Announcements

Remaining grading material:

- Project: final version due 5/1
- Final Exam on Monday, May 8 at 9-11am -- cumulative test that covers all lectures (through April 25), recitation labs, homework assignments, and assigned reading

Please submit your course evaluation at https://bu.campuslabs.com/courseeval/

No office hours today.

Office hours moved to Friday 4/28/2023 from 1 to 3 pm.

Double-click (or enter) to edit

Lab 12: Oracles

The content of this lecture has been taken from ethereum developer page, Julien Thevenard's blog and Pedro Costa's blog

Introduction

Smart contracts are programs that run on the blockchain. Smart contracts can have access to the blockchain state. However, one of the limitations of blockchain is its inability to access external data sources, such as real-world events or off-chain data.

What if we want to write a smart contract that settles a sports bet between two parties? How will the blockchain know who won?

That's the job of an oracle. An oracle feeds outside data into the blockchain so that smart contracts can use it.

Oracle use cases:

- Provide market prices (decentralized finance ...)
- Provide natural events data (Insurance ...)

network infrastructure ...)

What is the oracle problem?

One can solve the problem of oracles by giving the information needed to the smart contract in a transaction. However, how do we know that the information given to the smart contract is the correct information **(correctness)** ? Also, how do we guarantee that one will give any infromation at all **(Availability)** ?

This means that we need to ensure that the data coming from the oracle is accurate, or else the smart contract might not work properly. We also need to ensure that we don't have to trust the oracle operators to provide accurate information, as that goes against the trustless nature of smart contracts.

Oracle properties

Though there is no flawless oracle, the effectiveness of an oracle should be evaluated on its ability to implement the following properties:

1. Correcness
2. Availability
3. Incentive compatibility

Correctness (Authenticity + Integrity):

An oracle should not cause smart contracts to trigger state changes based on invalid off-chain data. For this reason, an oracle must guarantee authenticity and integrity of data—authenticity means the data was gotten from the correct source, while integrity means the data remained intact (i.e., it wasn’t altered) before being sent on-chain.

Availability (The data is always available when needed):

An oracle should not delay or prevent smart contracts from executing actions and triggering state changes. This quality requires that data from an oracle be available on request without interruption.

Incentive compatibility (Incentives must align)

An oracle should not have incentive in submitting false or no information. This often involves attributability and accountability. Attributability allows for correlating a piece of external information to its provider, while accountability bonds data providers to the information they give, such that they can be rewarded or penalized based on the quality of information provided.

Types of oracles

1. Centralized
2. Decentralized (or try to be)

Centralized oracles:

We already live in this world (somewhat). We trust the goverment/exchanges/companies to give us data. Why not let those entites send us the data directly on the blockchain (assuming we know their public key)?

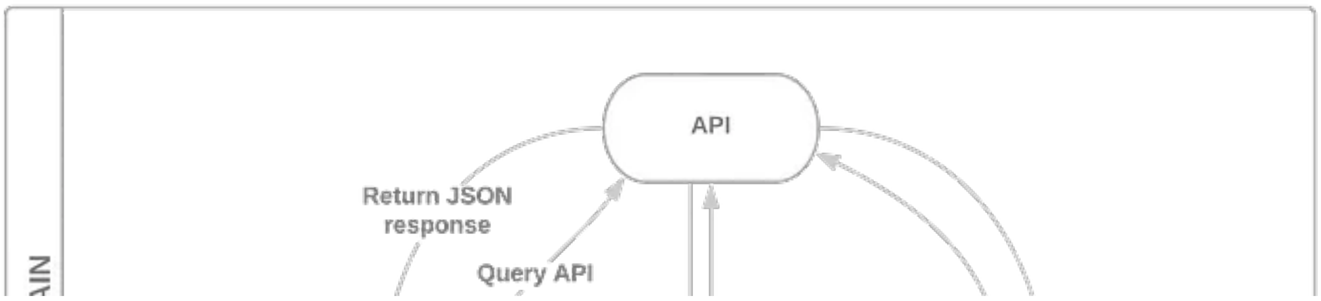
An oracle can send it's data signed on the blockchain directly.

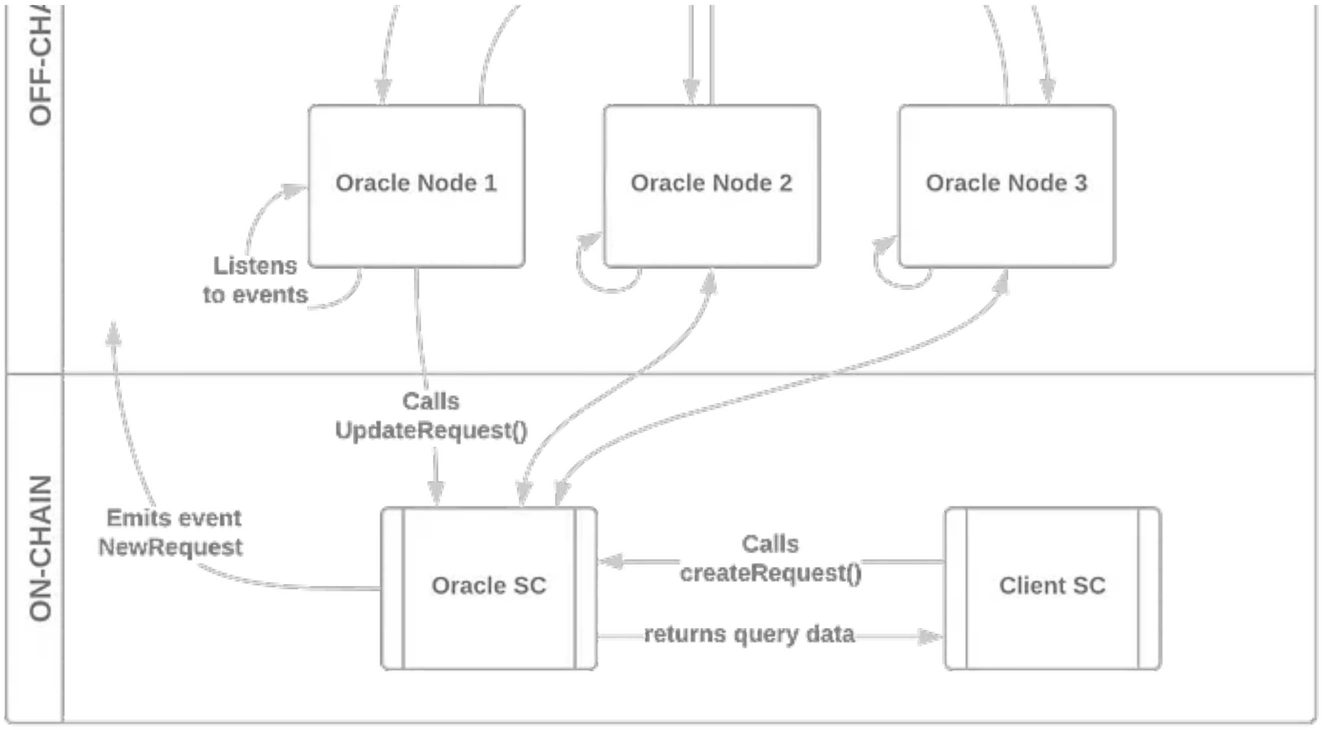
Can anyone see the pros and cons of such a plan? How well does it meet our oracle properties?

Decentralized oracles

There is more than one oracle that we listen to. We then can take some function over the output of the oracles (maybe take the average, the median, pick a random one, pick based on reputation etc...)

Decentralized oracle design:





two out of three oracle:

Picture taken from [Pedro Costa's blog]

Would this design meet our oracle properties better?

Can someone argue why this is not decentralized and is no better than a centralized oracle?

Implementation example:

Taken from Pedro costa's blog

```
contract Oracle {
  Request[] requests; //list of requests made to the contract
  uint currentId = 0; //increasing request id
  uint minQuorum = 2; //minimum number of responses to receive before declaring final resu
  uint totalOracleCount = 3; // Hardcoded oracle count
}


// defines a general api request
struct Request {
  uint id;                            //request id
  string urlToQuery;                  //API url
  string attributeToFetch;            //json attribute (key) to retrieve in the response
  string agreedValue;                 //value from key
```

```
    mapping(uint => string) anwers;   //answers provided by the oracles
    mapping(address => uint) quorum;  //oracles which will query the answer (1=oracle hasn't
  }


  // Client requesting data from the oracles (using a specific api)

  function createRequest (
    string memory _urlToQuery,
    string memory _attributeToFetch
  )
  public
  {
    uint lenght = requests.push(Request(currentId, _urlToQuery, _attributeToFetch, ""));
    Request storage r = requests[lenght-1];

    // Hardcoded oracles address
    r.quorum[address(0x6c2339b46F41a06f09CA0051ddAD54D1e582bA77)] = 1;
    r.quorum[address(0xb5346CF224c02186606e5f89EACC21eC25398077)] = 1;
    r.quorum[address(0xa2997F1CA363D11a0a35bB1Ac0Ff7849bc13e914)] = 1;

    // launch an event to be detected by oracle outside of blockchain
    emit NewRequest (
      currentId,
      _urlToQuery,
      _attributeToFetch
    );

    // increase request id
    currentId++;
  }


  //event that triggers oracle outside of the blockchain
   event NewRequest (
    uint id,
    string urlToQuery,
    string attributeToFetch
   );


  //called by the oracle to record its answer
  function updateRequest (
    uint _id,
    string memory _valueRetrieved
  ) public {

    Request storage currRequest = requests[_id];

    //check if oracle is in the list of trusted oracles
    //and if the oracle hasn't voted yet
    if(currRequest.quorum[address(msg.sender)] == 1){
```

```
    if(currRequest.quorum[address(msg.sender)] == 1){

      //marking that this address has voted
      currRequest.quorum[msg.sender] = 2;

      //iterate through "array" of answers until a position if free and save the retrieved va
      uint tmpI = 0;
      bool found = false;
      while(!found) {
        //find first empty slot
        if(bytes(currRequest.anwers[tmpI]).length == 0){
          found = true;
          currRequest.anwers[tmpI] = _valueRetrieved;
        }
        tmpI++;
      }

      uint currentQuorum = 0;

      //iterate through oracle list and check if enough oracles(minimum quorum)
      //have voted the same answer has the current one
      for(uint i = 0; i < totalOracleCount; i++){
        bytes memory a = bytes(currRequest.anwers[i]);
        bytes memory b = bytes(_valueRetrieved);

        if(keccak256(a) == keccak256(b)){
          currentQuorum++;
          if(currentQuorum >= minQuorum){
            currRequest.agreedValue = _valueRetrieved;
            emit UpdatedRequest (
              currRequest.id,
              currRequest.urlToQuery,
              currRequest.attributeToFetch,
              currRequest.agreedValue
            );
          }
        }
      }
    }
```