Anonymous Paper-Review System Based On Block-Chain

Reporter: Hongyu Chen

Group: Hongyu Chen, Zhiyang Lv, Jinjin Chen

Environment: Win 8, Python 3

Background:

Almost every research academic is aware of the current environment in which both institutions and individuals are subject to some form of assessment and a considerable proportion of the scoring is often weighted on the production of scholarly outputs, typically as papers in reputable journals.

Apart from the obvious authors and readers, these journals typically have a small number of editors who do most of the day-to-day management of the intellectual material; there might also be a larger editorial pool from the discipline who oversee the journal, and then there are those people who take on the role of reviewers which could include all of the previously listed groups of people.

The production of a journal also include publishers who publish for profit or publish through a non-profit entity such as a learned society. Irrespective of the details of the process, the production of an academic journal is an economic system and like any such system the publishing process presents a number of intended and unintended incentives.

Due to the anonymous nature of peer review, people seem to have lesser incentive to carry out this part of the publication process. Although some journals will list those that have carried out reviews on its behalf, it is more difficult to provide a measure of the number, quality and timeliness of those reviews. The paper considers how it might be possible to provide a greater incentive to carry out peer review and to raise the quality of those reviews using some of the currently available internet-based technologies. The proposal also provides a mechanism to track the review process and has the potential for wider application within the academic publishing environment as an alternative metric to those used at present.

Proposed System :

Block-chain :

The incentive to improve the peer review process might to be to pay the reviewers some real money but obviously it's not a good option here for there are potential conflicts of interest. Instead, the technology of Block-Chain allows for alternative currency exchange mechanisms, and we can found a new exchange system using the cryptocurrency similar to bit-coin (called r-coin in the paper referred to 'ReviewCoin')

The Block-Chain has the advantages:

I. It is a distributed accounting system not owned by any specific organization. No one has the only right to modify the system.

II. item It is difficult to corrupt and transactions are anonymous, which means that it is difficult for 'bad guys' to modify the review due to interfere the publication of a paper.

R-coin system :

In order for this exchange system to work, authors need to register an ID using a system such as ORCID to confirm their identifications and then submit their ID to the journal or through Publons when doing a peer review. Then they are paid with r-coins. The exchange mechanism will be talked at next part.

The new journals needs a registration process to join the r-coin currency in case of unscrupulous people from setting up publications that do not conform to the rules agreed by the community. Also, a lot of journals, are not so fabulous that should be excluded from participating. A body that decides which journals were acceptable is needed.

Finally, the r-coin is just 'present' for review. The r-coin system only provides a "right to publish" mechanism. It couldn't be access to the 'real' money transaction.

Transaction Mechanism :

I. Pay for publication: The author should pay some r-coin to require a peer review, but this will have many conditions. When multiple authors are involved in submitting a paper, then should the cost be shared equally, or only the corresponding author pay or the authors could decide their own r-coin contribution that could be a reflection of their contributions to the paper.

The cost of submitting a paper to a journal would need to be agreed and whether that cost was the same for all journals or journals could set their own price.

II. Earn from review : The participant will only earn the r-coins from review. However, the payment of the amount of r-coin could also be related to timeliness where less r-coin is paid if a review is late. The reviewers would be paid more r-coins per review the more reviews that were completed.

III. Expansion of currency pool : There are two ways to expand the r-coins pool:

New registration : An author would be given a certain amount of r-coin during the ID registration process. Every new author will have an initial r-coin budget so that they can then start submitting papers to journals. This also resolve the lack of budget for a new research.

Review : The r-coin is paid once an editor accepts the review in terms of meeting the journal's quality requirements. But the payment of the amount of r-coin can be changed in each transaction(review) which depends on the factors mentioned above, thus the r-coins that an author pay is not equal to the total of what the reviewers receive. In this method, the pool expand and it also encourage peer review.

My Work :

Achieving the rudimentary frame of the system by code, which comprises the part of follows:

I. Part of contributor and its relevant functions

II. Part of reviewers and its relevant functions

III. Creation of block-chain

IV. Recording of the currency

V. The incentive mechanism

Details :

Block-Chain: According to the properties of block-chain, its pivotal part the the hash value, which corresponded with its index, time stamp, data and the hash value of the previous block. The class is defined as follows:

```
class Block:
    review = 0  # check if reviewed: 0-not review | 1-pass | 2-no pass
    value = 3  # the review reward carried is 3 Bitcoin

    def __init__(self, index, timestamp, data, previous_hash):
        self.index = index
        self.timestamp = timestamp
        self.data = data
        self.previous_hash = previous_hash
        self.hash = self.hash_block()

    def hash_block(self):
        sha = hasher.sha256()
        sha.update((str(self.index) +
                    str(self.timestamp) +
                    str(self.data) +
                    str(self.previous_hash)).encode("utf-8"))
        return sha.hexdigest()
```

'value' is the currency saved in the block and 'review' represents the review status of the the block.

Contributor: In this class, its main function is contributing, and the ceiling of the number of the paper one contributor contributes one time can be set with a threshold. What's more, for each paper, the contributor should pay a sum of money and its definition is as follow:

```
class Contributor:

    def __init__(self, id, dep):
        self.id = id
        self.dep = dep

    def contribute(self, num_paper):
        if self.dep < num_paper:
            print("contributor {} deposit not enough!".format(self.id))
            return
        self.dep -= num_paper * 3  # default cost is 3 per paper
        for _ in range(0, num_paper):
            contribute_pool.append(self.id)
```

It also carries the info of the deposit of the contributor.

Reviewer: The crucial function of this character should be the reviewing. Also, the definition of the class should record the reviewer's deposit and for the review function, there are some extra details to be considered: Whether the

paper pass the review? The maximum of the paper a contributor should review at one time in case of the malicious brush? The distribution of the fortune saved in the block?

For the first question, we can use a flag to mark if a paper passed. For the second question, a perfect solution is set the ceiling and use the Producer-Consumer model to achieve the real time application, and I simplified the model and just use a threshold. For the third question, since the sole way to increase the currency in the system is the creation of contributor, in which process the system will give a ration of money to the new user. So in order to achieve the self-supply function, we should guarantee the system will get a quota of money in the block during the distribution. The proportion of the money that reviewer can acquire and the system and acquire need careful consideration to make profits, this work is suspended at present and we temporarily define the system will get one third and reviewer will get two third of the money.

The definition of reviewer is as follow:

```
class Reviewer:
    fortune = 0

    def __init__(self, id, paper_store):  # paper_store is a list(save the block)
        self.id = id
        self.paper_store = paper_store

    def review(self):
        global deposit_system
        if len(self.paper_store) == 0:
            return
        while self.paper_store:
            to_review = self.paper_store.pop()
            self.fortune += (to_review.value - 1)  # get a part of value of the paper
            deposit_system += 1  # put 1 to the deposit pool
            blockchain[to_review.index].value = 0
            # simulate the review process(set delay)
            time.sleep(random.uniform(0, 0.5))
            blockchain[to_review.index].review = random.randint(1, 2)
```

Distribution: In my work, I use a random algorithm to distribute the papers to reviewers: We choose a qualified reviewer randomly(he hasn't get his ceiling) and then give him the paper, and meanwhile, the system will get a third of the money in the block which carries the paper and save it in the system's deposit pool(SDP). This process is defined with review process as follow:

```
def distribution_review():
    # distribution
    print("begin distribution: ========================================\n")
    full_load = False  # check if full loaded which means each reviewers has its maximum number of papers to review
    while True:
        if review_finish():
            break
        if full_load:
            break
        for item in blockchain:
            if full_load:
                break
            if item.review == 0:
                who = random.randint(0, len(reviewer)-1)
                if len(paper_pool[who]) < 13:  # one contributor can at most choose 13 papers
                    paper_pool[who].append(item)
                    blockchain[item.index].review = 1  # temporary mark, representing this paper has been chosen
                else:
                    iteration_time = 0
                    while len(paper_pool[who]) >= 13:
                        if iteration_time == len(reviewer):
                            print("CAUTION : Need more reviewers ! There are surplus papers to be distributed !\n")
                            full_load = True
                            break
                        who = (who + 1) % len(reviewer)
                        iteration_time += 1
                    if not full_load:
                        paper_pool[who].append(item)
                        blockchain[item.index].review = 1  # temporary mark, represent this paper has been chosen
    # review
    print("begin review: ========================================\n")
    print("Some time needed ! Waiting...\n")
    for o in range(0, len(reviewer)):
        reviewer[o].paper_store = paper_pool[o]
    for p in reviewer:
        p.review()

    # record the fortune of reviewers
    with open("reward_reviewer.txt", "w") as f:
        for j in range(0, len(reward)):
            if j < len(reviewer):
                f.write("{}\n".format(reviewer[j].fortune))
            else:
                f.write("0\n")
```

The final sub-part is used to record the deposit of reviewers.

Feedback: The is the final crucial part, which used to exert the influence of incentive layer. Specifically, in the light of the status of paper reviewed by the reviewers, if the the paper passed, the corresponding contributor will get the remuneration from the SDP; nonetheless, the corresponding contributor will get nothing is the paper failed in reviewing. And the definition of the process is as follow:

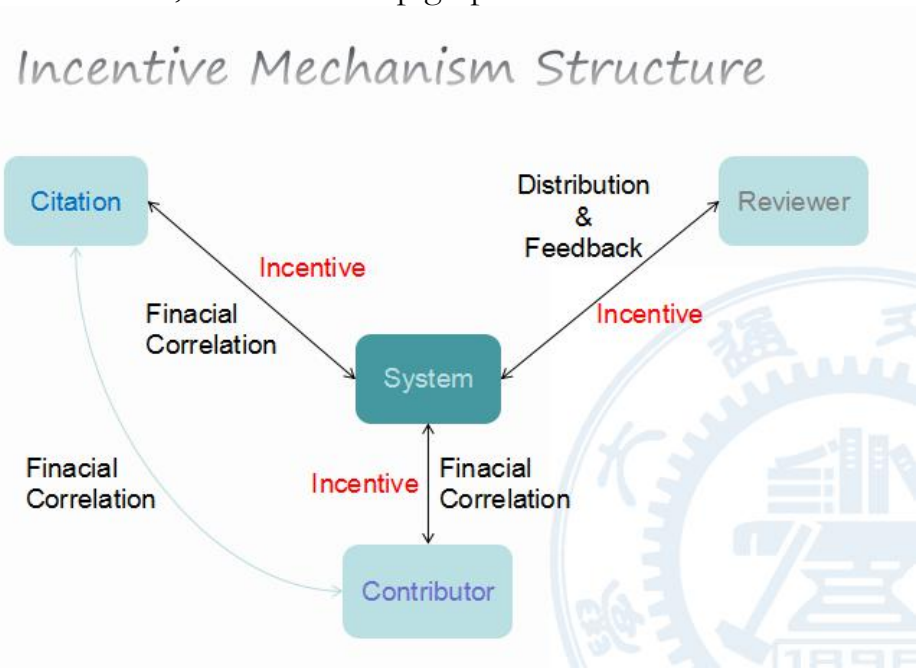
```
# ---------------------------------------------------------------------
# about feedback -- if some contributor's paper have been passed, then he will get remuneration
def feedback():
    global deposit_system
    print("feedback process: ========================================\n")
    for b in blockchain:
        if b.review == 1:
            if deposit_system < 4:
                print("NO MONEY IN REWARD POOL !!!")  # check if there has enough money
                return
            deposit_system -= 4  # if pass, the money is given by deposit pool
            contributor[int(b.data.split(" ")[-1])].dep += 4  # if pass, contributor will get 2 Bitcoin per paper

    # record the deposit of contributors
    with open("deposit_contributor.txt", "w") as f:
        for j in range(0, len(deposit)):
            if j < len(contributor):
                f.write("{}\n".format(contributor[j].dep))
            else:
                f.write("0\n")

    # record the deposit of pool
    with open("deposit_pool.txt", "w") as f:
        f.write("{}\n".format(deposit_system))
```

Compendium :

What I've done is achieve the skeleton of this system, especially its incentive mechanism, the relationship graph is as follow:



The work I've done can maintain the rudimentary frame and satisfy the fundamental requirement of the incentive layer of the system.

The concrete implement is achieved and the running result is consistent with the anticipated result, which means it exerts influence in the anonymous paper-review system successfully.

Code :

Since there is no way to submit the accessory of code. My whole works (code form) are as follows:

```
# -*- coding:utf-8 -*-
"""
Info:       This is a rudimentary frame of a anonymous system base on the idea of blockchain. You can run it to
            simulate the process of the real system. All parts are encapsulated well already, the command will
            be illustraed when you run it, and the key word is 'chy'.

Author: 陈泓宇

Last modified time: 24th/May/2018
"""


import hashlib as hasher
import datetime as date
import random
import time


# ----------------------------------------------------------------
# about initializing


deposit = []    # record the deposit of contributor
reward = []    # record the reward of each reviewer
deposit_system = 1000    # record the total deposit of system


# initialize the reward list (reviewer's fortune list), default initialized number is 100
def initialize_reward():
    with open("reward_reviewer.txt", "w") as f:
        for _ in range(0, 100):
            f.write("0\n")


# initialize the reward list (reviewer's fortune list), default initialized number is 100
def initialize_deposit():
```

```
    with open("deposit_contributor.txt", "w") as f:
        for _ in range(0, 100):
            f.write("100\n")


# initialize the reward pool, default initialized number is 1000
def initialize_deposit_pool():
    with open("deposit_pool.txt", "w") as f:
        f.write("1000\n")


# load the reward info saved in txt
def load_reward_data():
    """
    with open("reward_reviewer.txt", "w") as f:
        for _ in range(0, 100):
            f.write("0\n")
    """
    for line in open("reward_reviewer.txt", "r"):
        reward.append(int(line))


# load the deposit info saved in txt
def load_deposit_data():
    """
    with open("reward_reviewer.txt", "w") as f:
        for _ in range(0, 100):
            f.write("0\n")
    """
    for line in open("deposit_contributor.txt", "r"):
        deposit.append(int(line))


# load the deposit pool info saved in txt
def load_deposit_pool():
    """
    with open("reward_reviewer.txt", "w") as f:
        for _ in range(0, 100):
            f.write("0\n")
    """
    global deposit_system
```

```
    for line in open("deposit_pool.txt", "r"):
        deposit_system = int(line)


# ----------------------------------------------------------------------
# About chainblock creation


class Block:
    review = 0    # check if reviewed: 0-not review | 1-pass | 2-no pass
    value = 3    # the review reward carried is 3 Bitcoin

    def __init__(self, index, timestamp, data, previous_hash):
        self.index = index
        self.timestamp = timestamp
        self.data = data
        self.previous_hash = previous_hash
        self.hash = self.hash_block()

    def hash_block(self):
        sha = hasher.sha256()
        sha.update((str(self.index) +
                    str(self.timestamp) +
                    str(self.data) +
                    str(self.previous_hash)).encode("utf-8"))
        return sha.hexdigest()


def create_genesis_block():
    # Manually construct a block with
    # index zero and arbitrary previous hash
    s = "Genesis Block! This block is contributor " + str(paper[0].split(":")[0])
    return Block(0, date.datetime.now(), s, "0")


paper = []    # suppose for one paper-save pool we have 100 papers


def contributor_pool_to_paper():
    for p in range(0, len(contribute_pool)):
        # in paper list, each unit is "contributor's id : Article "
```

```
            paper.append(str(contribute_pool[p]) + ":" + "Article info")


def next_block(last_block):
        this_index = last_block.index + 1
        this_timestamp = date.datetime.now()
        # data: contributor's id + article
        this_data = "This block is contributor " +
str(paper[this_index].split(":")[0])
        this_hash = last_block.hash
        return Block(this_index, this_timestamp, this_data, this_hash)


# create the genesis block
blockchain = []


# how many blocks should we add to the chain after the genesis block
# add blocks to the chain
def add_blocks_to_chain(num_of_blocks_to_add):
        previous_block = blockchain[0]
        for _ in range(0, num_of_blocks_to_add):
                block_to_add = next_block(previous_block)
                blockchain.append(block_to_add)
                previous_block = block_to_add
                # Tell everyone about it!


def print_chain():
        print("Block-Chain                                                    Info:
==========================================")
        for i in range(0, len(blockchain)):
                print("Block #{} has been added to the
blockchain!".format(blockchain[i].index))
                print("review: {}".format(blockchain[i].review))
                print("value: {}".format(blockchain[i].value))
                print("Data: {}".format(blockchain[i].data))
                print("timestamp: {}".format(blockchain[i].timestamp))
                print("Hash: {}\n".format(blockchain[i].hash))
```

```
# ------------------------------------------------------------------------------
# about contributor


class Contributor:

    def __init__(self, id, dep):
        self.id = id
        self.dep = dep

    def contribute(self, num_paper):
        if self.dep < num_paper:
            print("contributor {} deposit not enough!".format(self.id))
            return
        self.dep -= num_paper * 3    # default cost is 3 per paper
        for _ in range(0, num_paper):
            contribute_pool.append(self.id)


# contributors list
contributor = []
# save the contributed paper's owner's id
contribute_pool = []


def create_contributor(number):
    for t in range(0, number):
        people = Contributor(t, deposit[t])
        contributor.append(people)


def contribution():
    print("begin                                                    contribution:
==========================================\n"
)
    for p in range(0, len(contributor)):
        # simulate the contribution process, one contributor at most can
contribute 5 papers
        contributor[p].contribute(random.randint(0, 5))
```

```
# ---------------------------------------------------------------------------
# about distribution and review


class Reviewer:
    fortune = 0

    def __init__(self, id, paper_store):    # paper_store is a list(save the block)
        self.id = id
        self.paper_store = paper_store

    def review(self):
        global deposit_system
        if len(self.paper_store) == 0:
            return
        while self.paper_store:
            to_review = self.paper_store.pop()
            self.fortune += (to_review.value - 1)    # get a part of value of
the paper
            deposit_system += 1    # put 1 to the deposit pool
            blockchain[to_review.index].value = 0
            # simulate the review process(set delay)
            time.sleep(random.uniform(0, 0.5))
            blockchain[to_review.index].review = random.randint(1, 2)


# reviewers list
reviewer = []
# in each pool units stores the papers to be reviewed by a certain reviewer
paper_pool = []


# create reviewers
def create_reviewer(number):
    for _ in range(0, number):
        paper_pool.append([])
    for t in range(0, number):
        people = Reviewer(t, paper_pool[t])
        # load the fortune data
        people.fortune = reward[t]
        reviewer.append(people)
```

```
# check if all the papers are reviewed
def review_finish():
	flag = True
	if len(blockchain) == 0:
		return flag
	for item in blockchain:
		if item.review == 0:
			flag = False
	return flag


# check if all the reviewers get the maximum number of papers
"""
def full_load():
	for item in paper_pool:
		if len(item) < 13:
			return False
	return True
"""


def distribution_review():
	# distribution
	print("begin                                                    distribution:
==========================================\n"
)
	full_load = False    # check if full loaded which means each reviewers has
its maximum number of papers to review
	while True:
		if review_finish():
			break
		if full_load:
			break
		for item in blockchain:
			if full_load:
				break
			if item.review == 0:
				who = random.randint(0, len(reviewer)-1)
```

```
                    if len(paper_pool[who]) < 13:    # one contributor can at most choose 13 papers
                        paper_pool[who].append(item)
                        blockchain[item.index].review = 1      # temporary mark, representing this paper has been chosen
                    else:
                        iteration_time = 0
                        while len(paper_pool[who]) >= 13:
                            if iteration_time == len(reviewer):
                                print("CAUTION : Need more reviewers ! There are surplus papers to be distributed !\n")
                                full_load = True
                                break
                            who = (who + 1) % len(reviewer)
                            iteration_time += 1
                        if not full_load:
                            paper_pool[who].append(item)
                            blockchain[item.index].review = 1          # temporary mark, represent this paper has been chosen
    # review
    print("begin                                                    review: ==========================================\n")
    print("Some time needed ! Waiting...\n")
    for o in range(0, len(reviewer)):
        reviewer[o].paper_store = paper_pool[o]
    for p in reviewer:
        p.review()

    # record the fortune of reviewers
    with open("reward_reviewer.txt", "w") as f:
        for j in range(0, len(reward)):
            if j < len(reviewer):
                f.write("{}\n".format(reviewer[j].fortune))
            else:
                f.write("0\n")


# ----------------------------------------------------------------------
# about feedback -- if some contributor's paper have been passed, then he will get remuneration
```

```
def feedback():
    global deposit_system
    print("feedback                                                    process:
==========================================\n"
)
    for b in blockchain:
        if b.review == 1:
            if deposit_system < 4:
                print("NO MONEY IN REWARD POOL !!!")    # check
if there has enough money
                return
            deposit_system -= 4    # if pass, the money is given by deposit
pool
            contributor[int(b.data.split("  ")[-1])].dep  +=  4      #  if  pass,
contributor will get 2 Bitcoin per paper

    # record the deposit of contributors
    with open("deposit_contributor.txt", "w") as f:
        for j in range(0, len(deposit)):
            if j < len(contributor):
                f.write("{}\n".format(contributor[j].dep))
            else:
                f.write("0\n")

    # record the deposit of pool
    with open("deposit_pool.txt", "w") as f:
        f.write("{}\n".format(deposit_system))


def pre_main():

    load_reward_data()
    load_deposit_data()
    load_deposit_pool()

    """
    for t in range(0, 20):
        print(reward[t])
    """
```

```
    create_contributor(50)    # how many contributors you want to hire to review, can change the 50
    contribution()    # contributors begin to contribute

    """
    print(contribute_pool)
    for item in contributor:
        print(item.dep)
    """

    contributor_pool_to_paper()        # associate contributor_pool[] with paper[]

    blockchain.append(create_genesis_block())    # add genesis block in chain
    add_blocks_to_chain(len(paper)-1)    # how many blocks you want to add in the chain associated with papers in theory

    print_chain()

    create_reviewer(10)    # how many reviewers you want to hire to review, can change the 10
    distribution_review()

    print_chain()

    feedback()

    """
    for item in contributor:
        print(item.dep)
    """


# Extra part : string      <----Transformation---->      binary code
# string to binary code
def encode(string):
    encode_str = ' '.join([bin(ord(c)).replace('0b', '') for c in string])
    return encode_str


# binary code to string
```

```
def decode(string):
    decode_str = ''.join(chr(i) for i in [int(b, 2) for b in string.split(' ')])
    return decode_str


def main():
    print("\nExtirpating previous data and initializing them ----- Command 1")
    print("Running the program based on the previous data ----- Command 2\n")
    command = int(input("Command : "))

    while command != 2:
        if command == 1:
            initialize_reward()    # when you need to eradicate the previous info and rerun at the very beginning
            initialize_deposit()    # when you need to eradicate the previous info and rerun at the very beginning
            initialize_deposit_pool()    # when you need to eradicate the previous info and rerun at the very beginning
            print("\nData has been initialized successfully !")
            command = int(input("Command : "))
        else:
            print("\nWrong command ! Input again !\n")
            command = int(input("Command : "))

    keyword = ""
    print("\nInput the keyword to enjoy this program !\n")
    while keyword != decode("1100011 1101000 1111001"):
        keyword = input("Keyword : ")
        print("\nWrong keyword ! Input again !\n")
    print("======================\nMain program begin running...\n")
    pre_main()


main()
```