ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ & ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ & ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

Κατανεμημένα Συστήματα

Εξαμηνιαία Άσκηση

Νεόφυτος Παπασάββας 03115702

Μηνάς Τράττου 03115710

Εισαγωγή - Σκοπός

Το blockchain είναι η τεχνολογία πίσω από τα περισσότερα κρυπτονομίσματα και αποτελεί στην πραγματικότητα μια κατανεμημένη βάση που επιτρέπει στους χρήστες της να κάνουν δοσοληψίες (transactions) μεταξύ τους με ασφάλεια, χωρίς να χρειάζονται κάποια κεντρική αρχή (π.χ. τράπεζα).
Σκοπός της εργασίας είναι να φτάξουμε το Noobcash, ένα απλό σύστημα blockchain, όπου θα καταγράφονται οι δοσοληψίες μεταξύ των συμμετεχόντων και θα εξασφαλίζεται το consensus με χρήση Proof-of-Work.

Σχεδιασμός Συστήματος

Το σύστημα υλοποιήθηκε με γλώσσα προγραμματισμού **Python**. Αποτελείται από 7 αρχεία, τα οποία είναι τα 5 που δίνονται, και προσθέσαμε ένα αρχείο για την υλοποίηση του CLI (Command Line Interface) και ένα αρχείο όπου θα αντιπροσωπεύει την κλάση blockchain, συμπεριλαμβάνοντας κάποιες βασικές λειτουργίες. Παρακάτω παρουσιάζονται αναλυτικά το κάθε αρχείο μαζί με μια σύντομη επεξήγηση για το τι υλοποιεί:

- **wallet.py**

  Στο αρχείο αυτό υλοποιείται η κλάση **Wallet** η οποία χρησιμοποιείται ως το πορτοφόλι του κάθε χρήστη, και έχει τα πεδία:

  - **private_key**, που χρησιμοποιείται από τον κάτοχο του πορτοφολιού για να υπογράψει την κάθε συναλλαγή και είναι γνωστό μόνο από αυτόν
  - **public_key** που αναπαριστά τη διεύθυνση του πορτοφολιού και είναι γνωστή σε όλους τους χρήστες του συστήματος ούτως ώστε να μπορούν κάνουν συναλλαγές μαζί του.

  Επιπρόσθετα στο αρχείο εμπεριέχεται και η συνάρτηση **balance()** η οποία υπολογίζει τα διαθέσιμα χρήματα που έχει το πορτοφόλι, αθροίζοντας όλα τα UTXOs του χρήστη. Επειδή τα αντικείμενα τύπου RSA δε μπορούν να μεταδοθούν, τα αποθηκεύουμε σε μορφή hex.

- **transaction.py**
  Στο αρχείο αυτό υλοποιείται η κλάση **Transaction** η οποία αναπαριστά τις συναλλαγές, και έχει τις ακόλουθες μεθόδους:
  - **sign_transaction()** κατά την οποία χρησιμοποιείται το private_key του αποστολέα για να υπογραφεί η συναλλαγή, το οποίο δεν αποθηκεύεται για λόγους ασφαλείας.
  - **verify_transaction()** κατά την οποία επαληθεύεται βάση του public_key του αποστολέα η υπογραφή της συναλλαγής.

  Το πεδίο transaction_id του αντικειμένου βγαίνει από το hash των υπόλοιπων πεδίων του Transaction, ούτως ώστε να εγγυάται η μοναδικότητά του.

- **blockain.py**

Στο αρχείο αυτό υλοποιείται η κλάση **Blockchain** η οποία αναπαριστά όλο το blockchain και έχει τη λίστα με όλα τα block, καθώς επίσης και τις ακόλουθες μεθόδους:

- **add_block()** η οποία προσθέτει ένα μπλοκ στο τέλος της αλυσίδας.
- **print_chain()** η οποία τυπώνει όλη την αλυσίδα.

- **cli.py**

  Στο αρχείο αυτό υλοποιείται το Command Line Interface, το οποίο δέχεται εντολές από τον χρήστη και τις τρέχει. Δέχεται σαν ορίσματα την πόρτα στην οποία ακούει ο εξυπηρετητής για τον οποίο θα τρέξει τις εντολές.

- **block.py**
  Στο αρχείο αυτό υλοποιείται η κλάση **Block** η οποία αναπαριστά τα blocks του blockchain και έχει τις ακόλουθες μεθόδους:
  - **myHash()** η οποία δημιουργεί το hash του block, χρησιμοποιώντας τα στοιχεία του.
  - **add_transaction()** η οποία δέχεται ένα Transaction και το προσθέτει στο block
  - **print_block()** η οποία τυπώνει τις βασικές πληροφορίες του αντικειένου. Χρησιμοποιήθηκε για debugging.

- **rest.py**

  Στο αρχείο αυτό έχουμε όλα τα mappings των endpoints, τα οποία χρησιμοποιεί το CLI για να δώσει εντολές στο σύστημα. Επίσης έχουμε endpoints που χρησιμοποιούνται και από το ίδιο το σύστημα για επικοινωνία των κόμβων μεταξύ τους. Κατά την εκκίνηση του αρχείου αυτού, δημιουργείται ένας κόμβος(node) και ξεκινάει να τρέχει ένας server που τρέχει στο ip και port που δίνεται σαν είσοδος κατά την εκτέλεση του αρχείου.

- **node.py**
  Στο αρχείο αυτό υλοποιούνται οι πλείστες λειτουργικές συναρτήσεις του συστήματος. Κάθε αντικείμενο αυτής της κλάσης αντιπροσωπεύει ένα κόμβο-χρήστη, και έχει αποθηκευμένα τόσο τα στοιχεία του ιδίου, όσο και στοιχεία ολόκληρου του δικτύου, για να μπορεί να επικοινωνεί με τους άλλους κόμβους και να κάνει validate transactions. Οι συναρτήσεις που υλοποιούνται είναι οι ακόλουθες:
  - **create_new_block()** η οποία δημιουργεί ένα καινούργιο block με τα στοιχεία που του δίνουμε.
  - **request_join_ring()** την οποία καλεί ο κάθε κόμβος κατά τη δημιουργία του, για να δηλώσει στον κόμβο bootstrap ότι έχει δημιουργηθεί, και να εισαχθεί στο ring.

- **generate_genesis()** η οποία δημιουργεί το genesis block
- **create_wallet()** η οποία δημιουργεί το wallet του κάθε node
- **register_me_to_ring()** η οποία καλείται από τον bootstrap κόμβο για να βάλει τον εαυτό του στο ring
- **register_node_to_ring()** η οποία καλείται από τον bootstrap για να προσθέσει τον νεοεισερχόμενο κόμβο στο ring.
- **set_id_and_update_ring()** η οποία καλείται από τον bootstrap κόμβο για να αναθέσει σε ένα νέο κόμβο το id του, να ενημερώσει όλους τους κόμβους για τον νέο κόμβο και να κάνουν όλοι update τα στοιχεία τους. Επίσης εδώ υλοποιείται και το transaction στο οποίο ο bootstrap κόμβος στέλνει τα πρώτα 100 NBC στον καινούριο κόμβο.
- **find_id_from_address()** μια βοηθητική συνάρτηση η οποία βρίσκει το id κάποιου κόμβου, χρησιμοποιώντας το address του.
- **create_transaction()** Η οποία δημιουργεί ένα νέο transaction και το κάνει broadcast σε όλο το δίκτυο.
- **broadcast_transaction()** Η συνάρτηση που καλέιται για να γίνει broadcast σε όλο το δίκτυο ένα transaction.
- **validate_transaction()** Η συνάρτηση που καλείται για να γίνει validate ένα transaction, να διαγρφούν τα UTXOs που χρησιμοποιήθηκαν για να αποδείξουν ότι ο κάτοχο του wallet έχει τα λεφτά που θέλει να στείλει, αλλά και για να προστεθούν τα UTXOs που παράγει το transaction σε αυτούς που τους αναλογεί.
- **add_transaction_to_block()** Η οποία προσθέτει ένα transaction στο τρέχον block. Εάν το τρέχον block είναι γεμάτο, ξεκινάει η διαδικασία για δημιουργία νέου.
- **mine_block()** Η συνάρτηση που τρέχει ο κάθε κόμβος μέχρι να βρει το hash το οποίο να ξεκινάει με difficulty μηδενικά. Ο πρώτος κόμβος που το βρίσκει, το κάνει broadcast σε όλο το δίκτυο για να το υιοθετήσουν και οι υπόλοιποι κόμβοι.
- **broadcast_block()** Η συνάρτηση που καλείται για να γίνει broadcast σε όλο το δίκτυο ένα block.
- **valid_proof()** Η συνάρτηση που χρησιμοποιείται για να ελέγξουμε αν το hash ενός block είναι valid (ξεκινάει με difficulty μηδενικά).
- **validate_block()** Η συνάρτηση που καλείται για να ελέγξουμε αν ένα block είναι valid.
- **validate_chain()** Η συνάρτηση που καλέιται για να ελέγξει αν τα blocks σε ένα chain έχουν σωστές τιμές στο current_hash και previous_hash.
- **resolve_conflicts()** Η συνάρτηση καλείται από κάποιο κόμβο όταν κάποιο block που δέχεται δε γίνει validate. Ο κόμβος αυτός παίρνει όλα τα blockchain, επιλέγει τη μεγαλύτερη από αυτές και εάν είναι μικρότερη από τη δική του αλυσίδα, την υιοθετεί.

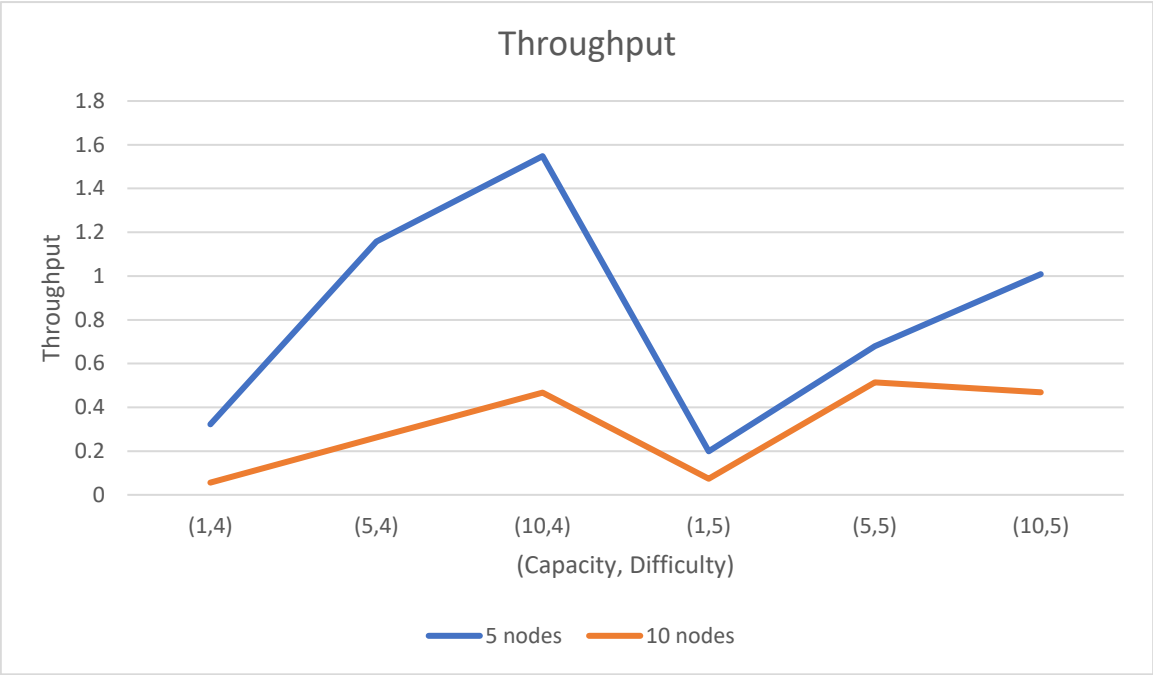
Κατά την εκτέλεση των πειραμάτων, αντιληφθήκαμε ότι είχαμε πρόβλημα στο συγχρονισμό των εντολών, και πιο συγκεκριμένα όταν κάποιος κόμβος ξεκινούσε να κάνει mine ένα νέο block, τότε ξεκινούσε και το επόμενο transaction να γίνεται. Ως εκ τούτου, δεν

προλάβαιναν οι κόμβοι να βρουν νέο hash για το καινούριο block, και έτσι είχαμε κάποια conflicts με ορισμένα transaction, που οδήγησαν σε απώλεια ορισμένων. Το πρόβλημα αυτό θα μπορούσε να λυθεί με καλύτερη χρήση χρονοπρογραμματισμού, με τη βοήθεια Thread και Lock.
Επίσης, όταν τρέχαμε το κώδικα στα μηχανήματα που δημιουργήσαμε στο Okeanos, δεν παίρναμε απάντηση από τους server τους οποίους σηκώναμε. Ακολουθήσαμε τις οδηγίες που δίνονταν στην εκφώνηση, αλλά το πρόβλημα δε διορθώθηκε. Ως εκ τούτου, όλα τα πειράματα έγιναν σε τοπικό υπολογιστή.

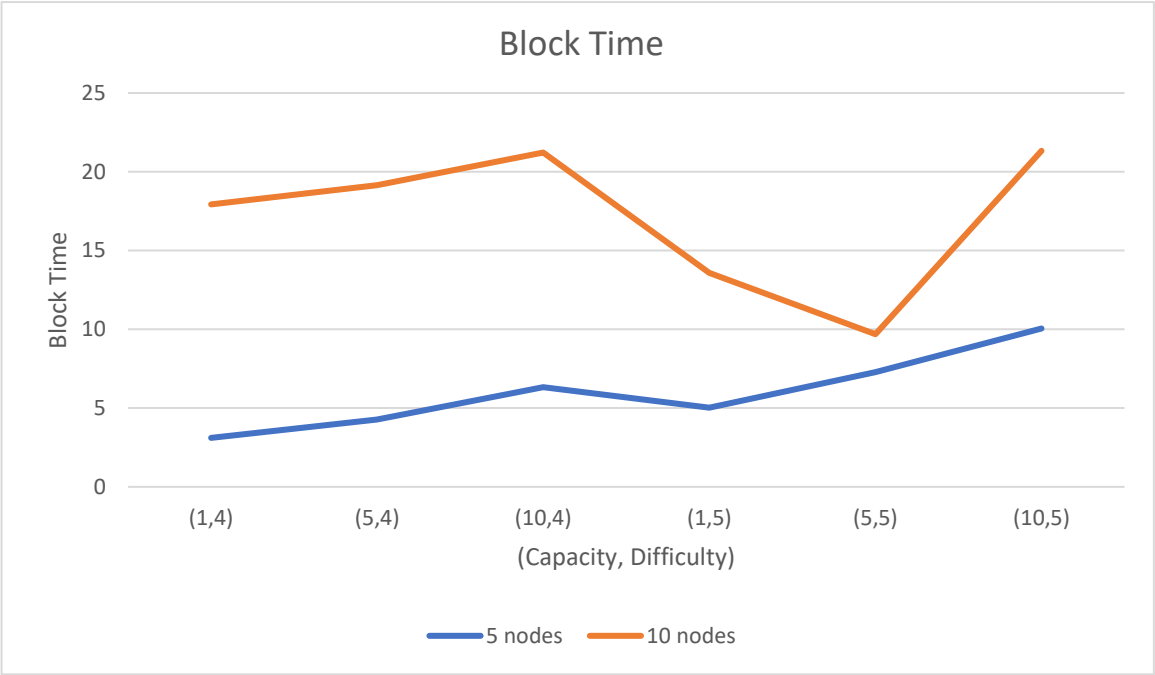
Πειράματα

Εκτελώντας τα πειράματα που ζητήθηκαν, για αριθμό κόμβων 5 και 10, και για χωρητικότητα των μπλοκ 1,5,10, πήραμε τις εξής μετρήσεις για ρυθμαπόδοση και χρόνο mining του κάθε μπλοκ:

Number of Nodes	Capacity	Difficulty	Throughput	Block Time
5	1	4	0.322	3.11
5	5	4	1.158	4.28
5	10	4	1.548	6.32
5	1	5	0.199	5.02
5	5	5	0.679	7.28
5	10	5	1.008	10.05
10	1	4	0.056	17.92
10	5	4	0.262	19.15
10	10	4	0.467	21.22
10	1	5	0.073	13.58
10	5	5	0.514	9.69
10	10	5	0.469	21.33



Παρατηρούμε ότι όσο μεγαλώνει το capacity, τόσο αυξάνεται και το throughput του συστήματος. Αυτό είναι απόλυτα λογικό, αφού το σύστημα ψάχνει hash για καινούριο block μετά από περισσότερα transactions.
Επίσης παρατηρούμε ότι όσο ανεβαίνει το difficulty, μειώνεται το throughput, γεγονός επίσης αναμενόμενο, αφού η εύρεση ενός hash του συστήματος γίνεται πιο δύσκολη, άρα και πιο χρονοβόρα. Έτσι το σύστημα αντί να εκτελεί transactions, χρησιμοποιεί τους πόρους του για να δημιουργήσει νέο block.
Τέλος, παρατηρούμε ότι η αύξηση του αριθμού των κόμβων έχει επιφέρει μεγάλη μείωση στη ρυθμαπόδοση, ιδιαίτερα για πιο χαμηλές τιμές difficulty. Αυξάνοντας τον αριθμό των κόμβων του συστήματος, ο χρόνος εύρεσης του hash ενός νέου block θα πρέπει να μειώνεται, αφού ψάχνουν περισσότεροι κόμβοι για να βρουν το σωστό nonce, άρα κατ' επέκταση και η ρυθμαπόδοση θα έπρεπε να ήταν πιο μεγάλη.



Παρατηρούμε ότι όσο μεγαλώνει το capacity, τόσο αυξάνεται και το Block Time του συστήματος. Αυτό είναι απόλυτα λογικό, αφού το σύστημα ψάχνει hash για καινούριο block μετά από περισσότερα transactions.

Επίσης παρατηρούμε ότι όσο ανεβαίνει το difficulty, αυξάνεται ελαφρώς το Block Time, γεγονός το οποίο δεν περιμέναμε.

Τέλος, παρατηρούμε ότι η αύξηση του αριθμού των κόμβων έχει επιφέρει αύξηση στο Block Time. Αυξάνοντας τον αριθμό των κόμβων του συστήματος, ο χρόνος εύρεσης του hash ενός νέου block θα πρέπει να μειώνεται, αφού ψάχνουν περισσότεροι κόμβοι για να βρουν το σωστό nonce, άρα κατ' επέκταση και το Block Time θα έπρεπε να πέφτει.

Συμπεράσματα

Σε γενικές γραμμές, λόγω του ότι τα πειράματα έτρεξαν τοπικά και όχι στον Okeanons, πήραμε λανθασμένα αποτελέσματα, με πολύ μικρή ρυθμαπόδοση. Επίσης λόγω προβλημάτων που είχαμε κατά τον συγχρονισμό των λειτουργιών του συστήματος, κάποια transactions παρατηρήσαμε ότι χάνονται κατά την εκτέλεση, γιατί προσπαθούσαν να εκτελεστούν παράλληλα με το ψάξιμο hash ενός νέου block.