(19)国家知识产权局

(12)发明专利

(10)授权公告号 CN 114153848 B
(45)授权公告日 2024.06.28

(21)申请号 202111443113.5
(22)申请日 2021.05.07
(65)同一申请的已公布的文献号
申请公布号 CN 114153848 A
(43)申请公布日 2022.03.08
(62)分案原申请数据
202110494903.X 2021.05.07
(73)专利权人 支付宝(杭州)信息技术有限公司
地址 310000 浙江省杭州市西湖区西溪路556号8层B段801-11
专利权人 蚂蚁区块链科技(上海)有限公司
(72)发明人 俞本权 卓海振 陆钟豪
(74)专利代理机构 北京博思佳知识产权代理有限公司 11415
专利代理师 李威
(51)Int.Cl.
G06F 16/22(2019.01)
G06F 16/23(2019.01)
G06F 16/27(2019.01)
G06F 21/62(2013.01)
G06Q 40/04(2012.01)
(56)对比文件
CN 112988908 B,2021.10.15
审查员 郭坚

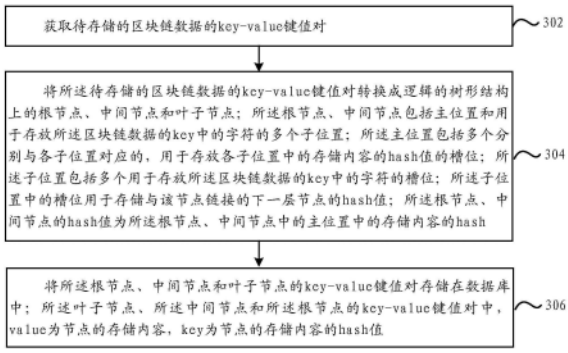
权利要求书3页 说明书30页 附图12页

(54)发明名称

区块链数据存储方法及装置、电子设备

(57)摘要

一种区块链数据存储方法及装置、电子设备,方法包括:获取待存储的区块链数据的key-value键值对;将待存储的区块链数据的key-value键值对转换成逻辑的树形结构上的根节点、中间节点和叶子节点;根节点、中间节点包括主位置和用于存放区块链数据的key中的字符的多个子位置;主位置包括多个分别与各子位置对应的,用于存放各子位置中的存储内容的hash值的槽位;子位置包括多个用于存放区块链数据的key中的字符的槽位;子位置中的槽位用于存储与该节点链接的下一层节点的hash值;将根节点、中间节点和叶子节点的key-value键值对存储在数据库中。



CN 114153848 B

1.一种区块链数据存储方法,所述方法包括:

获取待存储的区块链数据的key-value键值对;所述区块链数据的key对应的字符串包括字符前缀和字符后缀;

将所述待存储的区块链数据的key-value键值对转换成逻辑的树形结构上的根节点、中间节点和叶子节点;所述根节点、中间节点用于存储所述字符前缀中的字符;所述叶子节点用于存储所述字符后缀和所述区块链数据的Value;所述根节点、中间节点包括主位置和用于存放所述区块链数据的key中的字符的多个子位置;所述主位置包括多个分别与各子位置对应的,用于存放各子位置的hash值的槽位;所述子位置包括多个用于存放所述区块链数据的key中的字符的槽位;所述子位置中的槽位用于存储与该槽位所在的节点链接的下一层节点的hash值;所述子位置的hash值用所述子位置中的各个槽位所填充的hash值进行拼接后进行hash计算得到的hash值表示;所述根节点、中间节点的hash值为所述根节点、中间节点中的主位置中的存储内容的hash;

将所述根节点、中间节点和叶子节点的key-value键值对存储在数据库中。

2.根据权利要求1所述的方法,所述方法还包括:

确定所述树形结构上的叶子节点的存储容量是否满足节点分裂条件;

如果所述叶子节点的存储容量满足节点分裂条件,从所述叶子节点中分裂出至少一个所述中间节点;所述中间节点用于存储从所述叶子节点存放的字符后缀中分裂出的字符。

3.根据权利要求2所述的方法,所述根节点、所述中间节点存储的字符,为所述根节点、所述中间节点中的各子位置所代表的字符,与所述各子位置中填充了hash值的槽位所代表的字符,进行拼接生成的字符串。

4.根据权利要求3所述的方法,所述根节点、所述中间节点包括的子位置的数量,与所述子位置包括的槽位的数量相同。

5.根据权利要求4所述的方法,所述根节点、所述中间节点包括分别代表不同的16进制字符的16个子位置;所述子位置包括16个分别代表不同的16进制字符的槽位;所述主位置包括与各子位置对应的16个槽位。

6.根据权利要求1所述的方法,所述叶子节点为bucket数据桶;所述bucket数据桶包含若干条数据记录;所述数据记录中存储的数据内容包括所述字符后缀和所述区块链数据的Value。

7.根据权利要求2所述的方法,所述确定所述树形结构上的叶子节点的存储容量是否满足节点分裂条件,包括:

确定所述树形结构上的叶子节点是否发生数据更新;

如果所述树形结构上的任一叶子节点发生数据更新,在重新计算所述叶子节点的hash值,并基于该hash值与所述叶子节点上一层的节点进行重新链接之前,进一步确定所述叶子节点的存储容量是否满足节点分裂条件。

8.根据权利要求7所述的方法,所述方法还包括:

确定所述树形结构上的中间节点是否发生数据更新;

如果所述树形结构上的任一中间节点发生数据更新,在重新计算所述中间节点的hash值,并基于该hash值与该中间节点上一层的节点进行重新链接之前,确定所述中间节点的存储容量是否满足节点合并条件;

如果所述中间节点的存储容量满足节点合并条件,进一步将所述中间节点,合并至与该中间节点链接的下一层的叶子节点。

9.根据权利要求8所述的方法,所述节点分裂条件包括:

所述叶子节点中存储的数据记录的总数量大于阈值;和/或,所述叶子节点中存储的数据记录的总存储容量大于阈值;

相应的,所述节点合并条件包括:

所述中间节点中存储的字符的总数量小于或等于阈值;和/或,所述中间节点中存储的字符的总存储容量小于或等于阈值。

10.根据权利要求7所述的方法,从所述叶子节点分裂出至少一个中间节点,包括:

确定在当前的成块周期内写入所述叶子节点的最新数据记录;

将所述最新数据记录从所述叶子节点删除;以及,

从所述最新数据记录所包含的字符后缀中分裂出字符前缀,并创建用于存储分裂出的所述字符前缀的至少一中间节点,以及用于存储分裂出所述字符前缀之后的所述最新数据记录的至少一叶子节点。

11.根据权利要求7所述的方法,从所述叶子节点分裂出至少一个中间节点,包括:

确定所述叶子节点存储的若干条数据记录所包含的字符后缀之间的共享字符前缀;

将所述共享字符前缀从所述若干条数据记录中删除,并创建用于存储所述共享字符前缀的至少一中间节点。

12.根据权利要求8所述的方法,所述将所述中间节点,合并至与该中间节点链接的下一层的叶子节点,包括:

确定与所述中间节点链接的下一层的叶子节点;

将该中间节点存储的字符作为所述字符后缀写入所述叶子节点。

13.根据权利要求1所述的方法,所述区块链数据包括以下示出的任一:

区块中收录的交易;

与所述区块中收录的交易对应的交易收据;

与所述区块链中的区块链账户对应的最新账户状态数据;

智能合约账户的存储内容。

14.根据权利要求13所述的方法,所述逻辑的树形结构包括以下示出的任一:

用于存储区块中收录的交易的交易树;

用于存储与区块中收录的交易对应的交易收据的收据树;

用于存储与所述区块链中的区块链账户对应的最新账户状态数据的状态树;

用于存储智能合约账户的存储内容的存储树。

15.一种区块链数据存储装置,所述装置包括:

获取模块,获取待存储的区块链数据的key-value键值对;所述区块链数据的key对应的字符串包括字符前缀和字符后缀;

转换模块,将所述待存储的区块链数据的key-value键值对转换成逻辑的树形结构上的根节点、中间节点和叶子节点;所述根节点、中间节点用于存储所述字符前缀中的字符;所述叶子节点用于存储所述字符后缀和所述区块链数据的Value;所述根节点、中间节点包括主位置和用于存放所述区块链数据的key中的字符的多个子位置;所述主位置包括多个

分别与各子位置对应的,用于存放各子位置的hash值的槽位;所述子位置包括多个用于存放所述区块链数据的key中的字符的槽位;所述子位置中的槽位用于存储与该槽位所在的节点链接的下一层节点的hash值;所述子位置的hash值用所述子位置中的各个槽位所填充的hash值进行拼接后进行hash计算得到的hash值表示;所述根节点、中间节点的hash值为所述根节点、中间节点中的主位置中的存储内容的hash;

存储模块,将所述根节点、中间节点和叶子节点的key-value键值对存储在数据库中。

16.一种电子设备,包括:

处理器;

用于存储处理器可执行指令的存储器;

其中,所述处理器通过运行所述可执行指令以实现如权利要求1-14中任一项所述的方法。

17.一种计算机可读存储介质,其上存储有计算机指令,该指令被处理器执行时实现如权利要求1-14中任一项所述方法的步骤。

区块链数据存储方法及装置、电子设备

技术领域

[0001] 本说明书一个或多个实施例涉及区块链技术领域,尤其涉及一种区块链数据存储方法及装置、电子设备。

背景技术

[0002] 区块链技术,也被称之为分布式账本技术,是一种由若干台节点设备共同参与“记账”,共同存储和维护一份完整的分布式数据库的新兴技术。对于区块链的节点设备来说,需要存储和维护的区块链数据,通常包括区块数据、区块链中的区块链账户对应的账户状态数据;而区块数据又可以进一步包括区块头数据、区块中的区块交易数据、以及与区块中的区块交易数据对应的交易收据,等等。

[0003] 区块链的节点设备在存储以上示出的各种区块链数据时,通常可以将上述各种区块链数据以key-value键值对的形式,组织成Merkle树在数据库中存储。当需要查询节点设备存储的上述各种区块链数据时,可以通过将上述各种区块链数据的key作为查询索引,遍历上述Merkle树来高效的查询数据。

发明内容

[0004] 提出一种区块链数据存储方法,所述方法包括:

[0005] 获取待存储的区块链数据的key-value键值对;

[0006] 将所述待存储的区块链数据的key-value键值对转换成逻辑的树形结构上的根节点、中间节点和叶子节点;所述根节点、中间节点包括主位置和用于存放所述区块链数据的key中的字符的多个子位置;所述主位置包括多个分别与各子位置对应的,用于存放各子位置中的存储内容的hash值的槽位;所述子位置包括多个用于存放所述区块链数据的key中的字符的槽位;所述子位置中的槽位用于存储与该节点链接的下一层节点的hash值;所述根节点、中间节点的hash值为所述根节点、中间节点中的主位置中的存储内容的hash;

[0007] 将所述根节点、中间节点和叶子节点的key-value键值对存储在数据库中;所述叶子节点、所述中间节点和所述根节点的key-value键值对中,value为节点的存储内容,key为节点的存储内容的hash值。

[0008] 还提出一种区块链数据存储装置,所述装置包括:

[0009] 获取模块,获取待存储的区块链数据的key-value键值对;

[0010] 转换模块,将所述待存储的区块链数据的key-value键值对转换成逻辑的树形结构上的根节点、中间节点和叶子节点;所述根节点、中间节点包括主位置和用于存放所述区块链数据的key中的字符的多个子位置;所述主位置包括多个分别与各子位置对应的,用于存放各子位置中的存储内容的hash值的槽位;所述子位置包括多个用于存放所述区块链数据的key中的字符的槽位;所述子位置中的槽位用于存储与该节点链接的下一层节点的hash值;所述根节点、中间节点的hash值为所述根节点、中间节点中的主位置中的存储内容的hash;

[0011] 存储模块,将所述根节点、中间节点和叶子节点的key-value键值对存储在数据库中;所述叶子节点、所述中间节点和所述根节点的key-value键值对中,value为节点的存储内容,key为节点的存储内容的hash值。

[0012] 以上技术方案中,一方面,由于逻辑的树形结构中的每一个根节点和中间节点,均包括分别代表不同字符的多个位置;并且,各位置又进一步包括多个分别代表不同字符的槽位;因此,通过这种设计,使得每一个根节点和中间节点将具有更大的数据存储容量和数据承载能力,从而在将上述逻辑的树形结构中的根节点和中间节点写入数据库进行存储时;或者,在访问数据库中存储的逻辑的树形结构中的根节点和中间节点时,可以使得根节点和中间节点的数据存储容量,与承载上述数据库的存储介质的单次IO读写的容量更加适配,从而可以充分利用承载上述数据库的存储介质自身的IO读写能力,提升数据读写效率;而且,上述逻辑的树形结构上节点的数据存储容量和数据承载能力的提升,势必也会导致上述逻辑的树形结构整体的数据存储容量和数据承载能力的提升,使得上述逻辑的树形结构上可以存储更多的区块链数据;

[0013] 第二方面,由于逻辑的树形结构中的每一个根节点和中间节点均采用统一的数据结构;对于逻辑的树形结构中的根节点和中间节点来说,其实际存储的区块链数据的key的字符前缀的字符长度也将保持固定;因此,通过这种设计,可以避免由于根节点和中间节点其实际存储的字符长度不固定,而导致的节点的频繁分裂,从而可以确保逻辑的树形结构中所包含的根节点和中间节点的层数,始终处于一个相对稳定的状态;

[0014] 第三方面,由于上述根节点和中间节点的主位置包括多个分别与各子位置对应的,用于填充各子位置的hash的槽位;上述子位置又包括多个用于填充该字符节点链接的下一层节点的hash值的槽位;而且,上述根节点和中间节点的hash值可以用该根节点和中间节点中的主位置的hash表示;因此,当上述树形结构上的上述根节点或中间节点中的子位置中的任一槽位填充的hash值发生更新后,则在重新计算上述根节点或者中间节点的hash值时,只需要将该发生数据更新的子位置中各个槽位中填充的hash值进行拼接,并将拼接出的hash值作为计算参数重新进行hash计算,再将计算出的hash值填充至上述根节点或者中间节点中的主位置中与该子位置对应的槽位,然后再对主位置中的各槽位填充的hash进行拼接,再将拼接出的hash值作为计算参数重新进行hash计算即可得到上述根节点或者中间节点的hash;而对于未发生数据更新的其它子位置中各个槽位填充的hash值,将不再需要进行拼接后作为计算参数参与hash计算,从而可以降低在重新计算该根节点或者中间节点的hash值时的hash计算量和计算时长,提升hash计算的计算效率。

附图说明

[0015] 图1是一示例性实施例提供的一种将区块链的账户状态数据组织成MPT状态树的示意图;

[0016] 图2是一示例性实施例提供的一种将合约账户对应的存储空间中存储的合约数据组织成MPT storage树的示意图;

[0017] 图3是一示例性实施例提供的一种区块链数据存储方法的流程图;

[0018] 图4是一示例性实施例提供的一种FDMT树的树形结构图;

[0019] 图5是一示例性实施例提供的另一种FDMT树的树形结构图;

[0020] 图6是一示例性实施例提供的一种Treenode的结构图;

[0021] 图7是一示例性实施例提供的一种bucket数据桶的结构图;

[0022] 图8是一示例性实施例提供的一种为FDMT树上的节点设置节点ID的示意图;

[0023] 图9是一示例性实施例提供的一种对FDMT树上的Tree node进行递归压缩的示意图;

[0024] 图10是一示例性实施例提供的一种将账户状态数据写入Merkle状态存储树的示意图;

[0025] 图11是一示例性实施例提供的一种对bucket node进行节点分裂的示意图;

[0026] 图12是一示例性实施例提供的一种电子设备的结构示意图;

[0027] 图13是一示例性实施例提供的一种区块链数据存储装置的框图。

具体实施方式

[0028] 这里将详细地对示例性实施例进行说明,其示例表示在附图中。下面的描述涉及附图时,除非另有表示,不同附图中的相同数字表示相同或相似的要素。以下示例性实施例中所描述的实施方式并不代表与本说明书一个或多个实施例相一致的所有实施方式。相反,它们仅是与如所附权利要求书中所详述的、本说明书一个或多个实施例的一些方面相一致的装置和方法的例子。

[0029] 需要说明的是:在其他实施例中并不一定按照本说明书示出和描述的顺序来执行相应方法的步骤。在一些其他实施例中,其方法所包括的步骤可以比本说明书所描述的更多或更少。此外,本说明书中所描述的单个步骤,在其他实施例中可能被分解为多个步骤进行描述;而本说明书中所描述的多个步骤,在其他实施例中也可能被合并为单个步骤进行描述。

[0030] 区块链一般被划分为三种类型:公有链(Public Blockchain),私有链(Private Blockchain)和联盟链(Consortium Blockchain)。此外,还可以有上述多种类型的结合,比如私有链+联盟链、联盟链+公有链等。

[0031] 其中,去中心化程度最高的是公有链。加入公有链的参与者(也可称为区块链中的节点)可以读取链上的数据记录、参与交易、以及竞争新区块的记账权等。而且,各节点可自由加入或者退出网络,并进行相关操作。

[0032] 私有链则相反,该网络的写入权限由某个组织或者机构控制,数据读取权限受组织规定。简单来说,私有链可以为一个弱中心化系统,其对节点具有严格限制且节点数量较少。这种类型的区块链更适合于特定机构内部使用。

[0033] 联盟链则是介于公有链以及私有链之间的区块链,可实现“部分去中心化”。联盟链中各个节点通常有与之相对应的实体机构或者组织;节点通过授权加入网络并组成利益相关联盟,共同维护区块链运行。

[0034] 基于区块链的基本特性,区块链通常是由若干个区块构成。在这些区块中分别记录有与该区块的创建时刻对应的时间戳,所有的区块严格按照区块中记录的时间戳,构成一条在时间上有序的数据链条。

[0035] 对于链外产生的数据,可以将其构建成区块链所支持的标准的交易(transaction)格式,然后发布至区块链,由区块链中的节点设备对该交易进行共识,并在

达成共识后,由区块链中作为记账节点的节点设备,将这笔交易打包进区块,在区块链中进行持久化存证。

[0036] 在区块链领域,有一个重要的概念就是账户(Account);在实际应用中,通常可以将账户划分为外部账户和合约账户两类;外部账户就是由用户直接控制的账户,也称之为用户账户;而合约账户则是由用户通过外部账户创建的,包含合约代码的账户(即智能合约)。

[0037] 对于区块链中的账户而言,通常会通过一个结构体,来维护账户的账户状态。当区块中的交易被执行后,区块链中与该交易相关的账户的状态通常也会发生变化。

[0038] 在一个例子中,账户的结构体通常包括Balance,Nonce,Code和Storage等字段。其中:

[0039] Balance字段,用于维护账户目前的账户余额;

[0040] Nonce字段,用于维护该账户的交易次数;它是用于保障每笔交易能且只能被处理一次的计数器,有效避免重放攻击;

[0041] Code字段,用于维护该账户的合约代码;在实际应用中,Code字段中通常仅维护合约代码的hash值;因而,Code字段通常也称之为Codehash字段。

[0042] Storage字段,用于维护该账户的存储内容;对于合约账户而言,通常会分配一个独立的持久化的存储空间,用以存储该合约账户对应的存储空间中存储的合约数据;该独立的存储空间通常称之为该合约账户的账户存储。合约账户的存储内容通常会以key-value键值对的形式被构建成MPT(Merkle Patricia Trie)树的数据结构存储在上述独立的存储空间之中。MPT树是区块链领域用于存储和维护区块链数据的一种逻辑的树形结构,在这种树形结构中通常包括根节点、中间节点、叶子节点。

[0043] 其中,基于合约账户的存储内容构建成的MPT树,通常也称之为Storage树。而Storage字段通常仅维护该Storage树的根节点的hash值;因此,Storage字段通常也称之为Storage Root hash字段。其中,对于外部账户而言,以上示出的Code字段和Storage字段的字段值均为空值。

[0044] 对于大多数区块链模型,通常都会采用Merkle树;或者,基于Merkle树的数据结构的Merkle树变种等逻辑的树形结构,来存储和维护数据。例如,MPT树,就是一种用来存储和维护区块链数据的,融合了Trie字典树的树形结构的Merkle树变种。

[0045] 以下以使用MPT树,来存储区块链数据为例进行说明;

[0046] 在一个例子中,区块链中需要存储和维护的区块链数据,通常包括账户状态(state)数据、交易数据和收据数据;因此,在实际应用中,可以分别将上述账户状态数据、交易数据和收据数据以key-value键值对的形式,组织成MPT状态树(即world state)、MPT交易树和MPT收据树等三棵MPT树,分别进行存储和维护。

[0047] 其中,除了以上三棵MPT树以外,合约账户对应的存储空间中存储的合约数据,通常也会被构建成为一棵MPT Storage树(以下简称为Storage树)。该Storage树的根节点的hash值,会被添加到与该Storage树对应的合约账户的上述结构体中的Storage字段。

[0048] MPT状态树,是由区块链中所有账户(包括外部账户和合约账户)的账户状态(state)数据,以key-value键值对的形式组织成的MPT树;MPT交易树,是由区块链中的交易(transaction)数据,以key-value键值对的形式组织成的MPT树;MPT收据树,是区块中的交

易在执行完毕后生成的与每笔交易对应的交易(receipt)收据,以key-value键值对的形式组织成的MPT树。

[0049] 以上示出的MPT状态树、MPT交易树和MPT收据树的根节点的hash值,最终都会被添加至对应区块的区块头中。

[0050] 其中,MPT交易树和MPT收据树均与区块相对应,即每一个区块都有自己的MPT交易树和MPT收据树。而MPT状态树是一个全局的MPT树,并不与某一个特定的区块相对应,而是涵盖了区块链中所有账户的账户状态数据。区块链每产生一个最新区块,则在该最新区块中的交易被执行之后,区块链中这些被执行交易的相关账户(可以是外部账户也可以是合约账户)的账户状态,通常也会随之发生变化。

[0051] 例如,当区块中的一笔“转账交易”执行完毕后,与该“转账交易”相关的转出方账户和转入方账户的余额(即这些账户的Balance字段的字段值),通常也会随之发生变化。节点设备在区块链产生的最新区块中的交易执行完毕后,由于当前区块链中的账户状态发生了变化,因此节点设备需要根据区块链中所有账户当前的账户状态数据,来构建MPT状态树,用于维护区块链中所有账户的最新状态。

[0052] 也即,每当区块链中产生一个最新区块,并且该最新区块中的交易执行完毕后,导致区块链中的部分账户的账户状态发生了变化,节点设备需要基于区块链中所有账户最新的账户状态数据,重新构建一棵MPT状态树。换句话说,区块链中每一个区块,都有一个与之对应的MPT状态树;该MPT状态树,维护了在该区块中的交易在执行完毕后,区块链中所有账户最新的账户状态。

[0053] 请参见图1,图1为本说明书示出的一种将区块链中的各个区块链账户的账户状态数据以key-value键值对的形式组织成MPT状态树的示意图。

[0054] MPT树,是一种较为传统的经过改良的Merkle树变种,其融合了Merkle树和Trie字典树(也称之为前缀树)两种树形结构的优点。

[0055] 在MPT树中通常包括三种节点,分别为叶子节点(leaf node),扩展节点(extension node)和分支节点(branch node)。其中,MPT树的根节点通常可以是扩展节点;MPT树的中间节点通常可以是分支节点或者其它的扩展节点。

[0056] 其中,扩展节点和分支节点可以统称为字符节点,用于存储账户状态数据的key(即账户地址)对应的字符串的字符前缀部分;其中,对于MPT树而言,上述字符前缀部分通常是指共享字符前缀;所述共享字符前缀,是指所有账户状态数据的key(即区块链账户地址)所具有的相同的一个或者多个字符组成的前缀。而上述叶子节点,用于存储区块链数据的key对应的字符串的字符后缀部分和Value(即具体的账户状态数据)。

[0057] 扩展节点,用于存储账户地址的共享字符前缀中一个或者多个字符(即图1示出的shared nibble),和该扩展节点链接的下一层的节点的hash值(即图1示出的Next node)。

[0058] 分支节点,包含17个槽位,前16个槽位对应着key中的16个可能的十六进制字符,一个字符对应一个nibble(半字节),前16个槽位中的每一个槽位,分别表示一个账户地址的共享字符前缀中的一个字符,这些槽位用于填充该分支节点链接的下一层的节点的hash值。最后一个槽位为value槽位,一般为空值。

[0059] 叶子节点,用于存储账户地址的字符后缀(即图1示出的key-end),和账户状态数据的value(即以上描述的账户的结构体);其中,账户地址的字符后缀和账户地址的共享字

符前缀共同组成了一个完整的账户地址;所述字符后缀,是指除了账户地址的共享字符前缀以外的最后一个或者多个字符组成的后缀。

[0060] 假设需要组织成MPT状态树的账户状态数据如下表1所示:

[0061]

账户地址（Key）							账户状态（Value）			
							balance	nonce	codehash	storage root
a	7	1	1	3	5	5	45.0ETH	n1		
a	7	7	d	3	3	7	1.00WEI	n2		
a	7	f	9	3	6	5	1.1ETH	n3		
a	7	7	d	3	9	7	0.12ETH	n4	c1	s1

[0062] 表1

[0063] 其中,需要说明的是,在表1中,前三行的账户地址对应的区块链账户为外部账户,Codehash和Storage root字段为空值。第4行的账户地址对应的区块链账户为合约账户,Codehash字段维护了该合约账户对应的合约代码的hash值;Storage root字段维护了该合约账户的存储内容构成的Storage树的根节点的hash值。

[0064] 最终按照表1中的账户状态数据组织成的MPT状态树,如图1所示;该MPT状态树是由4个叶子节点,2个分支节点,和2个扩展节点(其中一个扩展节点作为根节点)构成。

[0065] 在图1中,prefix字段为扩展节点和叶子节点共同具有的前缀字段。该prefix字段的不同字段值可以用于表示不同的节点类型。

[0066] 例如,prefix字段的取值为0,表示包含偶数个nibbles的扩展节点;如前所述,nibble表示半字节,由4位二进制组成,一个nibble可以对应一个组成账户地址的字符。prefix字段的取值为1,表示包含奇数个nibble(s)的扩展节点;prefix字段的取值为2,表示包含偶数个nibbles的叶子节点;prefix字段的取值为3,表示包含奇数个nibble(s)的叶子节点。

[0067] 而分支节点,由于其是并列单nibble的字符节点,因此分支节点不具有上述prefix字段。

[0068] 扩展节点中的Shared nibble字段,对应该扩展节点所包含的键值对的key值,表示账户地址之间的共同字符前缀;比如,上表中的所有账户地址均具有共同的字符前缀a7。Next Node字段中填充下一个节点的hash值(hash指针)。

[0069] 分支节点中的16进制字符0~f字段,对应该分支节点所包含的键值对的key值;如果该分支节点为账户地址在MPT树上的搜索路径上的中间节点,则该分支节点的Value字段可以为空值。0~f字段中用于填充下一层节点的hash值。

[0070] 叶子节点中的Key-end,对应该叶子节点所包含的键值对的key值,表示账户地址的最后几个字符(账户地址的字符后缀)。从根节点搜索到叶子节点的搜索路径上的各个节点的key值,构成了一个完整的账户地址。该叶子节点的Value字段填充账户地址对应的账户状态数据;例如,可以对上述Balance,Nonce,Code和storage等字段构成的结构体进行编码后,填充至叶子节点的Value字段。

[0071] 进一步的,如图1所示的MPT状态树上的node,最终也是以Key-Value键值对的形式

存储在数据库中;

[0072] 其中,当MPT状态树上的node在数据库中进行存储时,MPT状态树上的node的键值对中的key,可以为node所包含的数据内容的hash值;MPT状态树上的node的键值对中的Value,为node所包含的数据内容。

[0073] 在将MPT状态树上的node存储至数据库时,可以计算该node所包含的数据内容的hash值(即对node整体进行hash计算),并将计算出的hash值作为key,将该node所包含的数据内容作为value,生成Key-Value键值对;然后,将生成的Key-Value键值对存储至数据库中。

[0074] 由于MPT状态树上的node,以Key-value键值对的形式进行存储;其中,Key可以是node所包含的数据内容的hash值,Value可以是node所包含的数据内容;因此,在需要查询MPT状态树上的node时,通常可以基于node所包含的数据内容的hash值作为key来进行内容寻址。

[0075] 请参见图2,图2为本说明书示出的一种将合约账户对应的存储空间中存储的合约数据组织成MPT storage树的示意图。

[0076] 请继续参见表1,表1中示出的账户地址为“a77d397”的账户为合约账户,因此该合约账户对应的存储空间中存储的合约数据会被组织成一颗storage树;其中,该storage树的根节点的hash值S1,会被添加到图1示出的MPT状态树中与该合约账户对应的叶子节点中的storage root字段里。

[0077] 假设该合约账户的存储空间中存储的合约数据如下表2所示:

[0078]

Key	Value

[0079]

3	3	5	b	2	e	4	张三的资产A余额(状态变量): 张三_A=20
7	c	2	5	9	8	8	李四的资产B余额(状态变量): 李四_B=50
f	a	9	9	3	6	5	王五的资产A余额(状态变量): 王五_A=35
f	a	6	b	e	3	3	staff(状态变量): staff=new

[0080] 表2

[0081] 需要说明的是,合约账户的存储空间中存储的合约数据,通常可以是状态变量的形式;在进行存储时,可以将状态变量名以key-value键值对的形式组织成如图2所示的storage树进行存储。

[0082] 例如,在一个例子中,可以将合约账户的账户地址和状态变量在合约账户的账户存储中的存储位置的hash值作为key,将状态变量对应的变量取值作为value。

[0083] 其中,图2示出的storage树的基本结构与图1示出的MPT状态树相似,在本说明书中不再赘述。通过上述图1和图2的描述可知,基于MPT树的树形结构设计,分支节点可以存储所有账户地址的共享字符前缀中的其中一个字符;而扩展节点则可以存储所有账户地址的共享字符前缀中的一个或者多个字符。

[0084] 在实际应用中,MPT树上存储的所有数据的key的共享字符前缀的字符长度,通常

并不固定;而且,当MPT树上写入了新增的数据之后,可能导致上述共享字符前缀的字符长度,也会随之发生变化;因此,这就可能会导致MPT树上的扩展节点发生分裂,分裂出新的分支节点;也即,MPT树上的节点的分裂条件是,上述共享字符前缀的字符长度发生变化;

[0085] 例如,以图1示出的MPT状态树为例,假设上述MPT状态树上新增了一个账户地址的前两位字符为“a8”的账户地址的账户状态数据,那么图1中示出的作为根节点的扩展节点的“Shared nibble”字段存储的共享字符前缀,就会由“a7”,改变为“a”;按照MPT状态树的节点的分裂条件,这个作为根节点的扩展节点,会分裂成一个存储的共享字符前缀为“a”的扩展节点;和一个字符“8”被占用的分支节点。

[0086] 一旦MPT树上的扩展节点发生分裂,可能会导致MPT树的节点层数也会发生变化,造成MPT树的节点层数不够稳定。由于上述MPT树上存储的所有数据的key的共享字符前缀的字符长度,会随着上述MPT树写入了新的数据,而频繁的发生变化;因此,按照以上示出的节点分裂方式,会造成节点的频繁分裂,进而会影响向MPT树写入新的数据时的数据存储效率。

[0087] 有鉴于此,本说明书提出一种新的逻辑的树形结构设计方案。

[0088] 在实现时,上述逻辑的树形结构仍然可以包括根节点、中间节点和叶子节点;其中,上述根节点、中间节点用于存储区块链数据的key中的字符;上述叶子节点用于存储区块链数据的value。

[0089] 与上述MPT树不同的是,上述根节点、中间节点可以包括主位置和用于存放所述区块链数据的key中的字符的多个子位置;所述主位置包括多个分别与各子位置对应的,用于存放各子位置中的存储内容的hash值的槽位;所述子位置包括多个用于存放所述区块链数据的key中的字符的槽位;所述子位置中的槽位用于存储与该节点链接的下一层节点的hash值;所述根节点、中间节点的hash值为所述根节点、中间节点中的主位置中的存储内容的hash。

[0090] 区块链中的节点设备在存储区块链数据时,可以获取待存储的区块链数据的key-value键值对,再将该待存储的区块链数据转换成上述逻辑的树形结构上的根节点、中间节点和叶子节点;然后,将上述根节点、中间节点和叶子节点的key-value键值对再进一步存储在数据库中;上述叶子节点、所述中间节点和所述根节点的key-value键值对中,value可以为节点的存储内容,key可以为节点的存储内容的hash值。

[0091] 一方面,由于逻辑的树形结构中的每一个根节点和中间节点,均包括分别代表不同字符的多个位置;并且,各位置又进一步包括多个分别代表不同字符的槽位;因此,通过这种设计,使得每一个根节点和中间节点将具有更大的数据存储容量和数据承载能力,从而在将上述逻辑的树形结构中的根节点和中间节点写入数据库进行存储时;或者,在访问数据库中存储的逻辑的树形结构中的根节点和中间节点时,可以使得根节点和中间节点的数据存储容量,与承载上述数据库的存储介质的单次IO读写的容量更加适配,从而可以充分利用承载上述数据库的存储介质自身的IO读写能力,提升数据读写效率;而且,上述逻辑的树形结构上节点的数据存储容量和数据承载能力的提升,势必也会导致上述逻辑的树形结构整体的数据存储容量和数据承载能力的提升,使得上述逻辑的树形结构上可以存储更多的区块链数据;

[0092] 第二方面,由于逻辑的树形结构中的每一个根节点和中间节点均采用统一的数据

结构;对于逻辑的树形结构中的根节点和中间节点来说,其实际存储的区块链数据的key的字符前缀的字符长度也将保持固定;因此,通过这种设计,可以避免由于根节点和中间节点其实际存储的字符长度不固定,而导致的节点的频繁分裂,从而可以确保逻辑的树形结构中所包含的根节点和中间节点的层数,始终处于一个相对稳定的状态;

[0093] 第三方面,由于上述根节点和中间节点的主位置包括多个分别与各子位置对应的,用于填充各子位置的hash的槽位;上述子位置又包括多个用于填充该字符节点链接的下一层节点的hash值的槽位;而且,上述根节点和中间节点的hash值可以用该根节点和中间节点中的主位置的hash表示;因此,当上述树形结构上的上述根节点或中间节点中的子位置中的任一槽位填充的hash值发生更新后,则在重新计算上述根节点或者中间节点的hash值时,只需要将该发生数据更新的子位置中各个槽位中填充的hash值进行拼接,并将拼接出的hash值作为计算参数重新进行hash计算,再将计算出的hash值填充至上述根节点或者中间节点中的主位置中与该子位置对应的槽位,然后再对主位置中的各槽位填充的hash进行拼接,再将拼接出的hash值作为计算参数重新进行hash计算即可得到上述根节点或者中间节点的hash;而对于未发生数据更新的其它子位置中各个槽位填充的hash值,将不再需要进行拼接后作为计算参数参与hash计算,从而可以降低在重新计算该根节点或者中间节点的hash值时的hash计算量和计算时长,提升hash计算的计算效率。

[0094] 请参见图3,图3是一示例性实施例提供的一种区块链数据存储方法的流程图。所述方法包括以下步骤:

[0095] 步骤302,获取待存储的区块链数据的key-value键值对;

[0096] 步骤304,将所述待存储的区块链数据的key-value键值对转换成逻辑的树形结构上的根节点、中间节点和叶子节点;所述根节点、中间节点包括主位置和用于存放所述区块链数据的key中的字符的多个子位置;所述主位置包括多个分别与各子位置对应的,用于存放各子位置中的存储内容的hash值的槽位;所述子位置包括多个用于存放所述区块链数据的key中的字符的槽位;所述子位置中的槽位用于存储与该节点链接的下一层节点的hash值;所述根节点、中间节点的hash值为所述根节点、中间节点中的主位置中的存储内容的hash;

[0097] 步骤306,将所述根节点、中间节点和叶子节点的key-value键值对存储在数据库中;所述叶子节点、所述中间节点和所述根节点的key-value键值对中,value为节点的存储内容,key为节点的存储内容的hash值。

[0098] 在本说明书中,为了避免节点频繁发生分裂,提出一种前N层的用于存储区块链数据的key中的字符节点的层数固定的逻辑的树状结构。

[0099] 其中,所谓逻辑的树形结构,是指在数据库的底层物理存储中,并不存在与树形结构完全对应的物理存储结构,而仅在数据库中存储上述树形结构上的各个节点的物理数据以及各个节点之间的链接关系数据,从而可以基于数据库中存储的各个节点的物理数据和链接关系数据,在逻辑层面上还原出上述树形结构。

[0100] 上述逻辑的树状结构,具体可以包括根节点、中间节点和叶子节点;其中,上述逻辑的树状结构上的节点,可以通过其自身的hash值与上一层的节点进行链接。上述根节点和中间节点,具体用于存储区块链数据的key-value键值对中的key中的至少一个字符;上述叶子节点具体用于存储区块链数据的value(即区块链数据的具体内容)。中间节点的层

数可以为一层也可以为多层,在本说明书中不进行特别限定。

[0101] 例如,在一个例子中,上述区块链数据的key,仍然可以包括字符前缀部分(Shared nibble)和字符后缀部分(key-end);在这种情况下,根节点和中间节点可以用于存储上述字符前缀中的字符;而上述叶子节点则可以用于存储上述字符后缀和区块链数据的value。

[0102] 一方面,由于上述逻辑的树形结构中的根节点和中间节点,可以存放区块链数据的key中的字符;因此,上述逻辑的树形结构具有Trie字典树的特点;另一方面,上述逻辑的树状结构上的节点,可以通过其自身的hash值与上一层的节点进行链接;因此,上述逻辑的树形结构也具有Merkle树的特点。综上,本说明书描述的逻辑的树形结构,实际上是一种类似于MPT树的融合了Trie字典树的树形结构的Merkle树变种。

[0103] 请参见图4,图4为本说明书示出的一种FDMT(Fixed Depth Merkle Tree)树的树形结构图。上述FDMT树是一种融合了Trie字典树的树形结构的Merkle树变种。

[0104] 如图4所示,在本说明书示出的FDMT树的树形结构中,包含前N层(图4示出的为3层,仅为示意性的)的Tree node,和最后一层的Leaf node(即叶子节点)。其中,在前N层的Tree node中,第一层为作为根节点的Tree node,其它层为作为中间节点的Tree node。

[0105] 其中,与以上描述的MPT树不同的是,上述FDMT树前N层的各个Tree node(即根节点和中间节点),将采用统一的数据结构,从而使得上述FDMT树前N层的各个Tree node,可以不再因为新写入了数据导致存储的字符的长度发生变化而发生节点分裂。

[0106] 如图4所示,上述FDMT树上的前N层的Tree node,均可以包括分别代表不同字符的多个block;上述block为用于存放区块链数据的key中的字符的“位置”。而每一个block可以进一步包括多个分别代表不同字符的槽位;上述槽位也用于存放区块链数据的key中的字符。例如,图4示出的为每一个Tree node包括N个block;每个block进一步包括N个slot。其中,上述FDMT树上各层的节点之间,仍然可以采用在上一层的节点中填充下一层的节点的hash值(hash指针)的方式,来进行节点间的链接。也即,上述FDMT树中的节点,通过其自身的hash值,链接至上一层的节点。相应的,上述槽位具体可以用于填充当前的Tree node所链接的下一层节点的hash值。Tree node的下一层节点,具体仍然可以是Tree node,也可以是Leaf node。

[0107] 其中,在实际应用中,当上述FDMT树上的Tree node中的任一block中的任一槽位填充的hash值发生更新后,通常需要重新计算该Tree node的hash值,并根据计算出的该hash值与上一层的节点重新进行链接。

[0108] 而在计算该Tree node的hash值时,通常需要将Tree node中填充的所有数据都作为计算参数进行hash计算;因此,在Tree node采用了如图4所示的数据结构的情况下,如果需要计算该Tree node的hash值,则需要先计算出该Tree node包含的各个block的hash,再将各block的hash拼接起来,再针对拼接起来的hash执行二次的hash计算。

[0109] 通过这种方式,当上述FDMT树上的Tree node中的任一block中的任一槽位填充的hash值发生更新,在重新计算该Tree node的hash值时,即便该Tree node中其它的各个block未发生数据更新,也仍然需要作为计算参数参与hash计算,显然存在hash计算的计算量较大的问题。

[0110] 基于此,在本说明书中,为了降低在计算Tree node的hash时的计算量,上述Tree node具体可以采用主block(即主位置)和多个子block(即子位置)的数据结构。

[0111] 请参见图5,图5为本说明书示出的另一种FDMT树的树形结构图。

[0112] 如图5所示,在本说明书示出的FDMT树的树形结构中,仍然包含前N层的Tree node,和最后一层的叶子节点。

[0113] 其中,与图4示出的上述FDMT树的结构不同的是,如图5中示出的上述FDMT树上的前N层的Tree node,均可以包括主block(即图5中示出的root block)和分别代表不同字符的多个子block;而每一个block还可以进一步包括多个槽位;例如,图5示出的为每一个Tree node均包括一个主block和N个子block;而每个子block又进一步包括N个slot。

[0114] 其中,上述主block和子block包含的槽位的功能,具体可以有所不同;如图5所示,对于主block而言,可以包含分别与各子block对应的多个槽位,每一个槽位具体可以用于填充对应的子block中存储的数据内容的hash;

[0115] 例如,在计算任一子block的hash时,具体可以将该子block中各槽位填充的hash值拼接起来,再针对拼接起来的hash执行二次的hash计算,得到该子block的hash。

[0116] 而对于子block而言,可以包含分别代表不同的字符的多个槽位,每一个槽位具体可以用于填充该Tree node链接的下一层节点的hash值。

[0117] 在本说明书中,按照图5示出的Tree node的结构,则可以用该Tree node中的主block的hash值来表示该Tree node的hash值;从而,在这种情况下,当上述FDMT树上的Tree node中的任一子block中的槽位填充的hash值发生更新后,则在重新计算该Tree node的hash值时,只需要将该发生数据更新的该子block中各个槽位中填充的hash值进行拼接,并将拼接出的hash值作为计算参数重新进行hash计算,再将计算出的hash值填充至该Tree node中的主block中与该子block对应的槽位,然后再对主block中的各槽位填充的hash进行拼接,再将拼接出的hash值作为计算参数重新进行hash计算即可得到该Tree node的hash。

[0118] 在整个过程中,对于未发生数据更新的其它子block中各个槽位填充的hash值,将不再需要进行拼接后作为计算参数参与hash计算,从而可以降低在重新计算该Tree node的hash值时的hash计算量和计算时长,提升hash计算的计算效率。

[0119] 需要解释的是,图4和图5中示出的上述FDMT树上各层的节点之间的链接关系仅为示意性的,并不是指对上述FDMT树上各层的节点之间的链接关系的一种特殊限定。

[0120] 请继续参见图4和图5,图4和图5示出的上述FDMT树上的各Tree node,均可以用于存储上述区块链数据的key中的至少部分字符。

[0121] 对于图4示出的上述FDMT树上的各Tree node,实际存储的字符,具体可以是该Tree node中的block(即至少有一个槽位填充了hash值的非空block)所代表的字符,与该block中填充了hash值的槽位(即非空槽位)所代表的字符,进行拼接生成的字符串。

[0122] 而对于图5示出的上述FDMT树上的各Tree node,实际存储的字符,具体可以是该Tree node中的子block所代表的字符,与该block中填充了hash值的槽位所代表的字符,进行拼接生成的字符串。

[0123] 其中,需要说明的是,在实际应用中,Tree node中的每一个block,可以仅代表一位字符;也即,基于图4和5示出的Tree node的存储格式,每一个Tree node实际存储的上述区块链数据的key的字符前缀中的部分字符,为长度是2位字符的字符串。

[0124] 例如,请参见图6,图6为本说明书示出的一种Tree node的结构图;

[0125] 如图6所示,该Tree node包含16个代表不同的16进制字符的子block;每一个子block进一步包括16个分别代表不同的16进制字符的slot(图6中只示出了block6包含的16个槽位);假设该Tree node中的子block6(代表了16进制字符6)为非空block,该子block中的slot4(代表了16进制字符4)、slot6(代表了16进制字符6)和slot9(代表了16进制字符9)为填充了该Tree node链接的下一层节点的hash值的非空slot;则该Tree node存储的上述区块链数据的key的字符前缀中的部分字符,分别为16进制字符串“64”、“66”和“69”。

[0126] 其中,上述Tree node中所包含的子block的数量,以及每一个子block所包含的槽位的数量,在本说明书中不进行特别限定;在实际应用中,可以基于上述区块链数据的key对应的字符串所包含的字符元素的类型数,来确定上述Tree node所包含的子block的数量;以及,子block所包含的槽位的数量;

[0127] 例如,假设上述区块链数据对应的key为16进制字符串,此时上述区块链数据的key对应的字符串所包含的字符元素的类型数为16;则上述Tree node中所包含的子block的数量,以及每一个子block所包含的槽位的数量都可以为16。

[0128] 需要说明的是,在实际应用中,上述Tree node所包含的子block的数量,和上述子block所包含的槽位的数量,可以保持相同;

[0129] 例如,在一个例子中,以上述区块链数据的key对应的字符串为16进制字符串为例,在这种情况下,上述前N层的Tree node可以均包括16个分别代表不同的16进制字符的子block;而每一个子block可以进一步包括16个分别代表不同的16进制字符的槽位。

[0130] 通过这种方式,则上述FDMT树前N层每一层中的单个Tree node,可以具有16*16=256个槽位,显然图4示出的FDMT树上的Tree node相对于图1示出的MPT树上用于存储字符前缀的节点而言,将具有更大的存储容量。

[0131] 当前,在实际应用中,上述Tree node所包含的子block的数量,和上述子block所包含的槽位的数量,具体也可以不相同;

[0132] 例如,在实际应用中,上述区块链数据的key对应的字符串,具体也可以是由两种不同进制字符构成的字符串;比如,上述区块链数据的key对应的字符串具体可以是由16进制的字符和10进制的字符混合构成的字符串,在这种情况下,上述前N层的Tree node可以均包括16个分别代表不同的16进制字符的子block;而每一个子block可以进一步包括10个分别代表不同的10进制字符的槽位;或者,上述前N层的Tree node可以均包括10个分别代表不同的10进制字符的子block;而每一个子block可以进一步包括16个分别代表不同的16进制字符的槽位。

[0133] 在示出的一种实施方式中,上述FDMT树包含的Tree node的层数,具体可以是一个固定值;在实际应用中,上述N的取值具体可以是一个大于或者等于1的整数;也即,上述FDMT树具体可以是一棵包含至少一层Tree node,并且包含的Tree node的层数相对固定的Merkle树。

[0134] 例如,在一个例子中,以上述区块链数据的key为区块链账户地址为例,假设区块链系统支持的区块链账户地址被设计为,前6位地址字符可以相同,那么在这种情况下,可以将区块链账户地址前6位的地址字符,作为区块链账户地址的字符前缀;而且,由于Tree node存储的区块链账户地址的字符前缀中的字符的长度为2位字符;因此,上述FDMT树可以被设计成包含三层Tree node的树状结构。

[0135] 基于图4和图5示出的FDMT树的树形结构设计,一方面,由于FDMT树中的每一个Tree node,均包括分别代表不同字符的多个block;并且,各block又进一步包括多个分别代表不同字符的槽位;因此,通过这种设计,使得每一个Tree node将具有更大的数据存储容量和数据承载能力,从而在将上述FDMT树中的Tree node写入数据库进行存储时;或者,在访问数据库中存储的上述FDMT树中的Tree node时,可以使得Tree node的数据存储容量,与承载上述数据库的存储介质的单次IO读写的容量更加适配,从而可以充分利用承载上述数据库的存储介质自身的IO读写能力,提升数据读写效率;而且,上述FDMT树上单个Tree node的数据存储容量和数据承载能力的提升,势必也会导致上述FDMT树整体的数据存储容量和数据承载能力的提升,使得上述FDMT树上可以存储更多的区块链数据;

[0136] 例如,以承载上述数据库的存储介质为单个物理扇区为4KB大小的磁盘为例,假设该磁盘单次IO读写的容量为4kb(一个扇区),对于图1描述的MPT树上的1个Branch node而言,假设Branch node包含的16个字段均填充了32byte的hash值,则该Branch Node的数据存储容量是32字节*16=512byte左右;显然针对Branch node的一次IO读取的容量最大只能是512byte左右,远小于该磁盘一次IO读取最大可以读取到4Kb的读取能力,这显然无法充分利用磁盘的IO读取能力,存在严重的性能浪费。

[0137] 但是,如果采用如图4和图5所示出的FDMT树的树形结构设计,假设上述前N层的Tree node可以均包括16个分别代表不同的16进制字符的block;每一个block可以进一步包括16个分别代表不同的16进制字符的槽位,则上述FDMT前N层每一层中的单个Tree node,可以具有16*16=256个槽位;假设每一个槽位填充的是32byte的hash值,则在所有槽位满负荷的情况下,一个Tree node的最大存储容量则是256*32byte=8192byte=8kb,刚好是两个扇区的容量。显而易见的,图4和图5示出的FDMT的树形结构中的每一个Tree node的数据存储容量,与上述磁盘的单次IO读写的容量更加适配,可以充分利用该磁盘自身的IO读写能力,提升数据读写效率。

[0138] 而且,上述FDMT上单个Tree node的数据存储容量和数据承载能力的提升,势必也会导致上述FDMT整体的数据存储容量和数据承载能力的提升,使得上述FDMT上可以存储更多的区块链数据;

[0139] 例如,假设图4和图5示出的FDMT的树形结构包含3层Tree node,每一层的Tree node可以均包括16个分别代表不同的16进制字符的子block;每一个block可以进一步包括16个分别代表不同的16进制字符的槽位;那么,每一层中的单个Tree node,可以具有16*16=256个槽位;则三层Tree node共计可以承载256*256*256=16.77M个字符组合,共计可以链接16.77M个bucket数据块;假设用户定义每一个bucket数据块可以承载16条数据记录,则整棵FDMT树最多可以承载16.77M*16条区块链数据;显而易见的,相较于图1示出的MPT树,图4和图5示出的上述FDMT可以存储更多的区块链数据,具有更大的数据承载能力。

[0140] 第二方面,由于FDMT中的每一个Tree node均采用统一的数据结构;对于上述FDMT中各层的Tree node来说,其实际存储的区块链数据的key的字符前缀的字符长度也将保持固定;因此,通过这种设计,可以避免由于各层Tree node其实际存储的字符长度不固定,而导致的节点的频繁分裂,从而可以确保FDMT的树形结构中所包含的Tree node的层数,始终处于一个相对稳定的状态。

[0141] 第三方面,由于通过这种设计,每一个Tree node将具有更大的数据存储容量和数

据承载能力;并且,上述FDMT的树形结构中包含的Tree node的层数,将处于一个相对稳定的状态;因此,Tree node存储容量的提升和Tree node的层数相对稳定,在某种程度上可以确保上述FDMT的Tree node将具有更少的层数;从而,在Tree node具有更大的数据存储容量和数据承载能力,以及上述FDMT的Tree node具有更少的层数的基础之上,系统在进行冷启动,需要从承载上述数据库的存储介质中,将上述FDMT的前N层的Tree node作为需要频繁读取的数据加载到内存中时,则可以显著降低将上述存储介质存储的上述前N层的Tree node读取到内存中时的IO读取次数,以及将FDMT的前N层的Tree node加载到内存时的整体读取时长,进而从根本上缩短系统冷启动时的启动时延。

[0142] 例如,以图4和图5示出的FDMT为3层Tree node固定,且每一层的Tree node包括16个分别代表不同的16进制字符的block;每一个block进一步包括16个分别代表不同的16进制字符的槽位为例,在所有槽位满负荷的情况下,一个Tree node的最大存储容量则是256*32byte=8192byte=8kb;对于图1示出的MPT树而言,由于其需要进行频繁的节点分裂,MPT树的层数不固定;而且,由于其单个Branch Node节点的存储容量512byte,远小于图4和图5示出的FDMT上的Tree node;势必造成MPT树具有更大的层数(比如MPT树最大可以达到64层,远大于3层)。而系统在冷启动,将FDMT树前N层读取到内存时,通常是一层一层读取的;因此,基于图1示出的MPT树,显然需要更多的读取次数。

[0143] 而且,由于MPT树上的单个Branch Node节点的存储容量512byte,仅为承载上述数据库的单个物理扇区为4KB的八分之一,读取效率很低;因此,即便按照图1的MPT树和图4和图5示出的FDMT树存储同样的数据,系统在冷启动针对MPT树的IO读取次数,也可能是针对图4和图5示出的FDMT树的IO读取次数的至少8倍。

[0144] 显而易见的,系统在冷启动时,针对图4和图5示出的上述FDMT树的IO读取次数,将远小于针对图1示出的MPT树的IO读取次数;因此,图4和图5示出的上述FDMT树的树形结构设计对系统冷启动将更加友好。

[0145] 在示出的一种实施方式中,上述区块链数据的key对应的字符串仍然可以包括字符前缀和字符后缀;在这种情况下,上述Tree node可以用于存储所述区块链数据的key的字符前缀中的字符;上述叶子节点可以用于存储所述区块链数据的key的字符后缀和上述区块链数据的Value。

[0146] 其中,需要说明的是,由于上述叶子节点实际存储的数据,相对于上述Tree node来说,通常具有更大的数据容量;比如,上述叶子节点实际存储的上述区块链数据的value,通常为上述区块链数据的原始内容,上述区块链数据的原始内容相对于上述区块链数据的字符前缀而言,其所占用的存储空间也更大;因此,在本说明书中,为了确保上述叶子节点能够具有更大的数据容量,上述叶子节点具体可以采用大数据块的形式来存储数据。

[0147] 其中,上述数据块的具体形式以及存储结构,在本说明书中不进行特别限定;

[0148] 在示出的一种实施方式中,上述叶子节点具体可以是bucket数据桶的形式;其中,上述bucket数据桶具体可以是用于存储数据的容器或者存储空间。

[0149] 请参见图7,图7为本说明书示出的一种bucket数据桶的结构图;

[0150] 如图7所示,在上述bucket数据桶(即图7中示出的bucker node)中,可以包括若干条数据记录;其中,需要说明的是,上述bucket数据桶中包含的上述若干条数据记录,在逻辑上可以不是一个整体,而是在逻辑上分离的若干条不同的数据记录。每一条数据记录都

分别对应一条区块链数据,用于存储上述区块链数据的value;也即,一条数据记录,是指一条存储的数据内容包括所述区块链数据的value的存储记录。

[0151] 需要说明的是,上述bucket数据桶中包含的上述若干条数据记录,在逻辑上不是一个整体,是指各条数据记录都可以分别对应一个独立的查询键值(key),从而可以基于各条数据记录的查询键值,来针对上述bucket数据桶中的各条数据记录分别进行精确查询,而并不要对上述bucket数据桶中存储的所有的数据记录进行整体的读取。

[0152] 其中,各条数据记录分别对应的查询键值的具体形式和具体内容,在本说明书中不进行特别限定,可以是任意形式的能够作为各条数据记录的查询索引的字符串。

[0153] 在示出的一种实施方式中,各条数据记录对应的查询键值具体可以是各条数据记录包含的数据内容的hash值;上述bucket数据桶中包含的上述数据记录,则具体可以包括由上述区块链数据的value的hash值,和上述区块链数据的value对应的数据内容构成的key-value键值对。

[0154] 当然,在实际应用中,各条数据记录对应的查询键值也可以是hash值以外的其它形式的能够作为各条数据记录的查询索引的字符串,在本说明中不进行特别的限定;例如,在一个例子中,各条数据记录对应的查询键值具体也可以是由节点设备为各条数据记录分别设置的一个唯一的标识(比如编号)。

[0155] 在示出的一种实施方式中,如果上述Tree node用于存储上述区块链数据的key的字符前缀中的字符;则上述叶子节点用于存储上述区块链数据的key的字符后缀和上述区块链数据的Value;

[0156] 相应的,上述bucket数据桶中包含的上述数据记录,具体可以是将上述区块链数据的key中的字符后缀和上述区块链数据的value对应的数据内容整体进行hash计算得到的hash值,和上述区块链数据的key的字符后缀和上述区块链数据的value对应的数据内容构成的key-value键值对。

[0157] 需要说明的是,在实际应用中,上述数据记录具体也可以是key-value键值对以外的其它类型的数据形式,在本说明书中不再进行一一列举。

[0158] 通过以上实施方式,由于FDMT树中的叶子节点中所包含的若干条数据记录,在逻辑上不再是一个整体,而是在逻辑上分离的若干条key-value键值对;因此,上述FDMT树中的叶子节点所包含的数据,将不再作为数据库中的一个整体的存取单位进行整体存取,上述叶子节点所包含的每一条key-va lue键值对将作为数据库中的一个独立的存取单位,从而使得针对上述数据库的数据存取将更加灵活;

[0159] 例如,以上述区块链数据的value为区块链中的区块链账户对应的最新账户状态数据,上述区块链数据的key为区块链账户地址为例,在这种情况下,上述bucket数据桶中包含的上述数据记录对应的key-value键值对的key,可以是区块链账户地址的字符后缀和对应的账户状态数据两部分数据内容的hash值;该key-value键值对的value,可以是区块链账户地址的字符后缀和对应的账户状态数据两部分的数据内容。

[0160] 假设需要读取上述bucket数据桶中包含的某一个特定的账户地址的字符后缀和对应的账户状态数据,由于上述bucket数据桶中的各条key-value键值对都是一个独立的存取单位;因此,只需要基于该特定的账户地址的字符后缀和对应的账户状态数据这两部分数据内容的hash值,在数据库进行内容寻址即可,而不再需要从数据库中将该叶子节点

中所包含的所有数据内容,整体读取到内存中,再在内存中进一步查找需要读取的账户地址的字符后缀和对应的账户状态数据;

[0161] 相应的,如果需要向上述bucket数据桶写入一个新的账户地址的字符后缀和对应的账户状态数据,或者对上述bucket数据桶中包含的一个特定的账户地址对应的账户状态数据进行更新,则可以直接根据新的账户地址的字符后缀和对应的账户状态数据构建key-value键值对,并将该key-value键值对写入上述bucket数据桶即可;或者,基于该特定的账户地址的字符后缀和对应的账户状态数据这两部分数据内容的hash值,在数据库进行内容寻址,查找到对应的key-value键值对,然后写入该特定的账户地址对应的更新后的账户状态数据,对该key-value键值对原有的value进行更新即可。

[0162] 其中,需要说明的是,上述bucket数据桶中所包含的数据记录的条数,在本说明书中不进行特别限定;在示出的一种实现方式中,上述bucket数据桶中包含的数据记录的条数,具体可以由用户进行自定义配置。

[0163] 例如,以上述区块链数据为区块链中的区块链账户对应的最新账户状态数据,上述区块链数据的key为区块链账户地址为例,在这种情况下,上述bucket数据桶中的每一条数据记录,分别与一个区块链账户的账户状态相对应;上述bucket数据桶中的数据记录的条数,实际上代表着该bucket数据桶可以容纳区块链账户的账户承载能力;因此,用户通过对上述bucket数据桶能够容纳的数据记录的条数进行自定义,可以实现对上述bucket数据桶的账户承载能力进行灵活的定制;比如,在一个例子中,上述bucket数据桶中包含的数据记录的条数可以由用户配置为16条或者64条,从而使得单个bucket数据桶可以承载16或者64个区块链账户的状态数据。

[0164] 其中,还需要说明的是,上述bucket数据桶中所存储的数据记录的总存储容量,在本说明书中也不进行特别限定;在示出的一种实现方式中,上述bucket数据桶中所存储的数据记录的总存储容量,具体可以由用户进行自定义配置。

[0165] 例如,在实现时,以承载上述数据库的存储介质为单个物理扇区为4KB大小的磁盘为例,在这种情况下,用户可以将上述bucket数据桶中所存储的数据记录的最大存储容量,设置为4KB;或者,4KB的整数倍,以使上述bucket数据桶的最大存储容量,能够与上述存储介质的单个物理扇区的存储容量相适配。

[0166] 在本说明书中,如前所述,上述FDMT树上前N层的各Tree node,由于其采用统一的数据结构;因此,上述FDMT树上前N层的各Tree node,将不再发生节点分裂。而上述FDMT树上的叶子节点,仍然可以在满足一定的分裂条件时进行节点分裂。

[0167] 与上述MPT树不同的是,在本说明书中,上述FDMT树上的叶子节点在进行节点分裂时,可以不再按照共享字符前缀的字符长度发生变化来进行分裂,而是按照叶子节点的存储容量进行分裂。当上述FDMT树上的叶子节点的存储容量满足节点分裂条件时,再从该叶子节点中分裂出中间节点。需要说明的是,分裂出的中间节点,为上述叶子节点的上层节点。对同一个叶子节点而言,可以进行多次分裂。

[0168] 其中,上述节点分裂条件,可以包含与上述叶子节点的存储容量相关的任意形式的条件,在本说明书中不进行特别限定;

[0169] 在示出的一种实施方式中,上述节点分裂条件,具体可以是以下示出的条件1和条件2中的任意一个;或者,也可以是以下示出的条件1和条件2中的组合:

[0170] 条件1:上述叶子节点中存储的数据记录的总数量大于阈值;

[0171] 条件2:上述叶子节点中存储的数据记录的总存储容量大于阈值。

[0172] 例如,在一个例子中,可以将上述节点分裂条件设置为,当上述叶子节点中存储的数据记录的总数量大于64条;和/或,上述叶子节点中存储的数据记录的总存储容量大于4KB时,则从该叶子节点中分裂出中间节点。

[0173] 在本说明书中,上述Extend node,也是上述FDMT树的中间节点,可以采用与以上描述的Tree node完全相同的数据结构,具体用于存储从上述叶子节点存储的字符后缀中分裂出的字符;

[0174] 例如,从叶子节点中分裂出的Extend node,也可以包括分别代表不同字符的多个block;而每一个block可以进一步包括多个分别代表不同字符的槽位;该槽位具体可以用于填充该节点链接的下一层节点的hash值。比如,对于Extend node而言,其下一层节点为上述叶子节点;因此,Extend node中的block所包含的槽位中,可以用于填充该Extend node链接的下一层叶子节点的hash值。

[0175] 在本说明书中,对于从上述叶子节点中分裂出的Extend node而言,还可以按照Extend node的存储容量与其链接的下层的叶子节点进行合并。当上述FDMT树上的Extend node的存储容量满足节点合并条件时,可以将该Extend node合并至与其链接的下层的叶子节点。

[0176] 其中,将Extend node合并至与其链接的下层的叶子节点,是指将该Extend node中存储的字符作为上述字符后缀,写入与该Extend node链接的下层的一个或者多个叶子节点。

[0177] 其中,上述节点合并条件,也可以包含与上述Extend node的存储容量相关的任意形式的条件,在本说明书中不进行特别限定;

[0178] 在示出的一种实施方式中,上述节点合并条件,具体可以是以下示出的条件3和条件4中的任意一个;或者,也可以是以下示出的条件3和条件4中的组合:

[0179] 条件3:上述Extend node中存储的字符的总数量小于或等于阈值;也即,上述Extend node中各个block中非空slot(即填充了hash值的slot)的总数量;

[0180] 条件4:上述Extend node中存储的字符的总存储容量小于或等于阈值。也即,上述Extend node中各个block中非空slot中填充的hash值的总存储容量;

[0181] 例如,在一个例子中,可以将上述节点合并条件设置为,当上述Extend node中存储的字符的总数量小于或者等于1条;和/或,上述Extend node中存储的字符的总存储容量小于或者等于1KB时,则将该Extend node存储的字符作为字符后缀,写入该Extend node链接的下层的一个或者多个叶子节点。

[0182] 通过以上描述可知,本说明书中描述的Extend node,可以是上述FDMT树上动态可伸缩的中间节点;当上述FDMT树上的任一叶子节点的存储容量满足节点分裂条件时,可以从该叶子节点中分裂出至少一个Extend node,以及至少一个叶子节点;当任一Extend node的存储容量满足节点合并条件时,可以将该Extend node合并至与其链接的下层的叶子节点。

[0183] 在本说明书中,待存储的区块链数据,具体可以包括至少以下四类数据:

[0184] 区块中收录的交易;在区块中的交易执行完毕后,与所述区块中收录的交易对应

的交易收据;在区块中的交易执行完毕后,与所述区块链中的区块链账户对应的最新账户状态数据;智能合约账户的存储内容;

[0185] 相应的,上述FDMT树具体也可以包括:

[0186] 用于存储区块中收录的交易的交易树;用于存储与区块中收录的交易对应的交易收据的收据树;用于存储与所述区块链中的区块链账户对应的最新账户状态数据的状态树;用于存储智能合约账户的存储内容的存储树。

[0187] 当然,在实际应用中,也可以只利用上述FDMT树的树形结构来存储上述四种数据中的部分类型的数据;比如,只利用上述FDMT树来存储上述区块链账户对应的最新账户状态数据,其他类型的数据可以利用其他形式的二叉树来存储(比如MPT或者其它形式的二叉树)。

[0188] 其中,上述交易树、收据树和状态树的根节点的hash值,可以存储在区块头中;上述存储树的根节点的hash值,可以存储在与该存储树对应的合约账户的结构体中的Storage字段中。

[0189] 上述区块链数据的key,具体可以是指区块链数据在数据库中对应的查找键值;相应的,上述区块链数据的Value,具体可以是上述区块链数据的原始内容;

[0190] 其中,在实际应用中,上述查找键值具体可以是一个与区块链数据对应的字符串;当上述区块链数据为不同类型的数据时,其对应的key也会存在一定的差异;

[0191] 例如,当上述区块链数据为区块中收录的交易;或者,在区块中的交易执行完毕后,与该区块中收录的交易对应的交易收据时,此时与该区块链数据对应的key,具体可以是交易在区块中的序号;或者,其它形式的交易标识。

[0192] 当上述区块链数据为在区块中的交易执行完毕后,与区块链中的区块链账户对应的最新账户状态数据时,此时与该区块链数据对应的key,具体可以是区块链账户的账户地址。

[0193] 当上述区块链数据为智能合约账户的存储内容时,此时与该区块链数据对应的key,具体可以是合约账户的账户地址和上述存储内容在合约账户的账户存储中的存储位置的hash值;比如,在一个例子中,上述存储内容通常可以是状态变量,则可以将合约账户的账户地址和状态变量在合约账户的账户存储中的存储位置的hash值作为key。

[0194] 在本说明书中,当区块链中的节点设备在存储区块链数据时,可以先获取待存储的区块链数据的key-value键值对;例如,在一个例子中,节点设备可以在获取到待存储的区块链数据时,将该区块链数据处理成key-value键值对;

[0195] 在获取到待存储的区块链数据的key-value键值对之后,可以将该待存储的区块链数据的key-value键值对转换成上述逻辑的树形结构上的根节点、中间节点和叶子节点;

[0196] 例如,在一个例子中,上述节点设备可以搭载一个与上述FDMT树对应的存储接口或者存储服务,该存储接口或者存储服务具体可以用于将待存储的区块链数据的key-value键值对,转换成为FDMT树上的节点;则该节点设备在获取到待存储的区块链数据的key-value键值对之后,可以通过调用该存储接口或者存储服务,按照图4或者图5示出的FDMT树的树形结构,将待存储的区块链数据的key-value键值对,转换成为图4或者图5上的根节点、中间节点和叶子节点。

[0197] 在将存储的区块链数据的key-value键值对转换成上述逻辑的树形结构上的根节

点、中间节点和叶子节点之后,可以将这些根节点、中间节点和叶子节点以key-value键值对的形式存储在数据库中;

[0198] 例如,上述数据库通常是存储在上述节点设备上搭载的持久化存储介质(比如存储磁盘)中的;上述存储介质为与上述数据库对应的物理存储;在将上述FDMT树在数据库中进行存储时,具体可以通过执行commit命令,将上述FDMT树上的节点,以Key-Value键值对的形式,从节点设备的内存中,进一步写入承载上述数据库的存储介质。

[0199] 其中,上述数据库的具体类型,在本说明书中不进行特别限定,本领域技术人员可以基于实际的需求进行灵活的选择;

[0200] 在一种实现方式中,上述数据库,具体可以是Key-Value型数据库;例如,在一个例子中,上述数据库可以为采用多层存储结构的LevelDB数据库;或者,基于LevelDB架构的数据库;比如,Rocksdb数据库就是一种典型的基于LevelDB数据库架构的数据库。

[0201] 需要说明的是,上述FDMT树上的节点,在以Key-Value键值对的形式在上述数据库中进行存储时,上述Key-Value键值对的key,具体可以是上述FDMT树上的节点的节点ID;

[0202] 其中,上述节点ID具体可以包括能够唯一标识上述FDMT树上的节点的标识信息;

[0203] 例如,在一种实现方式中,上述节点ID具体可以是上述FDMT树上的节点包含的数据内容的hash值;在这种情况下,在需要查询上述FDMT树上的节点时,可以基于节点所包含的数据内容的hash值作为key来进行内容寻址。

[0204] 在另一种实现方式中,上述节点ID具体也可以包括上述FDMT树上的节点在上述FDMT树中的路径信息;也即,将上述FDMT树上的节点在上述FDMT树中的路径信息作为该节点的节点ID;其中,上述路径信息具体可以包括,能够描述节点与其它节点之间的链接关系,以及节点在上述FDMT树上的位置的任意形式的信息;

[0205] 例如,请参见图8,图8为本说明书示出的一种为FDMT树上的节点设置节点ID的示意图;对于如图8所示包含三层Tree node的FDMT树,假设作为根节点的tree node的节点ID用0x00表示,那么图8示出的bucket node的节点ID,可以表示成0x00123456。

[0206] 其中,0x00为根节点的节点ID;123456是指上述bucket node在FDMT树上从根节点到该bucket node的路径信息;12表示第一层tree node的第一个block的第2个槽位;34表示第二层tree node的第3个block的第4个槽位;56表示第三层tree node的第5个block的第6个槽位。

[0207] 基于该节点ID,可以明确该bucket node与其它node之间的链接关系,以及该bucket node在上述FDMT树上的具体位置;比如,基于该节点ID,可以明确该bucket node链接在第三层tree node的第5个block的第6个槽位;而第三层的tree node又链接在第二层tree node的第3个block的第4个槽位;第二层的tree node又进一步链接在第一层作为根节点的tree node的第1个block的第2个槽位。

[0208] 通过这种方式,可以在使用节点ID来检索FDMT树上存储的节点时,可以精确定位到该节点在FDMT树上所在的具体槽位。

[0209] 在另一种实现方式中,上述节点ID具体也可以包括上述FDMT树上的节点在上述FDMT树中的相对位置,和该节点所包含的数据内容的hash值(此时该节点ID也可以作为该节点的hash标识);也即,将上述FDMT树上的节点在上述FDMT树中的相对位置,和该节点所包含的数据内容的hash值作为该节点的节点ID。

[0210] 例如,在实现时,可以将节点在上述FDMT树中的相对位置,和该节点所包含的数据内容的hash值拼接生成的字符串,作为该节点的节点ID;当然,在实际应用中,在进行拼接的过程中,也可以进一步引入上述相对位置和上述hash值以外的其它类型的信息来生成节点ID;在本说明书中不再一一列举。

[0211] 通过这种方式,除了可以基于节点所包含的数据内容的hash值作为key来进行内容寻址以外,还可以在使用节点ID来检索FDMT树上存储的节点时,可以精确定位到该节点在FDMT树上所在的具体槽位。

[0212] 其中,需要说明的是,上述节点设备上用于承载上述数据库的存储介质,具体可以是持久性存储介质;例如,可以是能够对数据进行持久性存储的磁盘、内存或者其它形式的存储介质,在本说明书中不再进行一一列举。

[0213] 在本说明书中,在将上述FDMT树上的节点的key-value键值对分别写入上述数据库之前,还可以预先对上述FDMT树上的节点进行编码处理,然后将编码处理后的节点的key-value键值对在数据库中进行存储。

[0214] 其中,由于上述FDMT树上的Tree node和Extend node均采用了,包含多个block,每一个block又进一步包含多个槽位的数据结构;因此,在本说明书中,在针对上述FDMT树进行编码时,可以针对上述FDMT树上的Tree node和Extend node中的block执行bitmap编码。

[0215] 需要说明的是,在针对上述FDMT树上的Tree node和Extend node中的block(可以包括主block和子block)执行bitmap编码时,具体可以统计出该Tree node和Extend node中的blcok中的各槽位是否填充了hash值,然后再将统计结果利用特殊的编码字符表示出来得到bitmap编码信息,最后再将该bitmap编码信息添加至block中,完成针对该block的bitmap编码过程;

[0216] 例如,在一个例子中,上述bitmap编码信息,具体可以是一个用于标识block中的各槽位是否填充了hash值的16进制字符串;假设待编码的block(可以是主block,也可以是子block)共计包含16个槽位;在这16个槽位中第0位、第8位、第9位、第10位、第11位分别填充了hash值;则该block中各个槽位是否填充了hash的统计结果,可以用二进制字符串0000 1111 0000 0001表示;其中,该二进制字符串最左边是最高位第15位,最右边是最低位第0位;最后,可以将该二进制字符串转换成为16进制字符串0x0f01;此时该16进制字符串0x0f01即为该block的bitmap编码信息。

[0217] 其中,需要说明的是,在针对上述FDMT树上的Tree node和Extend node中的block执行bitmap编码时,无论是主block还是子block,采用bitmap编码处理过程是完全相同的。

[0218] 通过这种方式,由于编码后的上述FDMT树中的Tree node和Extend node包含的block中被添加了,能够指示该block中的各槽位是否填充了hash值的bitmap编码信息;因此,在需要查询数据库中存储的各Tree node中的block里填充的数据时,通过该block中的bitmap编码信息,就可以确定该block里的非空槽位,而不再需要遍历该block中的各个槽位,从而可以提升查找效率。

[0219] 其中,由于上述FDMT树上的叶子节点采用了与Tree node和Extend node完全不同的数据结构;因此,在对上述FDMT树上的叶子节点进行编码时,也可以采用与上述Tree node和Extend node完全不同的编码方式。

[0220] 在示出的一种实施方式中,在针对上述FDMT树上的叶子节点进行编码时,具体仍然可以采用RLP(Recursive Length Prefix,递归长度前缀编码)编码;其中,RLP编码是如图1示出的MPT树常用的一种编码方式,其具体的编码过程,在本说明书中不再进行详述。

[0221] 当然,在实际应用中,针对上述FDMT树上的叶子节点进行编码时所采用的编码方式,也可以是RLP编码以外的其它形式的编码方式,在实际应用中,可以灵活的选择,在本说明书中不再进行一一列举。

[0222] 在本说明书中,在针对上述FDMT树上的节点编码完成后,可以将上述FDMT树上的节点的key-value键值对存储至上述数据库。

[0223] 其中,在示出的一种实施方式中,对于上述FDMT树上的Tree node和Extend node而言,在将编码完成的Tree node和Extend node的key-value键值对存储到数据库之前,可以根据编码处理后的Tree node和Extend node中的block被添加的bitmap编码信息,来确定该block中未填充hash值的空槽位,然后将确定出的空槽位从该block中删除。

[0224] 通过这种方式,可以在将各Tree node和Extend nodekey-value键值对存储至数据库中时,删除各block中的空槽位,可以进一步节约将各Tree node和各Extend node存储至数据库时所占用的存储空间,提升存储效率。

[0225] 其中,需要说明的是,对于编码后的block,在计算该block的hash值时,可以将编码后的block所包含的bitmap编码信息和非空字符槽位填充的hash值作为一个整体进行hash计算;当然,也可以将bitmap编码信息排除在外。

[0226] 在本说明书中,在将上述FDMT树上的节点key-value键值对分别写入上述数据库之后,如果上述FDMT树上写入的某一条区块链数据对应的value发生更新,此时可能需要对上述FDMT树上用于存储该区块链数据的字符后缀和value的叶子节点,从该叶子节点中分裂出的Extend node,以及用于存储该区块链数据的key的字符前缀的Tree node分别进行更新。

[0227] 当然,如果用于存储该区块链数据的字符后缀和value的叶子节点没有发生过分裂,可以只需要对上述FDMT树上用于存储该区块链数据的字符后缀和value的叶子节点,以及用于存储该区块链数据的key的字符前缀的Tree node分别进行更新即可。

[0228] 在这种情况下,节点设备可以在上述数据库中查找与上述需要数据更新的区块链数据对应的节点;其中,具体的查找方式不再赘述;再将查找到的节点从上述数据库读取到内存中,在内存中对这些节点进行修改更新,然后再将更新后的节点,进一步写入上述数据库对原有的更新节点进行更新。

[0229] 例如,对上述FDMT树上发生数据更新的叶子节点进行更新,具体可以包括对该叶子节点中存储的上述区块链数据的value进行更新。在更新完成后,可以再重新计算该叶子节点的hash值,并基于该hash值与该节点上一层的节点进行重新链接。

[0230] 而对上述FDMT树上发生数据更新的叶子节点的上层的Extend node或者Tree node进行更新,具体可以包括对该Extend node或者Tree node中填充了该叶子节点的hash值的槽位进行更新;在更新完成后,可以再重新计算Extend node或者Tree node的hash值,并基于该hash值与该节点上一层的节点继续进行重新链接。

[0231] 在示出的一种实施方式中,针对上述FDMT树上的叶子节点的分裂判断以及具体的节点分裂操作,可以在重新计算该叶子节点的hash值,并基于该hash值与该叶子节点上一

层的节点(可以是Tree node,也可以是Extend node)进行重新链接的阶段来完成。

[0232] 在这种情况下,节点设备可以确定上述FDMT树上的叶子节点是否发生数据更新,如果上述FDMT树上的任一叶子节点发生数据更新,则节点设备可以在重新计算该叶子节点的hash值,并基于该hash值与该叶子节点上层的Tree node或者Extend node进行重新链接之前,进一步确定该叶子节点的存储容量是否满足节点分裂条件;如果该叶子节点的存储容量满足节点分裂条件,可以进一步从该叶子节点分裂出至少一个Extend node。

[0233] 例如,在一个例子中,在叶子节点的存储容量满足节点分裂条件时,为了快速降低叶子节点的存储容量,可以将该叶子节点分裂为,一个Extend node和多个叶子节点。

[0234] 其中,从上述叶子节点中分裂出Extend node的分裂策略,在本说明书中也不进行特别限定,在实际应用中,用户可以根据具体的分裂需求,来进行自定义设置;

[0235] 在示出的一种实施方式中,上述分裂策略具体可以包括:将在当前的成块周期内写入叶子节点的最新的数据记录,从该叶子节点中分裂出去,并基于该最新的数据来额外创建Extend node和叶子节点。

[0236] 在这种情况下,在对任一目标叶子节点进行节点分裂时,可以确定在当前的成块周期内写入该目标叶子节点的最新的数据记录;再将这些最新的数据记录从该目标数据节点中删除,并从这些最新的数据记录所包含的字符后缀中分裂出字符前缀;

[0237] 例如,在一个例子中,默认可以将这些最新的数据记录的前两位分裂出去;或者,在这些最新的数据记录包含的字符后缀中存在共享字符前缀,并且该共享字符前缀的长度达到两位时,将该共享字符前缀分裂出去。

[0238] 然后,再进一步创建用于存储分裂出的该字符前缀的至少一Extend node,以及用于存储分裂出所述字符前缀之后的所述最新的数据记录的至少一叶子节点。

[0239] 通过这种方式,由于可以将当前的成块周期内写入该目标叶子节点的最新的数据记录,从该目标叶子节点中删除,并基于这些删除的最新的数据记录重新创建至少一个Extend node和至少一个叶子节点作为分裂出去的节点;因此,对于该目标叶子节点而言,在其进行节点分裂之后,该目标叶子节点实际存储的数据记录,将与上一个成块周期该目标叶子节点存储的数据记录完全一致;该目标叶子节点的hash值相对于上一个成块周期不会发生任何变化。从而,当该目标叶子节点在当前的成块周期写入了新的区块链数据,导致该目标叶子节点的存储容量满足了节点分裂条件时,只需要将新写入的数据记录分裂出去重新创建额外的Extend node和叶子节点即可,而不再需要重新计算该目标叶子节点的hash值与上一层的节点重新进行重新链接;通过这种分裂方式,不仅可以保证满足节点分裂条件的叶子节点的hash值处于一个稳定的状态,而且可以减少重新计算该叶子节点的hash值的计算次数。

[0240] 在示出的另一种实施方式中,上述分裂策略具体可以包括:将叶子节点存储的若干条数据记录所包含的字符后缀之间的共享字符前缀,从该叶子节点中分裂出去,并基于该共享字符前缀来额外创建Extend node。

[0241] 在这种情况下,在对任一目标叶子节点进行节点分裂时,可以确定目标叶子节点存储的若干条数据记录所包含的字符后缀之间是否存在共享字符前缀;如果该目标叶子节点存储的若干条数据记录所包含的字符后缀之间存在共享字符前缀,可以将该共享字符前缀从上述若干条数据记录中删除,并创建用于存储该共享字符前缀的至少一Extend node。

[0242] 例如,由于上述Extend node也可以采用与Tree node相同的数据结构;因此,上述Extend node存储的字符,也可以是block代表的字符和block中填充了hash值的slot所代表的字符拼接起来的,长度为两位的字符串;则在目标叶子节点存储的若干条数据记录所包含的字符后缀之间存在共享字符前缀,并且该共享字符前缀的长度达到两位时,可以将该共享字符前缀从上述若干条数据记录中分裂出去。

[0243] 在示出的一种实施方式中,针对上述FDMT上的Extend node的合并判断以及具体的节点合并操作,也可以在重新计算该Extend node的hash值,并基于该hash值与该Extend node上一层的节点进行重新链接的阶段来完成。

[0244] 在这种情况下,节点设备可以确定上述FDMT树上的Extend node是否发生数据更新,如果上述FDMT树上的任一从叶子节点中分裂出的Extend node发生数据更新,则节点设备在重新计算该Extend node的hash值,并基于该hash值与该Extend node上层的节点进行重新链接之前,可以进一步确定该Extend node的存储容量是否满足节点合并条件;

[0245] 如果该Extend node的存储容量满足节点分裂条件,可以进一步将该Extend node,合并至与该Extend node链接的下一层的叶子节点。

[0246] 例如,在实现时,首先可以确定该Extend node链接的下一层的叶子节点;比如,可以根据该Extend node包含的block中的非空slot中填充的hash值,来确定该Extend node链接的下一层的叶子节点;然后,可以将该Extend node存储的字符作为上述字符后缀的一部分,写入确定出的该Extend node链接的下一层的叶子节点。

[0247] 当然,在实际应用中,当一个叶子节点发生多次节点分裂时,此时该叶子节点可能分裂出多层Extend node;在这种情况下,当任一Extend node满足节点合并条件,如果其下层节点仍然是Extend node,不是叶子节点,则该Extend node暂时无法进行节点合并;在这种情况下,可以等待其下层的Extend node与更下层的叶子节点完成合并形成新的叶子节点之后,再与下层的该新的叶子节点进行二次合并,具体的合并方式不再赘述。

[0248] 在本说明书中,由于上述FDMT树上的Tree node采用了包含多个block,每一个block又进一步包含多个槽位的数据结构;因此,上述FDMT树上的Tree node将具有更大的数据存储容量。

[0249] 然而,由于上述FDMT树上的Tree node将具有更大的数据存储容量,在利用该FDMT树存储区块链数据时,势必会导致该FDMT树上存储的区块链数据过于“稀疏”;例如,利用该FDMT树存储区块链数据时,可能会高频出现单个Tree node只有一个block填充了数据,以及单个block只有一个槽位填充了数据的情况,这就造成了该FDMT树上存储的区块链数据所在的槽位之间间隔较大,导致该FDMT树上存储的区块链数据在逻辑上较为稀疏的问题。

[0250] 而该FDMT树上存储的区块链数据过于“稀疏”,在将该FDMT树在数据库中进行存储时,对大量空槽位进行存储,势必会造成存储空间的浪费,无法充分利用数据库的存储空间。

[0251] 基于此,在本说明书中,在将该FDMT树上的节点的key-value键值对在数据库中进行存储的过程中,或者在将该FDMT树上的节点的key-value键值对在数据库存储完成之后,可以对各Tree node中非空槽位的数量为1的block进行压缩处理,将其压缩至上一级的block中。

[0252] 需要强调的是,对各Tree node中非空槽位的数量为1的block进行压缩处理的时

机,可以是在将该FDMT树在数据库中进行存储的过程中,也可以是在将该FDMT树在数据库存储完成之后,在本说明书中不进行特别限定。在将该FDMT树上的作为中间节点的Tree node或者Extend node的key-value键值对在数据库中进行存储的过程中,或者在将该FDMT树上的作为中间节点的Tree node或者Extend node的key-value键值对在数据库存储完成之后,可以确定作为中间节点的Tree node或者Extend node中的各block的非空槽位的数量是否为1;如果任一Tree node或者任一Extend node中的任一目标block的非空槽位的数量为1,可以对该目标block进行压缩处理,将其压缩到其上一级的block或者上一级的节点中对应的block中;

[0253] 在示出的一种实施方式中,如果上述Tree node采用图4示出的数据结构;即上述Tree node包括分别代表不同字符的多个block;而每一个block可以进一步包括多个分别代表不同字符的槽位;在这种情况下,如果作为中间节点的任一Tree node或者Extend node中的任一目标block的非空槽位的数量为1,首先可以确定与该Tree node或者Extend node链接的上一层节点中用于填充该Tree node或者Extend node的hash值的目标槽位。

[0254] 然后,再将该目标位置的hash标识作为该Tree node或者Extend node的hash值填充至该目标槽位,并删除该Tree node或者Extend node。

[0255] 其中,上述hash标识,具体可以是指能够唯一标识FDMT树上一个中间节点;或者中间节点中的一个具体的block的标识信息;例如,当上述hash标识用于标识一个中间节点,则该hash标识可以作为该中间节点的节点ID。

[0256] 在本说明书中,hash标识具体可以包括上述唯一的非空槽位中填充的hash值,和与该非空槽位对应的压缩信息;也即,上述hash标识,具体是由上述唯一的非空槽位中填充的hash值,和与该非空槽位对应的压缩信息,拼接起来的一个二元组。

[0257] 其中,与该非空槽位对应的压缩信息,具体可以包括该非空槽位被压缩的次数和该非空槽位在上述作为中间节点的Tree node或者Extend node中的相对位置。所述相对位置具体可以能够定位出该非空槽位在中间节点上的相对位置的任意信息;比如,可以是将该非空槽位所在的block和具体的槽位号拼接起来的16进制或者其它进制的一串字符串。在示出的另一种实施方式中,如果上述Tree node采用图5示出的数据结构;即上述Tree node包括主block和分别代表不同字符的多个子block;而每一个block还可以进一步包括多个槽位:

[0258] 一方面,如果作为中间节点的任一Tree node或者Extend node中的任一目标子block的非空槽位的数量为1,则可以确定与该目标子block链接的上一级主block中用于填充该目标子block中的存储内容的hash值的目标槽位,并将该目标子block的上述hash标识作为该目标子block中的存储内容的hash值填充至上述目标槽位,并删除该目标子block。

[0259] 另一方面,如果作为中间节点的任一Tree node或者Extend node中的任一主block的非空槽位的数量为1,则可以确定与该Tree node或者Extend node链接的上一层节点中用于填充该Tree node或者Extend node的hash值的目标槽位,并将该主block的hash标识作为该Tree node或者Extend node的hash值填充至该目标槽位,并删除所述该Tree node或者Extend node。

[0260] 需要说明的是,以上描述的当存在只有一个槽位填充了数据的目标block时,对非空槽位的压缩过程,具体可以是一个递归压缩的过程;所谓递归压缩是指,将目标block的

非空槽位,压缩到该block的上一级block中后,该上一级block可能也存在相似的情况,只有一个槽位填充了数据;因此,针对该上一级block需要执行相同的压缩过程,将目标block的非空字符槽位逐级的不断的向更上级的block进行压缩。

[0261] 以下以上述Tree node采用图4示出的数据结构为例,通过一个具体的示例对以上示出的递归压缩过程进行详细描述。

[0262] 请参见图9,如图9所示包含三层block的FDMT树中,节点ID为0x00123456的叶子节点的hash值,最初保存在第三层的Tree nodeC中的第5个block的第6个槽位中;假设上述FDMT树的三层block中,均只有一个槽位填充了数据;第二层的Tree nodeB第3个block的第4个槽位填充了数据;第一层的Tree nodeA的第2个block的第2个槽位中填充了数据;

[0263] 那么,按照以上描述的递归压缩的方式,第三层的Tree nodeC的第5个block的第6个槽位中填充的是该叶子节点的hash值;

[0264] 由于Tree nodeC的第5个block只有第6个槽位填充了数据,因此该第6个槽位会被压缩到上一层的Tree nodeB的第3个block中,此时该第3个block中的第4个槽位填充的该第5个block的hash值,将会被替换为该第5个block的hash-ID。此时该hash-ID具体可以是该第5个block的压缩数据“0x01,0x65”和该第5个block的第6个槽位中填充的“hash值”拼接成的二元组;

[0265] 其中,“0x01”表示被压缩的次数为1次,“0x65”表示该槽位在Tree nodeC中的相对位置,即第5个block中的第6个槽位;“0x01,0x65”表示第一次压缩的是第5个block的第6个槽位。

[0266] 由于Tree nodeB的第3个block只有第4个槽位填充了数据,此时需要执行相同的压缩过程,该第4个槽位会被压缩到上一层的Tree nodeA的第1个block中,此时该第1个block中的第2个槽位填充的该第3个block的hash值,将会被替换为该第3个block的hash-ID。此时该hash-ID,具体可以是该第3个block的压缩数据“0x02,0x43,0x65”和该第3个block的第4个槽位填充的“hash值”拼接成的二元组;

[0267] 其中,“0x02”表示被压缩的次数为2次;“0x43”表示该槽位在Tree nodeB中的相对位置,即第3个block中的第4个槽位;“0x02,0x43,0x65”表示的是,第一次压缩的是第5个block的第6个槽位;第二次压缩的是第3个block的第4个槽位。

[0268] 其中,需要说明的是,以上示出的用于表示槽位的位置信息和压缩次数的表示方式,仅为示例性的,并不用于对本说明书的方案进行限定;在实际应用中,本领域技术人员可以灵活的定义槽位的位置信息和压缩次数的表示式,在本说明书中不再进行一一举例。例如,在实际应用中,为了进一步降低上述压缩信息所占用的比特数,可以将压缩信息中的压缩次数和位置信息编码成一个数值。

[0269] 需要说明的是,以上的示例是以上述Tree node采用图4示出的数据结构为例进行了说明,在实际应用中,当上述Tree node采用图5示出的数据结构时,上述递归压缩的过程与以上示例描述的过程类似;

[0270] 例如,当上述Tree node采用图5示出的数据结构时,图9中示出的Tree nodeC的第5个block、Tree nodeB的第3个block都将是子block;此时Tree nodeC的第5个block的上一级block是Tree nodeC的主block;Tree nodeB的第3个bloc的上一级block是Tree nodeB的主block;在这种情况下,可以先将Tree nodeC中的第5个block(子节点)的第6个槽位压缩

到Tree nodeC的主节点,再将Tree nodeC的主节点压缩到Tree nodeB的第3个block(子节点)的第6个槽位,然后重复以上过程,将子节点压缩到上一级的主节点的压缩方式可以参考以上示例,不再赘述。

[0271] 以下以上述区块链数据为区块链账户对应的最新账户状态数据,上述区块链数据的key为区块链账户地址为例,对将上述区块链数据以key-value键值对的形式,写入采用如图4和图5所示的FDMT树的具体过程,进行详细描述。

[0272] 在实现时,接入区块链的用户客户端,可以将交易数据打包成区块链所支持的标准的交易格式,然后发布至区块链;而区块链中的节点设备,可以基于搭载的共识算法与其它节点设备一起,对用户客户端发布至区块链的这些交易进行共识,以此来为区块链产生最新区块;其中,具体的共识过程不再赘述。

[0273] 区块链中的节点设备在执行了目标区块中的交易之后,区块链中与这些被执行的交易相关的目标账户的账户状态,通常也会随之发生变化;因此,节点设备在目标区块中的交易执行完毕后,可以获取上述目标账户发生账户更新后的最新账户状态数据(即目标区块中的交易执行后的账户状态),并将获取到的上述目标账户的最新账户状态数据处理成key-value键值对;其中,该key-value键值对的key为该目标账户的账户地址;该key-value键值对的value为该目标账户的最新账户状态。

[0274] 当将获取到的上述目标账户的最新账户状态数据处理成key-value键值对之后,可以将该key-value键值对转换成为上述FDMT树上的Tree node、Extend node和Leaf node,并将上述Tree node、Extend node和Leaf node的key-value键值对在数据库中进行存储。

[0275] 请参见图10,图10为本说明书示出的一种将账户状态数据写入FDMT状态树的示意图;

[0276] 在图10示出的上述FDMT状态树中,前三层为Tree node,最后一层为叶子节点;其中,上述叶子节点可以采用以上描述的bucket数据桶的存储结构。各层的Tree node均包括主block和16个分别代表不同的16进制字符0-f的子block;并且,各子block进一步包括16个分别代表不同的16进制字符0-f的槽位。

[0277] 假设上述目标账户的账户地址为“a71135125”,最新的账户状态数据为“state1”;此时,该账户地址的字符前缀(Shared nibble)为“a71135”;字符后缀(key-end)为“125”。

[0278] 在将该账户地址“a71135125”的账户状态数据“state1”以key-value键值对的形式写入上述FDMT树时,首先可以在数据库中定位该FDMT树上作为根节点的Tree node(比如根据区块头中填充的根节点的hash来定位根节点);如果是首次数据写入,根节点尚未创建,此时可以创建根节点;再从第一层的根节点Tree node1开始,依次在三层Tree node中确定与该账户地址的字符前缀“a71135”对应的字符槽位。如图10所示,与该账户地址的字符前缀“a71135”对应的字符槽位,具体包括:Tree node1的第11个子block(代表字符a)中的第8个槽位(代表字符7);Tree node2的第2个子block(代表字符1)中的第2个槽位(代表字符1);Tree node3的第4个子block(代表字符3)中的第6个槽位(代表字符5)。

[0279] 在确定以上槽位后,可以在Tree node3的第4个子block的第6个槽位链接的bucket node中,写入由字符后缀“125”和状态数据“state1”构成的数据记录;当然,如果bucket node中已经存在与字符后缀“125”对应的数据记录,表明FMDT树上写入过上述账户

地址“a71135125”的历史账户状态数据对应的value;此时可以将该数据记录中存储的Value更新为“state1”即可。

[0280] 数据写入完成后,可以重新计算该bucket node包含的数据内容的hash值,将该hash值填充到Tree node3的第4个block的第6个槽位中,对该槽位原有的hash值进行更新。

[0281] 进一步的,在将Tree node3的第4个block的第6个槽位中原有的hash值进行更新之后,再重新计算Tree node3的主block包含的数据内容的hash值,将该hash值填充到Tree node2的第2个block的第2个槽位,对该槽位原有的hash值进行更新。

[0282] 然后,在将Tree node2的第2个block的第2个槽位原有的hash值进行更新之后,再重新计算Tree node2的主block包含的数据内容的hash值,将该hash值继续填充到Tree node1(即根节点)的第11个block的第8个槽位,对该槽位原有的hash值进行更新。

[0283] 当根节点Tree node1的第11个block的第8个槽位原有的hash值也更新完成后,此时再重新计算根节点Tree node1的主block包含的数据内容的hash值,并基于该hash值对区块头里存储的FDMT状态树的根节点的hash值进行更新。当区块头里存储的该FDMT状态树的根节点的hash值更新完成后,此时针对该FDMT状态树的更新完成,上述账户地址“a71135512”和对应的账户状态数据“state1”构成的key-value键值对,成功写入到FDMT状态树。

[0284] 请参见图11,假设图10示出的bucket node满足了以上描述的节点分裂条件,并且该bucket node包含的若干条数据记录中的字符后缀存在共享字符前缀“12”;则图10示出的bucket node,可以进一步分裂出一个如图11所示出的用于存储该共享字符前缀“12”的Extend node,具体的分裂过程不再详述。

[0285] 需要强调的是,以上述区块链数据为区块链账户对应的最新账户状态数据为例,仅为示例性的;在实际应用中,当上述区块链数据为以上描述的4种区块链数据的其它类型的区块链数据时,将其写入相应的FDMT树的具体过程,与以上述描述的实现过程类似,在本说明书中不再进行一一赘述。

[0286] 与上述方法实施例相对应,本申请还提供了装置的实施例。

[0287] 与上述方法实施例相对应,本说明书还提供了一种区块链数据存储装置的实施例。

[0288] 本说明书的区块链数据存储装置的实施例可以应用在电子设备上。装置实施例可以通过软件实现,也可以通过硬件或者软硬件结合的方式实现。以软件实现为例,作为一个逻辑意义上的装置,是通过其所在电子设备的处理器将非易失性存储器中对应的计算机程序指令读取到内存中运行形成的。

[0289] 从硬件层面而言,如图12所示,为本说明书的区块链数据存储装置所在电子设备的一种硬件结构图,除了图12所示的处理器、内存、网络接口、以及非易失性存储器之外,实施例中装置所在的电子设备通常根据该电子设备的实际功能,还可以包括其他硬件,对此不再赘述。

[0290] 图13是本说明书一示例性实施例示出的一种区块链数据存储装置的框图。

[0291] 请参考图13,所述区块链数据存储装置130可以应用在前述图12所示的电子设备中,所述装置130包括:

[0292] 获取模块1301,获取待存储的区块链数据的key-value键值对;

[0293] 转换模块1302,将所述待存储的区块链数据的key-value键值对转换成逻辑的树形结构上的根节点、中间节点和叶子节点;所述根节点、中间节点包括主位置和用于存放所述区块链数据的key中的字符的多个子位置;所述主位置包括多个分别与各子位置对应的,用于存放各子位置中的存储内容的hash值的槽位;所述子位置包括多个用于存放所述区块链数据的key中的字符的槽位;所述子位置中的槽位用于存储与该节点链接的下一层节点的hash值;所述根节点、中间节点的hash值为所述根节点、中间节点中的主位置中的存储内容的hash;

[0294] 存储模块1303,将所述根节点、中间节点和叶子节点的key-value键值对存储在数据库中;所述叶子节点、所述中间节点和所述根节点的key-value键值对中,value为节点的存储内容,key为节点的存储内容的hash值。

[0295] 在本实施例中,所述区块链数据的key对应的字符串包括字符前缀和字符后缀;所述根节点、中间节点用于存储所述字符前缀中的字符;所述叶子节点用于存储所述字符后缀和所述区块链数据的Value。

[0296] 在本实施例中,所述装置130还包括:

[0297] 分裂模块,确定所述树形结构上的叶子节点的存储容量是否满足节点分裂条件;如果所述叶子节点的存储容量满足节点分裂条件,从所述叶子节点中分裂出至少一个所述中间节点;所述中间节点用于存储从所述叶子节点存放的字符后缀中分裂出的字符。

[0298] 在本实施例中,所述根节点、所述中间节点存储的字符,为所述根节点、所述中间节点中的各子位置所代表的字符,与所述各子位置中填充了hash值的槽位所代表的字符,进行拼接生成的字符串。

[0299] 在本实施例中,所述根节点、所述中间节点包括的子位置的数量,与所述子位置包括的槽位的数量相同。

[0300] 在本实施例中,所述根节点、所述中间节点包括分别代表不同的16进制字符的16个子位置;所述子位置包括16个分别代表不同的16进制字符的槽位;所述主位置包括与各子位置对应的16个槽位。

[0301] 在本实施例中,所述叶子节点为bucket数据桶;所述bucket数据桶包含若干条数据记录;所述数据记录中存储的数据内容包括所述字符后缀和所述区块链数据的Value。

[0302] 在本实施例中,所述分裂模块:

[0303] 确定所述树形结构上的叶子节点是否发生数据更新;

[0304] 如果所述树形结构上的任一叶子节点发生数据更新,在重新计算所述叶子节点的hash值,并基于该hash值与所述叶子节点上一层的节点进行重新链接之前,进一步确定所述叶子节点的存储容量是否满足节点分裂条件。

[0305] 在本实施例中,所述装置130还包括:

[0306] 合并模块,确定所述树形结构上的中间节点是否发生数据更新;

[0307] 如果所述树形结构上的任一中间节点发生数据更新,在重新计算所述中间节点的hash值,并基于该hash值与该中间节点上一层的节点进行重新链接之前,确定所述中间节点的存储容量是否满足节点合并条件;如果所述中间节点的存储容量满足节点合并条件,进一步将所述中间节点,合并至与该中间节点链接的下一层的叶子节点。

[0308] 在本实施例中,所述数据分裂条件包括:

[0309] 所述叶子节点中存储的数据记录的总数量大于阈值;和/或,所述叶子节点中存储的数据记录的总存储容量大于阈值;

[0310] 相应的,所述数据合并条件包括:

[0311] 所述中间节点中存储的字符的总数量小于或等于阈值;和/或,所述中间节点中存储的字符的总存储容量小于或等于阈值。

[0312] 在本实施例中,所述分裂模块:

[0313] 确定在当前的成块周期内写入所述叶子节点的最新数据记录;

[0314] 将所述最新数据记录从所述叶子节点删除;以及,

[0315] 从所述最新数据记录所包含的字符后缀中分裂出字符前缀,并创建用于存储分裂出的所述字符前缀的至少一中间节点,以及用于存储分裂出所述字符前缀之后的所述最新数据记录的至少一叶子节点。

[0316] 在本实施例中,所述分裂模块:

[0317] 确定所述叶子节点存储的若干条数据记录所包含的字符后缀之间的共享字符前缀;

[0318] 将所述共享字符前缀从所述若干条数据记录中删除,并创建用于存储所述共享字符前缀的至少一中间节点。

[0319] 在本实施例中,所述合并模块:

[0320] 确定与所述中间节点链接的下一层的叶子节点;

[0321] 将该中间节点存储的字符作为所述字符后缀写入所述叶子节点。

[0322] 上述实施例阐明的系统、装置、模块或单元,具体可以由计算机芯片或实体实现,或者由具有某种功能的产品来实现。一种典型的实现设备为计算机,计算机的具体形式可以是个人计算机、膝上型计算机、蜂窝电话、相机电话、智能电话、个人数字助理、媒体播放器、导航设备、电子邮件收发设备、游戏控制台、平板计算机、可穿戴设备或者这些设备中的任意几种设备的组合。

[0323] 在一个典型的配置中,计算机包括一个或多个处理器(CPU)、输入/输出接口、网络接口和内存。

[0324] 内存可能包括计算机可读介质中的非永久性存储器,随机存取存储器(RAM)和/或非易失性内存等形式,如只读存储器(ROM)或闪存(flash RAM)。内存是计算机可读介质的示例。

[0325] 计算机可读介质包括永久性和非永久性、可移动和非可移动媒体可以由任何方法或技术来实现信息存储。信息可以是计算机可读指令、数据结构、程序的模块或其他数据。计算机的存储介质的例子包括,但不限于相变内存(PRAM)、静态随机存取存储器(SRAM)、动态随机存取存储器(DRAM)、其他类型的随机存取存储器(RAM)、只读存储器(ROM)、电可擦除可编程只读存储器(EEPROM)、快闪记忆体或其他内存技术、只读光盘只读存储器(CD-ROM)、数字多功能光盘(DVD)或其他光学存储、磁盒式磁带、磁盘存储、量子存储器、基于石墨烯的存储介质或其他磁性存储设备或任何其他非传输介质,可用于存储可以被计算设备访问的信息。按照本文中的界定,计算机可读介质不包括暂存电脑可读媒体(transitory media),如调制的数据信号和载波。

[0326] 还需要说明的是,术语“包括”、“包含”或者其任何其他变体意在涵盖非排他性的

包含,从而使得包括一系列要素的过程、方法、商品或者设备不仅包括那些要素,而且还包括没有明确列出的其他要素,或者是还包括为这种过程、方法、商品或者设备所固有的要素。在没有更多限制的情况下,由语句“包括一个……”限定的要素,并不排除在包括所述要素的过程、方法、商品或者设备中还存在另外的相同要素。

[0327] 上述对本说明书特定实施例进行了描述。其它实施例在所附权利要求书的范围内。在一些情况下,在权利要求书中记载的动作或步骤可以按照不同于实施例中的顺序来执行并且仍然可以实现期望的结果。另外,在附图中描绘的过程不一定要求示出的特定顺序或者连续顺序才能实现期望的结果。在某些实施方式中,多任务处理和并行处理也是可以的或者可能是有利的。

[0328] 在本说明书一个或多个实施例使用的术语是仅仅出于描述特定实施例的目的,而非旨在限制本说明书一个或多个实施例。在本说明书一个或多个实施例和所附权利要求书中所使用的单数形式的“一种”、“所述”和“该”也旨在包括多数形式,除非上下文清楚地表示其他含义。还应当理解,本文中使用的术语“和/或”是指并包含一个或多个相关联的列出项目的任何或所有可能组合。

[0329] 应当理解,尽管在本说明书一个或多个实施例可能采用术语第一、第二、第三等来描述各种信息,但这些信息不应限于这些术语。这些术语仅用来将同一类型的信息彼此区分开。例如,在不脱离本说明书一个或多个实施例范围的情况下,第一信息也可以被称为第二信息,类似地,第二信息也可以被称为第一信息。取决于语境,如在此所使用的词语“如果”可以被解释成为“在……时”或“当……时”或“响应于确定”。

[0330] 以上所述仅为本说明书一个或多个实施例的较佳实施例而已,并不用以限制本说明书一个或多个实施例,凡在本说明书一个或多个实施例的精神和原则之内,所做的任何修改、等同替换、改进等,均应包含在本说明书一个或多个实施例保护的范围之内。

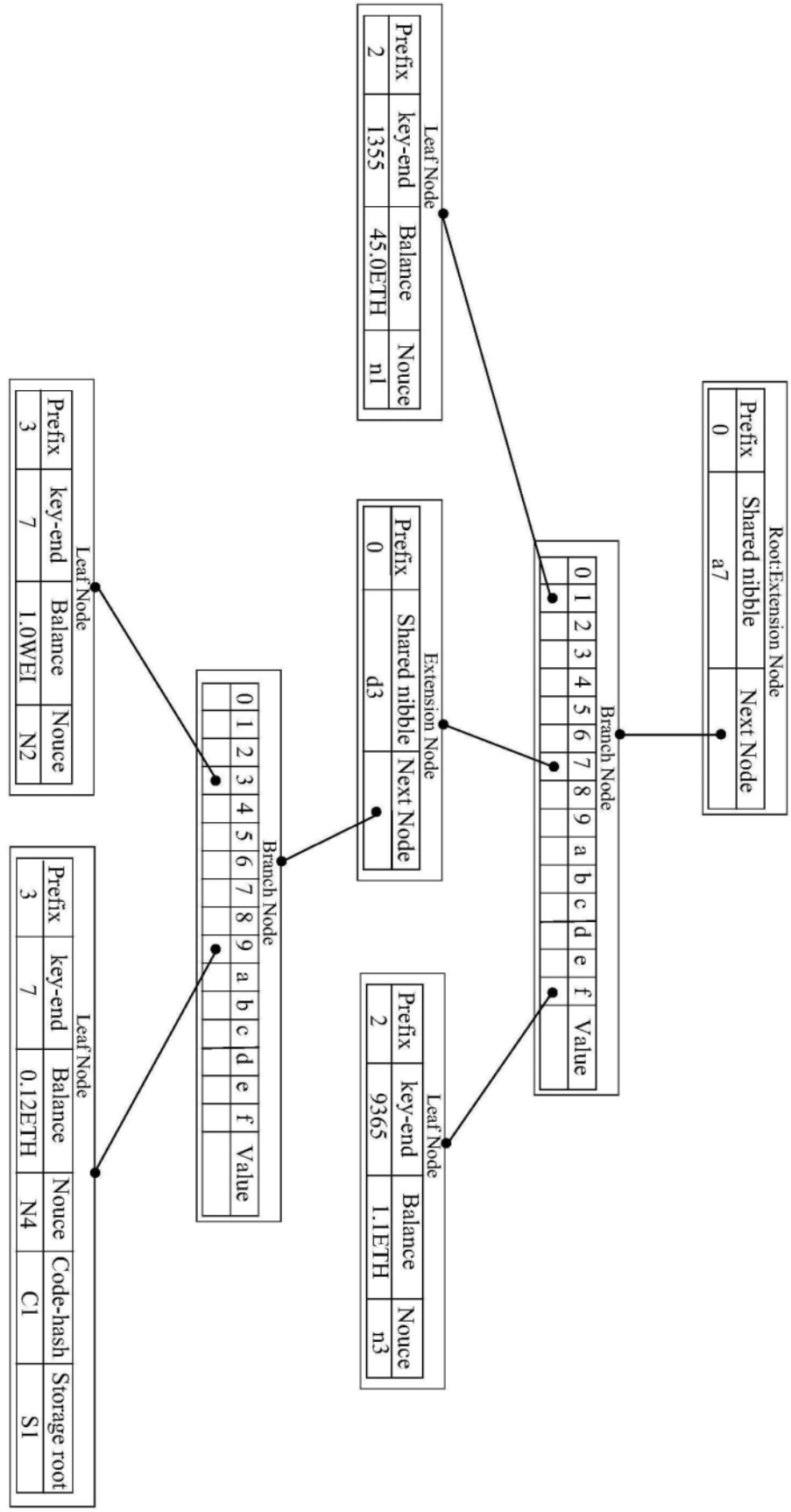

Root:Extension Node
Prefix
Shared nibble
Next Node
0
a7
Branch Node
0 1 2 3 4 5 6 7 8 9 a b c d e f Value
Leaf Node
Prefix
key-end
Balance
Nouce
2
1355
45.0ETH
n1
Extension Node
Prefix
Shared nibble
Next Node
0
d3
Branch Node
0 1 2 3 4 5 6 7 8 9 a b c d e f Value
Leaf Node
Prefix
key-end
Balance
Nouce
3
7
1.0WEI
N2
Leaf Node
Prefix
key-end
Balance
Nouce
2
9365
1.1ETH
n3
Leaf Node
Prefix
key-end
Balance
Nouce
Code-hash
Storage root
3
7
0.12ETH
N4
C1
S1

图1

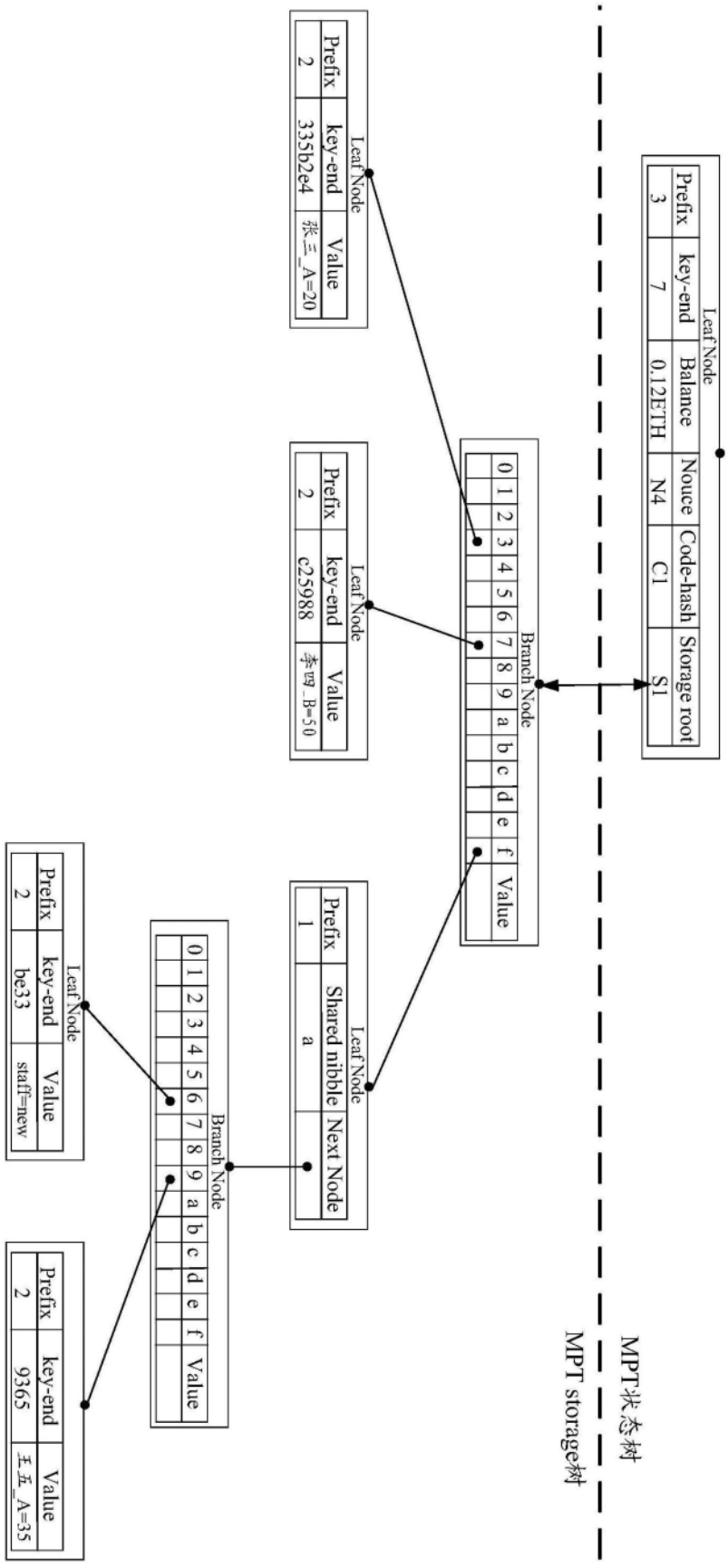

MPT状态树
MPT storage树
Leaf Node
Prefix
key-end
Balance
Nouce
Code-hash
Storage root
3
7
0.12ETH
N4
C1
S1
Branch Node
0 1 2 3 4 5 6 7 8 9 a b c d e f
Value
Leaf Node
Prefix
key-end
Value
2
335b2e4
张三_A=20
Leaf Node
Prefix
key-end
Value
2
c25988
李四_B=50
Leaf Node
Prefix
Shared nibble
Next Node
1
a
Branch Node
0 1 2 3 4 5 6 7 8 9 a b c d e f
Value
Leaf Node
Prefix
key-end
Value
2
be33
staff=new
Leaf Node
Prefix
key-end
Value
2
9365
王五_A=35

图2

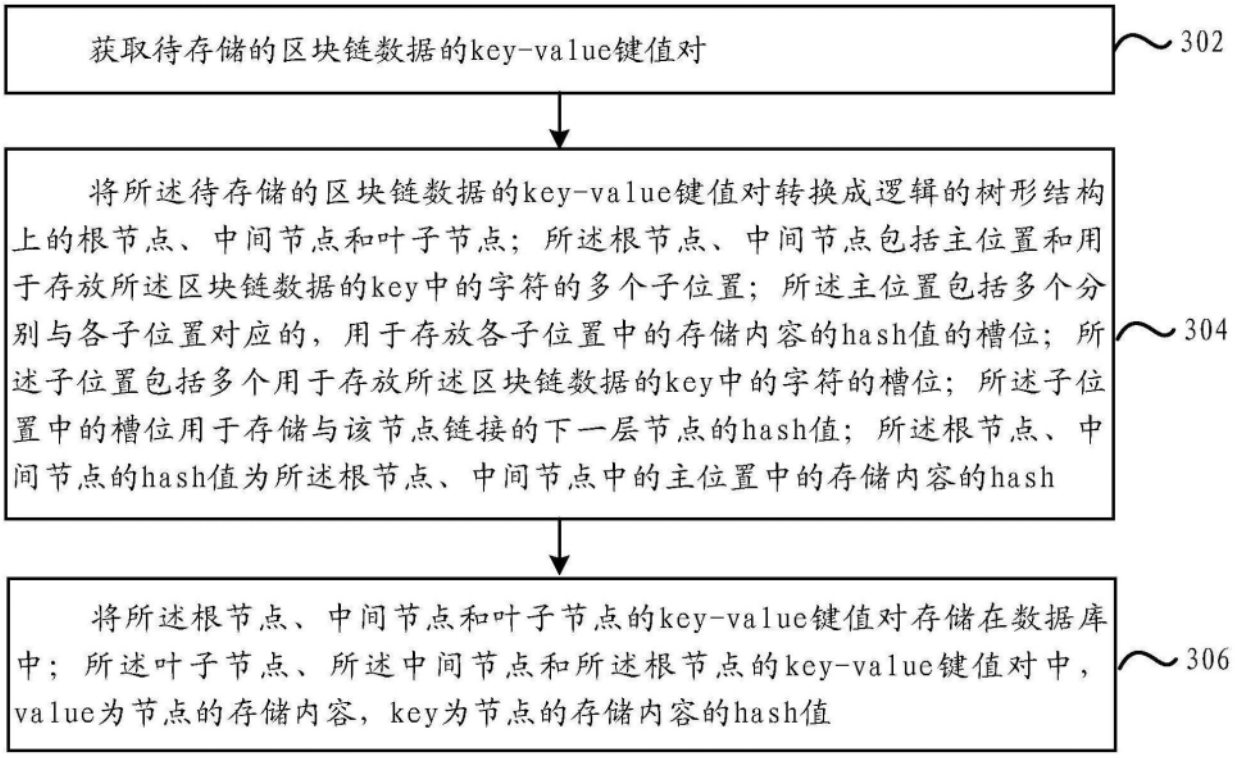

获取待存储的区块链数据的key-value键值对
302
将所述待存储的区块链数据的key-value键值对转换成逻辑的树形结构上的根节点、中间节点和叶子节点；所述根节点、中间节点包括主位置和用于存放所述区块链数据的key中的字符的多个子位置；所述主位置包括多个分别与各子位置对应的，用于存放各子位置中的存储内容的hash值的槽位；所述子位置包括多个用于存放所述区块链数据的key中的字符的槽位；所述子位置中的槽位用于存储与该节点链接的下一层节点的hash值；所述根节点、中间节点的hash值为所述根节点、中间节点中的主位置中的存储内容的hash
304
将所述根节点、中间节点和叶子节点的key-value键值对存储在数据库中；所述叶子节点、所述中间节点和所述根节点的key-value键值对中，value为节点的存储内容，key为节点的存储内容的hash值
306

图3

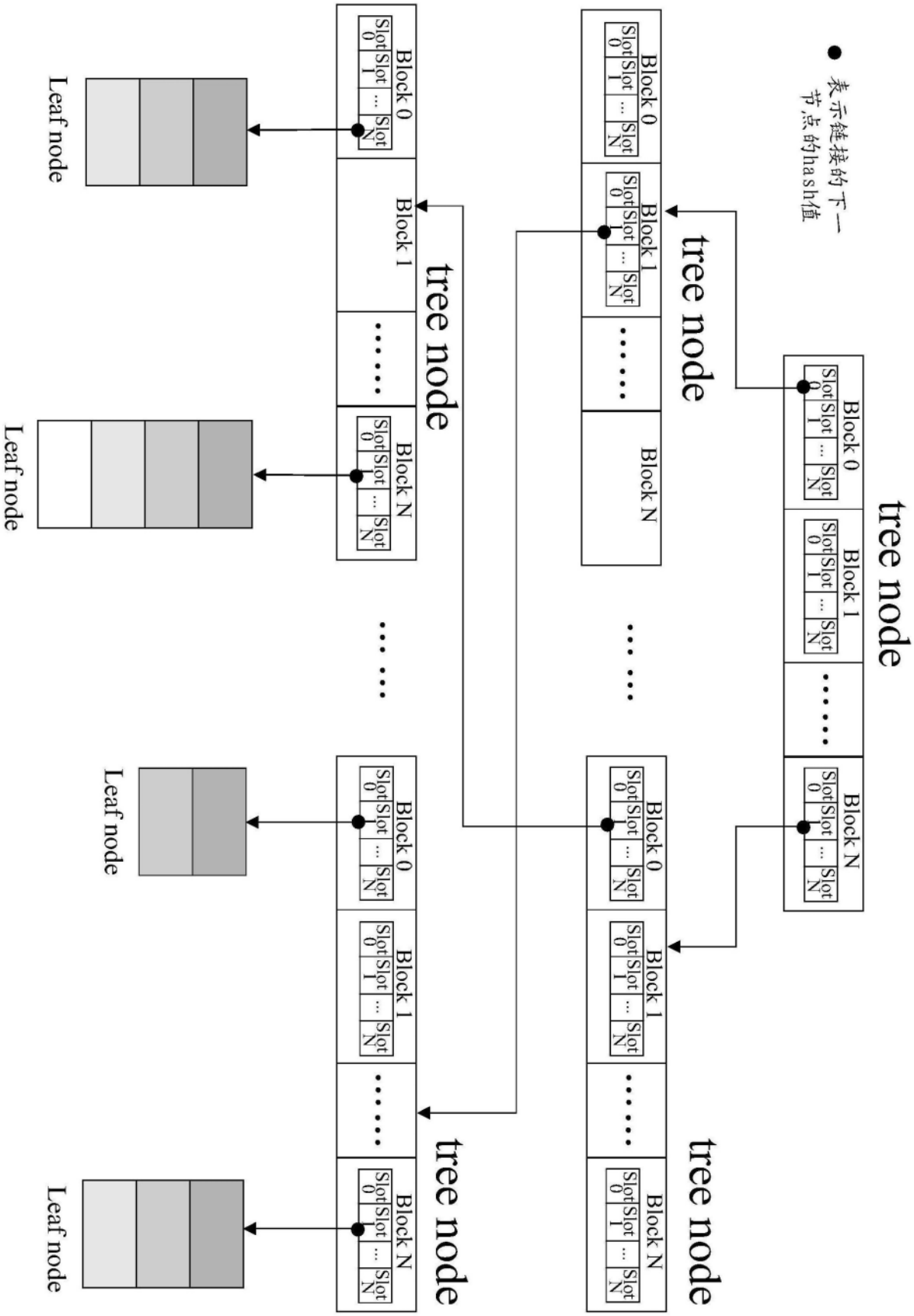

表示链接的下一节点的hash值
tree node
Block 0
Block 1
Block N
Slot 0
Slot 1
Slot N
Leaf node

图4

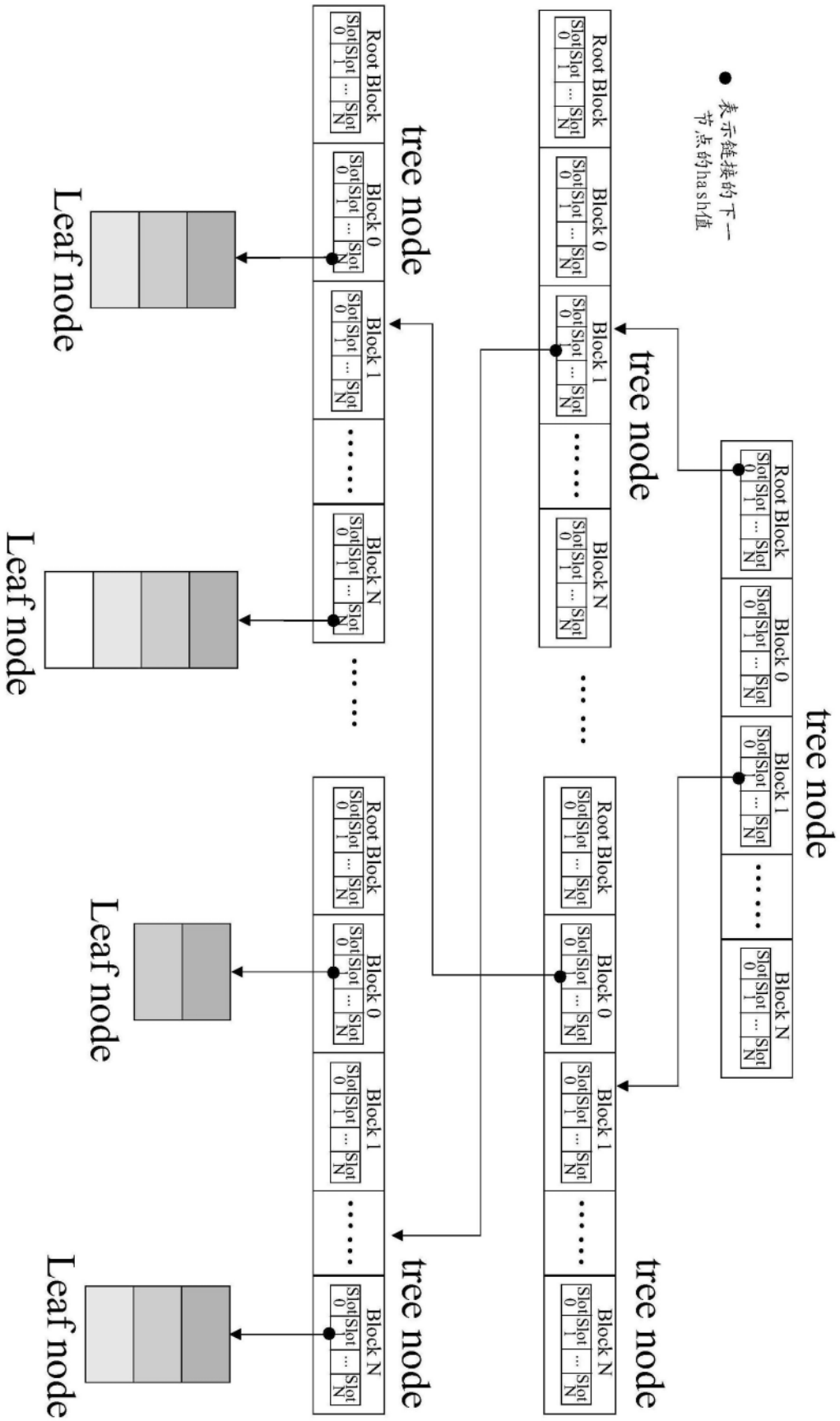

表示链接的下一节点的hash值
tree node
Root Block
Block 0
Block 1
Block N
Slot 0
Slot 1
Slot N
Leaf node

图5

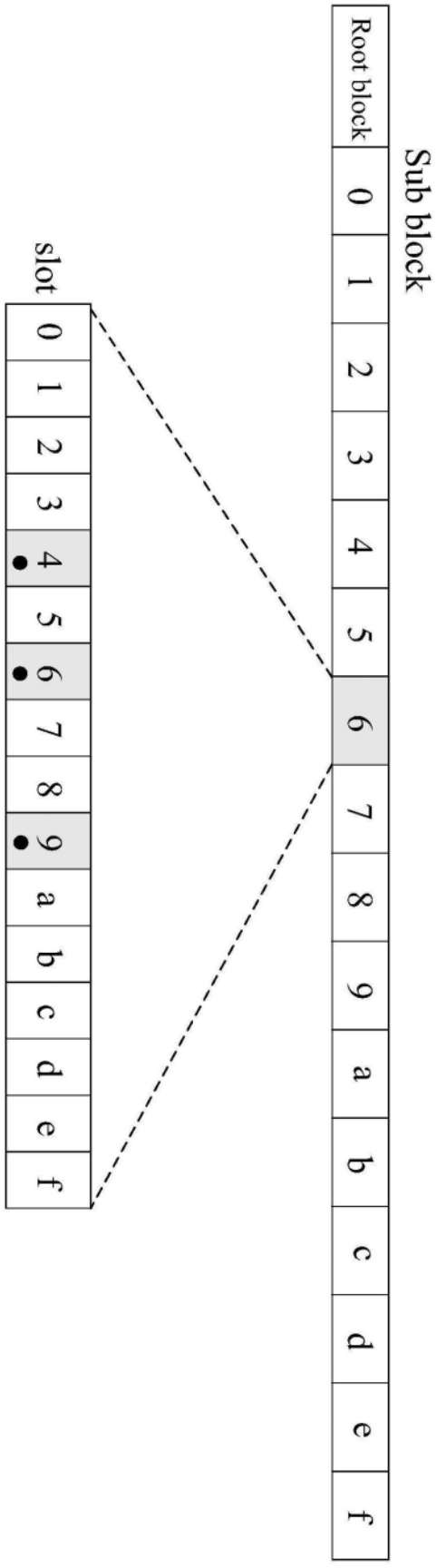


图6

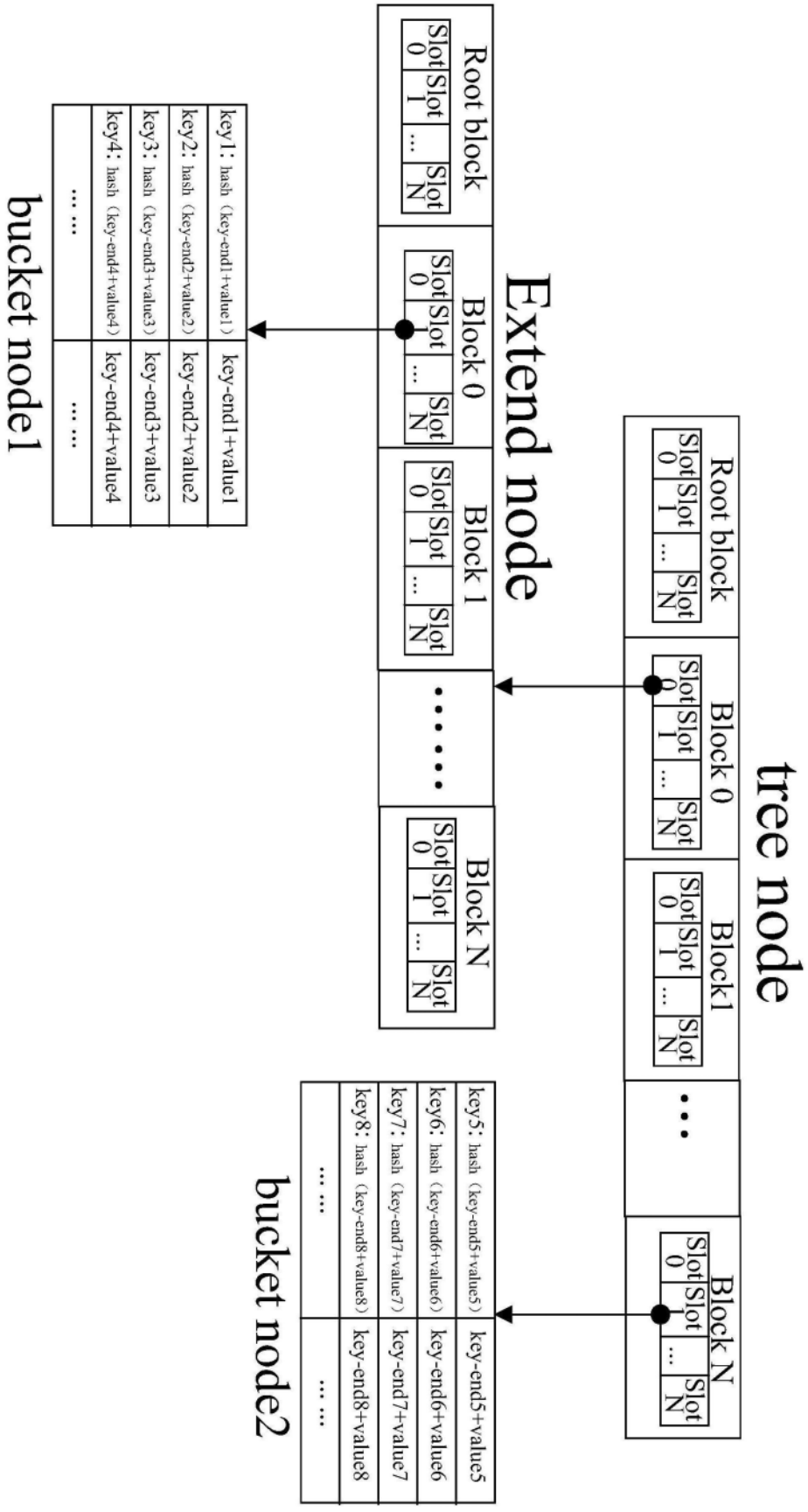

tree node
Root block
Slot 0
Slot 1
...
Slot N
Block 0
Block1
Block N
Extend node
Root block
Block 0
Block 1
Block N
bucket node1
key1: hash（key-end1+value1）
key-end1+value1
key2: hash（key-end2+value2）
key-end2+value2
key3: hash（key-end3+value3）
key-end3+value3
key4: hash（key-end4+value4）
key-end4+value4
bucket node2
key5: hash（key-end5+value5）
key-end5+value5
key6: hash（key-end6+value6）
key-end6+value6
key7: hash（key-end7+value7）
key-end7+value7
key8: hash（key-end8+value8）
key-end8+value8

图7

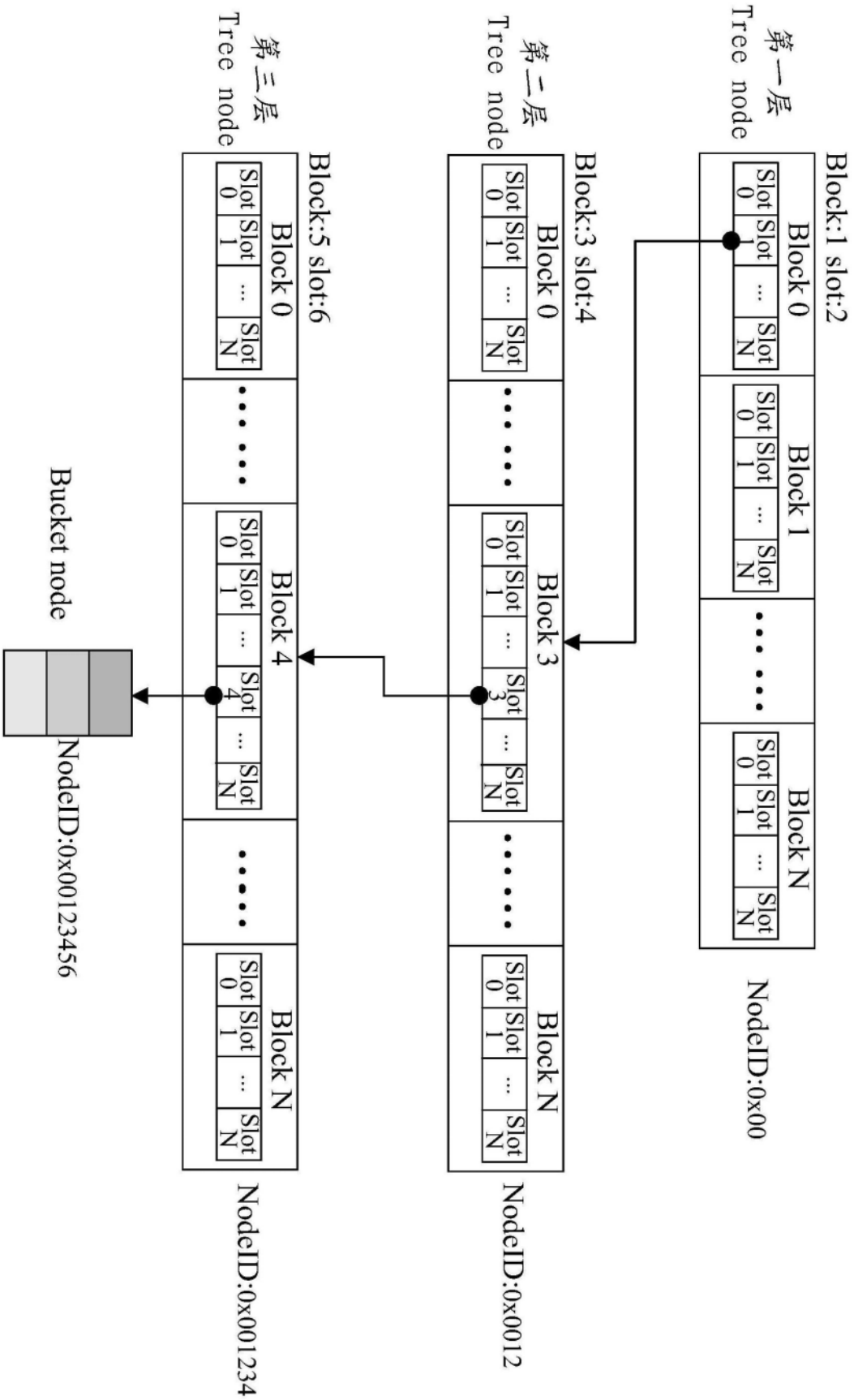

第一层
Tree node
Block:1 slot:2
Block 0
Block 1
Block N
Slot 0
Slot 1
Slot N
NodeID:0x00
第二层
Tree node
Block:3 slot:4
Block 3
NodeID:0x0012
第三层
Tree node
Block:5 slot:6
Block 4
NodeID:0x001234
Bucket node
NodeID:0x00123456

图8

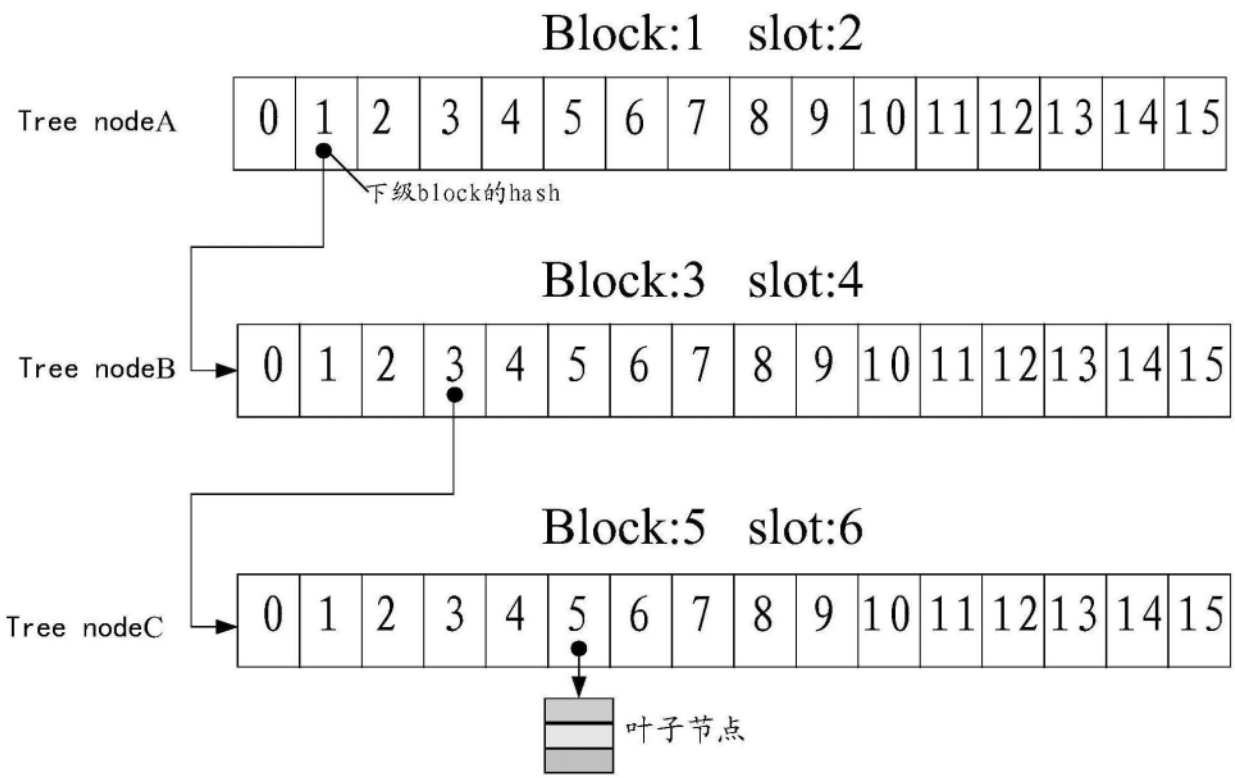

Block:1 slot:2
Tree nodeA
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
下级block的hash
Block:3 slot:4
Tree nodeB
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Block:5 slot:6
Tree nodeC
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
叶子节点

图9

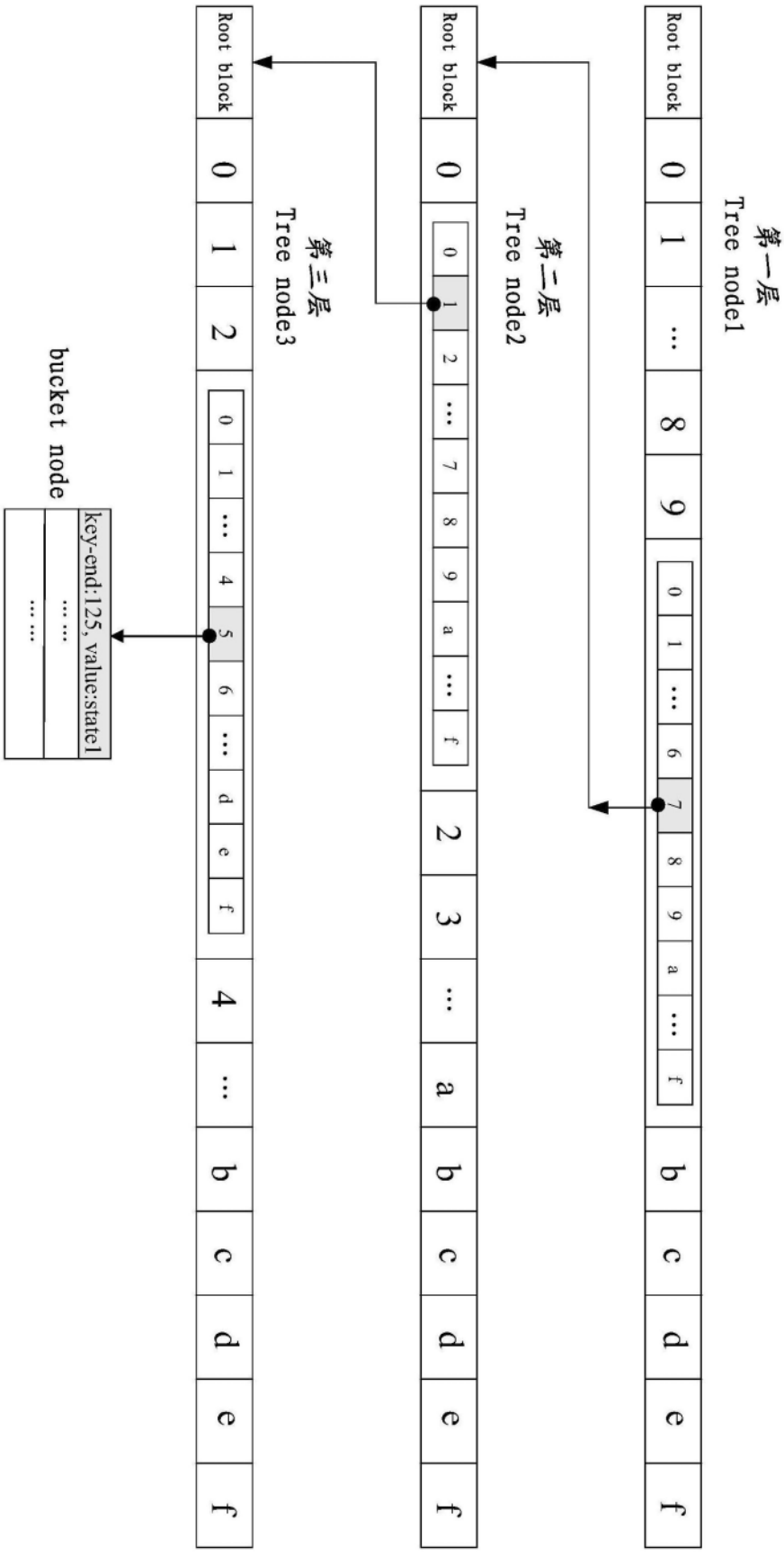

第一层
Tree node1
Root block
第二层
Tree node2
第三层
Tree node3
bucket node
key-end:125, value:state1

图10

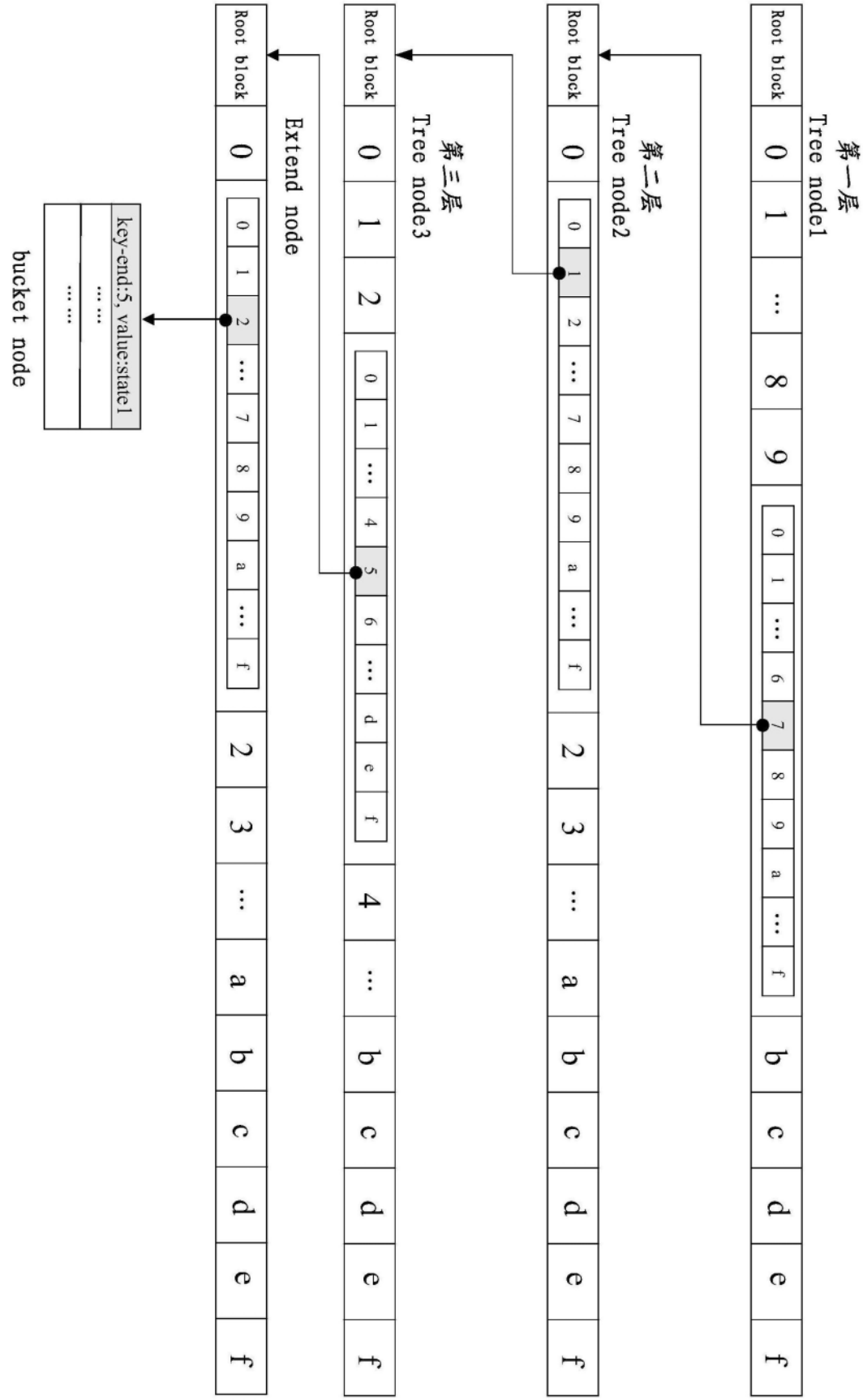

第一层
Tree node1
Root block
第二层
Tree node2
第三层
Tree node3
Extend node
key-end:5, value:state1
bucket node

图11

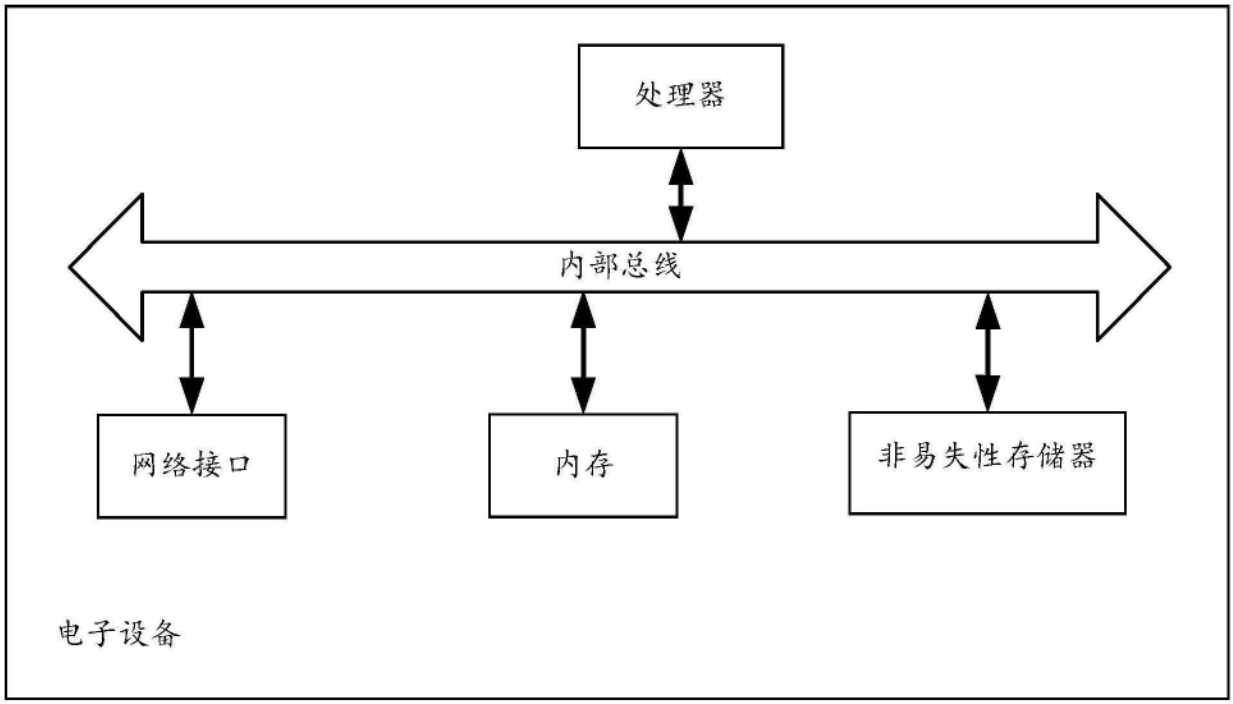

处理器
内部总线
网络接口
内存
非易失性存储器
电子设备

图12

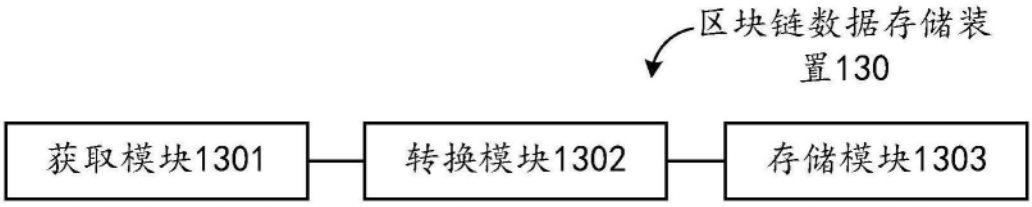

区块链数据存储装置130
获取模块1301
转换模块1302
存储模块1303

图13