BSS: 一种联盟链存储优化方案

阎 萌[1,2],杨 颖[1,2],王 刚[1,2*],刘晓光[1,2]
(1. 南开大学计算机学院,天津 300350;2. 天津市网络与数据安全技术重点实验室,天津 300350)

摘 要: 针对联盟链系统普遍采用的全副本存储模式导致存储可扩展性及安全性降低的问题,提出一种基于动态弹性区块散布的存储分片方案. 在全网划分若干存储组,同一组内节点以协作方式维护固定数量区块链副本,从而在大幅降低存储开销的同时保证存储可靠性及数据访问性能. 具体地,初始散布机制为每个新区块随机分配固定数量初始副本;根据实时区块访问热度变化,动态复制机制弹性增加热区块副本数量,存储优化机制则将冷区块由副本态切换为编码态. 根据模拟实验结果,与使用全副本存储模式相比,具有2节点容错能力的存储分片系统可将节点存储开销降低约71%,同时区块访问性能保持在较为良好的水平.

关键词: 联盟链;区块链可扩展性;分布式存储;数据散布;存储分片
基金项目: 国家自然科学基金项目(No.62141412, No.61872201);天津市科技发展计划项目(No.20JCZDJC00610, No.19YFZCSF00900)
中图分类号: TP302 **文献标识码:** A **文章编号:** 0372-2112(2024)04-1364-13
电子学报URL:http://www.ejournal.org.cn **DOI:**10.12263/DZXB.20220219

BSS: A Storage Optimization Scheme for the Permissioned Blockchain

YAN Meng[1,2], YANG Ying[1,2], WANG Gang[1,2*], LIU Xiao-guang[1,2]
(1. *College of Computer Science*, *Nankai University*, *Tianjin* 300350, *China*;
2. *Tianjin Key Laboratory of Network and Data Security Technology*, *Nankai University*, *Tianjin* 300350, *China*)

Abstract: For the permissioned blockchain, the commonly-used full-replication storage mode impacts the storage scalability and security. To solve the problem, a blockchain storage sharding scheme was proposed, which offered good performance, high reliability, and low storage overhead. The network is divided into several storage units, each of which holds a certain number of blockchain copies through cooperation among nodes. The initial allocation distributes each new block to fixed-number of nodes. According to the changing access pattern, the adaptive replication increases replicas of popular blocks, and the storage optimization switches unpopular blocks into encoded mode. Experimental results show that, compared with the full-replication mode, the proposed scheme reduces the storage overhead of full nodes by on average 71%, with the performance kept at a proper level.

Key words: permissioned blockchain; blockchain scalability; distributed storage; data dispersion; storage sharding
Foundation Item(s): National Natural Science Foundation of China (No.62141412, No.61872201); Tianjin Science and Technology Development Plan Project (No.20JCZDJC00610, No.19YFZCSF00900)

1 引言

近年来,区块链技术在各领域得到广泛应用,其中联盟链技术凭借高吞吐率、隐私性及多场景支持性应用最广[1,2]. 区块链系统要求全节点存储完整数据副本,而这逐渐引发新的问题. 截至2021年,比特币数据总量超过350 GB[3],2019年以太坊数据总量超过2 TB. 联盟链交易吞吐率高于公有链,因此数据增长更快. 假设某联盟链平均区块大小为2 MB,出块间隔为2 s,则1个月将产生2.5 TB数据. 较大的存储开销提高了全节点的加入门槛,并导致全节点数量锐减. 这将影响区块链系统去中心性和可扩展性,进而威胁系统安全. 因此,优化区块链尤其是联盟链

收稿日期:2022-03-02;修回日期:2022-06-20;责任编辑:王天慧
*通讯作者:王刚

存储模式,降低全节点存储开销成为一个重要研究方向[4].

许多研究工作致力于解决上述问题.例如,简单支付验证机制允许轻节点在存储少量数据[5]甚至不存储额外数据[6,7]的情况下验证交易合法性,不过轻节点只能使用部分系统功能且无法参与共识.修剪技术允许节点只保存少量区块[8~10]或交易[11,12]且不影响节点参与共识,但存储可靠性和数据检索效率大大降低.2018年,CUB[13]提出存储分片思想:在全网划分若干存储组,通过区块散布机制使组内节点以协作方式维护一份或多份区块链副本.存储分片能够在节点存储开销、存储可靠性和数据检索性能之间取得良好平衡,但相关研究工作仍有待进一步探索和完善.例如,一些方案[14~16]提出的区块散布算法主要针对静态场景且散布过程依赖中心化节点,导致方案适用性受限.另一些方案虽适用于动态场景,但采用了中心化散布[17],或者未结合区块访问热度进行性能优化[18~20].

本文深入研究适用于动态场景的联盟链存储分片方案.结合副本技术和纠删码技术,根据区块访问热度变化弹性调整副本个数、副本位置及存储状态,以尽可能低地存储开销,保证较高的存储可靠性和访问性能.具体地,设计初始散布机制为每个新区块分配固定数量初始存储节点,以保证数据可靠性;设计动态复制机制弹性调整历史区块的副本数量,以提升数据访问性能;设计存储优化机制将最近较少访问的历史区块由副本态切换为编码态,在不影响可靠性的前提下进一步降低存储开销.区块散布过程无需中心化管理节点.

本文的主要贡献如下:(1)专注于动态场景下的联盟链存储分片研究,提出动态弹性区块散布方案,对区块链存储模式优化进行了创新性的探索;(2)充分利用区块链数据特点及数据访问特性,根据访问热度变化及时调整副本数量和存储状态,在低存储开销、高访问性能、高存储可靠性之间取得良好平衡;(3)针对所提出策略进行全面充分的实验测试,从存储消耗、系统性能、自适应性等多角度对方案进行分析评估.

2　相关工作

2.1　区块链存储可扩展性

区块链目前采用的全副本存储模式导致全节点存储开销过大,并引发如下两方面问题.(1)全节点数量减少,系统安全性降低.由于使用普通硬件设备难以胜任全节点所负责的数据存储任务,全网范围内全节点数量减少.这将削弱系统去中心性,并影响分布式系统安全.(2)新加入节点同步数据困难.对新节点来说,同步数据的时间开销和带宽开销过大,且扫描区块建立世界状态时的计算开销过大.针对上述问题,许多研究工作致力于提升区块链系统的存储可扩展性.已提出的方案可分为如下三类.

(1)SPV方案　简单支付验证(Simplified Payment Verification,SPV)使得区块链轻节点仅存储少量数据便可验证交易合法性.例如,Bitcoin[5]轻节点仅需存储区块头.当验证一笔交易时,轻节点向全节点索要默克尔证明,以证明该笔交易已被打包到最长链.NIPoPoWs[6]轻节点无需存储任何数据.当验证一笔交易时,轻节点向全节点索要两个证明:(1)后缀证明,以证明该全节点持有当前最长链;(2)前缀证明,以证明指定交易已被打包到最长链.不过,NIPoPoWs方案只在挖矿难度保持不变时有效.FlyClient[7]方案在挖矿难度变化的情况下仍旧有效,且证明长度比NIPoPoWs更短.总的来看,SPV方案允许资源受限用户使用区块链系统,增强了系统可扩展性,但轻节点无法参与共识,系统去中心性并未增强.

(2)修剪方案　区块链修剪技术(pruning)允许节点删除一部分历史数据并且不影响节点参与共识.已提出方案可归纳为如下两类.(*a*)区块修剪.比特币允许节点只存储最近新产生区块[8]或某个高度之后的区块[9].CoinPrune[10]允许节点定期创建区块链状态快照并删除快照之前所有区块.不过,若系统中大部分节点执行了区块修剪,则历史数据的可访问性和可靠性大大降低,给数据审计带来困难.(*b*)交易修剪.Jidar[11]允许节点只保存与自己相关的交易,即自己是发送方或接收方的交易.文献[12]提出联盟链节点可以删除不影响区块链状态且预计未来一段时间不会被访问的交易.不过,交易修剪的弊端在于,节点在尝试恢复完整区块的过程中需要与多节点进行交互,导致数据恢复效率低下.

(3)存储分片　存储分片可视为特殊的修剪策略,因为节点之间的行为是协作式的:将全网划分若干存储组,在组内使用区块散布机制保证每个节点存储的区块集合各不相同,从而所有节点可以"拼凑"出完整的区块链副本.已提出的存储分片方案可归纳为如下几类.(*a*)优化问题求解.以CUB[13]为代表,这类方案对区块散布问题进行形式化定义并求近似最优解.优化目标是降低区块访问时延[14,15]或保证区块链数据可用性[16].不过,上述方案主要针对静态场景(如区块链数据不增长、访问概率分布已知等),导致方案适用性受限.另外,所提出的区块散布算法需要中心化管理节点,影响系统去中心性和安全性.(*b*)基于编码.这类方案针对区块链增长的动态场景,将新区块先编码再散布.GCBlock[17]提出基于纠删码的区块散布方案,但散

布过程需要中心化管理节点. BFT-Store[18]结合了纠删码和副本两种存储技术,且实现拜占庭容错,但并未考虑对不同热度的区块弹性调整散布策略. (c)复制率调度算法. ElasticChain[19]和SE-Chain[20]提出动态调整不同区块的复制率以保证系统安全性,但并未考虑节点间数据请求的效率问题.

较之于修剪方案,存储分片方案能够更好地在系统安全性、数据可靠性和数据检索性能间取得平衡. 已提出的存储分片方案各有所长,但针对动态场景的分布式区块散布算法还有待进一步研究和探索,尤其应结合区块访问热度进行性能和存储方面的优化.

2.2 数据散布

在点对点(P2P, Peer-to-Peer)存储系统中,给定若干存储节点及数据对象,数据散布算法确定每个数据对象的副本个数及存储位置. 根据适用系统类型的不同,数据散布算法可分为三类. (1)结构化P2P系统算法[21,22]. 数据对象的存储位置及查找过程遵循分布式哈希表(Distributed Hash Table, DHT). 数据检索性能指标为成功检索某一数据对象平均所需最小DHT跳数. (2)非结构化P2P系统算法[23,24]. 数据散布位置无特定规则,数据查找基于洪泛(flooding)等技术. 数据检索性能指标为成功检索某一数据对象平均所需问询的节点数. (3)P2P社区[25,26]算法. 假设P2P社区网络与一外部网络(如广域网)相连,若数据对象在社区内部检索失败则可在外部网络中继续检索. 数据检索性能指标为数据对象在P2P社区内的命中率. 数据复制常用来提升P2P存储系统的数据访问性能. 例如,路径复制将被请求数据复制在搜索路径上的全部或一部分节点[27],邻居复制将被请求数据(或数据索引)复制在数据持有节点的邻居节点[28].

本文提出的数据散布和检索方法借鉴了Kademlia DHT[29]的设计思想,并针对存储分片系统进行了改进和优化. (1)传统P2P系统一般包含成百上千个节点且节点流动性大,因此传统散布算法多采用高冗余度随机散布策略,以在较大概率下保障数据可用性. 作为对比,联盟链存储分片系统只包含数十个节点且节点流动性低,沿用传统散布策略可能导致数据冗余度过大且数据访问性能不佳. 本文所提出的数据散布策略经过精心设计以达到较高的数据访问性能和较低的存储开销,同时保证区块副本分布与区块访问概率分布高度吻合. (2)传统P2P系统的研究目标是在节点流动性较高的情况下保证数据高可用性,而本文的研究目标是以最小的存储开销来保证较高的数据可靠性和数据访问性能. 作为总结,本文的研究工作可以看作是传统P2P数据散布和数据复制在联盟链存储分片这一新场景下的拓展和应用.

3 设计概览

3.1 数据模型

对区块链数据结构进行如下定义. 如图1所示,一个区块包含两部分,其中区块头记录元数据,区块体记录一段时间内产生的交易(按默克尔树结构打包). 每个区块对应唯一标识符blockID,即区块头SHA256(Secure Hash Algorithm, SHA)哈希值;每个交易对应唯一标识符txID,即交易SHA256哈希值. 区块头prev Hash字段记录前一个区块的blockID,由此所有区块前后链接为链式结构. 区块头root Hash字段记录区块体包含所有交易的哈希摘要,由此交易之间、区块体与区块头之间形成链式结构. 在链式结构及共识机制的共同作用下,成功篡改区块链数据而不被发现的概率很低.

本文提出以区块(而不是交易)为单位对区块链账本进行切分,以使得区块内部设计精巧的链式结构得以保留,从而数据不可篡改性得以最大程度保留,系统设计也较为简洁. 相反地,假如以交易为单位切分账本(即每个节点保存交易集合),为保证数据不可篡改性必须引入新的交易验证协议和区块验证协议,且节点需要额外存储每个交易的存在性证明,系统复杂程度大大增加.

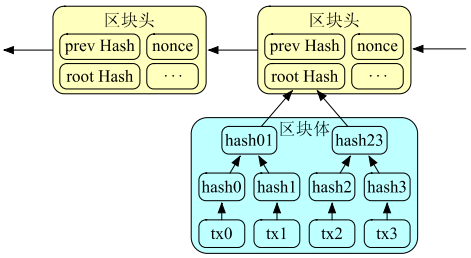


图1 区块链数据结构

3.2 系统模型

一个区块链存储分片系统(Blockchain Storage Sharding, BSS)由联盟链中的一部分节点构成,是一个相对独立的存储组. 全网内包含若干个这样的存储组(或称"BSS系统"). 全节点可以选择加入某个BSS系统从而与其它节点协作式地维护区块链副本,降低存储开销. 这将改变节点的数据存储模式,但并不会影响共识层协议. 称加入BSS系统的全节点为"服务节点"而它们所服务的轻节点为"用户节点".

(1)服务节点负责维护区块链、存储区块链数据并提供数据查询服务. 一个BSS系统一般包含数十个服务节点. 服务节点负责存储全部区块头,一部分区块体,以及一部分交易索引. 服务节点通过协作维护固定

数量区块链副本,并通过数据请求访问本地未保存区块. 每个服务节点被分配256 bit长度的唯一标识符nodeID,即节点公钥SHA256哈希值.

(2)用户节点使用区块链服务但不维护区块链数据. 每个用户节点连接到一个服务节点以对区块链数据进行写入和读取. 用户节点可能会创建新交易,部署智能合约,或触发执行一个已部署的智能合约. 用户节点可能会要求服务节点返回历史区块或历史交易,这时服务节点必须尽快响应.

对服务节点做出如下假设:假设联盟链有身份准入机制,服务节点间彼此互知;由于联盟链节点之间存在合作关系,因此假设服务节点半可信,即可能遭遇故障而不提供数据服务或提供错误数据,但不会故意作恶. 以下统一将"服务节点"简称为"节点",将"用户节点"简称为"用户".

3.3　系统工作流程

假设用户创建一笔新交易并发送给连接的节点. 该交易将被广播至全网,并根据共识协议打包到一个新区块. 新区块生成后,每个节点执行初始散布决定是否保存该区块. 如图2所示,区块4经过初始散布由节点b和d进行保存,而节点a和c仅保存区块4的区块头. 初始散布机制保证每个区块在系统内的副本数不少于某一固定值.

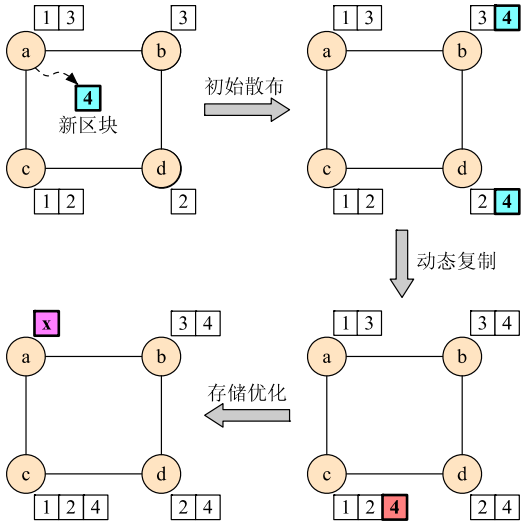


图2　系统工作流程

假设用户想要获取某个历史区块但所连接的节点并未存储该区块. 因此,节点在接收到用户请求后向其它节点进行请求,并在成功获取目标区块且验证有效后将区块返回给用户. 节点间区块请求可能会触发对被请求区块的动态复制. 如图2所示,动态复制使得区块4在系统中增加一个副本(由节点c存储). 动态副本设置有生命周期,因此一段时间后节点c会删除区块4的副本. 动态复制机制可以提升数据访问性能.

系统后台运行的存储优化机制将最近较少访问的历史区块成批编码并删除系统内多余副本,从而在不降低数据冗余度的基础上压缩存储开销. 如图2所示,存储优化机制使得节点a所保存的区块1、3被校验块x所替代. 存储优化机制可以降低存储开销.

4　BSS设计方案

4.1　初始散布

BSS数据散布规则基于节点与数据对象的逻辑距离. BSS系统中区块标识符、交易标识符和节点标识符都是256 bit长度字符串. 定义某节点与某区块或交易之间"距离"为相应标识符按位异或的结果. 以长度为4位的标识符举例,nodeID为0010的节点与blockID为0101的区块之间距离为7,而同一节点与txID为0001的交易之间距离为3. 可以看到,两个标识符的公共前缀越长则距离越近.

每个节点维护一个节点列表来记录系统内所有其它节点的信息. 如图3所示,节点列表呈前缀树结构. 每个叶子节点对应一个BSS节点,记录该节点nodeID、IP地址和端口. 从树根节点到叶子节点的路径即相应BSS节点nodeID的最短唯一前缀. 给定blockID或txID,使用树状结构的节点列表可以快速找到距离相应区块或交易最近的节点. 在图3所示的例子中,给定以"010"为前缀的blockID,可以找到距离该区块最近的BSS节点,即叶子结点"011"和"000"对应的节点.

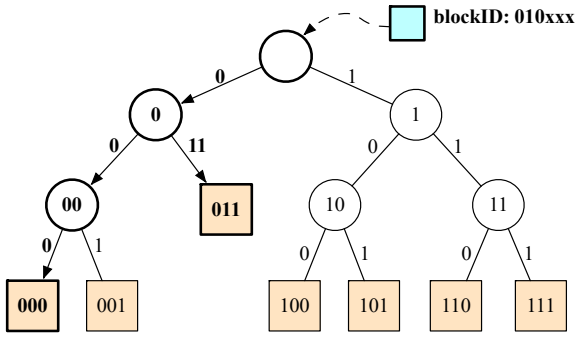


图3　节点列表示意图

(1)初始散布过程　在一轮共识完成、一个新区块被确认后,节点执行初始散布.

(*a*)散布区块. 节点检索本地节点列表找到距离新区块最近的*r*个节点,亦即该区块的"初始宿主". 若节点是初始宿主之一则存储完整区块数据{blockID, blockData};若节点不是初始宿主则只存储区块头数据{blockID, blockHeader}. 初始宿主不能够删除初始副本,除非相应区块被切换为编码模式(具体内容见第

4.3节).

(b)散布交易索引. 对于新区块包含的每一笔交易,节点检索本地节点列表找到距离该交易最近的r个节点,亦即该交易的"索引宿主". 若节点是索引宿主之一则存储索引项{txID, blockID};若节点不是索引宿主则什么都不做. 索引宿主不能够删除所存储的索引项. 对索引项进行散布是为了支持交易检索功能(具体内容见第5.1节).

(2)分析 初始散布机制具有如下优点.(a)简洁:节点无需记录区块或索引项的存储位置. 若需确认初始宿主或索引宿主,只需检索节点列表找到距离该数据最近的r个节点.(b)可靠性:给定初始散布参数r,则系统能够容忍$r-1$个节点故障.(c)存储负载均衡:由于数据被随机散布,各节点存储负载相对均衡. 然而,仅仅执行初始散布还不够. 由于区块链上不同区块访问热度差异较大,系统内通信负载和数据传输负载不均衡,导致节点间数据访问性能表现不理想. 因此,需要根据区块访问热度动态调整不同区块的副本数量,从而在性能方面进行优化.

4.2 动态复制

BSS执行动态复制的依据是区块访问热度. 每个节点维护一个局部热度表来记录本地保存的所有区块在最近两个纪元(epoch)内的访问频次. 热度表中id指区块高度,变量last指区块在上一epoch的全局平均访问次数,变量curr指区块在当前epoch的本地局部访问次数(包含节点对自己的访问). 节点间通过协作来统一更新last值,过程如下.(1)在每个epoch结束时,每个节点向其它节点广播局部热度表{id, curr}. 假设共有n个节点,则每个节点会收到$n-1$个局部热度表.(2)节点统计本地保存的所有区块的全局总访问次数,并将总访问次数除以节点数得出全局平均访问次数.(3)节点用算出的全局平均访问次数更新last值,并将curr值置0. 自last值更新完成至下一个epoch结束前,节点只更新curr值(伴随数据访问实时更新). 图4给出上述更新过程的一个示例(假设BSS系统中包含3个服务节点). 定义一个区块的实时热度值等于last值加上curr值.

(1)请求者复制 每隔固定时间,每个节点从最近向其它节点请求过的区块中选择热度最高者复制到本地. 为执行请求者复制,节点在每次进行数据请求时不仅要求被请求节点返回区块,还要求返回区块当前热度值. 以下称正在执行请求者复制的节点为"主节点". 执行过程如下.

(a)确定待复制区块 自上一次执行请求者复制后所有被主节点请求过的区块将作为候选区块. 对所有候选区块按热度降序排序并选择前$1/n$比例作为待复制区块(假设系统共有n个节点).

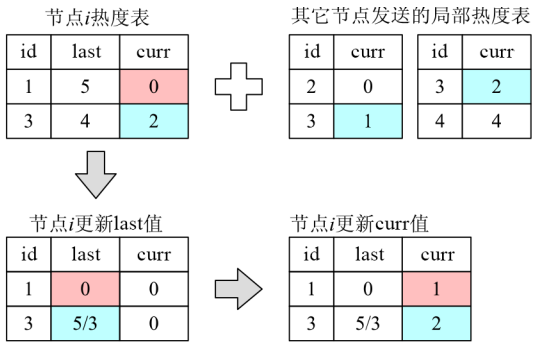


图4 热度表更新过程

(b)选择动态宿主 对每个待复制区块,选择尚未保存该区块的节点中距离该区块最近者作为动态宿主.

(c)执行复制 主节点通知动态宿主复制对应区块,并向所有节点广播新的复制状态. 为追踪全局复制状态,每个节点维护一个复制表{blockID, nodeID},即相应区块被动态复制到相应节点.

(2)持有者复制 若节点发现本地存储的区块访问热度过高,便将该区块复制给其它节点以缓解本地访问负载压力. 以下将正在执行持有者复制的节点称为"主节点". 执行过程如下.

(a)选择区块 给定系统参数p和p',主节点检查热度表选出热度大于的p区块并将选中区块划分等级:热度区间在$[p, p+p')$的区块级别为g=1,热度区间在$[p+p', p+2p')$的区块级别为g=2,热度大于等于$p+2p'$的区块级别为g=3.

(b)确定副本个数 级别g=1, 2, 3的区块将分别增加1, 2, 3个动态副本.

(c)选择动态宿主 对于某个待复制区块,从尚未保存该区块的节点中选择g个距离该区块最近者作为动态宿主.

(d)执行复制 主节点通知动态宿主复制对应区块,并向所有节点广播新的复制状态.

每个动态副本的生命周期设置为T,即动态副本将在被保存T个epoch后被删除. 删除操作将进行广播以便系统内所有节点更新复制表.

(3)分析 动态复制机制具有如下优点:(a)请求者复制使得最近访问过的热区块在系统内副本增加,提升了热区块的访问性能.(b)持有者复制使得访问过载节点将通信和数据传输任务及时卸载给其它节点,避免请求者等待以及网络拥塞.(c)设置生命周期可以保证系统内动态副本的生成率等于删除率,使节点存储开销保持稳定.(d)实验测试结果显示,热度表的设计带来较大性能提升. 相比于节点仅统计本地局部访问次数,维护全局访问次数可使节点更准确地掌握区

块热度变化,实现精准复制,从而大幅提升系统性能.

虽然初始散布和动态复制机制保证了数据可靠性和数据访问性能,不过节点存储开销仍有可优化空间.考虑到在许多区块链系统中不同区块的访问概率差异较大(如较早区块的访问概率较低而较新区块的访问概率较高),可以对较少访问的区块进行数据压缩,从而进一步降低节点存储开销.

4.3　存储优化

BSS存储优化将最近较少访问的历史区块由副本态切换为编码态.区块编码由被编码区块的初始宿主合作完成,过程基于Reed-Solomon(RS)编码.编码参数(k, m)表示利用k个冷区块生成$k+m$个块,且使用任意k个块即可恢复初始的k个冷区块[30].对某个区块,规定其编码"主节点"为所有初始宿主中距离该区块最近者.每个节点持续监测其所主导编码的区块,若存在连续$2T$时间内未被访问的区块则将其标记为"冷区块"并放入编码缓存区.每个节点维护一个编码表来追踪被编码区块的信息.

(1)编码过程　若某节点编码缓存区中包含k个被存储在不同节点的区块,节点启动编码过程.图5(a)给出一个示例.假设初始散布参数r=3且RS码参数(k, m)=(4,2).节点a编码缓存区内区块2,5,8,9刚好被存储在4个不同的节点b、c、d、e,于是节点a启动编码.具体步骤如下.

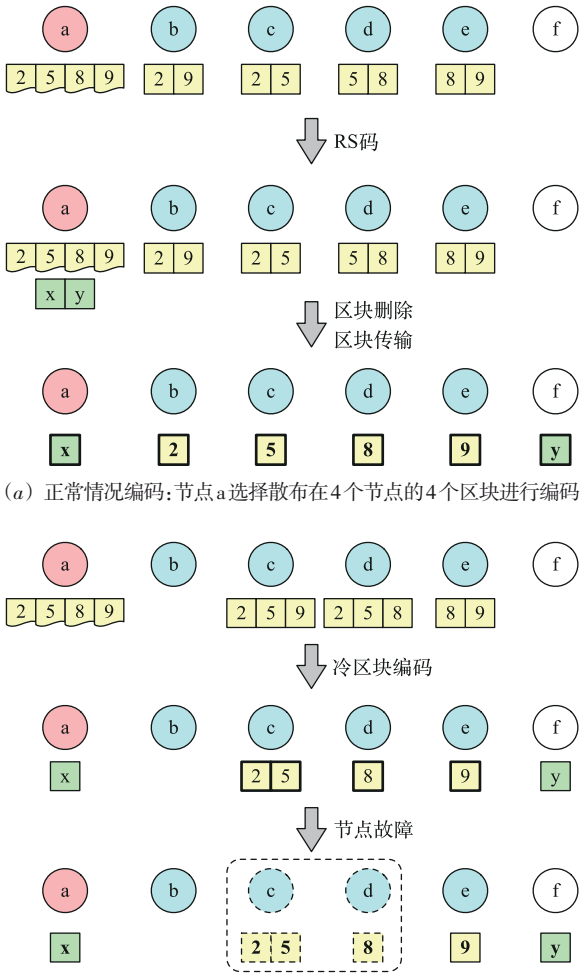
(a)节点a对区块2、5、8、9使用RS码,计算得到校验块x,y.

(b)节点a通知节点b,c,d,e执行区块删除,以使得删除后每个节点刚好存储一个不同的区块.

(c)节点a将校验块x存储在本地,并选择一个新的节点f存储校验块y.

(d)节点a广播新的编码状态{block 2,5,8,9 => node a,f},即区块2、5、8、9被作为一组进行编码且生成的校验块被存储在节点a、f.节点a无需广播区块2、5、8、9的存储位置,因为这4个区块都被存储在距离它们第二近的节点上.

需要强调,编码主节点必须等到编码缓存区内k个区块刚好被散布在k个不同节点时再启动编码,否则系统的容错能力会降低.图5(b)给出一个反例.区块2、5、8、9被散布在3个节点c、d、e,编码主节点执行编码后节点c负责存储区块2、5.假设某个时刻节点c、d崩溃,则由于可用数据块不足4个,区块2、5、8、9无法还原.此时系统无法容忍2节点故障.一个解决办法是在编码后将区块2或区块5传输给节点d保存,但这就打破了编码态节点存储在第二近区块的规则,因此也不可取.



(a) 正常情况编码:节点a选择散布在4个节点的4个区块进行编码

(b) 错误示范:节点a选择散布在3个节点的4个区块进行编码,导致系统无法容忍两节点故障

图5　存储优化机制:区块编码的正例与反例.

(2)分析　存储优化机制具有如下优点:(a)编码结束后,冷区块在系统内只有一个副本,并与其它冷区块共享若干校验块,降低了系统存储开销.(b)得益于RS码的特点,尽管存储开销降低,数据可靠性水平并未降低.(c)编码过程和删除过程都很简洁.(d)编码态区块在系统内的唯一副本存储在距离其第二近的节点上,因此无需额外记录其存储位置.执行存储优化后系统中存在两种数据存储状态:编码态和副本态.前者的访问时延可能会略大于后者,但这并不会对系统整体性能造成较大影响.

5　数据请求

5.1　数据请求

若某节点请求一个区块,第一步需要检查编码表以确认目标区块是否处于编码态.若目标区块处于编

码态,节点搜索节点列表并找到距离目标区块第二近的节点,即该区块副本的存储节点. 存储节点收到数据请求并返回区块后启动对被请求区块的动态复制. 尚未保存该区块的节点中距离该区块最近者将被选为动态宿主. 若目标区块处于副本态,节点向目标区块的若干个初始宿主或动态宿主发送消息,并在成功接收目标区块后再次发送消息以免重复发送. 成功接收到目标区块后,请求节点执行区块验证. 首先验证区块头,然后验证区块体. 区块验证通过,则本次数据请求成功.

若节点请求一笔交易,第一步需要向索引宿主获取索引项{txID, blockID},方法与向初始宿主获取区块的方法类似. 接收到索引项{txID, blockID}之后,请求节点使用blockID查询节点列表和复制表,获取目标区块的初始宿主和动态宿主,并向宿主请求数据. 宿主在返回目标交易的同时还会返回交易所对应的默克尔证明以证明该笔交易的有效性. 请求节点对默克尔证明执行验证,若验证通过则本次数据请求成功.

5.2 故障恢复

本节考虑节点出现瞬时故障的情况. 假设故障节点一段时间后恢复正常且其存储的数据将再次可用. 对于副本态区块,若少于$r-1$个节点出现故障则区块请求过程不会受到影响. 对于编码态区块,当出现节点故障时需要进行数据恢复. 本节主要讨论编码态区块的故障恢复. 如图6所示,在节点a对区块2、5、8、9执行编码得到校验块x、y后,某个时刻节点b故障. 节点c访问区块2失败,于是发消息通知其它节点. 节点a在接收到通知后启动对区块2的数据恢复. 节点a向节点c、d、e、f请求区块5、8、9、y. 只要有3个节点返回数据则节点a可以成功恢复出区块2. 在节点b再次可用前,节点a代为存储区块2,并向其它节点广播自己暂时存储区块2.

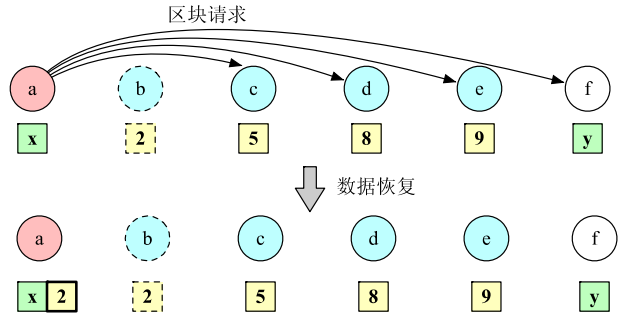


图6 编码态区块的故障恢复

5.3 节点加入或退出

新节点加入系统需要进行数据同步. 首先,新节点需要获取系统内所有节点的信息并建立节点列表. 然后,新节点向多个节点并行请求数据以获取全部有效区块头. 最后,新节点向其它节点请求复制表和编码表. 新节点的加入会导致一部分区块和交易索引的初始宿主发生改变,但是新节点无需主动同步相关数据,除非遇到以下两种情况:(1)与新节点连接的用户想要请求某个区块或者交易,而相关数据理应由新节点保存;(2)有其它节点向新节点请求某个区块或者交易索引,而相关数据理应由新节点保存. 这时,新节点执行第5.1节所述数据请求过程,将相应数据返回给用户或节点,并本地保存该数据.

若有节点想要退出系统,则退出节点必须将负责保存的所有初始副本传输给其它节点. 具体地,对于所有需要传输的区块或者交易索引,节点将数据迁移至尚未保存该数据且距离其ID最近的节点. 退出节点必须在完成全部传输任务之后再退出系统.

6 实验

对本文所提出的存储分片方案进行实验测试,评估其在性能、存储开销等方面的表现并与动态区块散布方案BFT-Store[18]进行对比,相关实验结果在第6.2节给出. 从性能角度与静态区块散布方案CUB[13]进行对比,相关实验结果在第6.3节给出.

6.1 实验设置

(1)区块链设置 参考真实Bitcoin数据模拟包含400个区块的区块链. 具体地,使用BigQuery[31]提供的Bitcoin数据库,截取高度654 000至654 399的区块大小作为模拟区块链中区块0至区块399的大小.

(2)节点设置 在PC机上模拟10个全节点的运行情况. PC机的硬件配置为Intel(R) Core™ i5-8265U处理器和8 GB RAM,操作系统为Windows 10. 根据正态分布设置节点间通信开销. 具体地,对任意两节点,从正态分布N(1, 1)中随机选择大于0的值作为其通信开销,单位是s/MB.

(3)访问分布 使用Zipf分布模拟区块链上各区块的访问概率分布. 具体地,对于一次区块访问,一个排名为i的区块被访问到的概率为

$$\mathrm{AP}_i=\frac{1}{\sum_{j=1}^{l}\left(\frac{1}{j}\right)^s\cdot i^s} \tag{1}$$

式(1)中:l表示区块链长度,s为预定义系统参数,区块排名i=1, 2, …, l. 可以看出,排名越小的区块被访问到的概率越大. 关于如何对区块排名,考虑两种方法:(a)高度越高则i值越小,即更新的区块有更大的访问概率. 由此得到的访问分布称为“Zipf分布”. 在以Bitcoin,NIPoPoWs和FlyClient为代表的基于SPV的区块链中常见Zipf分布. (b)随机分配i的值. 由此得到的访问分布称为“随机Zipf分布”. 在以Filecoin[32]和Storj[33]

为代表的P2P存储系统中常见随机Zipf分布. 由于在Zipf分布和随机Zipf分布下得到的实验结果非常接近,因此下文给出的大部分结果只展示Zipf分布的情况. 设置Zipf分布参数s=1.2.

(4)随机区块访问过程　使用泊松过程模拟节点访问区块的过程. 具体地,令k表示一个全节点在一个epoch内发起的区块访问个数,则k的值服从泊松分布:

$$P(k)=\frac{\lambda^k}{k!}e^{-\lambda},k=0,1,\cdots,K \tag{2}$$

参数λ的值等于k的期望,亦即λ的值决定了各个节点的平均访问负载. 在下文所有实验中设置λ=10或20,K=25.

(5)BSS实验设置　模拟两个BSS系统,其中$BSS^{(3)}$运行三副本初始散布和动态复制,$BSS^{(2)}$在$BSS^{(3)}$的基础上加入存储优化且编码后的区块保留2个副本. 这两个系统都具有2节点容错能力. 表1给出了系统实现的相关程序,具体执行流程如下. $BSS^{(3)}$系统:第h个epoch开始后每个节点执行NewBlock(b)和初始散布InitAlloc(b,r);执行持有者复制OwnerR(p,p')以及随机区块访问RqstSet(K)和BlockRqst(R);执行请求者复制RqstR(n)和区块删除Deletion(T). $BSS^{(2)}$系统:在$BSS^{(3)}$的基础上,执行完区块删除后,每个节点在满足编码条件时执行Encode(k, m)并删除1个初始副本. 实验开始前的初始状态设置为:区块链上有10个区块(区块0到9),其中每个区块被随机分配给3个初始宿主①. 实验过程中模拟从epoch 10到epoch 399的动态过程.

(6)BFT-Store实验设置　BFT-Store是一个动态区块散布方案,采用了纠删码与副本结合的存储模式. 纠删码的设计在低存储开销的前提下保证容错性,副本的设计可以提升数据访问性能. 模拟两个BFT-Store系统,其中系统$BFT^{(2)}$使用(8,2)的RS编码和二副本策略,与$BSS^{(2)}$进行对比;系统$BFT^{(3)}$使用(8,2)的RS编码和三副本策略,与$BSS^{(3)}$进行对比. 假设每个节点存储全部区块头,用于在数据请求阶段验证获取到的区块是否正确②. 这两个系统都能够容忍2个节点出现拜占庭错误③. 实验测试中,每隔8个epoch执行一次区块散布,过程如下:以8个区块为一个编码组生成2个校验块,将这10个块统一散布给系统内的10个节点,且每个块在系统中有2或3个副本. 图7给出系统$BFT^{(3)}$的散布过程示例. 在执行区块请求时,假设每个节点都会保存尚未执行散布的区块,因此节点只请求已经散布完成的区块.

表1　BSS系统实现的主要程序

程序名	描述
NewBlock(b)	更新区块链.向区块列表中添加新区块b.
InitAlloc(b, r)	初始散布.参数r表示初始宿主的个数,设置r=3.程序输出初始宿主的nodeID.
RqstSet(K)	获取访问集合.根据泊松分布从数值区间[0, K]随机选出一个值k,根据Zipf分布或随机Zipf分布从区块链上随机选出k个区块.程序输出当前epoch内将要访问的区块集合R.
BlockRqst(R)	区块请求.对于属于集合R且未在本地保存的区块,向其初始宿主和动态宿主发送请求.
RqstR(n)	请求者复制.给定节点在本epoch请求过的区块,从中选出1/n比例的区块复制在本地.
OwnerR(p, p')	持有者复制.程序检查热度表,选择热度超过p的区块并划分级别.设置p=2, p'=3.
Deletion(T)	副本删除.参数T表示动态副本生命周期,设置T=3.
Encode(k, m)	区块编码.程序以k个冷区块为一组进行编码并生成m个校验块,设置k=4, m=2.

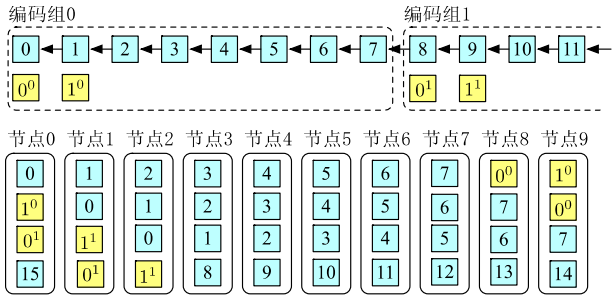


图7　BFT-Store区块散布示意图

6.2　测试结果

(1)存储测试　BFT-Store系统的存储开销是可以计算出来的. 对于$BFT^{(2)}$,每8个区块生成2×(8+2)=20个区块副本,因此复制度=20/8=2.5. 对于$BFT^{(3)}$,每8个

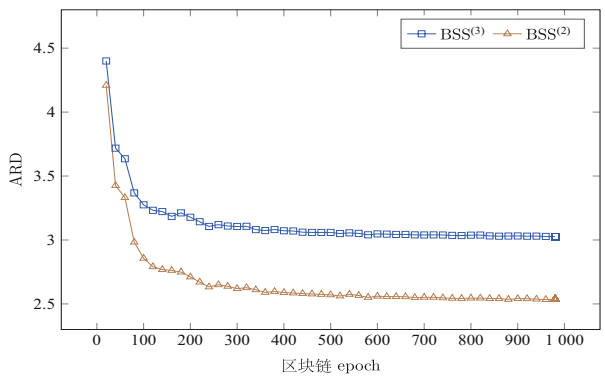
①设置如上初始状态是因为在使用式(1)为每个节点生成访问集合时区块链长度l必须大于0. 初始状态下,所有节点本地维护的区块列表中都包含10个条目,热度表中平均有3个条目. 由于尚未执行区块访问所以区块热度均为0.

②原文并没有这个假设,在这里对BFT-Store进行优化以使它与BSS方案"站在同一起跑线上". 根据原文,当系统内存在2个拜占庭节点时,节点必须获取到3个相同的副本才能够确认该数据正确,因此当使用参数为(k, m)的RS编码时系统能容忍m/2个节点发生拜占庭错误. 增加了存储区块头的假设之后,节点在区块请求阶段能够判断获取到的区块是否正确,因此同样使用参数为(k, m)的RS编码使得系统能够容忍m个拜占庭节点. 在存储开销不变的前提下,容错能力提高了.

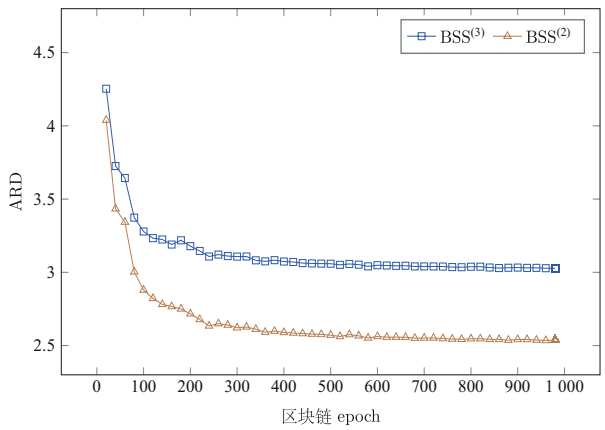
③增加区块副本并不能增强系统的容错能力,这是因为不同的副本被散布给了相同的10个节点.

区块生成30个区块副本，复制度=30/8=3.75. 对于BSS系统，通过实验测试得出各节点的平均存储开销. 在每个epoch内，执行完区块删除或是区块编码后，统计系统内各节点存储开销，再将所有节点总存储开销除以区块链总数据量得到平均复制度(Averaged Replication Degree，ARD). 测试了1 000个epoch内的ARD值，并每隔20个epoch取一个数据点，得到实验结果如图8所示. 可以看到，$BSS^{(3)}$的ARD值逐渐趋近于3.0而$BSS^{(2)}$的ARD值逐渐趋近于2.5. 将BFT-Store和BSS进行对比发现：(*a*)$BFT^{(3)}$和$BSS^{(3)}$都是三副本的设置，不过前者的复制度比后者高了约0.75；(*b*)$BFT^{(2)}$和$BSS^{(2)}$都是二副本的设置，两者的复制度几乎相同. 经计算，$BSS^{(2)}$系统测得的ARD平均值为2.9，即每个BSS节点平均存储的区块链数据比例为2.9/10=29%. 与全副本存储模式相比，每个BSS节点所需要存储的数据量降低了约71%.

(2)性能测试　BSS性能测试方法如下：在每个epoch内，执行完随机区块请求后，每个节点分别计算请求一个区块所需平均时间，再将10个节点的计算结果取平均值，得到平均区块请求时延(Averaged Request Latency，ARL). BFT-Store性能测试方法如下：在每个epoch内，若当前epoch序号为8的倍数则先执行区块散布再执行Zipf随机区块请求，否则直接执行随机区块请求；统计每个节点每次访问所花费的时间，并计算出ARL值. 测试400个epoch内的ARL值，并每隔5个epoch取一个数据点，得到实验结果如图9所示. 首先可以看到，$BSS^{(2)}$和$BFT^{(2)}$都采用了纠删码加二副本的混合散布策略，复制度也几乎一样，但前者的性能明显优于后者，这是因为BSS只对冷区块进行编码，热区块仍旧为副本态，而BFT-Store对所有区块都采取了相同的混合散布策略. 再者，虽然$BSS^{(3)}$系统的存储开销比$BFT^{(3)}$低，但前者的性能比后者更优. 这是因为BSS能够根据区块热度动态调整区块副本数量，从而大幅提升访问性能.

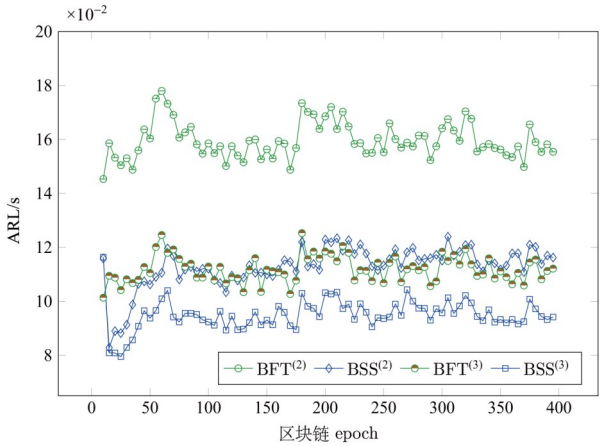


(*a*) λ=10

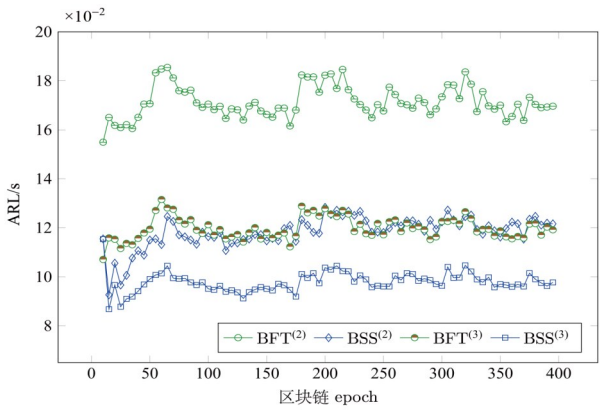


(*b*) λ=20

图8　BSS存储测试结果



(*a*) λ=10



(*b*) λ=20

图9　性能测试结果

(3)故障测试　模拟两类故障:crash故障指节点接收请求后不返回任何数据,byzantine故障指节点故意返回错误数据. 针对crash故障设置最大等待时长①,若请求节点等待到设定时间后未收到回复则认为被请求节点为crash节点. 针对byzantine故障,节点在接收到区块后利用本地存储的区块头执行数据验证,以确认接收到的数据是否正确. 针对$BSS^{(2)}$和$BFT^{(2)}$进行节点故障下的性能测试. 这两个系统都采用了编码加二副本的散布策略. 当系统中有1个节点发生故障时不需要执行数据恢复;当系统中有2个节点故障时,若被请求区块的2个副本的存储节点刚好均遭遇故障,则请求节点需要执行数据恢复. 设置两组实验,测试节点故障下的数据请求性能②. 第1组实验模拟1节点故障,并测试连续400个epoch下的ARL值(测试方法性能测试相同),实验结果如图10(a)所示. 对比无故障下的测试结果,$BSS^{(2)}$和$BFT^{(2)}$的ARL都有一定程度的增加. 第2组实验模拟系统中有2个节点发生故障的情况,测试连续400个epoch下的ARL值以及节点单次执行数据恢复的平均时间开销,实验结果如图10(b)和表2所示.

表2　数据恢复平均时长　　单位:s

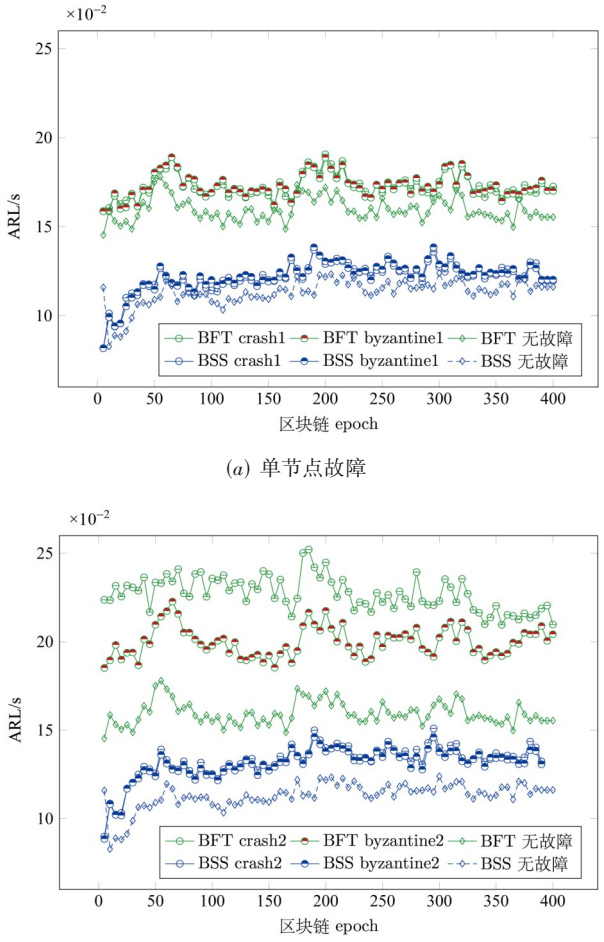
方案	crash2	byzantine2
BFT	2.760	1.103
BSS	2.849	1.269

从表2可以看出BSS单次数据恢复的时间开销稍大于BFT-Store,但是从图10(b)可以看到$BSS^{(2)}$测得的ARL上升幅度比$BFT^{(2)}$更小. 这是因为在$BSS^{(2)}$中少数区块处于编码态(只有2个副本),多数区块处于副本态(有不少于3个副本),而$BFT^{(2)}$中全部区块都只有2个副本. 在两节点故障的情况下,$BSS^{(2)}$系统执行数据恢复的次数更少,因而对数据请求性能的影响更小.

(4)自适应性　本实验用以评估动态复制过程是否及时将热区块的副本数增加,以使得区块副本分布的变化与区块访问概率变化吻合. 本实验针对$BSS^{(3)}$系统进行测试. 在第399个epoch执行结束后,统计各区块的副本个数,并将区块副本个数除以副本总数得到各区块的副本比率. 将副本比率曲线与两种访问分布曲线进行对比,得到测试结果如图11所示. 图中x轴代表区块序号,y轴代表区块的访问概率或者副本比率. 可以看到在Zipf和随机Zipf两种访问概率分布下,副本分布与访问分布都吻合良好. 这说明动态复制过程较为精准地复制了当前访问频次较高的区块.

6.3　CUB性能测试结果

本实验用以评估CUB静态散布策略在性能方面的表现,并与BSS系统进行对比. 实现了CUB中所提出的基于贪心算法的静态散布策略,并对其进行性能测试.



(a) 单节点故障

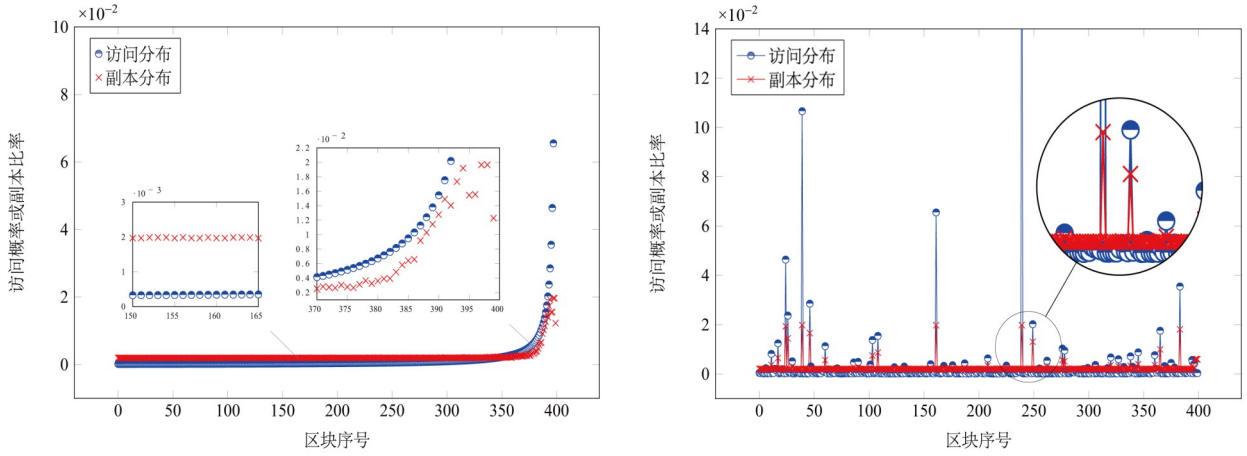
(b) 二节点故障

图10　节点故障下的性能测试结果

(1)实验设置　本部分实验中的区块链设置、节点设置与BSS实验相同. 所不同的是对区块访问分布的设置. CUB实验开始前需要为每个节点预先构建频繁访问集合(Frequently Access Set, FAS),即该节点在实验中将访问的区块及访问次数. 给定FAS包含的区块个数f,为每个节点从400个区块中随机选择f个不同的区块加入其FAS,并将FAS中每个区块的访问次数设为1. 参数f的作用类似于BSS实验中的参数λ,其决定了实验过程中的总体访问负载大小. 另外,将每个全节点的存储空间上限设置为区块链数据总量的五分之一(这是因为贪心算法要求输入节点存储空间上限).

①经过多轮测试得到节点间数据访问的最大时延,然后将最大等待时长设定为最大时延的1.5倍.

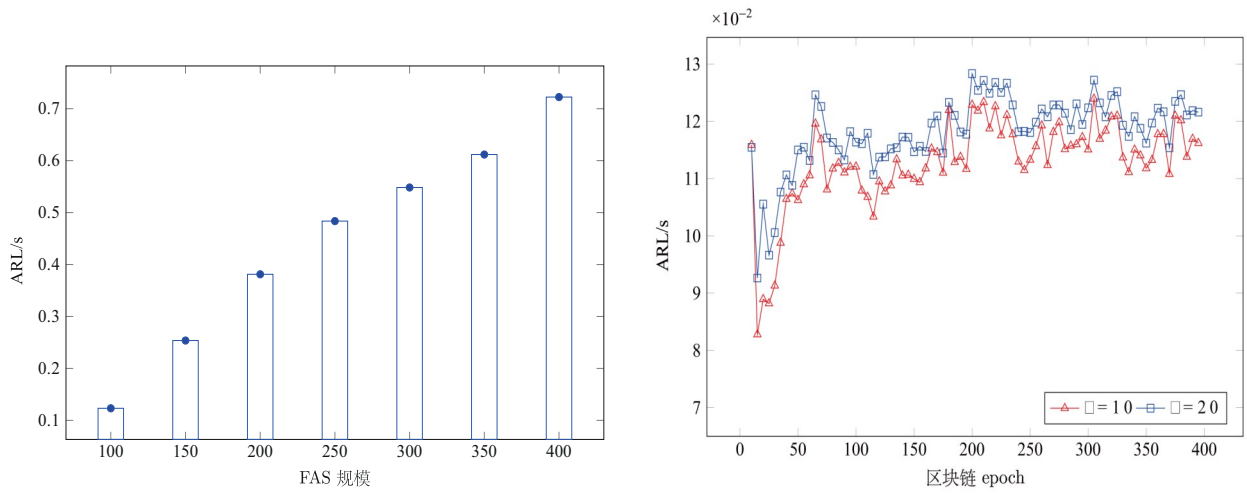
②在实验过程中每隔40个epoch切换1次故障节点.



(*a*) Zipf分布 (*b*) 随机Zipf分布

图11 自适应性测试结果

(2)测试结果 实验测试过程如下. 给定各节点FAS及存储空间上限,执行贪心算法完成区块散布. 然后,依照FAS执行区块访问,并统计每个节点每次访问所花费的时间. 计算各节点区块请求所花费的平均时延,再计算总体平均区块请求时延得到ARL. 图12展示了7种不同大小FAS下所测得的ARL结果. 将这一测试结果与BSS[(2)]的测试结果进行对比. 虽然CUB贪心算法得出的散布结果接近于最优解,但由于缺乏动态复制过程使得CUB测得的ARL伴随FAS呈线性增长. 作为对比,BSS中λ=20时的性能表现与λ=10时的性能表现相比变化不大. 当FAS大小设置为400时,CUB实验执行总访问次数为4 000,这与BSS系统(λ=20)中20个epoch所执行的总访问次数相近. 然而,CUB所测得ARL是BSS[(2)]测得ARL的5倍以上.



(*a*) CUB性能测试结果 (*b*) BSS性能测试结果

图12 CUB静态散布策略的性能测试结果对比BSS动态散布策略的性能测试结果

7 结束语

本文提出一种适用于联盟链系统的存储分片方案,使得在显著降低全节点存储开销的同时不损伤数据可用性和可靠性. 对比现有的全副本存储模式,BSS系统内全节点仅需存储约为r/n比例的区块链数据,同时全网范围内的区块链副本数量为rN(r指初始副本个数,n指BSS节点个数,N为存储组个数). 相比较于修剪类方法,BSS提供快速的数据访问服务,且能够保证即使$r-1$个节点发生故障时数据仍旧可用. 可考虑的未来工作是探索实现拜占庭容错的存储分片方案,即假设节点可能会崩溃、执行恶意行为等.

参考文献

[1] CONSENSYS. Blockchain use cases and applications by industry[EB/OL]. (2021)[2022]. https://consensys.net/blockchain-use-cases/.

[2] Hyperledger Foundation. Case studies: Browse various use cases powered by hyperledger technologies[EB/OL]. (2021)[2022]. https://www.hyperledger.org/learn/case-studies.

[3] Blockchain.com, Inc. Bitcoin blockchain size[EB/OL]. (2021)[2022]. https://www.blockchain.com/charts/blocks-size.

[4] SANK A I, CHEUNG R C C. A systematic review of blockchain scalability: Issues, solutions, analysis and future research[J]. Journal of Network and Computer Applications, 2021, 195: 1-25.

[5] NAKAMOTO S. Bitcoin: A Peer-to-peer Electronic Cash System[R/OL]. (2008)[2021]. https://bitcoin.org/en/bitcoin-paper.

[6] AGGELOS K, ANDREW M, DIONYSIS Z. Non-interactive proofs of proof-of-work[EB/OL]. (2018)[2021]. https://eprint.iacr.org/2017/963.pdf.

[7] BUNZ B, KIFFER L, LUU L, et al. FlyClient: Super-light clients for cryptocurrencies[C]//Proceedings of the 2020 IEEE Symposium on Security and Privacy. Piscataway: IEEE, 2020: 928-946.

[8] Bitcoin. org. Bitcoin core version 0.11.0[EB/OL]. (2021)[2021]. https://bitcoin. org/en/release/v0.11.0#block-file-pruning.

[9] Bitcoin. org. Bitcoin core version 0.14.0[EB/OL]. (2021)[2021]. https://bitcoin.org/en/release/v0.14.0#manual-pruning.

[10] MATZUTT R, KALDE B, PENNEKAMP J, et al. How to securely prune bitcoin's blockchain[C]//Proceedings of the 2020 IFIP Networking Conference. Piscataway: IEEE, 2020: 298-306.

[11] DAI X H, XIAO J, YANG W H, et al. Jidar: A jigsaw-like data reduction approach without trust assumptions for bitcoin system[C]//Proceedings of the 2019 IEEE 39th International Conference on Distributed Computing Systems. Piscataway: IEEE, 2019: 1317-1326.

[12] PALM E, SCHELEN O, BODIN U. Selective blockchain transaction pruning and state derivability[C]//Proceedings of the 2018 Crypto Valley Conference on Blockchain Technology. Piscataway: IEEE, 2018: 31-40.

[13] XU Z H, HAN S Y, CHEN L. CUB, A consensus unit-based storage scheme for blockchain system[C]//Proceedings of the 34th IEEE International Conference on Data Engineering. Piscataway: IEEE, 2018: 173-184.

[14] LI D, DAI J, JIANG R, et al. GAPG: A heuristic greedy algorithm for grouping storage scheme in blockchain[C]//Proceedings of the 2020 IEEE/CIC International Conference on Communications in China. Piscataway: IEEE, 2020: 91-95.

[15] CHEN J, GAI K, JIANG P, et al. Heuristic-based blockchain assignment: An empirical study[C]//Proceedings of the 2021 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking. Piscataway: IEEE, 2021: 916-923.

[16] ZHOU J, FENG G, SUN Y, et al. Intelligent block assignment for blockchain based wireless IoT systems[C]//Proceedings of the 2020 IEEE International Conference on Communications. Piscataway: IEEE, 2020: 1-6.

[17] QU B, WANG L, LIU P, et al. GCBlock: A grouping and coding based storage scheme for blockchain system[J]. IEEE Access, 2020, 8: 48325-48336.

[18] QI X D, ZHANG Z, JIN C Q, et al. BFT-Store: Storage partition for permissioned blockchain via erasure coding[C]//Proceedings of the 2020 IEEE 36th International Conference on Data Engineering. Piscataway: IEEE, 2020: 1926-1929.

[19] JIA D Y, XIN J C, WANG Z Q, et al. ElasticChain: Support very large blockchain by reducing data redundancy[C]//Proceedings of the 2nd International Joint Conference on Web and Big Data. Cham: Springer International Publishing, 2018: 440-454.

[20] JIA D Y, XIN J C, WANG Z Q, et al. SE-Chain: A scalable storage and efficient retrieval model for blockchain[J]. Journal of Computer Science and Technology, 2021, 36(3): 693-706.

[21] RAHMANI M, BENCHABA M. A comparative study of replication schemes for structured P2P networks[C]//Proceedings of the Ninth International Conference on Internet and Web Applications and Services. Piscataway: IEEE, 2014: 147-158.

[22] STOICA I, MORRIS R, KARGER D, et al. Chord: A scalable peer-to-peer lookup service for internet applications [C]//Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications. Piscataway: IEEE, 2001: 149-160.

[23] LV Q, CAO P, COHEN E, et al. Search and replication in unstructured peer-to-peer networks[C]//Proceedings of the 16th International Conference on Supercomputing. New York: ACM, 2002: 84-95.

[24] THAMPI S M. Survey of search and replication schemes in unstructured p2p networks[J]. Network Protocols & Algorithms, 2010, 2(1): 93-131.
[25] KANGASHARJU J. Internet content distribution[EB/OL]. (2021)[2021]. http://tubiblio.ulb.tu-darmstadt.de/4711/.
[26] KANGASHARJU J, ROSS K W, TURNER D A. Adaptive content management in structured P2P communities[C]// Proceedings of the 1st International Conference on Scalable Information Systems. New York: ACM, 2006: 24-33.
[27] ZHAO B Y, HUANG L, STRIBLING J, et al. Tapestry: A resilient global-scale overlay for service deployment[J]. IEEE Journal on Selected Areas in Communications, 2004, 22(1): 41-53.
[28] HUANG H P, ZHENG Y, CHEN H W, et al. PChord: A distributed hash table for P2P network[J]. Frontiers of Electrical and Electronic Engineering in China, 2010, 5: 49-58.
[29] MAYMOUNKOV P, MAZIERES D. Kademlia: A peer-to-peer information system based on the XOR metric[C]// Proceedings of the First International Workshop on Peer-to-Peer Systems. Heidelberg: Springer, 2002: 53-65.
[30] MACWILLIAMS F J, SLOANE N J A. The Theory of Error-Correcting Codes[M]. Amsterdam: Elsevier, 1977: 294-295.
[31] BIGQUERY. Bitcoin cryptocurrency[DB/OL]. (2021)[2021]. https://console.cloud.google.com/marketplace/pro duct/bitcoin/crypto-bitcoin?q=search&referer=search& project=hexo-calendar.
[32] PROTOCOL LABS. Filecoin: A Decentralized Storage Network[R/OL]. (2017)[2021]. https: //filecoin.io/filecoin.pdf.
[33] Storj Labs, Inc. Storj: A Decentralized Cloud Storage Network Framework[R/OL]. (2018)[2021]. https://storj.io/storjv3.pdf.

作者简介

阎 萌 女,1992年出生,天津人.南开大学博士生.主要研究方向为区块链、云存储数据保护等.
E-mail: yanm@nbjl.nankai.edu.cn

杨 颖 女,1998年出生,河南新乡人.南开大学硕士生.主要研究方向为数字货币技术、区块链存储优化等.
E-mail: yangy@nbjl.nankai.edu.cn

王 刚 男,1974年出生,北京人.博士,南开大学教授、博士生导师.主要研究方向为云存储系统、并行计算等.
E-mail: wgzwp@nbjl.nankai.edu.cn

刘晓光 男,1974年出生,河北安国人.博士,南开大学教授、博士生导师.主要研究方向为并行计算、存储系统、搜索引擎等.
E-mail: liuxg@nbjl.nankai.edu.cn