今天和大家分享一篇 vldb2019的 最佳论文，结合了数据库技术和区块链。
Fine-Grained, Secure and Efficient Data Provenance on Blockchain Systems
提出了一种细粒度、安全高效的区块链数据溯源技术

区块链是来自于bitcoin的一种数据结构。
现在许多的区块链系统，实际上可以看做一种特殊的分布式交易系统。

这类系统有一些基本的特点。
P2P，点对点网络。
拜占庭的节点网络环境。节点互相不信任，或者弱信任。
分布式账本。并且使用智能合约来实现区块链上数据流转的逻辑。
区块链的设计可以保证数据的不可篡改。对于区块链的数据检索是链下的行为。

我们来看一个链，第一个块部署了一个token合约，就是币；第二个块上有一笔交易，调用了这个合约里面的转账函数，A给B转账20；第三个块也是一笔转账交易。C给D转账20

这些交易由链结构连接起来，交易都是可追踪的。交易的具体逻辑是由智能合约来定义的。

说完前面的背景，来讲一下motivation。
能不能丰富智能合约的交易语义呢？使得智能合约可以使用交易来源的信息，文章后面会给出交易来源，历史交易依赖的定义。

解释一下智能合约中的数据依赖

原先智能合约中转账的条件是转账方的余额足够。这里给大家补充一点关于联盟链存储模型的介绍。

新的转账的条件：比如历史五个区块，账户的余额大于某个阈值。
收款人没有和黑名单中的账户做过交易。
这两个条件需要使用历史世界状态，和数据源依赖信息。

目前有三类区块链解决方案：

第一种，所有的数据都放在世界状态里面，这样子代价比较大，容易出错。

第二种，链上进行交易，链下对交易进行分析；这样会破坏序列化，会受到重排交易攻击。
第三种，Fabric 1.0版本之后的改进，不是协议级别的改进，主要是增加历史数据库，方便应用层的查询。但是这种数据库并不能实现数据防篡改。

整体分析一下：协议级别的改进可以带来安全；注重性能上的权衡可以带来高效。

分析一下面临的挑战：
区块链比起数据库还是新生事物，没有标准化的操作。

比如Hadoop里面的Map reduce；SQL里面的Select, join and aggregation这样的一些明确的转换语义。目前区块链上还是缺少的。

在拜占庭的节点网络情况下，保证区块链的完整性、不可篡改性。

区块链是一个持续增长的数据结构。对于以太坊来说，采用gas的机制来保证无法实现针对区块链智能合约进行恶意的DDOS攻击，但同时这样的设计使得所有节点都需要执行合约，这是一个消耗大量算力的操作，也被称作验证者困境。

区块链其实并不神秘。我们来具体看一下区块的结构。分成区块头和区块体。

在网络中区块头和区块体是分开来传输的。区块体主要是交易的列表。在以太坊里大概有上百笔交易，fabric里面从10-1000都是自己设置的。区块链头里有这些交易的摘要。如果是挖矿型的链包含一个随机数的值。

区块头还包含前一个区块的hash。这是区块链的特性，前后链接成为不可篡改的账本记录。还包含一个state摘要。state就是世界状态。在区块链系统的概念里面，区块链状态由key和value键值对组成。

看这张图比较清楚。一个key对应着一个value。
区块链上的交易可以对某个key的value进行更新。世界状态是最新区块高度下，所有的键值对。

左边是以太坊的区块头，右边是fabric的区块头。相同的部分有前一个区块的hash，交易hash摘要。以太坊里是通过merkle 帕特里克树来组织世界状态。世界状态是很多的。世界状态的摘要是最上面一层的hash。

这个树有什么特点呢？既是一颗Merkle树，也是一颗字典树。方便根据账户地址寻址。

fabric世界状态是merkle Bucket tree。叶子节点桶数量是可以改变的。平衡存储和查找效率，桶的数量可以设置成2或者5。
最上层root hash就是当前世界状态的摘要。

这张图给出了可以进行数据溯源的区块链架构。分成应用层、共识层、执行层和存储层。

在应用层，我们先解释这个数据溯源的概念。定义一种输入和输出之间的依赖。
比如A转账10块钱给B。首先要读取当前A和B的余额，然后进行写操作，A余额减10，B余额加10.
输入是A的状态和B的状态，输出是下一个区块A的状态和B的状态。输入和输出之间就有依赖关系。

在区块链里面是通过智能合约来实现这种应用层的转账逻辑的。实际上代码实现的时候，通过handler来读写世界状态的键值对。

history函数的功能是查找键值对的历史记录。

Backward函数，会返回key在某个区块高度下的后向依赖。
Forward函数，会返回key在某个区块高度下的前向依赖。

这里给出了一些智能合约的代码片段。利用之前新加的函数功能，可以实现更丰富的智能合约语义。

比如退款功能，使用history函数，如果在200到230的区块高度里面，账户的余额小于一个阈值，就进行退款。

比如黑名单功能，利用history函数确定200到230的一个区块区间。如果一个账户的前向依赖或者后向依赖中包含原本在黑名单中的账户，就说明这个账户曾经和他们进行过交易，则这个账户也被列入黑名单。

然后我们解释在执行层用到的技术。执行层接收合约调用和用户定义的依赖关系，能够计算出交易的结果，并返回明确的依赖元组。

主要利用了merkle DAG数据结构。DAG是一个有向无环图。用它来组织世界状态。它取代了原本的merkle树。实际上是在merkle树的叶子节点加入指针结构，指向后向依赖的key。

我们来具体分析这张图。v1 v2 v3是区块高度，是递增的。k1 k2 k3是不同的键。
k1在V1区块高度的时候，有一个值。由于交易1，在V2的高度时，k1的值更新了，变成版本2。在V3区块高度的时候，由于交易3，k1的值再次更新，变成第三个版本。

k1 v3依赖 k1 v2和 k2 v1，类似的k2 v4依赖k1 v2和k3 v2。merkle DAG是在原本merkle树的基础上在叶子节点里面，加入后向依赖的指针。

但是对于前向依赖有些问题。在状态更新期间，前向依赖是无法确定的。解决方式是在后继状态里面存储前向依赖指针。

接下去看存储层使用的技术。世界状态里存储的是一个状态最新区块高度下的值。索引历史值是不太高效的。借助一个跳表，来有效索引一个状态的历史值。

跳表的查询比较简单，就是用二分法查询。跳表的构建算法是需要解决的问题。

跳表的多层的。插入的是区块高度，是递增的。

插入12 ，是否要创建 L4；1除以4 5除以4；1除以8 10除以8；12除以8 和10除以8
在一个间隔里面，所以要创建L4

现在来看这种设计的性能测试。运行时的延时非常小，可以忽略不计
额外的存储空间，建立索引和hash指针只需要百分之2到4的存储空间。

和之前提过的，在链下分析区块链数据相比，查询区块的高度的增加不会给Linearchain带来延迟。
在时间延迟和tps上，都有显著的提升。

再来看存储。一个区块链系统，在存储层Block信息占主要比重，历史状态索引占第二，merkle DAG索引占比重比较小。

总结一下，LineageChain提出了一种细粒度、安全高效的区块链数据溯源技术，增加其安全性和效率。存储代价也比较小。

关键设计包括：提出用户自定义的依赖关系，用Merkle DAG替换原来的Merkle状态树。建立索引来查询历史状态。在Fabric0.6和1.3版本上都进行了实现。