 节点对应的部分编码分段集合 $S$，那么就要考虑如何让每个 $P$ 尽可能少且均匀地存储编码分段 $s$。本文设计了一种以均衡开销为目标的循环移位编码分段存储算法，计算出每个 $P$ 需要存储的编码分段数量为 $n_s$ 和编码分段集合 $S_j^i$。具体算法步骤如下：

1）确定存储的首个节点编号 $x$。

如果某一组织 $Org$ 中节点数量为 $n$，那么 $x\in[1,n]$，确定本轮区块文件存储的首个节点编号 $x$，那么下一个区块文件存储的首个节点编号为 $(x+1)\%n$。

2）计算 $P$ 的最大编码分段数量为 $n_s$。

已确定组织内节点数量 $n$、数据分段数量 $k$、校验分段数量 $r$，如果 $k+r$ 能够被 $n$ 节点平分，则每次存储区块文件时，所有节点存储的 $n_s$ 一致；否则，分别计算不同节点存储的编码分段集合，节点间 $n_s$ 最多相差 1。设 $\beta=(k+r)\%n$，编码分段数量为：

$$n_s=\begin{cases}(k+r)/n, & \beta=0\\ (k+r)/n+1, & \beta\neq 0\end{cases} \tag{8}$$

3）计算每个 $P$ 的编码分段集合 $S_j^i$ 及其编码分段 $s_y$。

$S_j^i$ 是 $P$ 的编号为 $j$ 时存储 $b_i$ 的部分编码分段集合，设 $t$ 为标记存储过的节点数量。

（1）当 $\beta=0$ 时，$S_j^i$ 存储的 $s_y$ 数量相同。根据 $t$ 和 $\gamma$，计算每个 $P$ 的 $S_j^i$：

$$S_j^i=S_j^i\cup\{s_y\},\ t\times n_s\leqslant\gamma<(t+1)\times n_s \tag{9}$$

接着计算下一个节点 $t=t+1$，$j=(j+1)\%n$，继续循环计算式（9），直到 $t+1=n$ 时，所有节点都结束计算。

（2）当 $\beta\neq 0$ 时，$S_j^i$ 存储的 $s_y$ 数量不相同。先循环计算式（9），此时结束条件为 $t+1=\beta$。此时，计算

$$S_j^i=S_j^i\cup\{s_y\},\ t\times(n_s-1)\leqslant\gamma<(t+1)\times(n_s-1) \tag{10}$$

接着计算下一个节点 $t=t+1$，$j=(j+1)\%n$，继续循环计算式（10），直到 $t+1=n$ 时，结束计算。

4）确定包含了校验分段的节点。

如果 $\gamma\geqslant k$，说明 $S_j^i$ 中包含了校验分段部分，使用 $b=1$ 标记，$b=0$ 则默认该 $S_j^i$ 不需要编码计算得到。标记的作用就是方便恢复区块文件，如果收集到的 $S_j^i$ 中，$b$ 都不为 1，则不需要解码，可快速恢复区块文件。

如果每个节点存储的编码分段数量刚好相同，则每次存储区块文件时，每个节点存储的内容大小都相同，否则不同。采用循环移位存储编码分段方法，可以有效避免节点存储大小不一样的问题。如图 5 所示，这种算法可使得各节点均匀地存储且不增加额外存储。

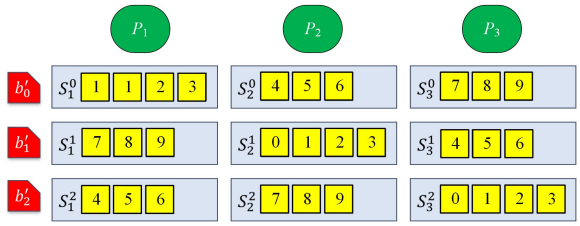



图 5　循环移位编码分段存储算法的简单示例

Fig. 5　A simple example of a segmental storage algorithm using cyclic shift encoding

## 5.2　收集恢复区块文件过程

用户在查询溯源交易过程中，需要恢复区块文件，采用 GS 节点存储区块文件会影响查询溯源功能。因此本文设计了 GS 节点收集恢复区块文件过程，如图 6 所示。本节通过描述查询溯源交易场景来分析该过程，具体方案如下：

Step1　发起查询交易请求

首先，客户端 $C$ 通过链码向组织 $Org$ 内的 $L$ 节点发起查询请求，查询用户请求的交易 $T$ 的信息。然后，$L$ 节点查询 $T$ 是否位于区块文件 $b_i$ 中，这是因为一个区块文件可以存储多个交易，每个交易对应一个区块。此外，$L$ 节点还需确定该编码分段集合 $S$ 已被存储到 $Org$ 内不同的 $P$ 节点中。

Step2　收集编码分段

本方法的特点之一是可以从组织内部或跨组织中收集编码分段 $s$。如果组内 $s$ 不足，本地组织内的 $L$ 节点才跨组织收集，直到收集到所需要的 $s$，然后解码恢复成对应的 $b_i$。一般会选择优先在组内收集 $b_i$，以提高恢复效率。

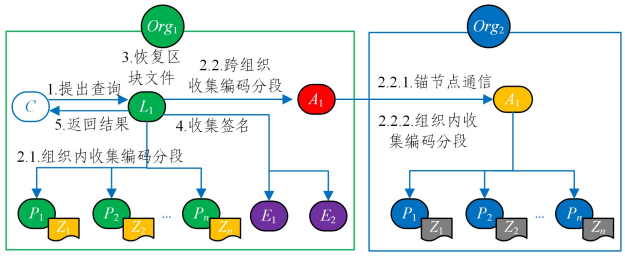



图 6　查询溯源交易信息过程

Fig. 6　Process of querying and tracing transaction information

1)组织内部收集

$L$ 节点向 $Org$ 内的每个 $P$ 节点广播请求收集 $b_i$ 相关 $s$ 的消息。当 $P$ 节点收到广播消息后,返回对应的 $s$ 给 $L$ 节点。如果 $L$ 节点收集到足够的 $s$,那么它会广播停止收集 $s$ 的消息。$L$ 节点优先收集 $k$ 个数据分段 $m$,这样就可以避免解码。从 4.1 节可知,$b=0$ 时,$S$ 仅含数据分段,数据分段恢复区块文件不产生解码开销。此外,$L$ 节点也可收集足够的数据分段和校验分段,利用纠删码解码算法恢复区块文件。

2)跨组织收集

如果 $Org$ 内未收集到 $k$ 个 $s$,那么可用跨组织收集缺失的 $s$ 返回给 $L$ 节点。接着,$L$ 节点向 $Org$ 内 $A$ 节点发出与其他组织 $Org$ 通信并获取缺失的 $s$ 消息。其他组织的 $A$ 节点收到返回的缺失 $s$ 消息后,通过算法找到存储 $s$ 的节点,然后返回给本地 $A$ 节点,再由 $A$ 节点将 $s$ 返回给原来的 $Org$。如图 6 所示,$Org_1$ 中的 $A_1$ 节点与 $Org_2$ 的 $A_1$ 节点通信,同时从 $Org_2$ 内获取缺失的 $s$。

Step3 解码恢复区块文件

若从上一步中的 $L$ 节点收集到足够的编码分段 $s$,则可利用 QC-LDPC 解码恢复区块文件,否则返回错误。编码分段 $s$ 分为两种:数据分段 $m$ 和校验分段 $c$。如果 $L$ 节点收集到足够的 $k$ 就可以恢复原来的区块文件 $b_i$,否则才需要通过 $c$ 解码恢复。计算恢复的 $b_i$ 的哈希值,并与编码前的值对比,可确认恢复的区块文件是否准确和检验是否被篡改。解码除了用到 4.3 节的原理外,还需要正确的解码矩阵。解码矩阵为编码分段对应的线性组合的逆矩阵,会提前保存在账本中。

Step4 收集足够背书签名

$L$ 节点从恢复的区块文件 $b_i$ 中查询交易 $T$,并广播给 $E$ 节点。接着,$E$ 节点根据链码背书签名该交易 $T$,$L$ 节点根据背书策略从不同 $E$ 节点收集足够的背书签名。

Step5 返回查询结果

从 $L$ 节点收集到足够的背书签名后,客户端才可以返回交易 $T$ 的相关查询溯源信息,最后整个查询流程结束。

需要注意的是,查询某个交易数据值的最新状态不需要恢复区块文件,可直接从状态数据库获取。如果溯源某个交易或查询某个区块导致解码区块文件,才需要恢复区块文件。当溯源查询操作比较频繁时,可采用暂时缓存的方式把对应数据存储在组织账本中。目前的缓存方法有多种,例如过期策略,即数据超过过期时间没有被访问则把它们删除,以此来提升区块链的查询溯源性能;此外还有一种扩展办法,即一个组内一个节点保持全节点模式,拥有整个账本数据,其余的多个节点可设为 GS 节点模式,这样在查询溯源时,可直接向全节点查询,可用性提高且基本不会有解码问题产生。该方法需要平衡区块链的存储空间和查询效率来满足实际情况的应用。

### 5.3 同步更新区块文件过程

原来超级账本的区块文件直接完整地存储在每个节点中,而本文则将区块文件进行切分和编码,然后每个节点分别存储编码后的区块文件的一部分,一个组存储完整的区块文件信息,以此减少区块文件的冗余存储,如图 4 所示。区块文件分组存储方案具体如下。

超级账本对区块同步采用的是 gossip 最终一致性协议。基于 GS 节点的超级账本网络中可能存在以下情况,需要进行区块文件同步更新。

1)某些 $P$ 节点长时间离线,导致一部分区块文件对应的编码分段缺失,即这些节点账本 $Z$ 不完整。$Org$ 只能通过索引库及状态数据库提供组织账本快照,却无法提供完整的原始文件以供离线节点进行区块同步。

2)当某组织离线时,需要与邻接的组织同步当前所有的区块文件。

为解决以上两种情况下同步区块文件问题,针对超级账本,本文提出了新的区块文件同步更新方案,如图 7 所示,具体步骤如下:

Step1 发起同步和恢复账本请求

如果组织 $Org$ 内的任意 $P$ 节点想同步最新区块文件的编码分段,先向 $L$ 节点发起同步恢复组织账本 $GZ$ 的请求。针对上面两种同步情况,只需要 $Org$ 内的任意 $P$ 节点发起同步请求,就可以实现任意节点或者整个组织账本更新,也可同步网络中的新区块文件。

Step2 恢复区块文件

$L$ 节点根据 3.3 节的流程收集和恢复区块文件,只需在上次同步过程的区块文件后面接着恢复区块文件即可。例如,上次已经同步到区块文件 $b_i$,意味着前面 $i$ 个区块文件已同步过且目前没被篡改,不需要再次同步,那么此次只需恢复 $b_{i+1}$ 到 $b_n$ 最新生成的区块文件即可,然后计算其账本快照。这种方法无须从第一个区块文件开始恢复,但如果是中间的区块文件缺失可能就不适用,需重新考虑是否恢复全部区块文件。

Step3 恢复组织账本

$L$ 节点恢复好组织内的区块文件后,生成当前 $Org$ 的账本,再计算账本快照。假设当前 $Org$ 与其他 $Org$ 的账本差别很大,索引库和状态库下的账本快照可能与实际不一致,需重新计算账本快照。另外,由于每个组织中的组织账本都一致,因此恢复本地组织账本有助于计算和更新账本中的差异。

Step4 广播账本快照

$L$ 节点广播 $Org$ 内的账本快照给其他组织的 $L$ 节点,其他组织收到请求后,可以在它们的索引库及状态数据库直接获取自己最新的账本快照,再发送给请求的节点。本地的 $L$ 节点收到其他组织账本快照后,与本地组织账本快照对比最高 hash、块 number 以及 worldstate 等参数,最后生成对比结果。

Step5 请求同步区块文件

如果对比结果不一致,则本地 $Org$ 的 $L$ 节点向对应组织发起同步区块文件的请求。对应组织发送相关的缺失的区块文件或编码分段给 $L$ 节点。

Step6 更新本地账本

最后,$L$ 节点获取新的区块文件,同步更新索引和状态数据库,更新组内每个节点的节点账本,最后整个同步更新流程结束。

考虑到加入联盟链的节点具有较高的可信度且动态性低，不易出现大量节点离线的情况。进一步，为了减少离线节点同步过程的开销，在区块文件编码存储前先确认节点是否离线，离线则先唤醒上线。当然，如果出现长期离线的节点，其同步过程所带来的开销对于整个区块网络而言也是可以承受的。

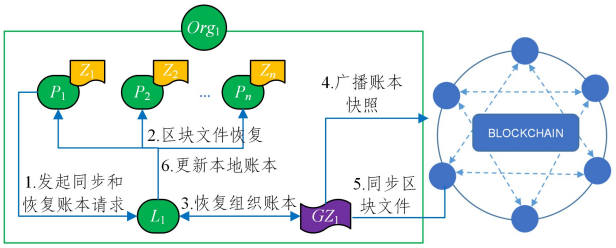



图 7 同步更新区块文件流程

Fig. 7 Process of synchronously updating block files

## 6 性能分析

本章首先验证所提的方法能够扩展区块链存储，降低节点的存储开销。其次，验证本方法在一定程度上可提高区块链的可靠性和可用性。最后分析区块文件解码产生的网络开销。

### 6.1 可扩展存储

设 $\theta$ 为压缩因子，$\theta=k/r$，用来调节存储空间大小，其中 $k$ 为数据分段的个数，$r$ 为校验分段 $c$ 的个数。图 8 给出了不同压缩因子 $\theta=1,2,3,4,5$ 对应的不同节点个数 $n=2,3,4,5$ 与传统冗余存储方案相比平均每个节点能够节约的存储空间。假设网络中每个组织只有 4 个 GS 节点，令 $\theta=5$，则每个 GS 节点仅需要存储编码分段原始区块文件的 30%，那么就节约了 70%的存储开销。显然，压缩因子越大，节点能够节约的存储空间就越多。然而，如果压缩因子过大，校验分段将会过少，可能不利于恢复原始区块文件。此外，组织内节点越多，可能需要更多的硬件资源和成本。

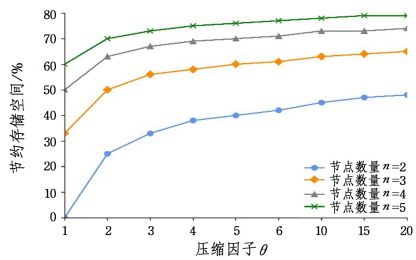



图 8 平均每个 GS 节点可节约的存储空间

Fig. 8 Average storage space saved by each GS node

假设区块链网络中有 $m$ 个组织，每个组织平均有 $n$ 个 GS 节点，原始区块文件大小为 $S_B$，网络有 $num_B$ 个区块文件，每个区块文件有 $k$ 个数据分段以及 $r$ 个校验分段，那么在使用基于 GS 节点的存储扩展方案后，单个节点存储区块文件的编码分段占用存储空间为：

$$S_{GS}=\frac{S_B\times num_B}{k}\times\frac{k+r}{n}=\frac{S_B\times num_B}{n}\times\left(1+\frac{1}{\theta}\right) \tag{11}$$

整个超级账本系统占用的存储空间为：

$$S_{net}=m\times S_{GS}\times n=m\times S_B\times num_B\times\left(1+\frac{1}{\theta}\right) \tag{12}$$

对于没有使用存储扩展方案的普通超级账本系统，其每个单节点所需的存储空间为 $S_B\times num_B$，全网存储节点的空间占用为 $m\times n\times S_B\times num_B$。由式(11)和式(12)可知，单个 GS 节点存储空间受组织中的节点数量及压缩因子的影响，而整个网络系统所占的空间主要受组织数量和压缩因子的影响。

考虑到实际情况，网络中有些组织只有 2 个以下的节点，节点规模较小，它们可保留一个节点存储一个账本的传统存储方式。如果采用 GS 节点，那么需要适量增加压缩因子，否则存储开销不会下降。对于有 2 个节点以上的组织，使用 GS 节点的存储方式优化效果会更明显。本存储方案在原有的区块链存储方案上扩展，并且兼容。如果网络节点规模大的组织采用的是 GS 节点的存储方式，节点规模小的组织采用的是全节点的存储方式，那么查询时可直接调用全节点的区块文件查询，这样就能在提高查询效率的同时降低整个网络的存储空间。

目前，每个组织的 GS 节点默认从编号为 1 的节点开始存储区块编码分段，也可以按照别的规律约定存储区块编码分段的首个节点；每个节点存储的数个编码分段默认是连续的，也可以是不连续的，即按其他规律进行存储。

### 6.2 可用性

区块文件的可用性是本文方法的基本要求之一。对于区块链网络这样拥有大量节点的分布式系统，任何节点都可能无法访问，原因可能是节点被攻击或者离线等。如果某节点无法被访问，那么其存储的编码分段在区块链网络中不可被获取，这样会导致其节点账本和所在组织的组织账本缺失。

为了满足在节点无法访问的情况下区块文件可用的要求，在基于 GS 节点的区块链网络系统中，我们设计每个组都维持相同的组织账本，让组内的成员承担该组织账本的部分存储开销，仅存储纠删码生成的区块文件编码分段，且各不相同。那么，当一个节点或者一个组织无法访问时，采用 GS 节点的区块链仍然能保证所有账本的完整性以及区块文件的可用性。其可以通过拷贝账本和纠删码技术恢复丢失的文件，恢复节点或者组织原来的存储功能，即组合任意组织里的任意节点提供 $k$ 个编码分段来恢复原来的区块文件。因此，对于具有大量 GS 节点的区块链系统来说，部分节点甚至组织无法访问，都不会显著地影响到区块文件的可用性。

为了恢复存储区块文件 $b$ 的编码分段数量，组织内最少收集 $k$ 个编码分段。如果要使得某个组织无法恢复所有区块文件，则要在组织内丢失大于或等于 $L$ 的存储空间：

$$L=\frac{r+1}{k+r}\times S_{GS} \tag{13}$$

进一步，如果区块链网络中的某一组织内丢失的存储空间大于或等于 $L$，且其他 $m-1$ 个组织的全部存储丢失，那么整个区块链网络才会无法恢复 $b$，如图 9 所示。当 $\theta=3$，$k=30$ 时，一个组织丢失 26%以上的存储内容，且其他 $m-1$ 个组织都丢失 100%的存储内容，要满足以上条件，才会使得区块文件不可用，这实际上是非常困难的，因此本文提出的扩展存储方案可保障区块文件的可用性。

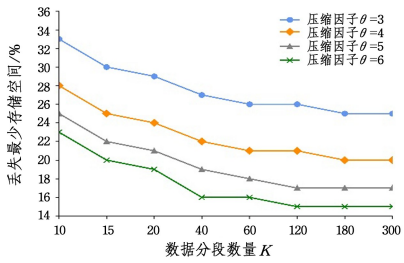



图 9 组织内使得区块文件不可恢复的最少丢失存储空间

Fig. 9 Minimum amount of lost storage space within an organization that makes block files unrecoverable

其他基于纠删码的区块链扩展系统[9-10]中，每个区块或者区块文件的编码块会被随机地存储在几个节点中，且整个网络只存储一次。与之前的系统相比，本系统的每个节点都存储每个区块文件的编码分段，且每个节点存储到校验分段 $c$ 的概率相等，相对而言安全性和可用性更高。假设本系统中，数据分段数量 $k=400$，每个区块压缩因子 $\theta=10$，那么每个区块文件产生 $r=3$ 个校验分段 $c$，则每个组内的每个节点分担 440 个编码分段，并会随机均匀地存储到每个节点。另外，当过于频繁地恢复某个区块文件时，会在该组织生成缓存的组织账本来存储这些命中率高的区块文件，以此来提高实用性。

值得注意的是，不同的 QC-LDPC 码编解码算法会给区块链网络带来一定影响。例如编解码算法不同，它们的编译码效率可能也不同。此外，不同的编解码算法，在码率和信噪比一定的情况下，它们的误码率也不一样。目前有许多相关的新的研究，本文采用一般的 QC-LDPC 码举例说明，实际可选择实用性更好的算法。

### 6.3 可靠性

GS 节点除了可以减少网络存储开销外，还可以让区块链更安全。这是因为区块链数据在 GS 节点中更难被篡改，并且还有助于识别恶意节点，从而提高区块链网络的可靠性。

1)防哈希碰撞。每个区块的区块头都包含前一个区块的哈希值，通过验证哈希值可以确认整个区块链的完整性。因此，任意恶意攻击节点想要篡改某个区块，必须调整随机数等参数，使得篡改完该区块后，该哈希值与原来的哈希值碰撞。由于我们采用 GS 节点存储区块文件的编码分段，恶意节点只能篡改其自身存储的部分编码分段，且它还要找出其他节点中哪些是解码过程中需要的编码分段。然后，恶意节点计算哈希碰撞，找到一个篡改过的编码分段，与原来的解码分段解码恢复的区块文件相比，它们的哈希值是一样的。这个攻击的难度比传统的寻找区块头的哈希碰撞问题更大。因此，使用 GS 节点扩展存储方案不仅可以减少节点的存储量，而且还可以提高系统的安全性和可靠性。

2)识别恶意节点。可以利用纠删码识别恶意节点是否篡改区块文件。如果某个编码分段被篡改，那么该编码分段就不可用来恢复成原来的区块文件，我们可以通过此方法来检验每个节点是否合法。此外，每个组织都维持一个账本，假设一个组织全部的节点都是恶意的，那么其可能与别的组织数据账本不同，可通过识别节点或组织是否合法来提高区块链网络系统的安全性及可靠性。值得注意的是，由于区块链有同步账本功能，如果 Leader 节点发出的信息有误，其他节点则无须验证。如果某个组织在某个时间点，从各节点收集到编码碎片、恢复成组织账本后，通过网络广播发现与别的组织账本不一致，就会重新自我更新恢复账本，继而可以检测哪个节点出错，再判断该出错节点是否是恶意节点。即使所有 Leader 都出错，还可以设计全副本节点用来保存账本，以减少事故发生。在实际应用中，同时应用 GS 节点和全副本节点可以获得更好的安全性。

### 6.4 网络负载

使用该存储扩展方案解码恢复会带来额外的网络消耗。每一次用户查询已编码的区块文件里的若干信息，客户端会向连接的 $P$ 节点发出查询申请，$P$ 节点会在组织内收集区块文件的若干个编码分段。单个节点存储一个区块文件的编码分段占用存储空间为：

$$S'_{GS}=\frac{S_B}{k}\times\frac{k+r}{n}=\frac{S_B}{n}\times\left(1+\frac{1}{\theta}\right) \tag{14}$$

假设只考虑组织内恢复区块文件的情况，GS 节点恢复一个区块文件至少传输的数据量为：

$$d_{\min}=\frac{(n-1)}{n}\times S'_{GS}=\frac{S_B}{n^2}\times(n-1)\times\left(1+\frac{1}{\theta}\right) \tag{15}$$

假设网络中每分钟有 20 次查询不同区块文件的信息请求，$\theta=3$，$S_B=12$ MB，$n=4$，时间 $t=60$ s，则组织内的每个节点至少需要额外负担的上行宽带为：

$$\frac{12}{4^2}\times(4-1)\times\left(1+\frac{1}{3}\right)\times20\div60=1\ \text{Mbps} \tag{16}$$

每个节点至少需要额外负担的下行宽带也同样为 1 Mbps。每次查询都会优先从组织内部恢复，跨组织解码的情况相对较少。假设要跨组织恢复解码，由于可以计算缺失的具体区块文件编码位置，因此只需要直接从别的组织的某个节点获取即可。通过分析，解码模块所带来的网络资源消耗主要受区块文件大小和节点数量的影响，当原始区块文件越大，节点数量越少，消耗的网络资源就越多。

## 7 实验分析

本实验先在 1 台物理机 PC 上搭建 1 台虚拟机，然后创建操作系统 CentOS7，其操作系统配置环境参数如表 1 所列。我们基于超级账本 Hyperledger Fabric v1.4.6 进行存储优化改进，加入 GS 节点相关存储扩展功能，简称为 Fabric-GS。未经过修改的 Hyperledger Fabric v1.4.6，简称为 Fabric。区块链架构相关参数如表 2 所列。下面将分析 Fabric 区块链账本，并对 Fabric-GS 的节点存储开销和编译码时间开销进行实验分析。实验所用的测试数据集是自己运行区块链网络所产生的交易区块。具体来说，首先基于 Fabric 官方提供的工程案例 fabric-samples 搭建区块链网络，再编写脚本调用链码生成大量交易数据。所调用链码的功能是在两个账户之间转账。经过共识与打包，这些交易数据以区块形式被记录在区块链账本中，为实验测试所用。为了实时获取区块链网络的各种状态数据，实验所采用的测试工具有 caliper，tape 和 jemeter。

表 1 实验系统参数配置信息

Table 1 Parameter configuration information of experimental system

| 参数 | 值 |
|---|---|
| CPU | Intel(R) Core(TM) i7-6500 CPU @ 2.50GHz |
| 内存/GB | 1.00 |
| 处理器/个 | 1 |
| 磁盘/GB | 20.00 |
| Linux 内核版本 | Linux version 3.10.0-1127.19.1.el7.x86_64 |
| 操作系统版本 | CentOS Linux release 7.8.2003(Core) |
| Docker 版本 | 19.03.13 |
| Go 版本 | 1.14.6 |

表 2 区块链架构参数信息

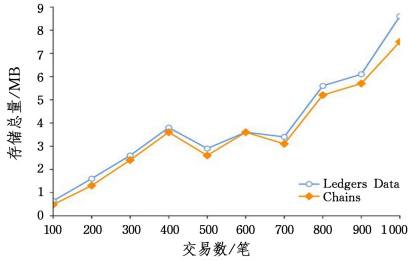
Table 2 Parameter information of blockchain architecture

| 参数 | 值 | 参数 | 值 |
|---|---|---|---|
| 共识机制 | Solo | 批处理等待时间/s | 2 |
| 组织数量 | 4 | 缓存最大交易数 | 10 |
| GS 节点数量 | 4×4 | 缓存大小上限/MB | 2 |
| 排序节点数量 | 1 | 缓存最佳大小/kB | 512 |

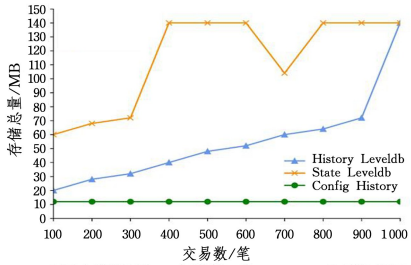
### 7.1 区块链账本分析

本实验先搭建 Fabric 网络，然后对单个节点的区块链账本进行分析。账本数据主要存储内容包括链、历史数据库、状态数据库和配置历史等。其中链包含索引和对应链码的区块文件存储。由实验结果可知，实际上每个区块的大小不是固定的，根据交易数据大小确定，该实验中每个区块的大小平均约为 3.9kB。每笔交易成功后其交易信息会被记录在账本中，区块高度便会增加 1，即新增 1 个区块。一个区块文件会存储多个区块。每次新增的区块会接着放入同一个区块文件中，直到超过区块文件设定的最大值，才会产生下一个区块文件接着存放新的区块。此外，需要注意实验中短时间内发出交易请求数量很多并且没有报错，并不意味着该时间内的所有交易被成功记录到区块链上，一般每笔交易最多等待 2s 延迟才会被成功记录。因此，需要检测区块高度和查询变量等检查交易是否成功，否则会使得测试的存储空间值偏低。

图 10(a)和图 10(b)给出了不同区块高度下，一个节点存储的不同账本内容的开销。




(a) Ledger data and chains




(b) History database, state database, and configuration history

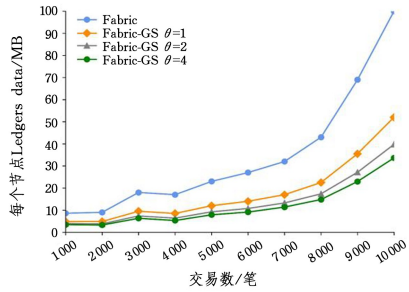
图 10 单个节点账本存储分析

Fig. 10 Analysis of ledger storage of a single node

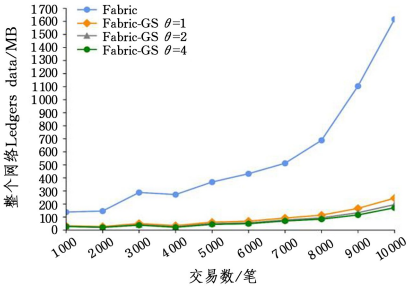
实验表明，大部分数据开销在链中，其中区块文件占用开销最大，1000 笔交易的存储总量为 7.5 MB，而历史数据库、状态数据库和配置历史信息等占用的比例较少，增长也少且慢，其中配置历史信息由于配置没变，一直在 12 KB，历史数据库和状态数据库 1000 笔交易的存储总量为 140 KB。Fabric 的数据存储并不是随着交易数线性增加，因为 Fabric 本身会自动优化其数据库，有时交易数大，存储总量反而下降。对区块文件的存储优化有助于减少区块链的存储开销，因此，本文提出的 GS 节点通过优化区块文件来减少节点和网络的存储占用是合理的。

### 7.2 节点存储开销

扩展方案中的 GS 节点在完成区块文件编码后，还要对各个区块文件的哈希值、校验矩阵以及补零位数等相关信息进行存储，我们将这些信息存到账本数据中，纳入测量范围中。图 11(a)和图 11(b)分别给出了 Fabric 和 Fabric-GS 的单个节点及整个网络的的账本数据空间测试情况。




(a) Comparison of single-node storage



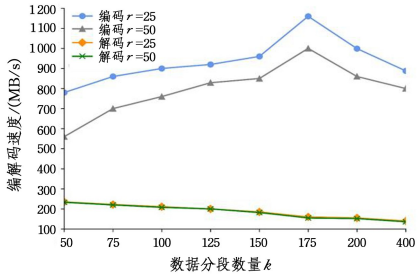

(b) Comparison of blockchain network storage history

图 11 存储开销测试

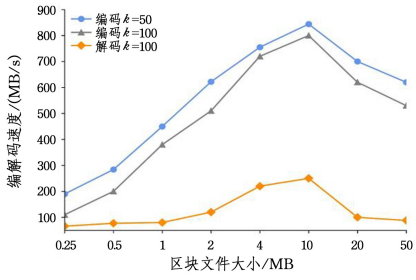
Fig. 11 Storage overhead test

Fabric-GS 单节点存储账本数据的空间占用随着交易数量的增加而缓慢增加，而 Fabric 交易数量越大增加越快。Fabric-GS 的压缩因子 $\theta$ 越大，冗余存储越少，每个节点存储开销越小。10000 笔交易中，Fabric 的单个节点占用约为100 MB，而 Fabric-GS 中 $\theta$ 为 4 的单个节点仅占用约 33.68 MB，约为 Fabric 的 1/3。与 Zhao 等[9]的方案相比，假设在1000 笔交易里，其方案单个节点存储量最少约为 1 MB，本方案在该测试环境下约为 3 MB。本文的存储量略偏大原因在于：1)存储总量大小不同，其方案的存储总量为 6MB，而本文方案的存储总量为 7.5MB，这可能是由实验环境和选取参数不完全相同所导致的；2)存储原理不同，其整个网络只存储一份账本，全网的节点数量影响单个节点的存储量，而本文方案则根据组织数量存储账本，单个组织的节点数量影响单个节点的存储量。综上，Fabric-GS 比文献[9]的单个节点

存储量更大，但测试结果在合理的范围内。虽然本文方案牺牲了一定的存储空间，但换来了更安全的数据存储。




(a) The cost when the block size is 3 MB




(b) The cost under different information amounts

图 12 编解码时间开销测试

Fig. 12 Encoding and decoding time cost test

随着交易数量的增加，Fabric 整个网络的账本数据快速增长，而 Fabric-GS 则缓慢增加，相比之下两者的差距越来越大，GS 节点存储优势越来越明显。这是由于 Fabric 每个节点的账本数据都要存储 $n$ 次，其存储开销为 $O(n)$，而 Fabric-GS 节点的账本数据只需存储 1 次，其存储开销为 $O(1)$。因此，与未修改的 Fabric 节点相比，采用 GS 节点存储数据能够减少单节点及全网对存储空间的占用，并且节点和组织数量越多，优势越明显。

### 7.3 编解码时间开销

GS 节点对区块文件编解码会产生一定的时间开销，当区块文件一定，数据分段数量 $k$ 和校验分段数量 $r$ 不同会对区块文件的编码速度产生影响，如图 12(a)所示。关于编码开销，当 $r$ 不变时，随着 $k$ 逐渐增加，编码速度逐渐提升，但是 $k>175$ 时，速度有所下降。进一步分析，$k$ 数值过小时，编码分段的信息尺寸较大，存储在数组所占用的空间高于 $k$ 值相对较大的时候。当 $k$ 不变时，随着 $r$ 的增加，编码分段速率明显下降，因为编码计算数量增多了。关于解码开销，当校验数量分段 $r$ 值一定时，解码速率随着数据分段数量 $k$ 的增加而下降，从 230 Mbps 降到 140 Mbps 左右。解码计算相对于编码计算复杂度更高，因此速率会较低。此外，解码速率与数据分段数量 $k$ 有关，而校验分段数量 $r$ 对其几乎没有影响。查询历史数据时恢复区块文件可能需要解码，但在一些情况下，系统可以忽略解码计算：1)如果区块文件的所有数据分段没有丢失，则可以直接合成区块文件；2)已经以缓存存储等方式读取数据；3)网络中存在至少 1 个全节点，其他节点可快速调用全节点数据查询。以上 3 种情况的存在会大大降低解码给系统带来的影响。

区块文件的大小同样会影响编解码速度，假设压缩因子 $\theta$ 为 1，在区块文件大小不同的情况下，编解码速率各有不同，如图 12(b)所示。当 $r$ 不变，随着区块文件变大，编解码速率会逐渐提升，但当区块文件大小超过一定程度时，编解码速率下降。这是因为区块文件太小，但是文件分段较多，因此产生文件的输入输出开销大，导致速率过慢；而文件太大，会导致数据读取慢，因此编解码速率也会降低。此外，在压缩因子和区块文件大小不变的情况下，$k$ 值越大，编码速率相对偏低，这是因为切分数据分段数量多，编码速率受到了影响。

根据实验结果可知，编解码区块文件会对区块链网络造成影响，但是超级账本生成过万笔的交易才会生成一个大小约为 100 MB 的区块文件，等待时间较长。如果想要一个区块文件达到 1 GB 以上才开始编码，那么等待时间会更长。因此，编码一个区块文件时间是比较充足的，对现有区块网络不会造成太大负担。

**结束语** 本文的存储优化方法提出了一种新的区块链节点——分组存储节点，它设定每组节点只存储一个账本，来解决区块链节点冗余存储的问题，从而有效减少区块链网络的存储开销。在全网大部分节点失效的情况下，GS 节点利用纠删码和分组存储特性来恢复和同步原始区块文件。此外，利用 GS 节点对超级账本文件存储系统进行扩展，对区块文件进行分组存储，对解码恢复和同步更新的过程进行优化，从而实现了 GS 节点的具体应用。除了超级账本外，GS 节点的原理可以应用到其他区块链架构中，如企业以太坊联盟 EEA 等。我们还对 GS 节点的可扩展存储、可用性、可靠性和网络负载进行了分析。分析表明该节点可降低系统开销，其区块文件在系统可用，可使系统更安全可靠。最后，实验结果表明，区块文件占用区块账本空间最大，本文方案对其进行了优化，组织内的每个节点的存储空间不会随节点的增多而快速增加，反而还会减少，从而可以有效解决区块链节点冗余存储。此外，GS 节点编解码区块文件不会对现有区块网络造成太大负担。

## 参 考 文 献

[1] NAKAMOTO S. Bitcoin: A peer-to-peer electronic cash system [J]. Decentralized Business Review, 2008: 37(13): 2889-2897.

[2] LI S, SONG B Y, LI D, et al. Composite blockchain associated event tracing method for financial activities [J]. Computer Science, 2022, 49(3): 346-353.

[3] GUO X, WANG Y Y, FENG T, et al. Blockchain-based role-delegation access control for industrial control system[J]. Computer Science, 2021, 48(9): 306-316.

[4] ZHOU H, JIANG H, ZHAO Y, et al. Study on optimal scheduling of power blockchain system for consensus transaction of each unit[J]. Computer Science, 2022, 49(6A): 771-776.

[5] ETHERSCAN. Ethereum Full Node Sync(Archive) Chart [EB/OL]. https://etherscan.io/chartsync/chainarchive.

[6] PERARD D, LACAN J, BACHY Y, et al. Erasure code-based low storage blockchain node[C]// 2018 IEEE International Conference on Internet of Things(iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing(CPSCom) and IEEE Smart Data (SmartData). IEEE, 2018: 1622-1627.

[7] WILKINSON S, BOSHEVSKI T, BRANDOFF J, et al. Storj a peer-to-peer cloud storage network [EB/OL]. https://www.storj.io/storjv2.pdf.

[8] TRÓN V, FISCHER A, NAGY D A, et al. Swap, swear, and swindle: Incentive system for swarm [EB/OL]. https://ethersphere.github.io/swarm-home/ethersphere/orange-papers/1/sw%5E3.pdf.

[9] ZHAO G F, ZHANG M C, ZHOU J H, et al. Research and application of block file storage model based on blockchain system of erasure code[J]. Netinfo Security, 2019, 19(2): 28-35.

[10] YIN F R, ZHU C Y, ZHAO B, et al. Erasure code partition storage based on CITA blockchain[J]. Journal of East China Normal University(Natural Science), 2021(5): 48-59.

[11] ANDROULAKI E, BARGER A, BORTNIKOV V, et al. Hyperledger Fabric: A distributed operating system for permissioned blockchains[C]// Proceedings of the Thirteenth EuroSys Conference. 2018: 1-15.

[12] BENET J. IPFS-content addressed, versioned, P2P file system [J]. arXiv: 1407.3561, 2014.

[13] SUN X Z, ZHANG X, XIANG F, et al. Survey of storage scalability on blockchain [J]. Journal of Software, 2021, 32(1): 1-20.

[14] ZHENG Q, LI Y, CHEN P, et al. An innovative IPFS-based storage model for blockchain[C]// 2018 IEEE/WIC/ACM International Conference on Web Intelligence (WI). IEEE, 2018: 704-708.

[15] SHARMA P, JINDAL R, BORAH M D. Blockchain technology for cloud storage: A systematic literature review[J]. ACM Computing Surveys(CSUR), 2020, 53(4): 1-32.

[16] WANG Y J, XU F L, PEI X Q. Resarch on erasure code-based fault-tolerant technology for distributed storage [J]. Chinese Journal of Computers, 2017, 40(1): 236-255.

[17] 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.

[18] TIAN J, DAI Y F. Study on durable peer-to-peer storage techniques[J]. Journal of Software, 2007(6): 1379-1399.

[19] TAN S, JIA Y, HAN W H. Research and development of provable data integrity in cloud storage [J]. Chinese Journal of Computers, 2015, 38(1): 164-177.

[20] CAI Z H, LIN J Y, LIU F. Blockchain storage: Technologies and challenges[J]. Chinese Journal of Network and Information Security, 2020, 6(5): 11-20.

[21] GALLAGER R. Low-Density Aarity-Check codes [J]. IEEE Transactions on Information Theory, 1962, 8(1): 21-28.

[22] LIANG T, ZHANG P, LIU C, et al. Efficient encoding of quasi-cyclic low-density parity-check codes[C]// 2018 IEEE 3rd Advanced Information Technology, Electronic and Automation Control Conference(IAEAC). IEEE, 2018: 1189-1193.

[23] NGUYEN T T B, NGUYEN TAN T, LEE H. Efficient QC-LDPC encoder for 5G new radio[J]. Electronics, 2019, 8(6): 668.

[24] XIE T, LI B, YANG M, et al. Memory compact high-speed QC-LDPC decoder[C]// 2017 IEEE International Conference on Signal Processing, Communications and Computing (ICSPCC). IEEE, 2017: 1-5.

[25] LIN S, COSTELLO D J. Error control coding, second edition [M]. Prentice-Hall, Inc. 2004.

[26] BOSE R C. On the construction of balanced incomplete block designs[J]. Annals of Eugenics, 1939, 9(4): 353-399.

[27] ZHANG F, MAO X, ZHOU W, et al. Girth-10 LDPC codes based on 3-D cyclic lattices[J]. IEEE Transactions on Vehicular Technology, 2008, 57(2): 1049-1060.

**ZHANG Yushu**, born in 1987, Ph.D, professor, Ph.D supervisor, is a member of China Computer Federation. His main research interests include blockchain, multimedia information security and artificial intelligence.

(责任编辑:何杨)