软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2020,31(3):680–694 [doi: 10.13328/j.cnki.jos.005914]

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

面向区块链的高效物化视图维护和可信查询*

蔡 磊, 朱燕超, 郭庆兴, 张 召, 金澈清

(华东师范大学 数据科学与工程学院,上海 200062)

通讯作者: 张召, E-mail: zhzhang@dase.ecnu.edu.cn

摘 要: 区块链具有去中心化、不可篡改和可追溯等特性,可应用于金融、物流等诸多行业.由于所有交易数据按照交易时间顺序存储在各个区块,相同类型的交易数据通常会散布在诸多区块之中,降低了面向历史区块的追溯查询的处理效率.索引构建和物化视图是提升查询性能的两种典型方法,但当待处理数据分布于多个区块时,使用索引无法改善 I/O 访问效率,而物化视图可有效应对这个问题.然而,由于区块链系统的特点明显区别于关系数据库,传统的面向关系数据库的物化视图技术无法被直接应用到区块链之中.鉴于此,首次提出一种面向区块链的高效物化视图机制,具有如下特征:(1) 将视图维护操作与共识过程同时执行,降低该操作对系统性能的影响;(2) 使用字典树加快以区块为单位的多物化视图维护进程;(3) 以默克尔验证的方式确保物化结果不被恶意篡改,进而确保查询结果可信.所提出的物化视图维护机制已经被集成到一个区块链系统中,并通过实验来验证该机制的高效性.

关键词: 物化视图;区块链;增量更新;视图维护;默克尔树

中图法分类号: TP18

中文引用格式: 蔡磊,朱燕超,郭庆兴,张召,金澈清.面向区块链的高效物化视图维护和可信查询.软件学报,2020,31(3):680–694. http://www.jos.org.cn/1000-9825/5914.htm

英文引用格式: Cai L, Zhu YC, Guo QX, Zhang Z, Jin CQ. Efficient materialized view maintenance and trusted query for blockchain. Ruan Jian Xue Bao/Journal of Software, 2020,31(3):680–694 (in Chinese). http://www.jos.org.cn/1000-9825/5914.htm

Efficient Materialized View Maintenance and Trusted Query for Blockchain

CAI Lei, ZHU Yan-Chao, GUO Qing-Xing, ZHANG Zhao, JIN Che-Qing

(School of Data Science and Engineering, East China Normal University, Shanghai 200062, China)

Abstract: The blockchain system is favored by many fields, such as finance and logistics due to several unique properties, including decentralized architecture, data immutability and data traceability. Transactions belonging to the same type are commonly distributed in massive blocks because all transactions are stored in chronological order of transaction committing, which lowers the efficiency to process tracing queries where a huge number of historical blocks are involved. Although indexing and materialized view are two typical ways to boost query performance, indexing cannot lower the I/O cost if the data to be processed are widely distributed in the system. Fortunately, materialized view suits for this scenario well. Furthermore, as traditional materialized view technologies for RDBMS cannot be directly adopted to blockchain due to significant difference between them, a set of materialized view mechanisms is firstly proposed for blockchain with the following properties: (1) To lower the impact to the system, the view maintenance operation is executed in parallel with consensus process; (2) Trie-Tree is used to speed up multi-materialized view maintenance process in blocks; (3) the query results is guaranteed credible by ensuring the materialized results not falsified with Merkle verification. After integrating the proposed materialized view maintenance mechanism into a blockchain system, experimental results show that the proposed method is convenient and efficient.

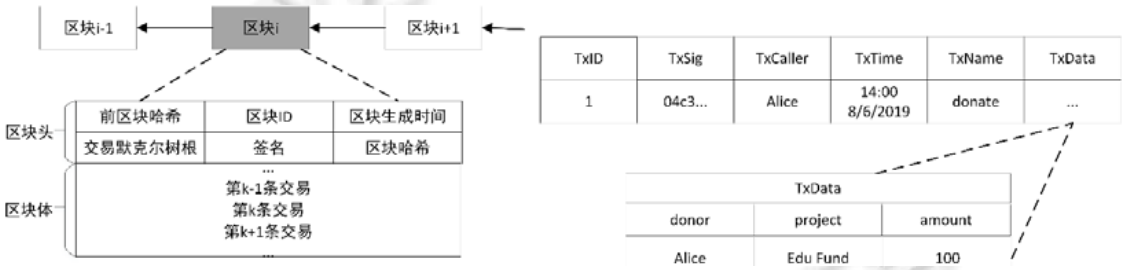
* 基金项目: 国家自然科学基金(U1811264, U1911203, 61972152, 61532021)
Foundation item: National Natural Science Foundation of China (U1811264, U1911203, 61972152, 61532021)
本文由人工智能赋能的数据管理、分析与系统专刊特约编辑李战怀教授、于戈教授和杨晓春教授推荐.
收稿时间: 2019-08-15; 修改时间: 2019-09-10, 2019-11-25; 采用时间: 2019-12-18; jos 在线出版时间: 2020-01-10
CNKI 网络优先出版: 2020-01-10 13:35:01, http://kns.cnki.net/kcms/detail/11.2560.TP.20200110.1334.014.html

Key words: materialized view; blockchain; incremental update; view maintenance; Merkle tree

作为一种在不可信环境中由多方共同维护的分布式账本,区块链已被应用在金融、物流等领域.然而,当前的区块链技术在数据管理方面存在着无法支持复杂查询、查询接口单一和响应太慢等局限性.

为了弥补现有区块链平台在数据管理性能方面的不足,一些课题组尝试融合数据管理和区块链技术,例如ChainSQL[1],BigchainDB[2],FlureeDB[3],SEBDB[4]等.ChainSQL 将关系数据库和 Ripple 区块链网络相结合,借助关系数据库的访问接口为链上数据访问提供便利,并使用区块链技术来提升数据异地多活的容错能力.BigchainDB 是一种添加了区块链特征的数据库,它集成了 MongoDB 和 Tendermint 区块链网络.FlureeDB 将区块链技术中不可篡改和高容错特性集成到图形数据库.以上工作尽管提供了更丰富的查询功能,但是并未聚焦查询性能优化.SEBDB[4]在面向传统行业的联盟链背景下,为区块数据添加了关系语义,将每种交易类型转化为一张关系表,将该交易类型的参数转化为相应关系表的列,从而有效融合关系语义和区块数据.

在区块链中,区块包括区块头(block header)和区块体(block body),如图 1(a)所示:区块头由前一个区块的哈希值、区块 ID、区块生成时间、交易默克尔树根、签名和本区块的哈希值组成;区块体包含多个交易,每个交易由交易 ID(TxID)、交易签名(TxSig)、智能合约调用者(TxCaller)、交易时间(TxTime)、交易表名(TxName)和表数据(TxData)组成.图 1(b)为一个交易示例,表示用户 Alice 调用智能合约中的 donate 函数向教育基金项目 Edu Fund 捐助了 100 元.在 SEBDB[4]中,交易表名相当于关系数据库中的表名,表数据包含若干列,相当于关系数据库中的一行记录.



(a) 区块结构 (b) 交易结构

Fig.1 Structure of block and transaction

图 1 区块和交易结构

由于交易数据在区块中以交易提交的时间顺序依次存储,属于同一关系表的交易数据往往会分散在不连续的多个区块中,这会降低针对区块数据的查询的执行效率.SEBDB[4]通过在区块链上建立 B+树、位图等索引提高查询性能,但是当查询涉及到多个区块时,使用索引无法持续降低 I/O 访问开销,因而无法进一步改善查询处理效率.在数据库中,建立物化视图是另一种提高查询效率的方法,物化视图通过物化查询结果来提高特定查询的处理效率.因此,如何在区块链中使用物化视图也值得思考.

尽管物化视图已被研究多年,如何维护物化视图仍旧是一个开放问题.在关系数据库中,增量刷新的物化视图维护策略可划分为立即维护[5]和延迟维护[6]两大类.立即维护策略的优点是实现较为简单,在单数据源下不存在一致性问题;然而该策略将物化视图维护过程嵌入到更新事务之中,延长了更新事务的提交时间,这在高并发的情况下易发生死锁.延迟维护策略解耦合视图维护和更新事务,在 OLTP 场景下,可以通过合并无关更新[6]的方法缩短视图维护时间;但是此策略存在一致性问题,若视图未更新完毕则不可使用.在延迟维护策略的诸多实现方法中,按需维护[7]较为常见,即:等待查询到来之后,只维护与查询相关的物化视图.由此可见,各种策略的优缺点显著,如何合理选择视图维护策略非常重要.

面对被赋予了关系语义的区块数据,采用关系数据库中普遍使用的建立物化视图的方式来提升查询性能是一种可行的方法.在区块链中,系统查找某张表的数据需要扫描所有的区块,当数据量庞大时,即使扫描索引

也会产生非常大的开销.鉴于此,如果将物化视图运用于区块链,将会优化查询的处理效率.

然而,关系数据库与区块链系统在存储模型和更新操作上有显著不同,区块链系统以区块为单位进行更新,单个区块包含多条交易,并且区块链系统中的交易需要通过共识来完成.区块链系统和关系数据库相比,在区块链上建立、维护物化视图将面临以下 3 个挑战.

(1) 如何选择物化视图的写入时机.区块链的写入性能受到分布式共识、智能合约执行限制,而物化视图的维护开销对系统的性能带来额外影响.因此,如何合理选择视图维护的时机来降低视图维护对系统整体性能的影响,是一个需要考虑的问题;

(2) 如何以区块为单位维护视图.区块是区块链的基本数据追加单位,各区块包含多种类型的交易,对于一个区块可能需要同时维护多个视图.因此,设计的方案必须支持批量的物化视图维护,并且使得物化视图维护的开销尽可能小;

(3) 如何确保查询结果的可信性.由于数据上链需要经过较为昂贵的共识过程,为了提升查询效率,物化视图并不保存在区块链上.与此同时,将物化视图保存在本地会面临数据被篡改的风险,需要实施相应措施来确保查询结果可信.

针对以上挑战,本文的主要贡献包括:首次将物化视图运用于区块链,提出了一种视图维护和共识过程并行的方法,降低物化视图的维护开销.区块链的共识过程主要消耗网络带宽,在此期间,CPU 和 I/O 资源消耗相对较少,而视图维护过程却主要消耗 CPU 和 I/O 资源.因此,将视图维护和共识过程并行执行可减少视图维护对写入性能的影响.提出了基于字典树的方法,以区块为单位批量维护视图,并且支持多种维护策略.本文使用字典树作为索引加快查找不同表名的更新记录,可对相同表名的更新记录只进行一次视图维护操作.并且本文支持闲时维护和按需维护的维护策略.提出了基于默克尔树的查询结果验证方法,确保结果可信.为物化视图构造默克尔树.当查询使用物化视图时,系统扫描物化视图建立默克尔树,并与预先保存的默克尔树根进行比较,以此确保物化视图的正确性与完整性.

本文第 1 节说明本文的系统架构.第 2 节阐述物化视图的维护时机.第 3 节描述物化视图维护的具体过程.第 4 节详述基于默克尔树的查询验证方法.第 5 节展示实验结果.第 6 节回顾与本文相关的研究工作.最后,第 7 节给出简短总结.

1 系统架构

本文原型系统架构如图 2 所示,包括应用层、查询层、存储层、共识层和网络层:应用层包括查询 API、访问控制和智能合约;查询层具有查询引擎,负责对查询的解析、优化、执行,包括物化视图的维护;存储层包括区块链和链下数据(物化视图、索引等);共识层负责交易的共识,运用的协议为 PBFT[8];最后,网络层采用 Gossip 协议.本文专注于查询层、存储层和共识层:物化视图的更新记录来自于共识返回的结果,查询层负责物化视图的维护工作,并将更新后的物化视图存于存储层.此外,查询的结果来源于存储层的区块数据或物化视图.

Fig.2 System architecture

图 2 系统架构

在此架构下,面向添加了关系语义的联盟链,我们首次提出一种高效的物化视图维护方法以提高查询的效率,并且提出一种验证方法来确保查询结果的正确性.当系统应用层接收到客户端发来的智能合约调用请求时,查询层处理请求,然后调用智能合约产生一条新的交易,交易通过共识后被打包进区块保存在区块链中.另一方面,查询层获取共识成功的交易进行视图维护,视图维护完毕后,将更新后的物化视图存于存储层的磁盘中.而当系统接收到客户端的查询请求时,查询层判断该请求是否可以运用物化视图:若可以,则获取物化视图数据返回给应用层;若不能使用物化视图,则需扫描区块链查找结果.接下来我们将回答 3 个问题:何时进行物化视图的维护、如何进行物化视图的维护以及如何保证物化视图结果的正确性.

2 物化时机的选择

在区块链中,交易在系统中达成共识需要在各节点之间进行网络通信.比如,广泛应用于联盟链的协议 PBFT 需要进行 3 轮网络交互,耗费时间较长.在共识阶段主要消耗网络资源,而 CPU 和 I/O 资源相对空闲.因此,视图维护与共识阶段可以并行执行.当上一轮共识的数据已经有效时,系统在执行新一轮交易共识的过程中对上一轮产生的数据进行物化操作.这样,物化视图维护和共识过程同时进行,互不干扰,从而大幅度减少视图维护对系统整体性能的影响.

该方案需考虑两种情况:一是视图的维护时间相对于共识时间比较少;二是视图的维护时间大于共识时间,即上轮共识通过的交易相关视图不能在此轮共识时间内完成.假设每个区块中的交易平均属于 k 张表,每张表的视图个数为 n,每张表的的平均视图维护时间为 $t_{mvi}(i\in[1,n])$,共识的时间为 T,那么需要考虑:

$$k\cdot(t_{mv1}+t_{mv2}+\ldots+t_{mvn})\leqslant T \tag{1}$$

或

$$k\cdot(t_{mv1}+t_{mv2}+\ldots+t_{mvn})>T \tag{2}$$

如果维护时间满足公式(1),如图 3 所示,系统直接对上一轮共识通过的数据进行视图维护;如果满足公式(2),则将剩余未维护的记录暂存在缓冲区中,等待 CPU 空闲进行闲时维护.以上方法中,正在进行维护更新的物化视图暂时不可用,因为最新的数据还未更新,查询得到的结果不完整.因此,当有可以引用物化视图的查询到来时,系统采用按需维护策略,优先维护查询相关的物化视图以快速响应查询.对于视图维护中的一致性保证,我们将在第 4.2 节详细叙述.

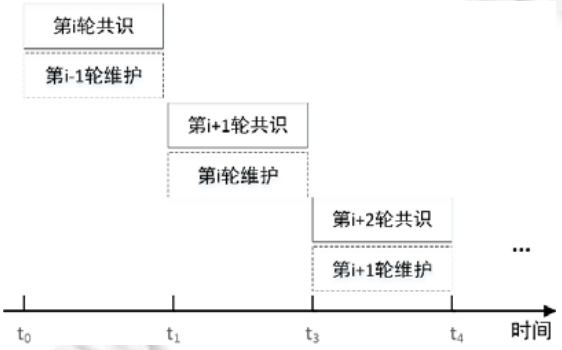


Fig.3 Maintenance timing of materialized views

图 3 物化视图的维护时机

3 视图维护过程

3.1 维护视图的基本步骤

在维护物化视图时,查询层的查询引擎获取共识成功的交易的表名、表数据、所属区块 ID 进行增量维护.系统创建的物化视图与基本表类似,只是在类型上加以区别.图 4 展示了物化视图维护的整体流程,该流程包括

4 个步骤:① 从共识模块获取已完成共识的多个交易,创建增量记录;② 查询层将增量记录按照表名分组,并存储在增量记录缓冲区中;③ 根据增量记录创建视图维护任务,当查询到来时,进行视图维护任务的调度;④ 根据物化视图的查询表达式计算更新的视图行集,再将新增的视图行集添加到视图中.

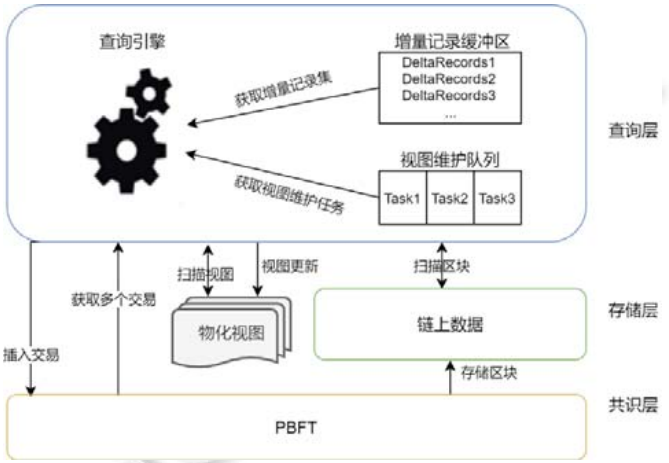


Fig.4 Maintenance process of materialized views

图 4 物化视图的维护过程

以下详细介绍相关步骤.

- 首先,在第①步骤中,需要创建增量记录.

增量记录是一个用于存储记录更新信息的三元组 *DeltaRecord*(*Row*,*TableName*,*BlockID*),其中,*Row* 是交易的表数据部分,*TableName* 是交易的表名,*BlockID* 是当前交易所在区块的区块高度.在针对交易进行新一轮共识时,系统获取上一轮各交易的表名和表数据字段,结合新区块的 ID 创建增量记录.该步骤可以线性复杂度执行完毕.

- 其次,为了提升数据检索效率,步骤②将增量数据分组后存放在增量数据缓冲区之中.

由于增量记录数目较多,可将其按照表名进行分组,分别构建增量记录集(*DeltaRecords*),并暂存于一个常驻内存的增量记录缓冲区(*DeltaRecordsCache*).为了提升增量记录的检索效率,采用字典树[9]对表名进行索引.换言之,该字典树包含了所有表名以及这些表名的部分前缀字符串.若字典树中某节点(包括叶节点和非叶节点)对应一个表名,则有一个指针指向与该表名相对应的增量记录集.图 5 显示了一个采用字典树索引增量记录的案例.在增量记录缓冲区中共有 3 个增量记录集,其中,视图维护任务 $Task_1$ 维护表名为 *donate* 的增量记录 *d*1,*d*2,*d*3,$Task_2$ 维护表名为 *transfer* 的增量记录 *t*1,*t*2,*t*3.当一个新的增量记录被添加到增量缓冲区时,我们在字典树上查找该增量记录的表名:若此增量记录表名不存在,则系统为该表名创建新的节点;若存在,则将此增量记录插入到叶子节点所指向的增量记录集.如果某个增量记录相关的所有视图维护完毕,则删除该增量记录.

- 然后,步骤③基于增量记录集来维护视图.

由于各视图之间相互独立,而且整体维护开销较大,可分别为各个视图分配一个维护任务,并由调度器进行调度.在此基础之上,可以动态设置维护任务的优先级,并且根据场景需求实时切换各维护任务的执行次序.比如说,假设包含相同基表的待维护视图包括 v_1 和 v_2,而新来的查询想要处理视图 v_1,则可以提升 v_1 的处理优先级,从而提升整体效率.各视图维护任务需要指定待维护的视图名称(*ViewName*)、维护任务优先级(*Priority*,默认为 1)和增量记录集.系统维护一个针对所有维护任务的优先级队列(即视图维护任务队列,*TaskQueue*),以确保高优先级的任务被优先执行.

- 最后,步骤④执行步骤③所创建的任务,计算需要更新到物化视图中的行集.

在物化视图创建时,为各视图创建一个数据结构,该数据结构保存视图的名称(*ViewName*)、视图的基表名称

(*TableName*)、视图的查询执行时的算子(*Operators*)和当前已维护到的 *BlockID*.本文将其命名为视图信息(*ViewInfo*).系统重新执行视图的查询算子,便能很快计算出结果.然后将行集写入物化视图中,至此,物化视图已被更新.

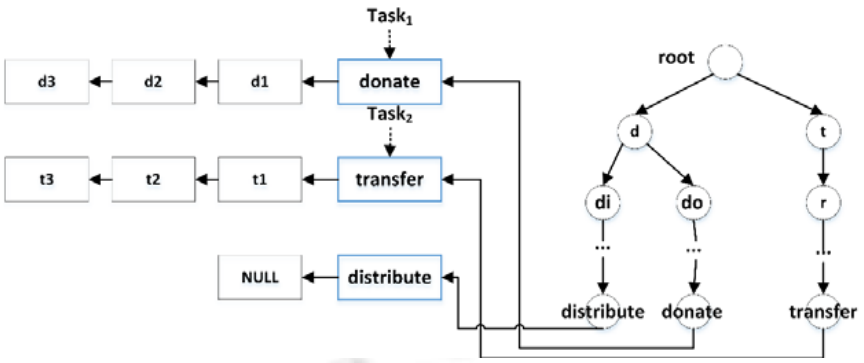


Fig.5 Storage of *DeltaRecord*

图 5 增量记录的存储

接下来具体描述算法步骤,包括算法 1 和算法 2.

算法 1. 创建视图维护任务.

Input:字典树 *TrieTree*,增量记录表名 *TableName*,物化视图信息 *ViewInfo*,视图维护队列 *TaskQueue*;

Output:无.

1: *DeltaRecords*←*TrieTree.search*(*TableName*); //查找字典树,获取对应的增量记录集

2: *Task*←创建一个新的物化视图维护任务;

3: *Task.DeltaRecords*←*DeltaRecords*;

4: *Task.ViewName*←*ViewInfo.ViewName*;

5: *TaskQueue.push*(*Task*); //将物化视图维护任务加入到视图维护任务队列

算法 2. 物化视图维护.

Input:待维护物化视图集合 *V*,待维护视图信息集合 *ViewInfoSet*,视图维护队列 *TaskQueue*;

Output:无.

1: **while** (!*TaskQueue.empty*(·))

2: *TaskHead*←*TaskQueue.front*(·); //当视图维护任务队列不为空则从队列头取出任务进行维护

3: *TaskQueue.pop*(·);

4: *v*←基于维护任务的视图名称 *TaskHead.ViewName* 在 *V* 中查找视图;

5: *ViewInfo*←基于维护任务的视图名称 *TaskHead.ViewName* 在 *ViewInfoSet* 中查找视图信息;

6: *Rows*←针对维护任务的增量记录集 *TaskHead.DeltaRecords* 执行 *ViewInfo* 的算子获得新增行集;

7: *update*(*v*,*Rows*); //将新增的行集 *Rows* 添加到视图 *v* 中

8: **if** (系统处于忙碌状态) **break**;

9: **endwhile**

算法 1 描述了创建视图维护任务的过程,其输入参数为:字典树 *TrieTree*,增量记录表名 *TableName*,物化视图信息 *ViewInfo* 和视图维护队列 *TaskQueue*.算法 1 第 1 行根据增量记录表名 *TableName* 查找字典树 *TrieTree*,得到增量记录集 *DeltaRecords*.这里省略了如何在字典树中查找增量记录集的过程.第 2 行~第 4 行表示创建一个新的视图维护任务 *Task*,并对其增量记录集和视图名称赋值.该算法最后将此任务加入到视图维护任务队列中.

调度器周期性评测当前系统的负载,一旦发现当前系统处于非忙碌状态(CPU 占用资源低于某一阈值),则从 *TaskQueue* 中依次调取优先级高的若干任务来执行;在执行过程之中,系统仍旧可以检测系统的资源占用情

况,当系统过于忙碌时,则暂时退出物化视图维护过程,留待下一周期执行.算法 2 描述了在非忙碌阶段调度器执行部分视图维护任务的过程,其输入参数为:待维护物化视图集合 V,待维护视图信息集合 *ViewInfoSet*,视图维护队列 *TaskQueue*.算法的第 2 行、第 3 行取出视图维护队列头部的视图维护任务.第 4 行根据视图维护任务包含的视图名称查找视图集合 V 中相应的视图.第 5 行表示根据视图维护任务包含的视图名称查找视图信息集合 *ViewInfoSet* 中相应的视图信息.第 6 行表示针对维护任务的增量记录集执行视图信息中的算子获得新增行集 *Rows*.第 7 行将第 6 行中产生的行集添加到视图 v 中.算法最后判断系统状态,若当前系统繁忙,则退出算法.

例 1:假设系统中存在两张物化视图 V_d 和 V_t,其中,物化视图 V_d 的基表为 *donate*,V_t 的基表为 *transfer*.系统中增量记录缓冲区的增量记录如图 5 所示,此时,视图维护任务队列含有 4 个视图维护任务.其中,$Task_1$ 负责维护基表为 *donate* 的物化视图,$Task_2$ 维护基表为 *transfer* 的物化视图.当系统收到查询物化视图 V_t 的请求,则提高视图维护任务 $Task_2$ 的优先级,那么视图维护任务队列中视图维护任务执行的顺序变为 $Task_2$,$Task_1$.当执行视图维护任务时,系统从视图维护任务队列头取出 $Task_2$,然后根据物化视图 V_t 的查询执行算子计算视图新增的行集并添加到 V_t 中,最后删除已维护完毕的维护任务 $Task_2$ 和增量记录 $t1$,$t2$,$t3$.

视图维护时,增量记录的 *BlockID* 提供了一致性保证.我们为每个物化视图存储已维护记录的最新 *BlockID*,如果该 *BlockID* 为查询到来时最新的区块号,则说明物化视图已经更新到最新状态.

3.2 多种维护策略的支持

许多数据库都采用单一的物化视图维护策略,例如:文献[6]采用闲时维护的策略,文献[7]采用按需维护的策略.闲时维护存在不能及时响应查询的情况,而按需维护可能存在一些物化视图累积增量记录过多、导致一次维护的执行时间长的问题.为了避免上述情况,本方法同时支持闲时维护和按需维护的策略:当 CPU 空闲时,查询引擎检查视图维护队列是否存在未执行的视图维护任务,若存在,则依次执行;如果查询到来,维护策略切换为按需维护.

当查询到来时,我们从共识层可以知道当前最新的 *BlockID*,而我们预先为每个视图维护一个最近更新的 *BlockID*.首先将两个 *BlockID* 比较,如果相等,则说明该物化视图已维护到最新状态;如果不等,则需要访问增量记录缓冲区和视图维护任务队列.此时,根据查询语句中的表名查找字典树,便可获取到与查询相关的增量记录,如果存在,则建立视图维护任务.如果视图维护任务队列中存在查询相关视图的维护任务,则系统将会提升这些视图维护任务的优先级,从而查询引擎优先维护查询相关的物化视图以快速响应查询.通过将多种维护策略相互协调,系统可以最大限度地降低查询的延迟时间.

4 可信任的物化视图

借助物化视图固然可显著提高区块链中查询的效率,但是当物化结果被存储于本地时,一旦本地节点被攻击,物化结果就有可能被恶意篡改,进而影响查询结果的正确性.鉴于此,本文利用默克尔树来确保基于物化视图的查询结果的正确性.

默克尔树[10]是基于哈希值的二叉树,其叶子节点是数据的哈希值,非叶子节点是对子节点哈希值进行串接之后再进行散列所获得的哈希值.因此,默克尔树具有防篡改特性.本方法预先为每个视图构建一棵默克尔树,并保存在内存中.在查询过程中,系统读取物化视图重新构建默克尔根,并与之前保存的默克尔根进行比对,以验证查询结果的正确性.每当物化视图更新时,均会针对所更新的数据产生一个哈希值,该哈希值将被添加到默克尔树的最右侧子节点,并向上更新默克尔树直至根节点.以图 6 为例,假定物化视图每次更新一行,则所生成的默克尔树共有 8 个叶子节点,其中,H_i 即为 Row_i 的哈希值.由于新增的交易总是更新默克尔树的最右节点,所以当默克尔树增长到超出可使用的内存大小时,系统可以只将默克尔树的最右路径上的节点保存在内存中,以便更新默克尔树.

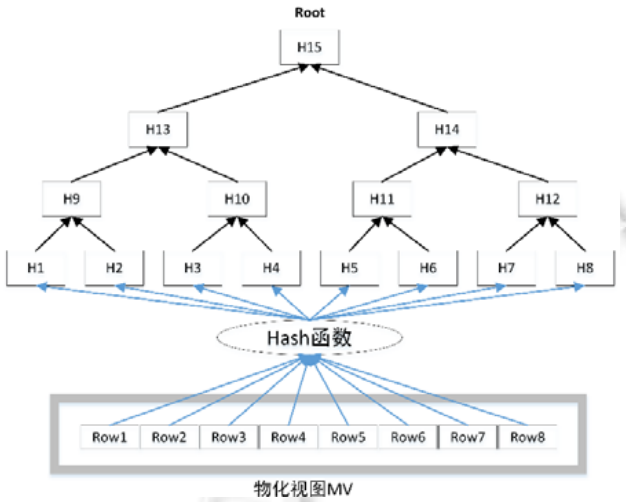


Fig.6 Structure of Merkle-tree

图 6 默克尔树结构

算法 3 描述了如何使用默克尔树验证基于物化视图的查询结果,输入参数为物化视图 *MV*、视图每次更新的数据长度数组 *sArray* 和预先建立的默克尔树的树根 *Root*,输出为验证标志 *Proofs*.算法第 1 行创建一个空数组 *ViewHashs* 以保存哈希值.第 2 行初始化视图 *MV* 的偏移位置.算法的第 3 行~第 8 行表示对于每次维护的记录行集 *Rows* 进行哈希操作,其中:第 4 行表示每次从视图的偏移位置读取长度为 *sArray*[*i*]的数据 *rows*;第 5 行、第 6 行表示将每次读取的数据 *rows* 进行哈希,并把得到的哈希值保存在 *ViewHashs* 中;第 7 行更新视图 *MV* 的偏移位置.算法的第 9 行使用 *ViewHashs* 建立默克尔树得到默克尔树根 *RowsRoot*.第 10 行验证新计算的默克尔树根 *RowsRoot* 是否与 *Root* 相同:若相同,则验证成功,算法返回验证结果 *Proofs* 为 true;反之验证失败,返回 false.由此,本方法保证了物化视图物化结果的正确性,从而保证基于视图的查询的结果也是正确的.

算法 3. 默克尔树查询验证.

Input:物化视图 *MV*,视图每次更新的数据长度数组 *sArray*,默克尔树根 *Root*;

Output:是否验证成功标志 *Proofs*.

1: *ViewHashs*←创建一个空数组;

2: *FilePos*←0; //初始化视图 *MV* 的偏移位置为 0

3: **for** *i*=0 to size of (s*Array*)−1 **do**

4: *rows*←从视图 *MV* 的偏移位置 *FilePos* 处读取 *sArray*[*i*]长度的数据;

5: *RowsHash*←*hash*(*rows*); //对 *rows* 进行哈希操作,得到 *RowsHash*

6: *ViewHashs*←将 *RowsHash* 添加到 *ViewHashs* 中;

7: *FilePos*←*sArray*[*i*]+*FilePos*;

8: **end for**

9: *RowsRoot*←根据 *ViewHashs* 建立默克尔树,得到默克尔根 *RowsRoot*;

10: *Proofs*←*equals*(*RowsRoot*,*Root*);

11: **return** *Proofs*;

例 2:以图 6 为例,查询获取视图结果时,系统按照每次视图写入的终止位置不断读取物化视图中的记录并进行散列,然后根据散列后的哈希值建立图 6 所示的默克尔树,如此得到默克尔树根 *H*15.算法 3 将它与预先保存的默克尔树根 *Root* 比对:若相同,则验证成功,表示物化结果没有被恶意篡改.

算法 3 的时间开销由内存中的计算时间和物化视图的读取时间组成.其中,内存计算的时间主要取决于哈希运算的次数.在算法 3 中的循环体中,哈希运算被运行 *n* 次(*n* 为物化视图维护的次数),循环体外建立默克尔树

哈希运算需要运行 $n-1$ 次,则哈希运算一共被运行 $2\times n-1$ 次,因此算法 3 内存中的时间复杂度为 $O(n)$.而对于视图的读取,设磁盘带宽为 x(MB/s),物化视图的大小为 y(MB),则视图的读取时间为 y/x(s).

5 实 验

5.1 实验环境

本文将提出的物化视图机制实现在区块链系统 SEBDB[4]中,以验证所提出方法的有效性.实验在 4 台机器组成的集群上进行,其中,每台机器配备 Intel Xeon(R) 2.10GHz 的 CPU,96G 的 RAM 和 3TB 的硬盘.区块链系统运行在 CentOS 7 上,区块链采用 Tendermint 共识,共识时间设置为 1s.

此外,我们采用了 SEBDB[4]中的模式,如图 7 所示,此模式的链上表由捐赠系统中的 *donate*,*transfer* 和 *distribute* 这 3 张表组成.其中,*donate* 表记录捐助者的捐款信息,*transfer* 记录捐赠组织间的资产转移信息,*distribute* 记录捐助组织给予受助者的援助信息.由于系统原型的局限性,原型暂不支持多表连接和聚集查询,因此,这里工作负载仅涉及单张表的查询或者两张表的连接,形式如 $Q1$,$Q2$ 和 $Q3$ 所示.本文实验的数据由 SEBDB 中的数据生成器生成,系统中各表的交易均匀地分布在区块链中.

Q1: SELECT * FROM *donate*;

Q2: SELECT * FROM *transfer* INNER JOIN *distribute* ON
transfer.organization=distribute.organization;

Q3: SELECT * FROM *transfer* INNER JOIN *distribute* ON
transfer.organization=distribute.organization
WHERE *distribute.amount*>10000;

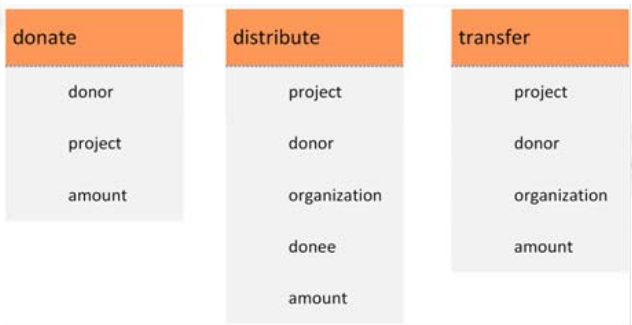


Fig.7 Database schema

图 7 数据库模式

5.2 物化视图维护性能

我们通过创建 $Q1$,$Q2$ 形式的物化视图来比较单表与等值连接视图维护所需要的时间.本实验设置区块个数 2 000~4 000,其中每个区块包含 5 条需要视图维护的交易,每条交易大小为 300 字节.图 8 显示了物化视图规模在 1 万~2 万条记录之间情况下的维护时间.对于 $Q1$ 形式的单表查询视图,维护开销缓慢增长,即使系统一次维护 2 万条数据,也仅需要 200ms 的时间.这种情况下,视图维护过程完全可以与共识过程并发处理中,视图维护对系统的影响非常小.$Q2$ 形式的等值连接视图维护过程中依然要扫描区块以获取另一张表的数据进行连接,所以相对单表视图的维护时间增长许多,对于 2 万条记录的维护已经超过共识设定的时间.这种情况下,系统采用闲时维护的策略.对于现阶段的区块链,交易的吞吐量最高为上千级别,并且现实情况中系统每张表的物化视图数量比较少,因此我们的方法完全可行.

图 9 具体分析了 $Q2$ 形式的等值连接视图在视图维护过程中,存储视图和除存储视图以外其他阶段消耗的时间,图中显示:存储物化视图的时间是短暂的,其余阶段的执行占据了视图维护的主要部分.这是因为在连接的另一张表不存在视图的情况下连接查询仍然需要扫描区块,但这可以通过使用索引的方法来避免.

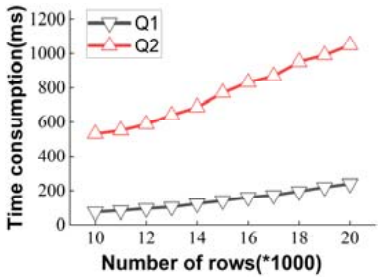


Fig.8 Cost to maintain views

图 8 视图维护开销

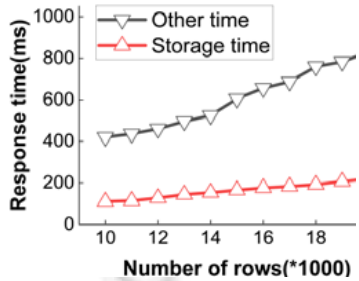


Fig.9 Storage time and other time

图 9 存储时间与其他时间

除此之外,我们测试了使用字典树和线性扫描两种方法查找增量记录的物化视图维护性能.为了更好地体现字典树的性能,实验中模拟加入了其他表名的增量记录,并且固定每个表名的增量记录为 100 个.如图 10 所示:使用字典树的视图维护方法的维护性能总是优于线性扫描增量记录缓冲区的维护方法,当维护的增量记录的表名越多、增量记录越多时,两者的差距则越大.这是因为字典树只需要一次字符串比较便可得到增量记录集的位置,而不使用字典树的情况下,系统每次获取增量记录集都需要遍历整个增量记录缓冲区.

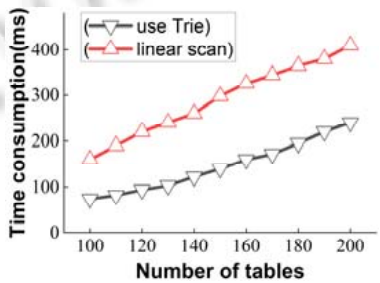


Fig.10 Using trie tree

图 10 使用字典树

5.3 使用物化视图减少查询的延迟

本节我们对比查询使用视图和查询扫描区块所需的响应时间.对于 *Q*1 查询,本实验固定区块个数为 1 000,每个区块中的交易数量为 200.图 11 显示:使用物化视图的查询响应时间增长缓慢,而扫描区块方式的查询时间快速增长.这是因为随着区块的增多,扫描区块的时间快速增长,而扫描物化视图需要更少的 I/O.图 12 显示 *Q*2 的查询响应时间,结果集固定为 20 000 行,区块个数从 1 000~5 000 增长,使用物化视图的查询响应时间远少于扫描区块的查询响应时间.而扫描区块的查询响应时间快速增长,这是因为 *Q*2 查询需要两次扫描区块,并且需要进行复杂的连接操作.

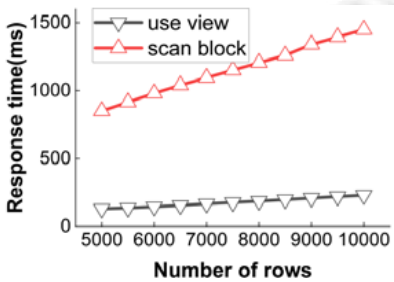


Fig.11 Response time on *Q*1

图 11 查询响应时间(*Q*1)

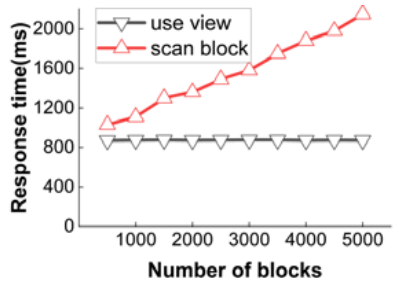


Fig.12 Response time on *Q*2

图 12 查询响应时间(*Q*2)

5.4 使用物化视图与使用索引的查询性能对比

本实验将本文提出的物化视图方法与SEBDB中的索引方法进行性能对比,两者皆采用相同的查询和数据.本实验固定区块个数为1 000,每个区块有200条交易,结果集从1万~2万行增长.由图13可知:对于*Q*1,使用物化视图的查询响应时间一直低于使用索引的查询响应时间;当结果集行数增加时,使用索引的查询响应时间增长较快,而使用物化视图的查询响应时间平缓增长.这是因为当数据均匀分散在各个区块中时,使用索引会引起更多的磁盘随机读取,而使用物化视图是对文件的顺序读取.

如图14所示:对于*Q*2,使用物化视图的查询始终优于使用索引的方式;基于索引的查询需要随机读取数据,并且需要进行连接操作,因此使用索引的查询响应时间相对使用视图的方式越来越长.实验证明,查询使用物化视图方法的性能优于使用索引的方法.

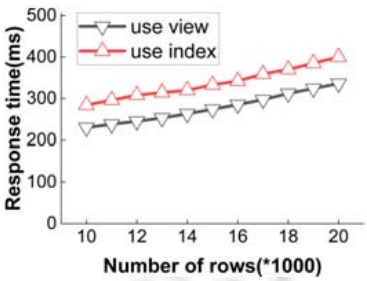


Fig.13 Index vs.materialized view on *Q*1

图13 索引和物化视图性能对比(*Q*1)

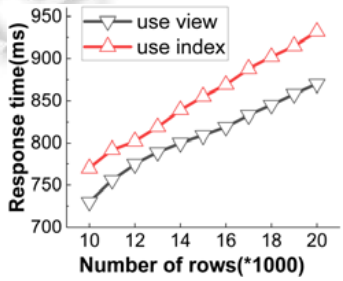


Fig.14 Index vs.materialized view on *Q*2

图14 索引和物化视图性能对比(*Q*2)

此外,我们使用 *Q*3 对比在查询具有选择条件的情况下,使用物化视图和索引的性能.本实验固定区块个数为2 000,每个区块的交易为1 000条,查询的选择率从0.1~1增长.实验结果如图15所示:当选择率为1时,使用索引的查询响应时间比使用物化视图的方法多 40ms;随着选择率的下降,物化视图的优势越明显;当选择率为0.1时,使用索引的查询响应时间为使用视图的方法的2倍多.

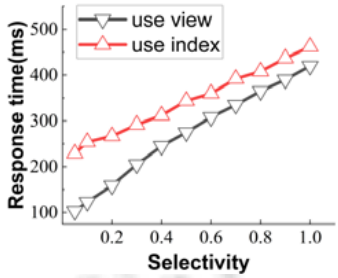


Fig.15 Index vs. materialized view on *Q*3

图15 索引和物化视图性能对比(*Q*3)

5.5 可验证查询的性能

对于使用物化视图的查询请求,我们构建默克尔树对视图进行验证.本实验采用的视图为*Q*1形式的单表视图.实验设置物化视图维护的总记录数从5 000增长到10 000,在扫描视图时生成默克尔树,并与内存中保存的默克尔树根进行比对.验证查询的延迟如图16所示:*NV*表示不加验证过程的查询响应时间,*YV*表示具有验证阶段的查询响应时间.图16中显示:结果集为10 000条记录时,*YV*比*NV*多了约100ms.这是可以接受的,因为这仍然比不使用视图的查询要快很多.

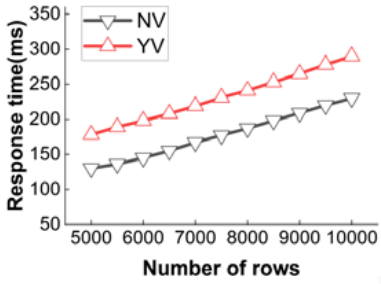


Fig.16 Verification cost

图 16 验证开销

总结以上实验,本文描述的物化视图维护方法可以很好地提升查询的性能;并且通过默克尔验证,保证了使用视图的查询请求结果的正确性.

6 相关工作

以支持智能合约为代表的区块链 2.0 平台的提出,使得区块链技术可以广泛使用到除电子货币以外的传统行业中.为了应对区块链在传统行业应用中所面临的数据管理方面的新需求,来自学术界和工业界的许多工作开始专注于区块链技术与数据管理技术的结合,这将促使区块链技术可以在传统行业领域有更广泛的使用.其中,ChainSQL[1]和 BigchainDB[2]在数据库的基础上使用区块链,使得数据库具有去中心化、防篡改的特点.FlureeDB[3]是一个结合了区块链技术的可扩展的图形数据库,虽然它们丰富了查询的功能,但未提高查询的性能.SEBDB[4]为区块数据添加了关系语义,并且使用索引来提升查询性能;但对于复杂查询或均匀分布的数据查询,响应时间依然较长.

相比于索引技术,物化视图是提升区块链数据库查询性能更直接的办法,但物化视图却会带来额外的维护开销.庆幸的是,数据管理领域的研究人员已经对如何降低物化视图的维护代价做了很多方面的工作.在视图维护策略方面,文献[11,12]采用立即维护的视图维护策略;而文献[6,7]使用延迟维护策略,使视图维护不阻塞更新事务.此外,文献[13,14]分别讨论了分布式数据库系统和 NOSQL 数据库系统中更新物化视图的问题,它们都支持增量的视图维护方法.在降低物化视图更新开销方面,早期的文献[5,15–19]提出了优化的物化视图更新算法,它们主要利用现有的物化视图和表达式进行增量更新.文献[6,20–23]提出了异步更新的视图维护工作,对于需要集成分布式数据源的数据仓库,这种方法的优势更加突出.文献[14,24]将分布式环境下的物化视图的维护工作分散到多个视图更新程序中,通过并行维护,提高了物化视图更新的效率.文献[25,26]优化了同时更新多个相关的物化视图的算法,它们考虑多个物化视图之间的表达式关系,利用多个物化视图表达式之间的公共子表达式,从而找到维护一组物化视图的最高效的维护计划.文献[27–29]提出了物化附加视图的方法,其中,文献[28,29]提出的方法对文献[27]中的方法做了优化,节省了空间开销.文献[30]提出了一种高阶形式的增量视图维护(HIVM)算法,该算法借鉴数学中微分的思想,递归地使用离散的前向差异(增量修改)进行视图更新.文献[31]提出了在分布式环境中进行批量更新的高效增量视图处理技术.文献[32]提出一种新的数据结构,并提出基于此结构的视图更新算法,该算法具有常量的时间复杂度和空间复杂度.在面向区块链的视图维护中,这些方法都是可以借鉴的,但它们并不完全适用于区块链的数据模型,比如不支持以区块为粒度的视图更新,并且区块链的共识过程使区块链上的视图维护变得更加复杂.

在可验证查询方面,文献[33]提出了 3 种可验证的连接算法,保证了连接结果的完整性和正确性.Vchain[34]提出了轻量级可验证查询框架,并保证查询结果的正确性和完整性,此外,Vchain 中还提出了基于双线性配对累加器的验证数据结构,以降低查询验证的代价.它们通过验证返回的结果来保证查询的完备和保真.而本文则验证查询的数据来源是否正确,使得验证更轻便、高效.

除了物化视图相关的工作,关于区块链数据管理方面的研究工作涉及得很广泛,文献[35–37]是关于区块链

技术和可信数据管理方面的探讨.为了提高区块链系统的扩展性和并发性,文献[39]提出一种智能合约并发执行的方法.

7 结束语

本文针对区块链系统中,面向区块数据的查询响应太慢的问题,提出了一套面向区块链中区块数据查询的物化视图构建、维护和访问机制:首先,本文选择合适的时机进行视图维护,使得视图维护过程隐藏于共识过程中,并以区块为粒度,采用字典树的元数据存储方式加快了视图维护的过程;其次,本文采用混合式的多种维护策略相结合的视图维护方式,降低了查询的延迟;最后,本文提出默克尔树验证方法使得本地物化数据不会被恶意篡改,进而保证了查询结果的有效性.

本文提出的面向区块链中区块数据查询的物化视图机制,实现在一个真实的区块链平台[4]中.实验结果表明,本文的方法高效可行.但是和关系模型相比,对于区块链中数据的链式存储模型,针对复杂查询的物化视图创建和维护面临着更多的挑战.比如:对于复杂的业务场景,为了保证查询效率,区块链需要创建的物化视图更多.因此,我们将继续探究物化视图的更新、选择在区块链中与关系数据库的不同之处,动态地为每个智能合约精准预创建物化视图,在保证系统查询性能下的同时,使得物化视图所占空间尽可能小.我们下一步的工作重点是研究支持复杂查询,例如多表连接、聚集查询等的有效物化视图维护机制,我们也将进一步探讨将基于物化视图的可信查询与可信硬件 SGX 结相合,使得可验证查询的响应时间更短.

References:

[1] ChainSQL. http://www.chainsql.net/

[2] BigChainDB. BigchainDB2.0: The Blockchain Database. White Paper, 2018.

[3] FlureeDB. https://flur.ee/

[4] Zhu YC, Zhang Z, Jin CQ, Zhou AY, Yan Y. SEBDB: Semantics empowered blockchain database. In: Proc. of the 35th ICDE. 2019. 1820−1831. [doi: 10.1109/ICDE.2019.00198]

[5] Gupta A, Mumick IS, Subrahmanian VS. Maintaining views incrementally. In: Proc. of the 1993 ACM SIGMOD. 1993. 157−166. [doi: 10.1145/170035.170066]

[6] Zhou JR, Larson PA, Elmongui HG. Lazy maintenance of materialized views. In: Proc. of the 33rd Int'l Conf. on Very Large Data Bases. 2007. 231−242.

[7] Duan HC, Hu HQ, Qian WN, Ma HX, Wang XL, Zhou AY. Incremental materialized view maintenance on distributed log-structured merge-tree. In: Proc. of the DASFAA. 2018. 682−700. [doi: 10.1007/978-3-319-91458-9_42]

[8] Castro M, Liskov B. Practical Byzantine fault tolerance. In: Proc. of the OSDI. 1999. 173−186.

[9] Weiner P. Linear pattern matching algorithms. In: Proc. of the 14th SWAT. 1973. 1−11.

[10] Merkle RC. A digital signature based on a conventional encryption function. In: Proc. of the CRYPTO. 1987. 369−378.

[11] Bello RG, Dias K, Downing A, Feenan J, Finnerty J, Norcott WD, Sun H, Witkowski A, Ziauddin M. Materialized views in Oracle. In: Proc. of the 24th VLDB. 1998. 659−664.

[12] Zaharioudakis M, Cochrane R, Lapis G, Pirahesh H, Urata M. Answering complex SQL queries using automatic summary tables. In: Proc. of the 2000 ACM SIGMOD. 2000. 105−116. [doi: 10.1145/342009.335390]

[13] Segev A, Park J. Maintaining materialized views in distributed databases. In: Proc. of the 5th ICDE. 1989. 262−270. [doi: 10.1109/ICDE.1989.47225]

[14] Agrawal P, Silberstein A, Cooper BF, Srivastava U, Ramakrishnan R. Asynchronous view maintenance for VLSD databases. In: Proc. of the 2009 ACM SIGMOD. 2009. 179−192. [doi: 10.1145/1559845.1559866]

[15] Blakeley JA, Larson PA, Tompa FW. Efficiently updating materialized views. In: Proc. of the 1986 ACM SIGMOD. 1986. 61−71. [doi: 10.1145/16894.16861]

[16] Larson PA, Yang HZ. Computing queries from derived relations. In: Proc. of the 11th Int'l Conf. on Very Large Data Bases. 1985. 259−269.

[17] Yang HZ, Larson PA. Query transformation for psj-queries. In: Proc. of the 13th Int'l Conf. on Very Large Data Bases. 1987. 245−254.

[18] Kawaguchi A, Lieuwen DF, Mumick IS, Ross K. Implementing incremental view maintenance in nested data models. In: Proc. of the DBPL. 1997. 202−221.

[19] Chaudhuri S, Krishnamurthy R, Potamianos S, Shim K. Optimizing queries with materialized views. In: Proc. of the 11th ICDE. 1995. 190−200. [doi: 10.1109/ICDE.1995.380392]

[20] Quass D, Widom J. On-Line warehouse view maintenance. In: Proc. of the 1997 ACM SIGMOD. 1997. 393−404. [doi: 10.1145/253260.253352]

[21] Salem K, Beyer KS, Cochrane R, Lindsay BG. How to roll a join: Asynchronous incremental view maintenance? In: Proc. of the 2000 ACM SIGMOD. 2000. 129−140. [doi: 10.1145/342009.335393]

[22] Zhang X, Ding LL, Rundensteiner EA. Parallel multisource view maintenance. The VLDB Journal, 2004,13(1):22−48. [doi: 10.1007/s00778-003-0086-0]

[23] Agrawal D, Abbadi AE, Singh AK, Yurek T. Efficient view maintenance at data warehouses. In: Proc. of the 1997 ACM SIGMOD. 1997. 417−427. [doi: 10.1145/253260.253355]

[24] Zhang Y, Power R, Zhou SY, Sovran Y, Aguilera MK, Li JY. Transaction chains: Achieving serializability with low latency in Geo-distributed storage systems. In: Proc. of the 24th SOSP. 2013. 276−291. [doi: 10.1145/2517349.2522729]

[25] Mistr H, Ro P, Sudarsha S, Ramamritham K. Materialized view selection and maintenance using multi-query optimization. In: Proc. of the 2001 ACM SIGMOD. 2001. 307−318. [doi: 10.1145/375663.375703]

[26] Segev A, Fang WP. Currency-Based updates to distributed materialized views. In: Proc. of the 6th ICDE. 1990. 512−520. [doi: 10.1109/ICDE.1990.113505]

[27] Ross KA, Srivastava D, Sudarshan S. Materialized view maintenance and integrity constraint checking: Trading space for time. In: Proc. of the 1996 ACMSIGMOD. 1996. 447−458. [doi: 10.1145/233269.233361]

[28] Luo G. Partial materialized views. In: Proc. of the 23rd ICDE. 2007. 756−765. [doi: 10.1109/ICDE.2007.367921]

[29] Zhou JR, Larson PA, Goldstein J, Ding LP. Dynamic materialized views. In: Proc. of the 23rd ICDE. 2007. 526−535. [doi: 10.1109/ICDE.2007.367898]

[30] Koch C, Ahmad Y, Kennedy O, Nikolic M, Nötzli A, Lupei D, Shaikhha A. Dbtoaster: Higher-order delta processing for dynamic, frequently fresh views. The VLDB Journal, 2014,23(2):253−278.

[31] Nikolic M, Dashti M, Koch C. How to win a hot dog eating contest: Distributed incremental view maintenance with batch updates? In: Proc. of the 2016 ACMSIGMOD. 2016. 511−526. [doi: 10.1145/2882903.2915246]

[32] Idris M, Ugarte M, Vansummeren S. The dynamic Yannakakis algorithm: Compact and efficient query processing under updates. In: Proc. of the 2017 ACMSIGMOD. 2017. 1259−1274. [doi: 10.1145/3035918.3064027]

[33] Yang Y, Papadias D, Papadopoulos S, Kalnis P. Authenticated join processing in outsourced databases. In: Proc. of the 2009 ACMSIGMOD. 2009. 5−18. [doi: 10.1145/1559845.1559849]

[34] Xu C, Zhang C, Xu JL. vChain: Enabling verifiable boolean range queries over blockchain databases. In: Proc. of the 2019 ACMSIGMOD. 2019. 141−158. [doi: 10.1145/3299869.3300083]

[35] Shao QF, Jin CQ, Zhang Z, Qian WN, Zhou AY. Blockchain: Architecture and research progress. Chinese Journal of Comptuers, 2018,41(5):969−988 (in Chinese with English abstract). [doi: 10.11897/SP.J.1016.2018.00969]

[36] Shao QF, Zhang Z, Zhu YC, Zhou AY. Survey of enterprise blockchains. Ruan Jian Xue Bao/Journal of Software, 2019,30(9): 2571−2592 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5775.htm [doi: 10.13328/j.cnki.jos.005775]

[37] Qian WN, Shao QF, Zhu YC, Jin CQ, Zhou AY. Research problems and methods in blockchain and trusted data management. Ruan Jian Xue Bao/Journal of Software, 2018,29(1):150−159 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5434.htm [doi: 10.13328/j.cnki.jos.005434]

[38] Qi XC, Zhu YC, Shao QF, Zhang Z, Zhou AY. A warehouse receipt management system based on blockchain technology. Journal of East China Normal University (Natural Science), 2018,201(5):152−161 (in Chinese with English abstract).

[39] Pang SF, Qi XD, Zhang Z, Jin CQ, Zhou AY. Concurrency protocol aiming at high performance of execution and replay for smart contracts. In: Proc. of the CoRR.abs/1905.07169, 2019.

附中文参考文献:

[35] 邵奇峰,金澈清,张召,钱卫宁,周傲英.区块链技术:架构及进展.计算机学报,2018,41(5):3–22. [doi: 10.11897/SP.J.1016.2018.00969]

[36] 邵奇峰,张召,朱燕超,周傲英.企业级区块链技术综述.软件学报,2019,30(9):2571–2592. http://www.jos.org.cn/1000-9825/5775.htm [doi: 10.13328/j.cnki.jos.005775]

[37] 钱卫宁,邵奇峰,朱燕超,金澈清,周傲英.区块链与可信数据管理:问题与方法.软件学报,2018,29(1):150–159. http://www.jos.org.cn/1000-9825/5434.htm [doi: 10.13328/j.cnki.jos.005434]

[38] 齐学成,朱燕超,邵奇峰,张召,金澈清.基于区块链的仓单管理系统.华东师范大学学报(自然科学版),2018,201(5):152–161.

蔡磊(1996－),男,山东莘县人,硕士生,主要研究领域为区块链.

朱燕超(1992－),男,博士生,主要研究领域为分布式数据库,区块链.

郭庆兴(1995－),男,硕士生,主要研究领域为区块链.

张召(1977－),女,博士,副教授,CCF 专业会员,主要研究领域为区块链,分布式数据管理.

金澈清(1977－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为区块链,海量数据挖掘.