Criação de um *Dataset* para Caracterização de Código de Contratos Inteligentes na Rede Ethereum

João Ricardo Pereira Marques[1], João Fabrício Filho[1], Rogério Aparecido Gonçalves[1]

[1]Universidade Tecnológica Federal do Paraná (UTFPR) – Câmpus Campo Mourão
Via Rosalina Maria dos Santos, 1233 – 87301-899 – Campo Mourão – PR – Brazil

joaoricardomarques@alunos.utfpr.edu.br,{joaof,rogerioag}@utfpr.edu.br

Abstract. *This article presents an approach to creating a dataset containing contract codes deployed on the Ethereum network. The selection criterion covers the three types of transactions: regular, creation, and execution of contract codes. The ethereumetl and DuckDB tools are used for data extraction and database creation. The database gathers block information, transactions, contract codes, and information about the instructions used in each contract such as how many times each contract was executed, and in which versions of the Ethereum Virtual Machine (EVM).*

Resumo. *Este artigo apresenta uma abordagem para a criação de um dataset com os códigos dos contratos implantados na rede Ethereum. O critério de seleção compreende os três tipos de transações: regulares, de criação e de execução do código dos contratos. As ferramentas ethereumetl e Duckdb são utilizadas para extração dos dados e a criação do banco de dados. A base de dados reúne informações de blocos, transações, código dos contratos, informações sobre as instruções utilizadas em cada contrato e quantas vezes cada contrato foi executado e em quais versões da Ethereum Virtual Machine (EVM).*

1. Introdução

A rede `Ethereum` foi desenvolvida tendo como propósito funções como a transferência de ativos entre redes e pessoas e o desenvolvimento de aplicações descentralizadas (DApps) [Wood et al. 2014]. É uma rede *peer-to-peer* formando um grande Sistema Distribuído[1] no qual os nós participantes mantêm a *blockchain*. Cada nó armazena uma cópia completa ou parte dos dados da *blockchain* e executa uma instância da *Ethereum Virtual Machine (EVM)*, contribuindo para o mecanismo de consenso sobre o estado da *blockchain*.

Uma das características principais é o uso de *Contratos Inteligentes* escritos em `Solidity`, uma linguagem de alto nível projetada para esse fim. Os códigos binários dos contratos são armazenados em transações na *blockchain* e quando invocados por outras transações são executados pela `EVM`.

As redes podem ser divididas em três tipos, com base nos requisitos e uso: *mainnet*, *testnets* e *private nets*. A `mainnet` é a versão principal da rede `Ethereum`. Existe um grande número de redes de testes disponíveis para `Ethereum`. Elas têm como objetivo fornecer um ambiente de testes para contratos inteligentes e aplicações descentralizadas *(DApps)* antes de serem implantados para rede principal de produção. Além disso,

[1]`https://etherscan.io/nodetracker#`

sendo redes de teste, elas permitem experimentos e pesquisa com tecnologias *Blockchain*. Atualmente, a *testnet* pública para desenvolvimento que é mantida é a `Sepolia`[2].

As redes privadas *(private nets)* podem ser criadas gerando-se um novo bloco inicial *(genesis block)*. Uma lista de redes `Ethereum` pode ser vista em `https://chainlist.org/`. Cada rede possui `IDs` de rede e de cadeia que são utilizados para identificar as redes. O ID da *mainnet* é 1.

Este trabalho propõe uma abordagem para criação de um conjunto de dados contendo códigos de contrato da rede *Ethereum*. Um explorador de blocos pode ser utilizado para apresentar informações detalhadas sobre blocos, transações e outras métricas relevantes sobre cada uma das redes. O `etherscan`[3], por exemplo, pode ser usado para explorar a *blockchain* da `Ethereum`.

2. Contratos Inteligentes

Contratos Inteligentes foram definidos por [Szabo 1997] como sendo um protocolo de transação eletrônico que executa os termos de um contrato.

A rede `Ethereum` trouxe o suporte à implementação desses contratos com a *Ethereum Virtual Machine (EVM)* e com a linguagem `Solidity`[4] frente às limitações da rede `Bitcoin` que utiliza uma linguagem *script* bem simples.

No desenvolvimento de contratos pode ser utilizado o compilador `solc` que converte código de alto nível dos contratos para *bytecode* da *EVM*[5] que apresenta um histórico de versões: *"cancun", "shanghai", "paris", "london", "berlin", "istambul", "petersburg", "constantinople", "byzantium", "spuriousDragon", "tangerineWhistle", "homestead"*. O conjunto de instruções para versão atual da `EVM` e cada uma das outras versões pode ser conferido em `https://www.evm.codes/?fork=cancun`.

Esse conjunto de tecnologias permite a execução automática segura e rápida de acordos e transações conforme as instruções no código. O Código 1 apresenta um exemplo de código do contrato *Addition.sol* escrito em `Solidity`.

Código 1. Exemplo de Código de Contrato em Solidity

```
1  pragma solidity ^0.8.21;
2
3  contract Addition {
4     uint8 x;
5
6     function addx(uint8 y, uint8 z) public {
7        x = y + z;
8     }
9     function retrievex() view public returns (uint8) {
10       return x;
11    }
12 }
```

[2] `https://ethereum.org/pt/developers/docs/networks/#sepolia`
[3] `https://etherscan.io`
[4] `https://soliditylang.org/`
[5] `https://ethereum.org/pt-br/developers/docs/evm/`

O código binário do contrato gerado pelo `solc` pode ser visto no Código 2. O código binário dos contratos é embutido em transações do tipo ***Contract Creation*** que são armazenadas em blocos da *blockchain* do `Ethereum`. Os contratos podem ser invocados tendo seu código executado pela `EVM` gerando transações do tipo ***Contract CALL***.

Código 2. Código Binário do Contrato

```
1  $ solc --bin Addition.sol
2  --> Addition.sol
3
4  ======= Addition.sol:Addition =======
5  Binary:
6  608060405234801561001057600080fd5b506101f680610020600039600f3fe6080604
7  052348015610010576000 80fd5b50600436106100365760003560e01c806336718d8014
8  61003b578063ac04e0a014610057575b600080fd5b610055600480360381019061005 09
9  1906100f2565b610075565b005b61005f61009e565b60405161006c9190610141565b60
10 05180910390f35b8082610081919061018b565b6000806101000a81548160ff02191690
11 8360ff160217905550505056 5b6000806000905490610100 0a900460ff16905090565b6
12 00080fd5b600060ff82169050919050565b6100cf816100b9565b81146100da57600080
13 fd5b50565b600081359050610 0ec816100c6565b92915050565b6000806040838503121
14 561010957610108610 0b4565b5b60006101178582860161 00dd565b9250506020610128
15 {restante do codigo suprimido}
```

3. Extração dos Dados

Para a extração dos dados dos contratos implantados na rede `Ethereum` é necessário que compreendamos os tipos de transações que podem existir na rede. Tais transações podem ser identificadas como: **transações regulares** de transferência de valores entre contas, **transações de implantação de contratos** *(Contract Creation)* e **transações de execução de contrato** *(Contract CALL)*.

Os dados podem ser extraídos de um nó `Ethereum` local ou de algum *endpoint* remoto. Neste estudo estamos extraindo de um *endpoint* da *mainnet* com acesso fornecido pela `infura`[6]. *Blocos* e *transações* podem ser recuperados utilizando a ferramenta `ethereumetl` [Day and Medvedev 2018] [Medvedev and the D5 team 2023]. Usando o comando **ethereumetl** com o parâmetro *export_blocks_and_transactions*, definindo-se o bloco inicial e bloco final do intervalo que deve ser extraído. Serão gerados dois arquivos no formato `CSV` contendo os *blocos* e as *transações* relacionadas a esses blocos. Um exemplo da linha de comando é apresentado no Código 3.

Código 3. Comando de Extração de Blocos e Transações

```
1  $ ethereumetl export_blocks_and_transactions --start-block 0 --end-
       block 500000 --provider-uri https://mainnet.infura.io/v3/${
       INFURA_KEY} --blocks-output blocks.csv --transactions-output
       transactions.csv
```

O projeto `ethereumetl` disponibiliza uma base de dados pública no `BigQuery` replicando a estrutura e os dados do *blockchain* da rede `Ethereum`[7]. O armazenamento é mantido pelo `Google`, mas há cobrança por consultas realizadas que excedam 1TB por mês.

[6] `https://docs.infura.io/network-endpoints`

[7] `https://bigquery.cloud.google.com/bigquery?p=bigquery-public-data&d=ethereum_blockchain&t=transactions`

4. Recuperando os Códigos dos Contratos

Para a análise dos dados, as informações de *blocos* e *transações* são importadas para o banco de dados `Duckdb`[8]. O `Duckdb` foi escolhido pela facilidade do uso combinado com `Python` e pelos recursos de manipulação de arquivos `CSV`.

Da tabela de *transações* é possível recuperarmos o código binário de cada contrato que pode ser desmontado. Entre as ferramentas instaladas com o cliente para `Ethereum` `geth` está o comando `evm` que é uma interface de linha comando para o `EVM`. O Código 4 apresenta o comando com o parâmetro para *disassembly*.

Código 4. Disassembly

```
1 $ evm disasm source.asm
2 6001600201600A03600402
3 00000: PUSH1 0x01
4 00002: PUSH1 0x02
5 00004: ADD
6 00005: PUSH1 0x0a
7 00007: SUB
```

Código 5. Contagem

```
1 $ python instructions-count.py
2 0 : PUSH1 : 3
3 1 : ADD : 1
4 2 : SUB : 1
5 {restante do codigo suprimido}
```

A contagem de instruções pode ser feita em *Python* utilizado o recurso de expressões regulares. O Código 5 apresenta o resultado da contagem das instruções que também será armazenado na base de dados para análise.

5. Considerações Finais

Este trabalho apresenta os resultados parciais do projeto que visa a criação de uma base de dados sobre contratos inteligentes implantados na rede `Ethereum`, contendo além das informações de blocos e transações, informações adicionais sobre o código dos contratos, como instruções utilizadas em cada contrato, assim como quantas vezes cada contrato foi executado, em quais versões da `EVM`.

A pesquisa trata-se de um estudo exploratório da rede `Ethereum`, sobre o tipo de código utilizado nos contratos e o conjunto de instruções mais utilizadas. Buscando otimizar os custos de execução para as instruções, que sejam mais justos em comparação com a execução em outros modelos arquiteturais. E ainda a proposta de melhorias para a `EVM` como a implantação de novas instruções.

Referências

[Day and Medvedev 2018] Day, A. and Medvedev, E. (2018). Ethereum in BigQuery: a Public Dataset for Smart Contract Analytics.

[Medvedev and the D5 team 2023] Medvedev, E. and the D5 team (2023). Ethereum ETL.

[Szabo 1997] Szabo, N. (1997). Formalizing and securing relationships on public networks. *First Monday*, 2(9).

[Wood et al. 2014] Wood, G. et al. (2014). Ethereum: A Secure Decentralised Generalised Transaction Ledger. *Ethereum project yellow paper*, 151(2014):1–32. `https://ethereum.github.io/yellowpaper/paper.pdf`.

[8] `https://duckdb.org/`