Mobile Network

Course Project

Final Report

Arbitrary Review System of Essay Based on Blockchain

Author:	*Student Number:*
赵倩妤	515030910555
韩雨轩	515030910552

May 27, 2018

Contents

1 Project Background

1.1 Current System

Bidirectional anonymous system is since long a mechanism used for reviewing essays by plenty of journal or conference. In this mechanism, author's information is hided from editor group while authors also know nothing about reviewers. This method helps to form an objective and fair estimation and to enhance the academic quality.

But there is an obvious shortcoming, that the chair of this system, who is in responce to allocate essays, can still control the matching process. That is, all the systems nowadays are not enough anonymous and arbitrary.

Since we know that the main character of blockchain technology is decentration, can we realize an totally arbitrary review system of submitted essay based on blockchain?

1.2 System Request

- Totally arbitrary allocation;
- Matched pair cannot be modified;
- Matching information among entire network broadcasting;
- Fair and uniform matching.

2 System Introduction

2.1 Blockchain Technology

Blockchain is a technology used for distributed data storage, it keeps features of encryption, decentration and anti-fake security.
The information is distributed stored in many data 'block's. The entire network takes part in maintaining a same 'record' and protecting it from being modified.
To encourage this security mechanism, some rewards are given to the one who successfully find a new block. And this step is usually called 'Mining'. All nodes inside this network can run an SHA-256 algorithm in order to achieve a certain standard whose difficulty can be changed.
Therefore, When many nodes participate in running block-finding algorithm, we won't exactly know which node will successfully find a new block. That is, when the entire network is working on mining, the achiever is arbitrary!

2.2 System Design

Encouraged by this arbitrary feature, our essay-review system's concepts are shown as following:

- Each reviewer is denoted as a node, while the entire editor group equals to a network;
- To keep all wait-for-reviewing-essays in a pool. Each time when a new block is found, we select an essay to the finder and record this matching pair;
- The matching pair of the previous block is the information which is going to be stored in this block;
- All nodes are forced to operating the block-finding algorithm.

2.3 Consensus Mechanism

Also, a consensus mechanism is used to identify who is the real new finder when multiple nodes succeed in accomplishing the algorithm inside a certain time slice. To ensure that essays are alllocated as balanced as possible, we choose a consensus mechanism similar to POS:

The total number of matched essays of each reviewer is recorded, when conflict, system will write the one whose number is smallest as a new finder.

2.4 Overview of the whole system

This system works like this :
1. Each reviewer uses previous block address, last matched pair information, an arbitrary number to mine a new block;
2. When a reviewer accomplishes, an essay is selected to him;
When multiple reviewers accomplish at the same time, an essay is selected to the one with smallest number of essays;
3. This matched information is recorded by the system and broadcasted to all reviewers.
4. When reviewer receives a broadcast, he updates the algorithm and keeps running for next turn.

3 Program Code

3.1 Block chain

First, to build the system, we consider to build up a block chain. In the code, a class named ”blockchain” is defined. In this class, there are seven functions, ”init”, ”register_node”, ”new_block”,”new_transaction”, ”last_block”, ”hash”, and ”valid_chain”.

- ”init”. This function initialize the class blockchain. Three attributes ”current_transaction”, ”chain”, ”nodes” and one function ”new_block” are defined. ”current_transaction” stored the information of the current matching between reviewer and essay. ”chain” is about the exit block chain we build. And ”nodes” stored all the names of the reviewer and it is stored in the container ”set”, which makes sure that there will not be two reviewers with the same name in the system.

```
def __init__(self):
    self.current_transactions = []
    self.chain = []
    self.nodes = set()

    # Create the genesis block
    self.new_block(proof=100)
```

- ”rigister_node”. This function takes the name of the reviewer as parameter and works for adding the name of reviewers to the set ”nodes”. If the name already exists, throw out an error to inform that the name already exists in the system.

```
def register_node(self, name):
    """
    Add a new node to the list of nodes
    :param name: the name of the reviewer, it can not be the same as an exited one
    """
    if name in self.nodes:
        raise ValueError('Name already exit!')
    else:
        self.nodes.add(name)
```

- ”new_block”. This function takes the proof we calculated by proof of work algorithm and creates a new block. Create a new block with attributes ”index”, ”timestamp” which is the time of the creation of the block, ”transactions” which is the match of the reviewer and essay that we try to store in block chain, ”previous_hash” which is the proof value of the last block, ”proof” which is the proof value of the current block. Take two conditions into consideration, while the block chain is empty and while it is not. If the block chain is empty, set the ”previous_hash” as a default value 1.

```
def new_block(self, proof):
    """
    Create a new Block in the Blockchain

    :param proof: The proof given by the Proof of Work algorithm
    :return: New Block
    """
    if(len(self.chain)!=0):
        block = {
            'index': len(self.chain) + 1,
            'timestamp': time.time(),
            'transactions': self.current_transactions,
            'previous_hash': self.hash(self.chain[-1]),
            'proof':proof,
                }
        # Reset the current list of transactions
        self.current_transactions = []

        self.chain.append(block)
        return block
    else:
        block = {
            'index': len(self.chain) + 1,
            'timestamp': time.time(),
            'transactions': self.current_transactions,
            'previous_hash': 1,
            'proof':proof,
        }
        # Reset the current list of transactions
        self.current_transactions = []

        self.chain.append(block)
        return block
```

- "new_transaction". This function takes the name of reviewer and essay as parameters and it works to add the match to the attribute "transaction" defined as part of "blockchian"

```
def new_transaction(self, reviewer, essay):
    """
    Creates a new transaction to go into the next mined Block

    :param reviewer: Name of the reviewer
    :param essay: Name of the essay
    """
    self.current_transactions.append({
        'reviewer': reviewer,
        'essay': essay,
    })
```

- "last_block". It returns the last block of the block chain

```
def last_block(self):
    return self.chain[-1]
```

- "hash". It is the hash function used in the proof of work algorithm. It calculated the hash value of the current stored information of the block.

```python
def hash(self,block):
    """
    Creates a SHA-256 hash of a Block

    :param block: Block
    """

    # We must make sure that the Dictionary is Ordered, or we'll have inconsistent hashes
    #block_string = json.dumps(block, sort_keys=True).encode()
    block_string = str(block['transactions']).encode()
    return hashlib.sha256(block_string).hexdigest()
```

- "valid_chain". This function works to verify that whether the chain is valid or not by comparing the proof calculated and the proof stored. If data on one of blocks has been changed, the proof will be changed and the function will return the value false. On the other hand, it will return true.

```python
def valid_chain(self):
    """
    Determine if a given blockchain is valid
    :return: True if valid, False if not
    """

    last_block = self.chain[0]
    current_index = 1

    while current_index < len(self.chain):
        block = self.chain[current_index]
        print(f'{last_block}')
        print(f'{block}')
        print("\n-----------\n")
        # Check that the hash of the block is correct
        last_block_hash = self.hash(last_block)
        if block['previous_hash'] != last_block_hash:
            return False

        # Check that the Proof of Work is correct
        #if not self.valid_proof(last_block['proof'], block['proof'], last_block_hash):
        #    return False
                                                    PEP 8: block comment should start with '# '
        last_block = block
        current_index += 1

    return True
```

Second, we try to build the system based on the block chain. To decide which reviewer the essay will be allocated to, the proof of work algorithm is used. In this section, different threads are used to represent different reviewers, and different delay time is given to simulated the different ability of each reviewers. In each threads, the function "proof_of_work" is used to calculate the proof, the code is shown as below.

```python
def proof_of_work(name, delay):
    """
    Simple Proof of Work Algorithm:
    - Find a number p' such that hash(pp') contains leading 4 zeroes
     - Where p is the previous proof, and p' is the new proof
    :param name: name of the reviewer
    :param delay: delay of each process
    :return: proof
    """

    last_block = blockchain.last_block()
    last_proof = last_block['proof']
    last_hash = blockchain.hash(last_block)

    proof = 0
    global flag
    while valid_proof(last_proof, proof, last_hash) is False:
        if flag:
            return proof
        proof += 1
        #print("proof2:" + str(proof))
        time.sleep(delay)
    '''
    flag=1
    print("name:" + name)
    global chosenreviewers
    if chosenreviewers == []:
        chosenreviewers.append(name)

    '''
    print("name",name)
    print("flag",flag)
    #print("prooftest:"+str(proof))
    flags[name] = 1
    '''
    while(1):
        if flag:
            break
    '''

    return proof
```

```python
def valid_proof(last_proof, proof, last_hash):
    """
    Validates the Proof

    :param last_proof: <int> Previous Proof
    :param proof: <int> Current Proof
    :param last_hash: <str> The hash of the Previous Block
    :return: <bool> True if correct, False if not.

    """

    guess = f'{last_proof}{proof}{last_hash}'.encode()
    guess_hash = hashlib.sha256(guess).hexdigest()
    #return guess_hash[:4] == "0000"
    return guess_hash[:1] == '0'
```

As we mentioned before, different numbers are given as time of delay to different reviewers, which is decided randomly. Moreover, to make sure the essay can allocated to different reviewers instead of allocating constantly to a specific one because of the random number, the number of essay a reviewer already allocated to is taken into consideration as part of the time of delay. By testing for 50 and 100 iterations and calculating the average variance of each round, it can be told by the result that the consideration of allocated number can make the distribution more average.

	delay = r			
	reviewer1	reviewer2	reviewer3	Variance
iteration = 50	5.1	3.24	3.66	3.15
iteration = 100	4.48	3.8	3.72	2.20

	delay = r + number *50			
	reviewer1	reviewer2	reviewer3	Variance
iteration = 50	4.02	3.94	4.04	0.20
iteration = 100	4.05	3.99	3.96	0.22

Ultimately, consider the situation when two or more reviewers happen to find out the proof and finish at the same time. In this situation, we use the proof of stake algorithm, which, in our system, is to allocate the essay to the reviewer with the same allocated essay. To achieve this goal, if one thread is completed, "flags[i]" is assigned to 1 and another thread is set to find out the reviewer with its flag being 1 every 1ms.

```
def exam(delay):
    '''
    while(True):
        testing = 0
        for name in flags:
            if flags[name]:
                testing = 1
                #chosenreviewers.append(name)


        global flag
        if testing:
            flag = 1
    '''

    while True:
        global flag
        for name in flags:
            if flags[name]:
                flag = 1

        time.sleep(delay)
        flag = 0
```

So the main function will be like below. Traverse every essay in the essay pool, start a thread for every reviewer, find the one that the essay should be allocated to and create a new block of the block chain to store the matching information.

```
while len(essaypool) != 0:
    #chosenreviewer=randomchoose()
    #print("chosen:"+chosenreviewer)
    flag = 0
    init(flags)
    print("initflags:")
    print(flags)
    chosenreviewers = []
    calculating(reviewers)


    print("flags:")
    print(flags)
    for name in flags:
        if flags[name]:
            chosenreviewers.append(name)

    print("chosen:")
    print(chosenreviewers)
    chosenreviewer=pos(chosenreviewers)

    #chosenreviewer=chosenreviewers[0]
    sequence.append(chosenreviewer)
    blockchain.new_transaction(chosenreviewer, essaypool[0])

    blockchain.new_block(totalproof)
    counting[chosenreviewer] = counting[chosenreviewer]+1
    distribution[chosenreviewer].append(essaypool[0])
    essaypool.remove(essaypool[0])
```

3.2 User Interface

Based on python environment, we choose Tkinter as UI frame language packet. Tkinter is a Python binding to the Tk GUI toolkit. As with most other modern Tk bindings, Tkinter is implemented as a Python wrapper around a complete Tcl interpreter embedded in the Python interpreter. Tkinter calls are translated into Tcl commands which are fed to this embedded interpreter, thus making it possible to mix Python and Tcl in a single application.

Generally we need two input window for us to add essays or reviewers. And also, there ought to be a function supporting query, the both directional matching query. Therefore totally three input boxes and four input confirm buttons are needed.

As for the output, the result of query should be shown as well as the real time matching information. So two show windows are placed.

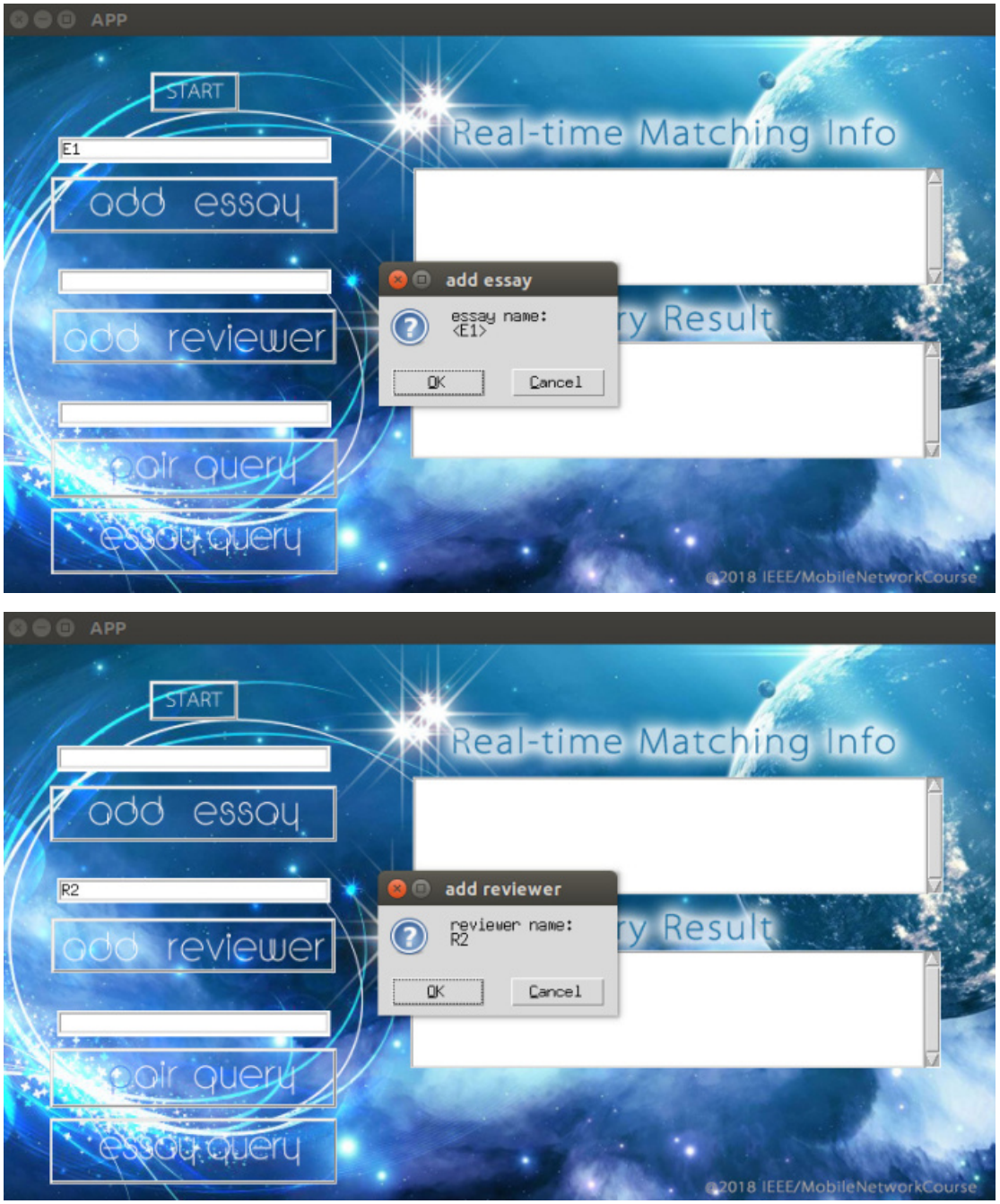
At first we set a timer, after a certain period of time the system will fresh itself and update the matching information. But there exists a problem, that input information and matching process will be out of order. That is, the matching process starts before user finish inputting required information. To avoid this chaos, we add a 'start' button. Each time the user finish inputting essay or reviewer information, he

can press this button to launch the matching system.

4 Program Exhibition

The user interface runs like this:

There will popup a confirm window while adding essay or reviewer:

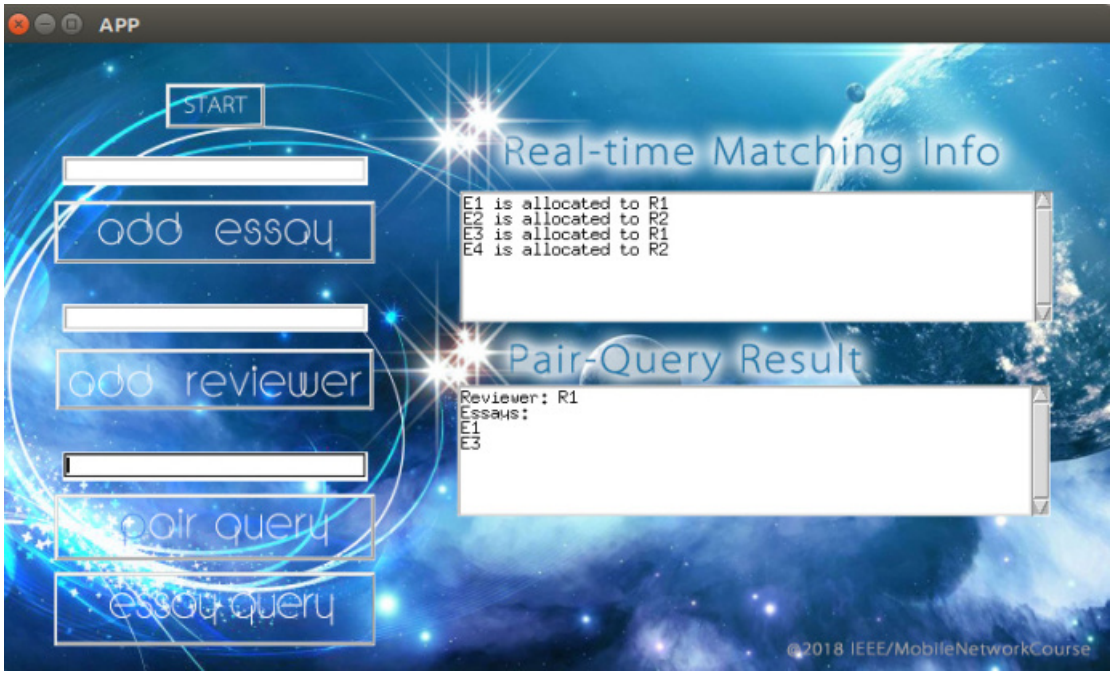


Press 'start' button, the system will begin allocating essays. When a pair is matched, the matching information will be shown on the up show-window.

The user can input essay or reviewer name to query the current matching informa-

tion:


APP
START
add essay
add reviewer
pair query
essay query
Real-time Matching Info
E1 is allocated to R1
Pair-Query Result
Essay: E1
Reviewer: R1
©2018 IEEE/MobileNetworkCourse


APP
START
add essay
add reviewer
pair query
essay query
Real-time Matching Info
E1 is allocated to R1
E2 is allocated to R2
E3 is allocated to R1
E4 is allocated to R2
Pair-Query Result
Reviewer: R1
Essays:
E1
E3
©2018 IEEE/MobileNetworkCourse

5 Task Division

赵倩妤: Blockchain Code, System Testing
韩雨轩: System Design, UI