Εθνικό Μετσόβιο Πολυτεχνείο

Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Κατανεμημένα Συστήματα

Αναφορά εξαμηνιαίας εργασίας

Μαυρογεώργης Κωνσταντίνος Α.Μ: 03115104
Ριζεάκου Χρύσα Α.Μ: 03117124
Μύτης Ιωάννης Α.Μ: 03113133

1.Εισαγωγή

Στην παρούσα εργασία υλοποιήθηκε ένα απλό σύστημα blockchain, το οποίο ονομάζεται Noobcash. Σκοπός του συστήματος είναι η δημιουργία και καταγραφή διαφόρων transactions και η εξασφάλιση του consensus με την χρήση του Proof-of-Work. Το σύστημα μας αποτελείται από ένα δίκτυο κόμβων, οι οποίοι είναι ταυτόχρονα miners και users που δημιουργούν transactions. Ο κάθε κόμβος είναι ουσιαστικά ένα backend, το οποίο υλοποιήθηκε με Rest api σε python με την χρήση του framework Flask. Για την επικοινωνία του χρήστη με το backend αναπτύξαμε και ένα frontend cli. Τέλος, για τον έλεγχο απόδοσης του συστήματος τρέξαμε κάποια πειράματα.
Παρακάτω παρουσιάζεται το backend, το cli καθώς και τα πειράματα και τα συμπεράσματα μας.

2.Backend

Το backend αποτελείται από έναν server σε Flask και κάθε τέτοιος server αντιστοιχεί σε έναν κόμβο του δικτύου μας. Ο server στήνεται στο αρχείο rest.py. Στο συγκεκριμένο αρχείο αρχικοποιήται η εφαρμογή μας (app) η οποία ακούει στο port και στο ip που δίνουμε σαν ορίσματα. Επιπλέον, για τον πρώτο κόμβο του δικτύου(bootstrap node) δίνουμε δύο παραπάνω ορίσματα, το bootstrap ως True και το nodes ως τον αριθμό από nodes που θα έχει το δίκτυο. Παρακάτω παρουσιάζονται τα routes στα αντιστοιχούν στα calls του api μας:

1. **/network/register**: Αυτό το route χρησιμοποιούν όλοι οι κόμβοι εκτός του πρώτου για να κοινοποιήσουν την την παρουσία τους μαζί με το ip και το port τους στον bootstrap
2. **/network/receive**: Αυτό το route χρησιμοποιεί ο bootstrap node, αφού πρώτα λάβει τις πληροφορίες όλων των άλλων κόμβων, για να στείλει τις πληροφορίες του δικτύου σε όλους του άλλους κόμβους.
3. **/network/ready**: Αυτό το route χρησιμοποιεί πάλι ο bootstrap για να ενημερώσει όλους τους άλλους κόμβους ότι έστειλε τα setup transaction και το δίκτυο είναι έτοιμο να δεχτεί transactions από τους users.
4. **/transaction/create**: Αυτό το route καλεί ο χρήστης για να δημιουργήσει ένα transaction.
5. **/transaction/receive**: Αυτό το route χρησιμοποιείται για να γίνει broadcast ένα transaction.
6. **/transaction/get**: Αυτό το route επιστρέφει τα transactions του τελευταίου block του blockchain.
7. **/block/receive**: Αυτό το route χρησιμοποιείται για να γίνει broadcast ένα block.
8. **/blockchain/send**: Αυτό το route επιστρέφει το blockchain για κάθε κόμβο και χρησιμοποιείται αν δεν σε περίπτωση που χρειάζεται consensus.
9. **/wallet/balance**: Επιστρέφει το wallet balance για κάθε κόμβο.
10. **/test/run**: Αυτό το route έχει νόημα μόνο στα πλαίσια της εργασίας και είναι υπευθυνο για να τρέχει τα πειράματα για κάθε κόμβο.

11. **/test/results**: Όπως και το /test/run, Αυτό το route έχει νόημα μόνο στα πλαίσια της εργασίας και είναι υπευθυνο για να επιστρέφει τα αποτελέσματα των πειραμάτων.

Επιπρόσθετα, το rest.py αρχικοποιεί ένα αντικείμενο της κλάσης node το οποίο ορίζεται στο αρχείο node.py. Το αντικείμενο αυτό είναι υπεύθυνο τόσο για την διαχείριση των transactions όσο και των blocks και του blockchain. Συγκεκριμένα, το node είναι υπεύθυνο να δημιουργεί και να λαμβάνει transactions, τα οποία βάζει σε μια ουρά. Στην συνέχεια, καλεί την queue_handler η οποία καλεί την add_transaction_to_block, που προσθέτει την συναλλαγή στο block και ελέγχει αν πρέπει να ξεκινήσει την διαδικασία mining. Έπειτα, κάνει mine είτε μέχρι να βρει το σωστό nonce είτε μέχρι να λάβει άλλο block. Αν λάβει άλλο block το κάνει validate και αν πετύχει το βάζει στο blockchain. Σε αντίθετη περίπτωση καλεί την resolve_conflicts, η οποία χρησιμοποιεί το /blockchain/send των υπόλοιπων κόμβων ώστε να βρει την blockchain με τους περισσότερους κόμβους και να κάνει consensus.
Για την υλοποίηση των παραπάνω το node έχει σαν μεταβλητές αντικείμενα των παρακάτω κλάσεων:

- wallet: η οποία έχει το public και private key.
- transaction: που αντιστοιχεί στα transactions και αποτελείται από sender address, receiver address, amount, signature, transaction id, transaction inputs και transaction outputs.
- block: που αντιστοιχεί στα blocks και αποτελείται από μια λίστα από transactions, καθώς και από ένα index, previous hash, που χρησιμοποιείται για το validation των blocks και το nonce που χρησιμοποιείται για την διαδικασία του mining.
- blockchain: που αντιστοιχεί στο blockchain κάθε κόμβου και είναι ουσιαστικά μια λίστα από αντικείμενα blocks.

Ακόμα, αξίζει να σημειωθεί ότι για την εύρεση του ballance κάθε κόμβου χρησιμοποιούνται τα unspent transaction outputs(utxos) τα οποία έχουν την μορφή

```
 {'id' : transaction_id,
'receiver' : sender_address,
'amount' : amount
}
```

Αυτά τα outputs ουσιαστικά δηλώνουν ότι το εξής amount έχει σταλεί στον receiver και αυτό προήλθε από το transaction με id transaction_id. Έτσι, κάθε transaction έχει κάποια inputs που καθορίζουν ποια utxos πρέπει να αφαιρεθούν από την λίστα του κάθε κόμβου καθώς και τα outputs που καθορίζουν ποια utxos πρέπει να προστεθούν στην λίστα κάθε κόμβου.

Τέλος, τα δεδομένα που ανταλλάσσονται μεταξύ κόμβων είναι σε μορφή json και για αυτόν το λόγο οι παραπάνω κλάσεις έχουν συναρτήσεις που μετατρέπουν τα αντικείμενα τους σε dictionaries ώστε να είναι json serializable.

3.Frontend - Cli

Το Cli είναι ένα πρόγραμμα σε python. Για να το τρέξουμε του δίνουμε όρισμα το ip και το port του κόμβου στον οποίο θέλουμε να συνδεθούμε. Τέλος, έχουμε ένα while στο οποίο το πρόγραμμα διαβάζει γραμμή γραμμή τις εντολές του χρήστη και κάνει requests με τον παρακάτω τρόπο:

```
Usage:
$ python cli.py -p PORT           Start as participant
Available commands:
* `t [recepient_id] [amount]`   Send `amount` NBC to `recepient`
* `view`                        View transactions of the latest block
* `balance`                     View balance of each wallet (as of last validated
block)
* `help`                        Print this help message
* `exit`                        Exit client (will not stop server)
Extra commands:
* `source [fname]`              Read and send transactions from `fname`
* `balance_all`                 View balance of each wallet (as of last received
transaction)
```

4.Αποτελέσματα Πειραμάτων και Συμπεράσματα

Με την χρήση των routes /test/run και /test/results πραγματοποιούμε τις συναλλαγές που βρίσκονται στα αρχεία testing_code/5nodes και testing_code/10nodes για 5 και 10 κόμβους αντίστοιχα. Για να το κάνουμε αυτό τρέχουμε τα αρχεία test_exec.py.

Παρακάτω παρουσιάζονται τα αποτελέσματα.

Για 5 κόμβους, capacity = 1 και mining difficulty = 4 έχουμε throughput = 0.7607612533253368 και block time = 1.2162480563056697

Για 5 κόμβους, capacity = 5 και mining difficulty = 4 έχουμε throughput = 1.2466764816625198 και block time = 3.2990502470518237

Για 5 κόμβους, capacity = 10 και mining difficulty = 4 έχουμε throughput = 1.1088906903132858 και block time = 6.584554166428431

Για 5 κόμβους, capacity = 1 και mining difficulty = 5 έχουμε throughput = 0.06448453828944851 και block time = 15.390125509726033

Για 5 κόμβους, capacity = 5 και mining difficulty = 5 έχουμε throughput = 0.10804808260205914 και block time = 44.899811175404764

Για 5 κόμβους, capacity = 5 και mining difficulty = 5 έχουμε
throughput = 0.1308290327632976 και block time = 73.50251286499429

Για 10 κόμβους, capacity = 1 και mining difficulty = 4 έχουμε
throughput = 1.139018327438119 και block time = 0.6259160381066818

Για 10 κόμβους, capacity = 5 και mining difficulty = 4 έχουμε
throughput = 2.82463507056242 και block time = 1.4557786900272356

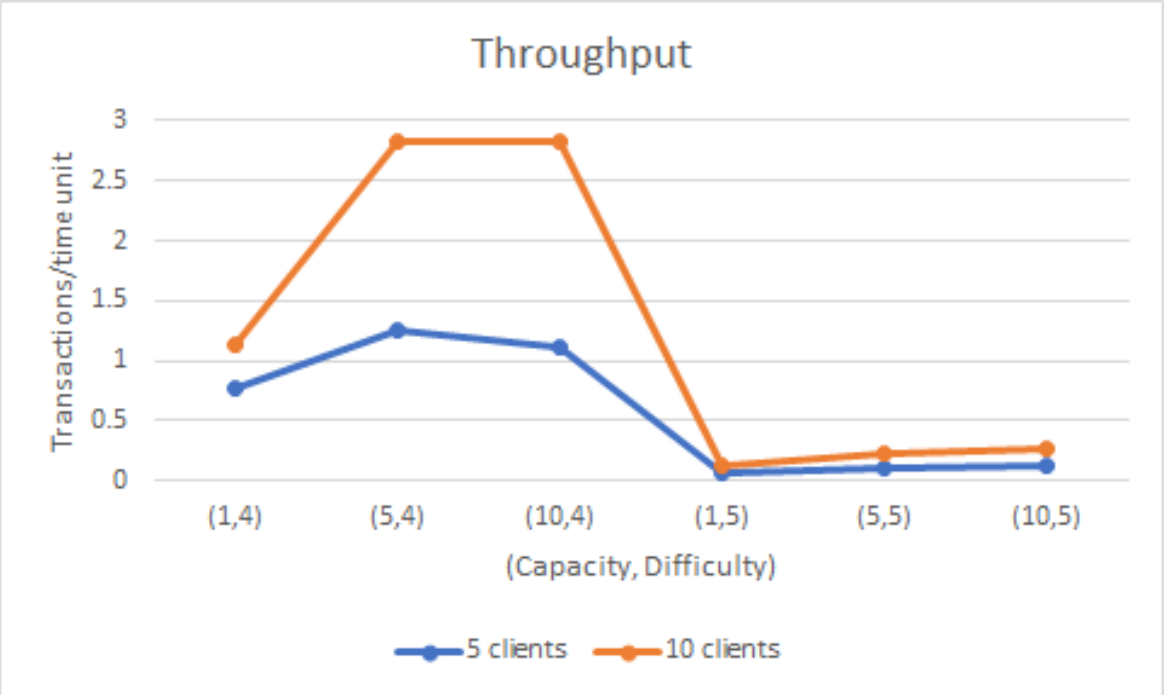
Για 10 κόμβους, capacity = 10 και mining difficulty = 4 έχουμε
throughput = 2.829131312759128 και block time = 2.5569321334835142

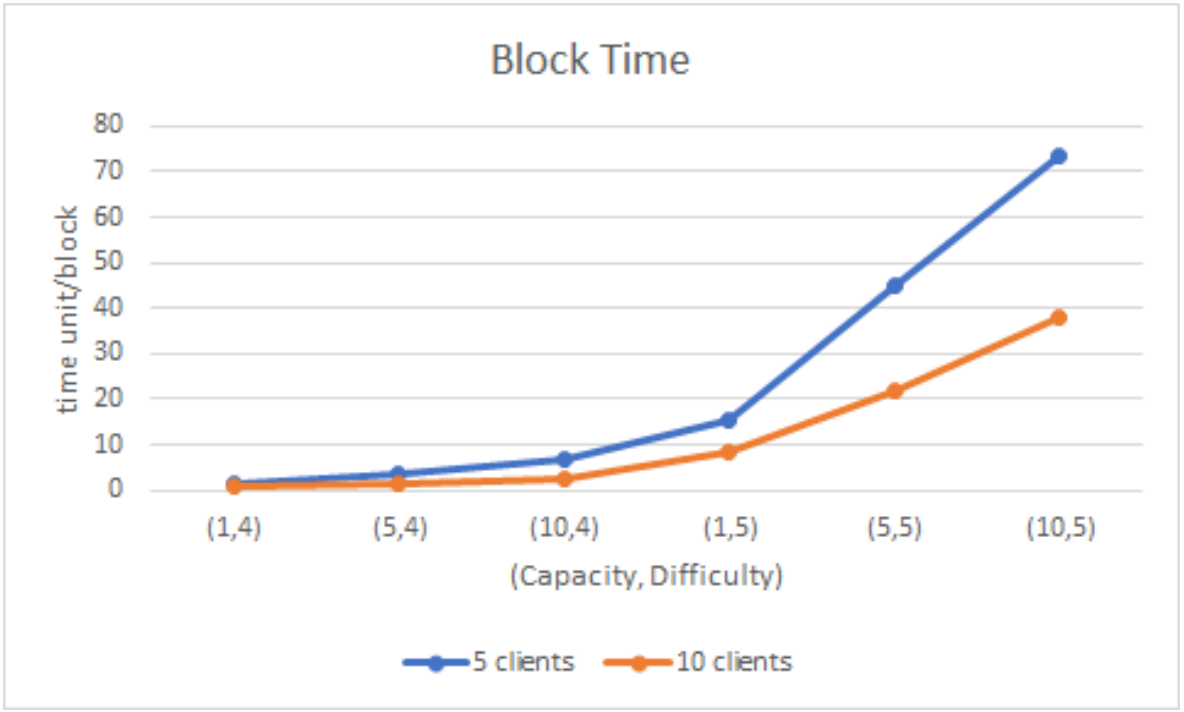
Για 10 κόμβους, capacity = 1 και mining difficulty = 5 έχουμε
throughput = 0.12119977331942784 και block time = 8.245352166066779

Για 10 κόμβους, capacity = 5 και mining difficulty = 5 έχουμε
throughput = 0.22939807321188538 και block time = 22.05506302157601

Για 10 κόμβους, capacity = 10 και mining difficulty = 5 έχουμε
throughput = 0.25861279892809985 και block time = 37.933660221776485

Στη συνέχεια ακολουθούν και τα διαγράμματα για throughput και block time.





Αρχικά, παρατηρούμε ότι για 10 nodes έχουμε βελτίωση στο throughput αλλά και στο block time συγκριτικά με τους 5 nodes. Αυτό είναι λογικό, δεδομένου ότι το κάθε block γίνεται mined από περισσότερους κόμβους με αποτέλεσμα να βρίσκεται το nonce γρηγορότερα και άρα να προστίθεται το κάθε block γρηγορότερα στο blockchain.
Έπειτα, παρατηρούμε ότι το mining difficulty επηρεάζει το block time. Αυτό το περιμέναμε επειδή αυξάνοντας την δυσκολία του mining αυξάνεται και ο χρόνος εισαγωγής του στο blockchain.
Επιπλέον, θα περιμέναμε ότι για αύξηση του capacity θα είχαμε βελτίωση του throughput. Αυτό συμβαίνει για difficulty 5 που ούτως ή άλλως το throughput είναι μικρό. Ωστόσο, για difficulty 4 το μεγαλύτερο capacity δεν οδηγεί πάντα σε καλυτερα αποτελέσματα. Αυτό δικαιολογείται επειδή όταν κάνουμε hash κάθε block προκειμένου να γίνει mine χρησιμοποιούμε ένα dictionary που περιέχει και την λίστα με τα transactions. Έτσι, όσο αυξάνονται τα transactions ανά block τόσο καθυστερεί το hash και στην περίπτωση του mining difficulty = 4 μας δημιουργεί bottleneck και δεν αυξάνει την απόδοση. Για τον ίδιο λόγο, βλέπουμε σύνδεση της αύξησης του capacity με αύξηση στο block time.