计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.200500105

基于区块并行的以太坊智能合约高速重放

陈自民[1]　卢艺文[2]　郭　燕[1]
1 中国科学技术大学苏州研究院　江苏 苏州 215000
2 安比实验室　江苏 苏州 215000
(czmmm@ustc.edu.cn)

摘　要　分析和研究以太坊上的区块、交易、账户和智能合约数据具有巨大价值,但是以太坊数据量大、数据种类多、存储结构各异,当前数据获取方法的获取速度慢而且获取的数据不全,因此充分利用这些数据非常困难。文中提出了基于区块并行的以太坊数据快速导出工具Geth-query,通过分析以太坊内部机制,利用区块世界状态快照消除区块之间的依赖关系,优化本机资源利用效率并行重放区块,实现了快速而全面地提取以太坊链上数据。实验证明,Geth-query提取的数据种类丰富,数据导出速度相比传统方法提升了10倍左右。为了使用方便,文中同时对导出的数据进行存储优化,并在前端页面进行数据展示,从而为分析和研究以太坊提供了数据基础。

关键词:以太坊;智能合约;数据抽取;并行重放
中图法分类号　TP315

High-speed Replay of Ethereum Smart Contracts Based on Block Parallel

CHEN Zi-min[1],LU Yi-wen[2] and GUO Yan[1]
1 Suzhou Research Institute,University of Science and Technology of China,Suzhou,Jiangsu 215000,China
2 SECBIT Lab,Suzhou,Jiangsu 215000,China

Abstract　Analyzing and researching blocks,transactions,accounts,and smart contract data on Ethereum is of great value,but Ethereum has a large amount of data,many types of data,and different storage structures. The current data acquisition methods are slow and the acquired data are incomplete,so it is very difficult to use these data. This paper proposes Geth-query,a fast data export tool for Ethereum based on block parallel. By analyzing the internal mechanism of Ethereum,it uses snapshots of the world state of the block to eliminate dependencies between blocks and optimize the efficiency of local resource utilization to parallel replay block,thus achieving fast and comprehensive extraction of data on the Ethereum chain. Experiments prove that the types of data extracted by Geth-query are rich,and the data export speed is about 10 times faster than traditional methods. For ease of use,this paper also optimizes the storage of the exported data and displays the data on the front-end page,thus providing a data foundation for the analysis and research of Ethereum.

Keywords　Ethereum,Smart contract,Data extraction,Parallel replay

1　介绍

以太坊是继比特币之后最成功的区块链平台,智能合约的引入使得以太坊不再是单一的分布式账本,从而为去中心化应用的发展提供了支持。通过分析以太坊中大量的区块、交易、账户以及各类智能合约数据,能够得出很多重要的信息,如发现重入攻击[1]、旁氏骗局[2]、低价格DoS攻击[3]、社交网络分析[4]等。

以太坊上数据种类众多,包括区块、交易、账户、智能合约字节码和交易执行字节码,各种数据的存储结构各异,按类分离和导出所有的数据很困难。当前很多以太坊数据的研究工作实际上都受到数据获取方法的限制,不能获得完整而丰富的数据。目前主要的数据获取方法包括:1)下载区块文件并解析[5-6];2)从以太坊全节点Web3 API接口[7-9]读取数据;3)使用爬虫从区块链浏览器[10-12]读取数据;4)从自定义以太坊客户端[13-14]读取数据。以上方法都存在获取的数据不完整或数据获取速度慢的缺点。

首先,方法1)—方法3)的数据获取不完整。方法1)不能获取交易票据数据和内部交易;方法2)不能获取内部交易,因此合约之间交互的数据不完整;方法3)中,区块链浏览器只能获取浏览器提供的数据,例如,当前最流行的区块链浏览器Etherscan[11]的API只提供最近10 000笔交易,每笔交易只提供1 000个trace数据,并且区块链浏览器通常都有反爬虫机制,因此,很难获得完整的链上数据。

其次,方法2)—方法4)都存在数据获取速度慢的问题。对于方法2),由于全节点在处理请求时会通过交易哈希找到

到稿日期:2020-05-22　返修日期:2020-08-21　本文已加入开放科学计划(OSID),请扫描上方二维码获取补充信息。
通信作者:郭燕(guoyan@ustc.edu.cn)

所处的区块，然后初始化交易的运行环境，在该区块中按顺序串行重放该交易之前的所有交易以获取该笔交易的 trace 数据，因此，在通过 debug_traceTransaction 接口获取交易 trace 信息时速度慢；方法 3）会受网络带宽的影响和区块链浏览器服务的限制，例如，区块链浏览器 Etherscan 限制 API 的查询频率为 5 次/秒[14]；对于方法 4），由于区块之间的状态存在相互依赖关系，后面区块的状态依赖于前面区块的结果状态，因此该方法在导出所有区块数据时需要串行重放区块中的交易，使得其数据导出速度慢。

为了解决上述问题，本文提出了 Geth-query 区块链数据导出工具。该工具基于 go-ethereum 的源代码，在 EVM 虚拟机中插桩增加数据捕获代码，以获取交易在 EVM 中执行的中间数据的特定数据，因此能够提取以太坊上所有的或指定的合约数据，以及交易中间状态数据；同时，其通过在以太坊全节点 archive 模式所保存的每个区块历史世界状态的基础上完成定点并行重放区块，来消除区块之间的依赖关系，优化本机资源利用效率，从而快速高效地进行数据提取。本文的主要贡献如下：

（1）获取了以太坊上种类丰富的数据，包括区块数据、交易数据、内部交易数据、交易 trace 数据、交易 event 数据和预编译合约调用数据。

（2）大幅提升了以太坊数据导出速度，使用该工具重放 9000000 个以太坊的区块原始数据仅用了 100 小时，而以太坊客户端 geth v1.9.0 同步同样数量的 archive 模式的全节点的区块至少需要 13 天[15]。

（3）对 Geth-query 工具导出的数据进行了存储优化，并搭建可视化的网站。

2 总体设计

为了快速全面地导出以太坊上的各类数据，从而为以太坊上各类数据分析的研究提供基础，本文提出了 Geth-query 工具。Geth-query 整体架构图如图 1 所示。其主要完成以下 3 方面的工作：1）区块世界状态自动快照；2）智能合约重放优化以及数据抽取；3）智能合约运行状态数据存储优化。

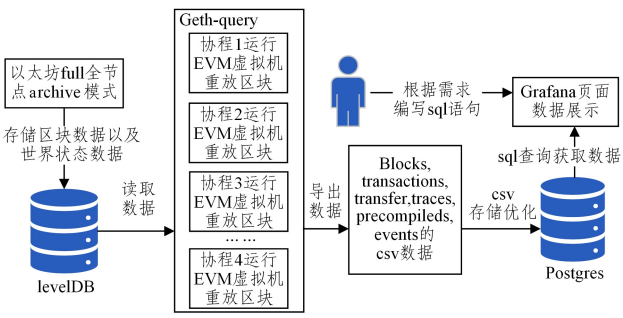


图 1 系统整体架构

Fig. 1 Overall system architecture

2.1 区块世界状态自动快照

由于智能合约运行在区块链虚拟环境中，不存在直观的标准化接口能够访问合约中的各类数据，因此需要保存合约的快照来访问数据。同时，单一合约状态的变化会影响整个区块链状态的变化，并随着合约代码的逐步调用执行而时刻发生变化，如合约的单步执行可导致自身、其他账户或其他合约的状态发生变化，并且这种变化会随着区块高度和交易数量的增加而不断增加。因此，区块之间存在相互依赖的关系，后面区块状态依赖前面区块状态。为了消除区块的这种依赖关系，需要保存每个区块执行完成之后的世界状态快照。

对于以太坊全节点而言，为了构建以太坊区块世界状态快照，需要使用全节点 archive 模式的 EVM 虚拟机串行执行所有区块中的交易数据，获取每一个区块的世界状态，然后通过 RLP 格式编码，存储到底层的 levelDB 数据库。Geth-query 工具依赖于全节点的 levelDB 数据库获取每个区块的世界状态。

2.2 智能合约重放优化以及数据抽取

区块链本质上是一个状态机系统，交易的逐步执行造成以太坊世界状态的改变，跟踪世界状态的变化就是跟踪合约执行的中间状态。Geth-query 工具在 EVM 虚拟机中捕获交易执行的中间过程信息，记录每一区块在 EVM 虚拟机中按顺序执行交易后世界状态的差异，实现以最小的代价维护整个区块链系统的关键状态转移信息。另外，其通过索引保存每个交易在 EVM 中执行过程的关键 trace 信息，保留交易的执行过程信息。这种设计可以实现以下多维度的执行过程重放、任意区块内的所有交易执行重放、任意交易的执行过程重放、任意合约执行过程中的虚拟机状态重放，最终通过此套方案快速恢复智能合约在历史任意时刻的状态。Geth-query 工具并行区块的重放工作，调整并行参数以最大限度地提高本机资源的利用效率，大大加快了区块重放的速度。

2.2.1 区块数据

区块数据主要涉及区块头数据和交易数据，这些数据直接通过 P2P 网络同步获取。区块头包含一个区块的基本信息，包括区块序号、区块哈希、父区块哈希、时间戳等。交易数据为每一个区块中包含的所有交易，交易信息包括发起方地址、接收方地址、交易金额、交易输入等。

2.2.2 合约状态数据

合约状态数据主要是交易在 EVM 中执行的中间数据，如内部交易数据、交易票据数据、交易执行的 trace 数据以及预编译合约调用数据。

内部交易数据是交易内部调用合约产生的数据。本文通过分析 EVM 虚拟机在执行被调用合约代码过程中修改世界状态处的代码，进行插桩添加 Capturetransfers 动作，从而获得 EVM 执行合约过程中导致账户世界状态变动的内部交易数据。EVM 以下动作会触发内部交易：1）CALL，CALLCODE，DELEGATECALL，STATISTIC 引起合约之间的相互调用；2）Create 创建合约地址并初始化余额值。

另外，为了便于校验世界状态余额的正确性，在实现中把以下导致世界状态变化的动作统计到内部交易中：1）外部账户之间的直接转账；2）SELFDESTRUCT 指令销毁智能合约后合约余额的退还；3）每笔交易产生的 gas 花费；4）矿工的出块奖励。

trace 信息对于分析合约创建情况、合约之间相互调用情况、合约读写 storage 空间情况以及优化合约 gas 资源消耗情况有着重要意义。交易 trace 数据主要是以下指令的执行信息：CREATE，CREATE2，CALL，CALLCODE，DELEGATE-

CALL，STATICCALL，SELFDESTRUCT，REVERT，SSTORE。上述指令执行的详细信息通过交易在 EVM 虚拟机执行过程中校验指定指令的运行信息。

预编译合约调用数据是交易中合约代码对预编译合约的调用数据。EVM 是基于堆栈的虚拟机，EVM 通过操作计算量来计算 gas 花费，因此，如果 EVM 中涉及非常复杂的运算，则 EVM 将会变得非常低效且消耗很多 gas。例如，在 zksnark[16] 中需要椭圆曲线上的加法和乘法以及双线性配对操作，这些操作都非常复杂，不适合在 EVM 中运行，因此以太坊提出了预编译合约。通过统计预编译合约的调用情况能够统计链上使用零知识证明技术[17] 的智能合约。在以太坊中预编译合约的合约地址固定在 0x01 到 0x09 之间，通过在 EVM 中校验被调用合约的合约地址来获取预编译合约调用数据。

交易 event 数据是交易在执行过程中产生的事件数据。交易 event 数据主要用于统计链上遵循 ERC20 或者 ERC721 标准的去中心化应用的运行情况。交易 event 数据主要通过交易执行完成之后返回的票据数据中的 Logs 字段来获取。

2.2.3 以太坊区块并行算法设计

以太坊全节点的 archive 模式保存每个区块的历史世界状态，消除并行重放区块过程中区块之间的依赖关系。Geth-query 启动后直接从 LevelDB 中读取每个区块的世界状态，开始从多个点并行重放区块，具体的执行步骤如下：

1）启动 Geth-query，Geth-query 读取配置文件 goMaxProcs（CPU 逻辑核心数）、batchBlock（任务单元区块数量）、goRoutine（协程池协程数）、startBlock（起始区块值）、endBlock（结束区块值）参数进行初始化配置。前三者的值对于 Geth-query 工具充分使用本机计算资源提高并行重放效率具有重大影响。

2）Geth-query 根据 batchBlock 值划分任务执行单元，并将所有单位任务按照区块顺序依次放到协程池中运行。当协程池中所有协程都在运行任务单元时，协程池的任务调度会受到阻塞；当协程池中有空余协程时，调度模块才会将剩余单位任务继续调度到协程池中运行，直到所有单位任务执行完毕。

3）Geth-query 中的每一个协程执行单位任务，首先读取全节点数据库 levelDB 中的区块数据和区块世界状态快照，恢复当前重放区块的世界状态，然后按照区块顺序为每笔交易新建一个 EVM 虚拟环境，开始执行交易。

4）Geth-query 工具中的 EVM 虚拟机通过合约账户的 code 域按顺序读取智能合约的指令执行，并且 Geth-query 工具通过在 EVM 中的插桩代码识别特定指令来获取内部交易。

5）每笔交易重放完成后会根据重放后的 from 地址余额以及内部交易涉及 from 地址的交易值，来计算重放前 from 地址余额，并通过校验 from 余额来证明数据的正确性。

6）所有交易执行结束，智能合约重放数据导出成 csv 文件加以存储。

根据上述设计，基于区块并行的以太坊智能合约高速重放算法如算法 1 所示。

算法 1 基于区块并行的以太坊智能合约高速重放算法

输入：GoMaxProcs，batchBlock，goRoutine，startBlock，endBlock

输出：以太坊数据

```
1. function RUN(GoMaxProc,batchBlock,goRoutine,startBlock,endBlock)
2.     goRoutinePool ← NewPoolWithFunc (GoMaxProc, REPLAYBATCHBLOCK(start,end))
3.   for i=startBlock→endBlock do
4.     goRoutinePool. Invoke(i,i+batchBlock-1)
5.     i←i+batchBlock
6.   end for
7. end function
8. function REPLAYBATCHBLOCK(start,end)
9.   result ←TRUE
10.  for i=start→ end do
11.    block←GetBlockByNumber(i)
12.    blockData←traceBlock(block)
13.    CSVWriter(blockdata)
14.  end for
15.  return result
16. end function
17. function REPLAYBLOCK(block)
18.  statedb,err:=blockchain. StateAt()
19.  for i=0← block. Transactions(). size()-1 do
20.    txData←block. Transactions(). get(i)
21.    vmenv=NewEVM(statedb)
22.    transitionDb()
23.    if msg. To()==nil then
24.      evm. Create()
25.      setcode()
26.      initContract()
27.    else
28.      evm. Call()
29.      setcode()
30.    end if
31.    checkPrecompiledContracts()
32.    precompileds←capturePrecompiledContractsData()
33.    interpreter. Run()
34.    loop execute opcode func()
35.      transfers←captureTransferData()
36.      traces←captureTraceData()
37.    end loop
38.    checkTransferBalanceValue()
39.    receipts←receipt
40.    blockData←block,txData,receipts,transfers,traces,precompiled
41.  end for
42.  return blockData
43. end function
```

2.3 智能合约运行状态数据存储优化

Geth-query 工具导出的区块数据和智能合约运行中间状态数据均保存在 csv 文件中，并最终导入到 postgres 数据库中。由于 Geth-query 导出的数据量较大（TB 级别），并且每个表中有数十亿条记录直接将所有数据存入数据库会导致数

据库查询过程非常低效。因此，需要在数据库中校验 Geth-query 工具导出数据的完整性，而且需要针对特定的查询需求对 postgres 数据库中的数据做存储优化。

对数据的存储优化主要有分表、建立索引以及根据索引建立聚类 3 个步骤。由于每个表中的数据记录多达数十亿条，执行 SQL 语句查询数据非常慢，因此需要对数据存储做优化。首先，按照每个子表存储 1 000 000 个区块数据的方式进行分表存储，使得数据记录尽可能均匀分布在各个子表中。其次，对于不同的子表，根据查询需求对经常查询的字段建立 B-tree 索引和表达式索引，查询语句可以组合多个索引条件以提高查询速度，目的是提高特定索引相邻数据的查询速度。最后，每个表根据特定索引将数据进行聚类。前端数据展示页面选用 Grafana 工具，该工具能够很方便地连接 Postgres 数据库，通过 SQL 语句查询数据，并将查询结果通过配置好的 Dashboard 进行数据展示。

3 实验结果

本节主要从导出数据的种类、区块重放效率等方面，将 Geth-query 与现有工具进行对比，可以看出 Geth-query 在获取数据种类、数据导出速度方面超过现有工具。

3.1 数据种类丰富度

表 1 列出了现有各种工具能够获取的数据种类。

表 1 数据获取种类

Table 1 Types of data acquired

数据类型	Etherscan	Ethereum-etl	DataEther	Geth-query
区块	√	√	√	√
外部交易	√	√	√	√
票据	√	√	√	√
合约字节码	▲	√	√	√
内部交易	▲	×	√	√
交易失败类型	√	√	√	√
预编译合约	×	×	√	√
trace 信息	▲	√	√	√
历史世界状态	▲	×	√	√
交易字节码	▲	√	√	√
…	×	×	√	√

注：“▲”代表能够获取部分数据

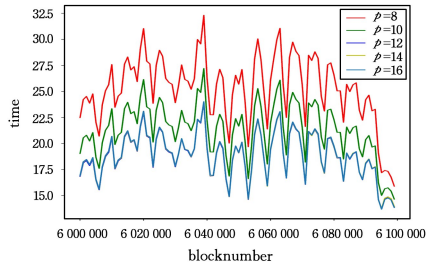
从表 1 可以看出，Geth-query 能够获取的数据种类比 Etherscan 数据源和 Ethereum-etl 工具提供的数据种类丰富。Dataether 和 Geth-query 的数据导出思路相同，因此导出的数据种类也相同，但从后面的实验可以看出，Geth-query 工具导出数据的速度更快。同时，理论上 Geth-query 能够通过 capture 模块获取任意 EVM 执行交易的中间状态数据。

3.2 Geth-query 重放效率

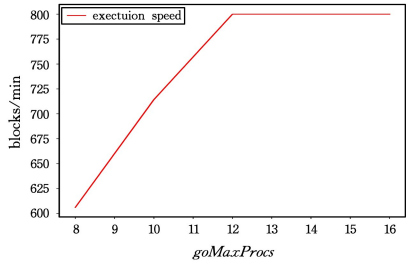
Geth-query 可以通过调整配置参数来优化本机系统资源利用效率，从而提升区块重放效率。Geth-query 的区块重放并行化效率主要受 goMaxProcs，batchBlock 和 goRoutine 参数的影响，通过调整 3 个参数值可使 Geth-query 实现最优并行重放区块效率。

本地实验环境为 CPU：intel i9-9900(8 核 16 线程)，memory：32G DDR4 2400，硬盘：4T samsung SSD RAID，4T intel SSD。

首先，由于 Geth-query 在本地并行重放合约的瓶颈主要在磁盘 IO，因此合适的 *goMaxProcs* 值有助于 Geth-query 在接近磁盘 IO 瓶颈的同时充分利用 CPU 资源。本文以以太坊区块号 6 000 000 到 6 100 000 的区块为本工具的性能测试区间，测试不同的 *goMaxProcs* 值对 Geth-query 执行速度的影响。其中，*batchBlock* 值为 1 000，*goRoutine* 值为 16，测试结果如图 3 所示。



(a)不同 *goMaxProcs* 值下，每个 batchBlock 的耗时



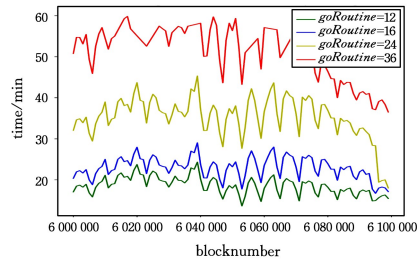
(b)不同 *goMaxProcs* 值下，每分钟重放区块数量

图 3 goMaxProcs 效率图

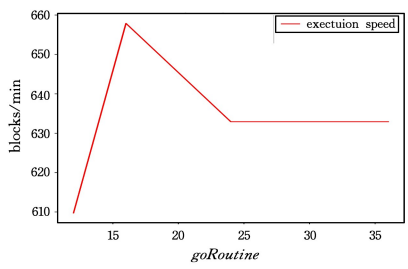
Fig. 3 goMaxProcs efficiency

由图 3 可知，当 *goMaxProcs* 值为 12 时，Geth-query 执行速度达到最大，继续增大 *goMaxProcs*，Geth-query 的执行速度没有明显提升，并且磁盘 IO 负担加大。*goMaxProcs* 值为 12，14，16 时曲线基本重合。

其次，*goRoutine* 值过大会导致 go 调度器频繁调度协程，从而降低执行效率。本文测试了不同的 *goRoutine* 参数值对 Geth-query 执行效率的影响。*goMaxProcs* 值为 12，*batchBlockNumber* 值为 1 000 时的测试结果如图 4 所示。



(a)不同 *goRoutine* 值下，每个 batchNumber 的耗时



(b)不同 *goRoutine* 值下，每分钟重放区块数量

图 4 goRoutine 效率图

Fig. 4 goRoutine efficiency

由图 4 可知，尽管 batchBlock 单位任务执行速度随着 *goRoutine* 值的减少而加快，但是对于 Geth-query 总体而言，单位时间内执行的区块数目在减少。当 *goRoutine* 的值超过 16 时，go 调度器由于频繁调度协程，使得 Geth-query 执行效率反而降低。因此对于本地实验环境而言，最优 *goRoutine* 值为 16。

再次，*batchBlock* 的值对 Geth-query 工具执行效率的影响主要是在程序结束时，可能存在极个别计算单元的计算量较大，导致大多数协程执行完毕后被销毁，但是少数协程仍然在执行的情况。该参数对 Geth-query 执行效率的影响较小，综合考虑程序并行读写 csv 文件以及任务调度器的开销，该值取 1 000。

最后，对 Geth-query 和 DataEther[13] 的数据抽取速度进行了对比。文献[13]给出的实验结果为：抽取 10 000 个交易 trace 的平均时间为 100.4 s，通过全节点的 RPC 接口获取相同的 10 000 个交易 trace 信息的平均耗时为 498.5 s。因此，本文也测试了抽取 10 000 个交易 trace 的平均时间。

另外，由于以太坊早期产生的区块包含的交易较少，计算密度小于中后期产生的区块，因此抽取不同区块的速度不同。因为 DataEther 中并未明确表明其获取的 10 000 个 trace 信息来自于哪些区块，为了充分对比其数据抽取速度，本文在 1～3 000 000，3 000 000～6 000 000，6 000 000～9 000 000 这 3 个区间内分别获取 10 000 个 trace 信息，并且该 10 000 个 trace 通过过滤 trace 信息获得，其执行的区块区间比不过滤信息获得的 trace 更大，因此数据抽取时间也更长。

首先获得的 10 000 个 trace 在各个区间内执行的区块区间分别为 2 100 000～2 101 700，5 100 000～5 100 030 以及 8 100 000～8 100 020。

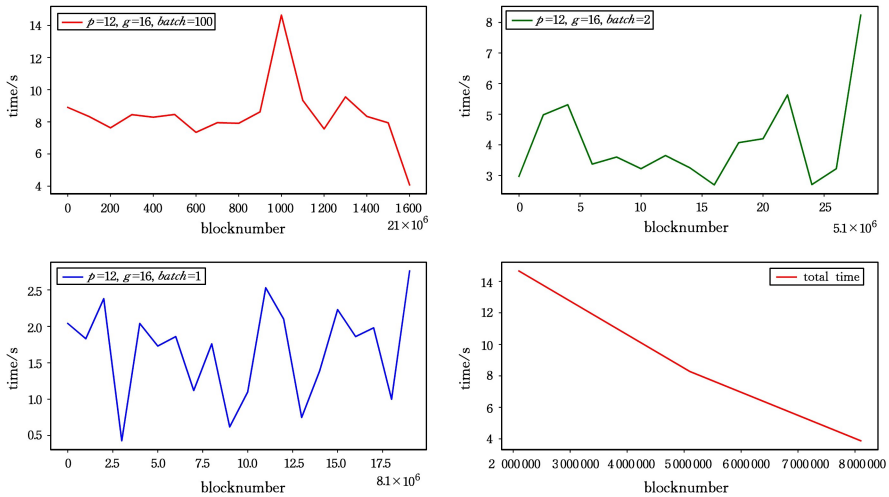


图 5 区间重放效率图

Fig. 5 Block replay efficiency

图 5 中绘制了 3 个区间每个 batchBlockNumber 的执行时间，因为 Geth-query 工具是并行重放区块，并且测试样本较小，因此 3 个区间执行的总时间几乎等于执行时间最长的单位任务所花费的时间。Geth-query 工具重放 3 个区块区间的用时分别为 14.6 s，8.2 s，3.8 s。执行时间依次递减的原因是，前面的区块交易大多是外部账户之间的转账，trace 数据较少，执行逻辑简单，因此获取10 000个 trace 信息包含的区块和交易较多；后面区块的交易大多是外部账户调用合约代码，执行逻辑复杂，单笔交易 trace 数据较多，获取 10 000 个 trace 信息包含的区块和交易较少。Geth-query 工具在交易之间频繁切换，为每笔交易初始化 EVM 运行环境需要大量时间，因此前面区块区间的执行时间长于后面区块区间。此外，本文实验环境与 DataEther 的实验环境存在差异，本文实验环境的机器性能优于 DataEther，这也是本文数据导出效率较高的原因之一。由于 DataEther 的代码没有开源，不能在本地测量其运行效率，因此表 2 中的数据直接引用文献[13]中的测试结果。使用本地最优值的参数，Geth-query 重放 1～9 000 000 区块一共用时 100 小时，相比以太坊 full 节点的 archive 模式重放 1～9 000 000 区块的 13 天 19 小时[17]，执行效率有非常大的提升。

表 2 重放效率对比

Table 2 Replay efficiency comparison

（单位：s）

	Etherscan	Web3 API	DataEther	Geth-query
获取 10 000 个 trace 数据的时间	×	498.5	100.4	8.9

注：“×”代表只能够获取部分数据

3.3 数据优化以及展示

Geth-query 工具执行了 1～9 000 000 的区块，导出 csv 数据大小为 1.6 TB。blocks，transactions，transfers，events，traces，precompileds 6 个表的记录条数及大小如表 3 所列。

表 3 导出数据结果

Table 3 Acquired data results

表名	记录条数	数据库表/GB	csv 数据/GB
blocks	9 000 000	3	3
transactions	590 040 664	248	266
transfers	1 805 979 725	552	511
events	494 388 397	266	239
traces	2 386 845 195	683	584
precompileds	44 399 903	13	11

为测试系统的可用性，简单列举了以下分析需求：1）以太坊每日区块产生数量；2）每日交易数量；3）每日活跃地址数；

4)每日创建地址数量;5)每日预编译合约调用数量;6)每日 Call 指令调用数量;7)每日 trace 调用失败的数量;8)USDT[18]每日活跃地址数;9)USDT 每日交易数量(各种 USDT 的操作);10)USDT 每日 transfer 交易频率。图 6 为数据展示图。

图 6 数据展示图

Fig. 6 Data presentation

结束语 本文只是提供了一种智能合约高速重放和数据抽取的方法,并未对数据进行详细的分析。后面的工作将会对导出合约的中间状态数据进行分析,并使用机器学习的相关方法提取出每一类攻击交易的数据特征,形成各类攻击交易特征库。

当前除了针对合约安全展开分析外,随着以太坊数据量的激增和应用爆发,链上数据的快速查询和分析也具有相当大的价值,这也是未来需要进一步探索的重要领域。

参 考 文 献

[1] ATZEI N, BARTOLETTI M, CIMOLI T. A survey of attacks on ethereum smart contracts (sok)[C]// International Conference on Principles of Security and Trust. Springer, Berlin, Heidelberg, 2017: 164-186.

[2] CHEN W, ZHENG Z, CUI J, et al. Detecting ponzi schemes on ethereum: Towards healthier blockchain technology[C]// Proceedings of the 2018 World Wide Web Conference. 2018: 1409-1418.

[3] CHEN T, LI X, WANG Y, et al. An adaptive gas cost mechanism for ethereum to defend against under-priced dos attacks [C]// International Conference on Information Security Practice and Experience. Springer, Cham, 2017: 3-24.

[4] LEE X T, KHAN A, SEN GUPTA S, et al. Measurements, analyses, and insights on the entire ethereum blockchain network [C]// Proceedings of The Web Conference 2020. 2020: 155-166.

[5] KIFFER L, LEVIN D, MISLOVE A. Stick a fork in it: Analyzing the ethereum network partition[C]// Proceedings of the 16th ACM Workshop on Hot Topics in Networks. 2017: 94-100.

[6] NIKOLI I, KOLLURI A, SERGEY I, et al. Finding the greedy, prodigal, and suicidal contracts at scale[C]// Proceedings of the 34th Annual Computer Security Applications Conference. 2018: 653-663.

[7] BARTOLETTI, M, LANDE S, POMPIANU L, et al. A general framework for blockchain analytics[C]// 1st Workshop on Scalable and Resilient Infrastructures for Distributed Ledgers. ACM, 2017.

[8] Google. blockchain-etl/ethereum-etl[EB/OL]. https://github.com/ blockchain-etl/ethereumetl. 2019.

[9] ZHENG W, ZHENG Z, DAI H N, et al. Xblock-EOS: Extracting and exploring blockchain data from EOSIO[J]. arXiv: 2003.11967, 2020.

[10] BISTARELLI S, MAZZANTE G, MICHELETTI M, et al. Analysis of ethereum smart contracts and opcodes[C]// International Conference on Advanced Information Networking and Applications. Springer, Cham, 2019: 546-558.

[11] ETHERSCAN. Etherscan[EB/OL]. https://cn. Etherscan. com/.

[12] BARTOLETTI M, CARTA S, CIMOLI T, et al. Dissecting Ponzi schemes on Ethereum: identification, analysis, and impact[J]. Future Generation Computer Systems, 2020, 102: 259-277.

[13] CHEN T, LI Z, ZHANG Y, et al. Dataether: Data exploration framework for ethereum[C]// 2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS). IEEE, 2019: 1369-1380.

[14] CHEN T, ZHU Y, LI Z, et al. Understanding ethereum via graph analysis[C]// IEEE Conference on Computer Communications. IEEE, 2018: 1484-1492.

[15] ETHEREUM F. Geth v1. 9. 0[EB/OL]. https://blog. ethereum. org/2019/07/10/geth-v1-9-0/.

[16] BOWE S, GABIZON A, GREEN M D. A multi-party protocol for constructing the public parameters of the Pinocchio zk-SNARK[C]// International Conference on Financial Cryptography and Data Security. Springer, Berlin, Heidelberg, 2018: 64-77.

[17] FEIGE U, FIAT A, SHAMIR A. Zero-knowledge proofs of identity[J]. Journal of cryptology, 1988, 1(2): 77-94.

[18] TETHER. Tether[EB/OL]. https://tether. to/. 2020.

CHEN Zi-min, born in 1995, master. His main research interests include blockchain and cryptography.

GUO Yan, born in 1981, lecturer. Her main research interests include information security, blockchain and NLP.