选择性压缩算法对区块链轻量级节点的优化研究

刘　云，陈路遥，宋　凯，朱鹏俊
(昆明理工大学信息工程与自动化学院，昆明 650500)

摘　要：区块链中的轻量级节点存在因不能存储完整区块链账本而无法参与区块链验证过程的局限，削弱了区块链分布式特性，可以通过压缩轻量级节点中存储的区块链账本以降低存储开销进而增强其区块链验证能力. 本文提出区块链选择性压缩(BSC) 算法，基于 Hyperledger Fabric 区块链架构，首先，针对区块链主链生成检查链，通过哈希指针链接检查链和主链并在检查链中存储用于验证主链区块完整性的哈希根路径；然后，对检查链中的检查点进行合并更新，降低检查链的存储开销；最后，选择性保留主链中的区块并计算验证主链完整性所需的哈希路径，降低主链的存储开销. 仿真结果表明，对比 EPBC 算法和 Snapshot 算法，BSC 算法在存储开销和验证能力方面有较好的表现.

关键词：区块链；检查链；哈希压缩；平衡二叉树
中图分类号：TP312　　**文献标识码**：A　　DOI：10.19907/j.0490-6756.2022.052001

Research on the optimization of lightweight nodes in blockchain by selective compression algorithm

LIU Yun, *CHEN Lu-Yao*, *SONG Kai*, *ZHU Peng-Jun*
(Faculty of Information Engineering and Automation, Kunming University of Science and Technology, Kunming 650500, China)

Abstract: Lightweight nodes in blockchain have the limitation that they cannot participate in the blockchain verification process because they cannot store a complete copy of the blockchain, which weakens the distributed characteristics of the blockchain. The verification ability can be enhanced by compressing the copy of the blockchain stored in the lightweight node to reduce storage overhead. The Blockchain Selective Compression (BSC) algorithm, which is based on the Hyperledger Fabric blockchain architecture, is proposed. In the proposed BSC algorithm, the check chain is first generated by the blockchain main chain, the two chains are linked through the hash pointers, and the hash root path is used to verify the integrity of the main chain blocks which stores in the check chain; secondly, the checkpoints in the check chain is merged and updated to reduce the storage overhead of the check chain; finally, the blocks in the main chain are selectively retained and the hash pathes are calculated for verifying the integrity of the main chain, which reduce the storage overhead of the main chain. The simulation results show that, comparing with the EPBC algorithm and the Snapshot algorithm, the BSC algorithm has a better performance in storage overhead and verification capabilities.

Keywords: Blockchain; Checkchain; Hash compression; Balanced binary tree

收稿日期：2021-09-01
基金项目：国家自然科学基金(61761025)；云南省重大科技专项计划(202002AD080002)
作者简介：刘云(1973—)，男，云南昆明人，副教授，主要从事数据挖掘、区块链等研究. E-mail：liuyun@kmust.edu.cn
通讯作者：陈路遥. E-mail：843858547@qq.com

1 引　言

区块链系统中存在完整和轻量级两类节点,两类节点都需要尽力存储区块链完整账本来保证区块链分布式特性[1-3].随着区块的累计[4],轻量级节点面临因为可存储空间不足而无法参与区块链验证的局限.已有研究可以通过压缩轻量级节点中存储的区块链账本,降低该节点的存储开销来保留其对区块链的验证能力[5,6].现有的压缩方法可以归纳为基于冗余的压缩算法[7-9]、基于模块的压缩算法[10]和基于哈希(hash)[11]的压缩算法[12],其中冗余压缩和模块压缩可以恢复原始区块链账本,但压缩开销大;哈希压缩开销小,并可以用压缩结果验证原始区块链账本.

Xu 等[12]提出了(Efficient Public Blockchain Compression,EPBC)压缩算法,该算法基于哈希压缩使用密码累加器压缩完整节点中存储的区块链以生成恒定大小的摘要,轻量级节点从完整节点接收并维护最新的摘要,同时接收证明来验证自身保留区块是否属于区块链.该方法能有效降低存储开销,但存在通信和验证方面的局限,首先使用EPBC 算法的节点必须定期与完整节点通信以接收和更新摘要;其次用于验证自身保留区块的证明会随着保留区块的增多累积;最终节点存储开销逐渐增大后导致可存储空间不足进而削弱节点的验证能力. Marsalek 等[13]提出了 Snapshot 压缩算法,该算法基于哈希压缩创建一条快照链来压缩现有主链,主链区块每累积一定数量后对应生成一个快照点,快照点中存储主链区块头用于验证主链完整性.该方法通过生成快照链来选择性保留主链区块降低存储开销,但随着主链区块的累积,快照点也出现累积,增大存储开销的同时削弱节点的验证能力.

本文提出区块链选择性压缩(Blockchain Selective Compression,BSC)算法,基于 Hyperledger Fabric[14]架构,首先提出轻量级节点容量优化模型并基于该模型提出检查点生成模型,针对区块链主链生成检查链;其次提出检查链更新子算法和主链哈希路径集合更新子算法,前者用于合并更新检查点,后者针对主链中的保留区块及更新后的检查点更新验证主链完整性所需的哈希路径集合(Block Merkle Paths,BMPs);最后将 BSC 算法与其比较算法在存储开销及验证能力方面进行对比仿真分析.仿真结果表明,对比 EPBC 算法和 Snapshot 算法,BSC 算法在存储开销和验证能力方面有较好的表现.

2 轻量级节点压缩模型

2.1 轻量级节点存储容量优化模型

针对区块链系统中轻量级节点面临的可存储空间不足,难以参与区块链验证过程的问题,本文提出区块链选择性压缩(BSC) 算法,使用该算法降低区块链系统中轻量级节点的区块存储开销进而保留其对完整区块链账本的验证能力.图 1 是 BSC 针对完整节点及轻量级节点的容量优化示例.

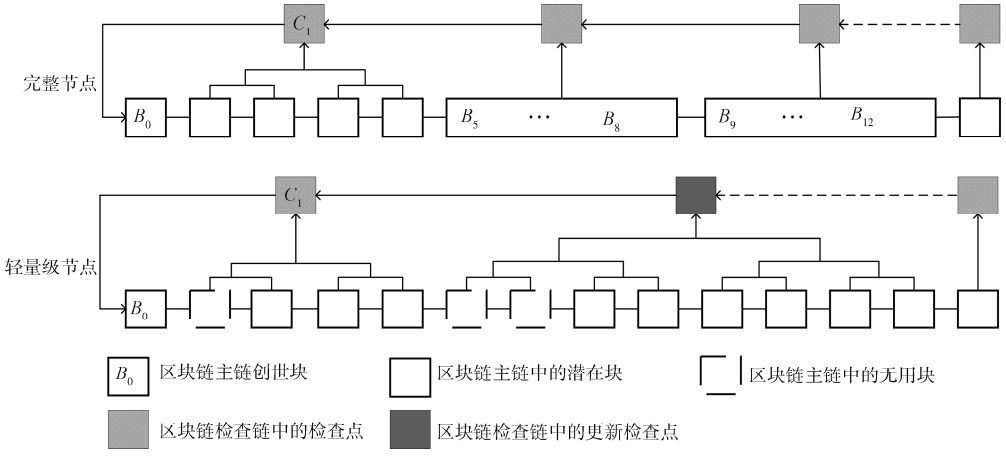

图 1　轻量级节点存储容量优化模型

Fig. 1　Lightweight node storage capacity optimization model

BSC 算法由检查点生成模型和 BSC 更新算法构成.首先通过主链压缩函数(Block Merkle Tree,BMT)生成主链区块验证根路径(Block Merkle Root,BMR),使用哈希指针(*hashes*)连接检查链

和主链；其次执行更新算法合并检查链中的相邻检查点；最后选择性保留主链中的区块并计算更新 BMPs.

在图 1 的示例中，首先定义主链 B（长度为 n）和检查链 C（长度为 m），其中两条链的创世块 $B_0 = C_0$. 示例中主链累积块（$n_c = 4$）时压缩生成一个检查点，检查链 C 的长度随着主链中累计区块的增多对应增长.

完整节点中存储主链和与轻量级节点保持同步的更新检查链，轻量级节点存储更新检查链，同时各节点独立的将主链中的区块分为潜在块和无用块，潜在块包含该节点存储的经常与之通信或交易的其他节点的数据，而无用块则相反. 每个节点选择性保留潜在块后擦除无用块并计算维护 BMPs，用于验证主链完整性.

图 1 中，轻量级节点在附加新的检查点后对之前的检查点进行合并更新；然后擦除主链中的无用块 $B_u, B_u \subset \{B_1, \ldots, B_n\}$；最终节点中仅保留潜在块 B_p.

2.2　检查点生成模型

我们规定区块链系统中各节点达成共识添加新区块且轻量级节点主链中累积的新区块数量达到 n_c 时，使用 BMT 对 n_c 个主链累积块进行压缩生成检查点（累计块 n_c 不包括等待共识确认的最新块和已被压缩为检查点的历史块）.

BMT 是 Merkle 树[15] 的扩展版本，用于压缩区块链中的多个块. 图 2 是包含完整性验证的 BMT 示例. 图 2 中，H_{ab} 是输入 H_a 和 H_b 后的哈希结果，$H_{ab} = H(H_a \text{ and } H_b)$，其中 $H(\cdot)$ 是用于压缩输入的哈希函数. 通过一系列哈希函数可以将多个区块压缩至单个 BMR，使用 BMR 验证具有 BMPs 路径集合的压缩区块.

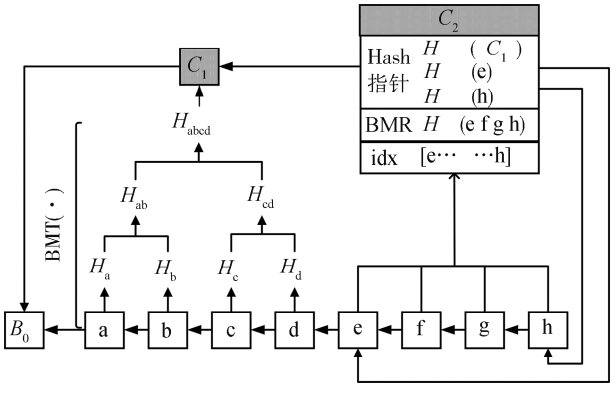

图 2　主链区块检查模型

Fig. 2　Main chain blocks check model

图 2 中，若仅保留区块 a 作为潜在块，可以使用 BMPs（即 H_b, H_{cd}）计算 BMR（即 H_{abcd}），如果计算出的 BMR 与检查点中存储的 BMR 相同，则可验证该 n_c 个区块的完整性同时验证潜在块 a 的内容完整性.

n_c 个主链区块对应生成一个检查点，该检查点存储在检查链中. 图 2 显示了检查点的存储内容. 每一个检查点中存储 hashes，BMR，idx 等三类数据，其中 hashes 链接前一个检查点和主链；BMR 验证主链完整性；idx 为 n_c 个压缩块的索引.

主链 B 末端 n_c 个压缩块对应的检查点 C_m 的索引 idx_m 的范围如下.

$$L_m \leqslant \text{idx}_m \leqslant R_m \tag{1}$$

式(1)中，$L_m = n - n_c + 1$ 和 $R_m = n$ 分别是该检查点对应的主链中最左侧和最右侧的块编号. C_m 的 BMR（root_m）如下.

$$\text{root}_m = \text{BMT}(B_{L_m}, \ldots, B_{R_m}) \tag{2}$$

式(2)中，BMT(·)是压缩函数，函数返回计算得到的 root_m. C_m 生成后被附加在检查点 C_{m-1} 的后端，新检查点 C_m 包含以下信息.

$$C_m = \{\text{hashes}_m, \text{root}_m, \text{idx}_m\} \tag{3}$$

式(3)中 $\text{hashes}_m = \{H(C_{m-1}), H(L_m), H(R_m)\}$ 是一个哈希集合. 哈希集合中存储前一个检查点的哈希值以及第 m 个 n_c 压缩块中最左侧和最右侧块的哈希值. 三个哈希值指明检查链对应的主链范围并将检查链链接到主链.

C_m 生成后，新的检查点将附加到长度为 m 的检查链的后端. 压缩函数用固定大小的检查点来验证 n_c 块的完整性. 随着压缩结果的累积会导致检查链的长度随着主链区块的增多逐渐增长，因此，检查链需要执行更新算法合并更新检查点.

3　BSC 更新算法

3.1　检查链更新子算法

BSC 更新算法包括检查链更新子算法和主链 BMPs 更新子算法. 前者用于对检查链中的检查点进行合并更新，防止检查点累积；后者用于计算主链中需要存储的 BMPs 信息及检查链更新后对应的 BMPs 更新信息.

为了快速验证主链完整性，需要保留 BMT 的平衡二叉树属性. 平衡二叉树要求左右子树的高度相同才能增加其高度. 因此，执行检查链更新子算法时需要两个相邻的检查点 C_{m-1} 和 C_m 的高度相同，通过 idx_{m-1} 和 idx_m 来获取相邻检查点的高度.

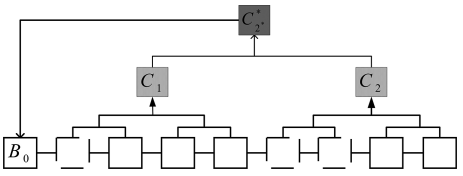

图 3　检查点更新模型

Fig. 3　Checkpoint update model

更新后,原有检查点将被合并为更新检查点.根据存储在每个检查点中的 BMR($root_{m-1}$ 和 $root_m$)计算出新的 BMR ($root_m^*$)并更新该新检查点中的 hashes 和 idx. 最后,新检查点将替换原有检查点. 检查链更新算法实现流程如算法 1 和算法 2 所示.

轻量级节点执行检查链更新算法将检查链中的检查点进行合并更新. 算法 1 遍历原有检查链,在算法 1 中调用算法 2 将所有满足平衡二叉树条件的相邻检查点进行合并,最终得到最简更新检查链.

算法 1 检查链更新算法

输入: 检查链 C

输出: 更新检查链 C^*

1) $i=2$;

2) $C_{temp}=\{C_1\}$; //首先在C_{temp}集合中存储检查点C_1^*

3) for($i=2;i\leqslant m;i++$) //使用 for 循环来遍历检查链 C 中的所有检查点

4) {

5) $C_{temp}=C_{temp}\cup\{C_i\}$; //循环每执行一次按检查链顺序引入下一个检查点

6) $C_{temp}=$output(C_{temp}^*) by Algorithm 2; //将算法 2 得到的更新检查集合C_{temp}^*赋值给C_{temp}

7) }

8) end

9) $C^*=C_{temp}$; //将循环结束后的C_{temp}集合作为更新检查链

10) return C^* //节点得到更新检查链

算法 2 检查点合并算法

输入: 检查点集合C_{temp}

输出: 更新检查点集合C_{temp}^*

1) while(1) //使用 while 循环更新检查链中相邻检查点

2) {

3) if $m\geqslant 2$ and $R_m-L_m==R_{m-1}-L_{m-1}$ then //判断C_{temp}集合中检查点个数是否大于 2 且集合中最右端的两个检查点的 idx 是否相等(即是否满足平衡二叉树高度要求)

4) $root^*=H(root_{m-1}$ and $root_m)$; //使用哈希运算得到更新检查点的$root^*$

5) $idx^*=[L_{m-1},R_m]$; //更新检查点的索引区间idx^*

6) $hashes^*=\{H(C_{temp_{m-2}}),H(B_{Lm-1}),H(B_{Rm})\}$; //改变更新检查点中三个 hash 指针所指的内容

7) $C_{m^*}^*=\{hashes^*,root^*,idx^*\}$; //得到更新检查点$C_{m^*}^*$

8) $C_{temp}=\{C_1,\ldots,C_{m-2},C_{m^*}^*\}$; //得到更新的$C_{temp}$集合

9) $m=m-1$; //每更新一次后C_{temp}集合的长度减 1

10) else //集合中检查点个数少于 2 或最右端两个检查点的 idx 不相等

11) $C_{temp}^*=C_{temp}$; //把完成检查点合并的C_{temp}集合称为C_{temp}^*

12) return C_{temp}^*//将C_{temp}^*返回给算法 1,等待算法 1 引入下一个检查点后继续使用算法 2 对检查点进行合并

13) end

14) }

15) end

3.2　主链 BMPs 更新子算法

各轻量级节点依据选择性保留的潜在块分别存储 BMPs 信息,并结合潜在块和 BMPs 计算 BMR 后与检查链中存储的 BMR 比对以验证主链 n_c 块的完整性. BMPs 的存储开销取决于潜在块的分布,当潜在块满足在主链所有 n_c 块中均匀分布时,BMPs 存储开销最大. 此时主链中潜在块数 r 表示如下式.

$$r=\sum_{i=1}^{m}r_i \tag{4}$$

式(4)中,$0\leqslant r_i\leqslant n_c$,$r_i$ 为检查点C_i中的潜在块数.

轻量级节点中检查链未更新时,bmp_{x,r_i} 表示检查点C_i对应的$n_c=2^x$个压缩块中r_i块的 BMPs 数量. 其值为每个 BMT 子树中从潜在块到子树根的

BMPs 数量和. 如果在C_i内$r_i=0$，则$bmp_{x,r_i}=0$，此时不需要 BMPs，bmp_{x,r_i}表示如下式.

$$bmp_{x,r_i}=\begin{cases}0, & \text{if} \quad r_i=0\\ r_i(x-k_i-1)+2^{k_i}, & \text{if} \quad r_i\leqslant\dfrac{n_c}{2}\\ n_c-r_i, & \text{otherwise}\end{cases} \tag{5}$$

式(5)中，$k_i=\lceil\log_2 r_i\rceil$.

轻量级节点执行 3.1 节中的检查链更新算法后，将所有具有相同高度 BMT 的相邻检查点合并更新，以减少检查点的数量，最终得到的更新检查点具有不同的 BMT 高度. 此外，由于更新算法在检查链中从链首向链尾更新，所以检查链中靠前的更新检查点的 BMT 具有更高的高度. 更新检查点的数量m^*按以下方式获得

$$m^*=\sum_{j=1}^{w}b_j \tag{6}$$

式(6)中，b_j代表二进制值，表示存在一个更新检查点；w表示由更新前检查链中的所有检查点数量之和m可以构造的 BMT 的最大高度. $w=\lfloor\log_2 m\rfloor+1$，$1\leqslant j\leqslant w$. 通过满足$\sum_{j=1}^{w}b_j\times 2^{j-1}=m$可以获得$b_j$集，其中$b_j\in\{0,1\}$. 最后，$bmp_{y_j,r_j^*}$是$n_c=2^{y_j}$个压缩块中$r_j^*$个块进行验证需要的 BMPs 数量，其中$y_j=x+j-1$，$r_j^*$是该更新检查点中的潜在块总数. 根据合并的检查点的范围，r_j^*可以表示为

$$r_j^*=\begin{cases}0, \text{ if } b_j=0\\ \sum_{i=L_j}^{R_j}r_i, \text{ otherwise}\end{cases} \tag{7}$$

式(7)中，$L_j=m-\sum_{l=1}^{j}b_l\times 2^{l-1}+1$和$R_j=m-\sum_{l=1}^{j}b_l\times 2^{l-1}+b_j\times 2^{j-1}$. $[L_j,R_j]$表示更新检查点中组合的原有检查点对应的主链区块范围. 在更新的检查点中$b_j=1$时，压缩块的数量为$n_c\times 2^{j-1}$，因此，r_j^*的范围为$0\leqslant r_j^*\leqslant n_c\times 2^{j-1}$. 根据下述公式获得基于式(5)的$bmp_{y_j,r_j^*}$.

$$bmp_{y_j,r_j^*}=\begin{cases}0, & \text{if } r_j^*=0\\ r_j^*(y_j-k_j^*-1)+2^{k_j^*}, & \text{if } r_j^*\leqslant\dfrac{2^{y_j}}{2}\\ 2^{y_j}-r_j^*, & \text{otherwise}\end{cases} \tag{8}$$

式(8)中，$k_j^*=\lceil\log_2 r_j^*\rceil$.

区块链系统中各轻量级节点达成共识添加新块且主链中新累计的区块达到n_c个时，对应生成第$m+1$个新检查点. 新检查点通过计算查询获取主链索引范围idx_{m+1}、BMT 压缩函数结果root_{m+1}及包含$H(C_m)$，$H(B_{Lm+1})$，$H(B_{Rm+1})$三个哈希指针的hashes_{m+1}指针集合，上述数据存储于C_{m+1}检查点中. 完成上述过程后，执行检查链更新算法合并更新检查点. 最后，各节点选择性保留主链中的潜在块计算 BMPs 后擦除无用块.

3.3　BSC 存储开销分析

轻量级节点使用 BSC 算法需要存储潜在块、检查点和 BMPs，存储开销V_{BSC}可以表示如下式.

$$V_{BSC}=m^*(4\times S_{hash}+2\times I)+\sum_{j=1}^{w}(r_j^*\times S+bmp_{y_j,r_j^*}\times S_{hash}) \tag{9}$$

式(9)中，$2\times I$表示一个索引 idx 的存储开销；S_{hash}表示一个哈希值的存储开销；S表示一个潜在块的存储开销.

对比算法 EPBC 的存储开销表示如下[10].

$$V_{EPBC}=|S_n|+r(S+|p^{(2)}|) \tag{10}$$

式(10)中，使用了参数S_n和$P^{(2)}$，参数定义如下.

$$S_n=g^{\prod_{i=1}^{n}H(blk_i,i)}\bmod N \tag{11}$$

$$P_i=\begin{cases}P_i^{(1)}=H(blk_i,i)\\ p_i^{(2)}=g^{\prod_{k=1}^{n}H(blk_k,k)/(blk_i,i)}\bmod N\end{cases} \tag{12}$$

式中，S_n表示完整主链的摘要；P_i是用于验证主链区块i的证明.

对比算法 Snapshot 的存储容量开销表示如下[11].

$$V_{Snapshot}=m(2\times S_{hash}+n_c\times S_{header})+r\times S \tag{13}$$

式(13)中，S_{header}表示一个主链区块头的存储开销.

4　仿真分析

4.1　仿真环境

本文使用 Python 针对 BSC、EPBC 和 Snapshot 等三种压缩算法进行数值仿真分析. 仿真分析包括两部分，首先计算针对同一轻量级节点使用三种压缩算法后的存储开销；其次计算节点允许存储空间有限时三种压缩算法的区块链验证能力.

在仿真分析中，我们为了模拟真实区块链系统的存储开销，集中定义部分参数如下所示：

(1) 每个S_{hash}的存储开销为 32 bytes；

(2) 每个S_{header}的存储开销为 80 bytes[13]；

(3) 每个潜在块的存储开销为 1 Mbytes；

(4) EPBC 算法中，N值存储开销为 128 bytes[10].

为了比较三种算法在存储开销及验证能力上的差异，分别定义部分参数如下所示：

(1) 存储开销分析中任一节点主链区块总数n

分别取 0.4 M 和 1 M;

(2) 验证能力分析中任一节点主链区块总数 n 分别取 25 K 和 1 M. 实验在配备 CPU AMD Ryzen 7 4800 H 和 16 GB RAM 的 PC 上运行.

4.2　存储开销仿真分析

我们比较使用 BSC、EPBC 和 Snapshot 等三种压缩算法后节点的存储开销,比较过程由 2 个子图构成,子图中主链区块压缩范围分别取 $n_c=2^4$ (图 4);$n_c=2^8$ (图 5),图中 x 轴代表一个轻量级节点中潜在块数量占该节点中主链区块总数的比值,规定潜在块数量与主链区块数量呈正比关系(即 $r\in[0.1\ n\sim0.9\ n]$);y 轴代表使用压缩算法后节点的存储开销. 依据 3.3 小节中三种算法的 V 值表达式计算任一轻量级节点的存储开销(三种算法针对同一节点计算开销,故计算中不考虑潜在块的存储开销),比较结果如图 4 和图 5 所示.

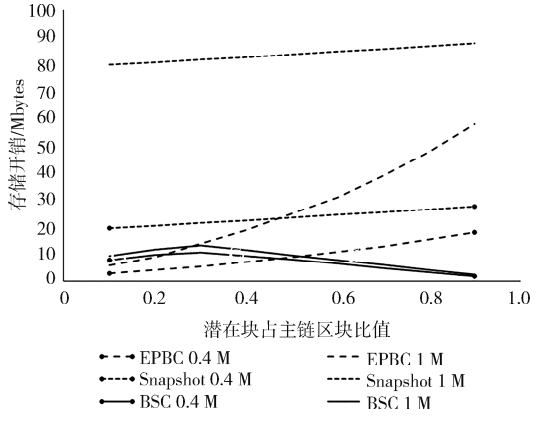

图 4　$n_c=2^4$ 时的存储开销对比

Fig. 4　Comparison of storage overhead when $n_c=2^4$

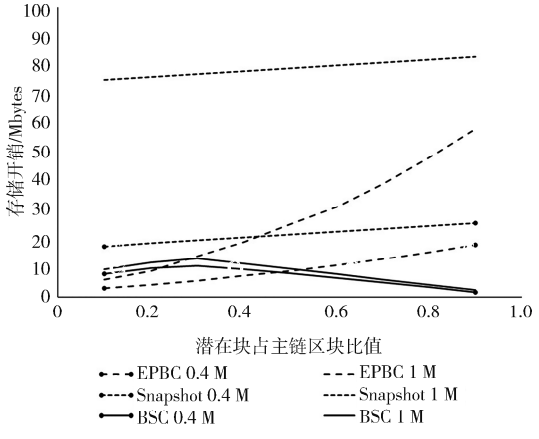

图 5　$n_c=2^8$ 时的存储开销对比

Fig. 5　Comparison of storage overhead when $n_c=2^8$

与对比压缩算法 EPBC 和 Snapshot 比较,当图 4 中 n=0.4 M 时,BSC 分别使存储开销平均减少了 27.02%和 70.12%; n=1 M 时,BSC 的平均存储开销分别减少了 69.20%和 89.85%. 图 5 n=0.4 M 时,BSC 分别使存储开销平均减少了 27.02%和 67.34%;n=1 M 的比较中,BSC 的平均存储开销分别减少了 69.20%和 89.21%.

使用 EPBC 算法压缩后的节点中存储恒定大小的区块链摘要及验证潜在块的证明,由于算法不使用检查链,其压缩开销不受 x 的影响,只随 r 的增加而增大,但因为 r 与 n 呈正比关系,因此该算法的存储开销最终如仿真结果所示随 n 和 r 的增多而增大. Snapshot 的快照链中存储主链区块头,其存储开销受 n、r 和 x 的影响,仿真结果显示其存储开销最终与 n 成线性比例增加.

BSC 算法的存储开销随着 n 的增多而微幅增大,且 n 和 r 都增多时,存储开销将减少. 首先,提出的算法构造了检查链,相似于 Snapshot 算法,但 BSC 任一检查点中只需要存储一个 BMR、一个 idx 和三个 hashes,存储开销远低于 Snapshot 快照点中存储的主链区块头;其次,借助 BMT 平衡二叉树形结构的二进制特性,存储开销会随着 r 占比 n 超过一半时而减少;最后,虽然检查点随时间推移逐渐增多,但 BSC 使用检查链更新算法合并更新检查点,有效降低了检查链存储开销,同时使用主链 BMPs 更新算法更新 BMPs 集合,有效降低了主链存储开销.

4.3　验证能力仿真分析

在节点可存储空间有限条件下比较使用三种压缩算法后节点对区块链的验证能力. 比较过程由 2 个子图构成(如图 6 和图 7),子图中 x 轴代表每个节点可用于存储区块的最大容量,y 轴代表使用压缩算法后所能保留的最大潜在块容量与该节点允许用于存储区块的最大容量的比值,将该比值定义为节点验证能力.

使用压缩算法后节点的存储开销主要集中于主链中的潜在块及为了验证主链完整性所需的额外存储开销(摘要、潜在块证明、快照链、BMPs、检查链). 节点为了验证主链完整性所需的额外存储开销越少,则节点可存储的潜在块越多(即节点可存储主链长度越长),节点验证能力越强.

图 6 和图 7 中,EPBC 算法的验证能力均为 1. 在图 6 中, BSC 和 Snapshot 在 x = 4 时进行对比,BSC 验证能力平均提高了 12.37%,在 x = 8 时进行对比,BSC 验证能力平均提高了 10.96%;在图 7 中,BSC 和 Snapshot 在 x=4 时进行对比,BSC 验证能力平均提高了 28.31%,在 x = 8 时进

行对比，BSC 验证能力平均提高了 26.03%.

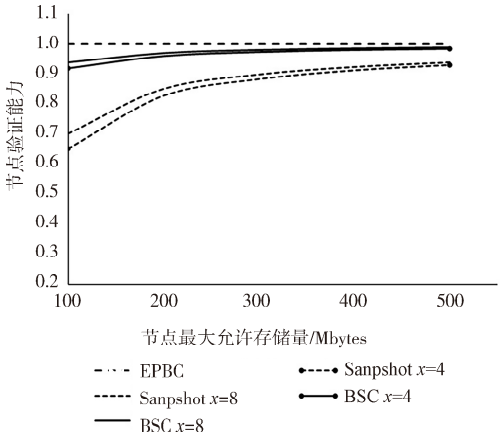

图 6　节点中主链区块总数 $n=25$ k

Fig. 6　The total number of node main chain blocks when $n=25$ k

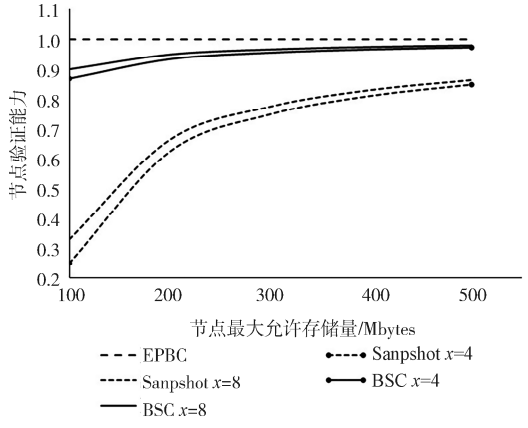

图 7　节点中主链区块总数 $n=1$ M

Fig. 7　The total number of node main chain blocks when $n=1$ M

EPBC 算法不使用检查链，且其用于验证区块链完整性的摘要和用于验证潜在块的证明均来自其它完整节点，轻量级节点存储空间可以全部用来存储潜在块. 但 EPBC 算法存在通信协议复杂和延迟验证的局限，即区块链系统每次添加新块后轻量级节点都需要依靠完整节点更新自身摘要和用于验证潜在块的证明. Snapshot 算法构造了检查链，随着 n 增多，检查链开销增大，且该算法不进行检查链更新，导致潜在块可存储空间减少，削弱了节点的区块链验证能力.

BSC 算法构建了检查链，相较于 EPBC 和 Snapshop 算法，BSC 使用检查链后无需依靠其它节点频繁更新自身压缩内容，且算法执行更新过程后可以大幅削减检查点数量，使得检查链开销远低于 Snapshot 算法快照链开销，最终实现更强的区块链验证能力.

综合 4.2 节和 4.3 节的分析结果，BSC 算法实现了有限存储空间下通过降低区块存储开销保留轻量级节点较强的区块链验证能力.

5　结　论

本文提出 BSC 算法针对轻量级节点存储的区块链账本进行压缩，首先，针对区块链主链压缩生成检查链；其次，合并更新检查链；最后选择性保留主链中的区块并计算验证主链所需的 BMPs. BSC 算法通过构造检查链实现了节点自主验证区块链账本，且其特有的更新过程使得检查链不随主链区块累积而增长，有效降低了节点的存储开销. 仿真结果表明，对比 EPBC 算法、Snapshot 算法，BSC 算法实现了较低的存储开销及较强的区块链验证能力.

参考文献：

[1]　张志威，王国仁，徐建良，等. 区块链的数据管理技术综述[J]. 软件学报，2020，31：283.

[2]　Bellini E，Iraqi Y，Damiani E. Blockchain-based distributed trust and reputation management systems：a survey [J]. IEEE Access，2020，8：21127.

[3]　曹傧，林亮，李云，等. 区块链研究综述[J]. 重庆邮电大学学报：自然科学版，2020，32：1.

[4]　BLOCKCHAIR. Blockchain size chart [EB/OL]. [2021-08-10]. https://blockchair.com/ethereum/charts/blockchain-size? compare=bitcoin.

[5]　赵羽龙，牛保宁，李鹏，等. 区块链增强型轻量级节点模型[J]. 计算机应用，2020，40：942.

[6]　Nadiya U，Mutijarsa K，Rizqi C Y. Block summarization and compression in bitcoin blockchain[C]// Proceedings of the 2018 International Symposium on Electronics and Smart Devices (ISESD). [S. l. ：s. n.]，2018.

[7]　Palai A，Vora M，Shah A. Empowering light nodes in blockchains with block summarization [C]// Proceedings of the 2018 9th IFIP International Conference on New Technologies，Mobility and Security (NTMS). [S. l. ：s. n.]，2018.

[8]　Chen X，Lin S，Yu N. Bitcoin blockchain compression algorithm for blank node synchronization [C]// Proceedings of the 2019 11th International Conference on Wireless Communications and Signal Processing (WCSP). [S. l.]：IEEE，2019.

[9]　王林景，高志宇，姚鹏帅. 基于时空相关性的传感器网络数据压缩算法[J]. 吉林大学学报：理学版，2020，58：337.

[10] Mei H, Gao Z, Guo Z, *et al*. Storage mechanism optimization in blockchain system based on residual number system [J]. IEEE Access, 2019, 7: 114539.

[11] 巫光福，曾宪文，刘娟，等. 基于纠错码的 Hash 函数的设计与分析[J]. 信息网络安全，2018(1)：67.

[12] Xu L, Chen L, Gao Z, *et al*. Efficient public blockchain client for lightweight users [EB/OL]. [2021-07-02]. https://arxiv.org/pdf/1811.04900.pdf.

[13] Marsalek A, Zefferer T, Fasllija E, *et al*. Tackling data inefficiency: compressing the bitcoin blockchain [C]// Proceedings of the 2019 18th IEEE International Conference on Trust, Security and Privacy in Computing And Communications/13th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE). [S. l.]: IEEE, 2019.

[14] Nathan S, Thakkar P, Vishwanathan B. Performance benchmarking and optimizing hyperledger fabric blockchain platform [C]// Proceedings of the 2018 IEEE 26th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS). [S. l.]: IEEE, 2018.

[15] 黄根，邹一波，徐云. 区块链中 Merkle 树性能研究[J]. 计算机系统应用，2020，29：7.

引用本文格式：

中　文：刘云，陈路遥，宋凯，等. 选择性压缩算法对区块链轻量级节点的优化研究[J]. 四川大学学报：自然科学版，2022，59：052001.

英　文：Liu Y, Chen L Y, Song K, *et al*. Research on the optimization of lightweight nodes in blockchain by selective compression algorithm [J]. J Sichuan Univ: Nat Sci Ed, 2022, 59: 052001.