Κατανεμημένα Συστήματα
Χειμερινό Εξάμηνο 2019-2020, Σχολή ΗΜ&ΜΥ
Εξαμηνιαία Εργασία - Αναφορά

Θεοδωρόπουλος Νικήτας	*Κασουρίδης Στυλιανός*	*Χαρδούβελης Γεώργιος-Ορέστης*
Α.Μ: 03115185	*Α.Μ: 03115172*	*Α.Μ: 03115100*
9ο Εξάμηνο	*9ο Εξάμηνο*	*9ο Εξάμηνο*

Στην παρούσα άσκηση δημιουργήσαμε ένα απλό σύστημα blockchain, το **noobcash**, στο οποίο καταγράφονται οι δοσοληψίες μεταξύ των συμμετεχόντων. Το σύστημα blockchain μπορεί να υλοποιεί όλες τις λειτουργίες που αναφέρονται στην εκφώνηση, όπως θα δούμε παρακάτω.

Σχεδιασμός Συστήματος

Το σύστημα που υλοποιήθηκε αποτελείται από τον server, που υλοποιήθηκε σε Flask, και τον noobcash client, ένα cli ο οποίος εκτελεί τις απαιτούμενες εντολές μέσω REST API. Το backend υλοποιήθηκε σε γλώσσα προγραμματισμού python.

Οι πόροι της εργασίας μας δόθηκαν από την υπηρεσία ~okeanos-knossos. Συγκεκριμένα, για τις ανάγκες της εργασίας είχαμε πρόσβαση σε 5 VMs.

Υλοποίηση Λειτουργιών στο Backend

Οι λειτουργίες του συστήματος σύμφωνα με την εκφώνηση είναι οι παρακάτω:

- Κάθε χρήστης θα έχει το δικό του noobcash wallet για να πραγματοποιεί transactions. Ένα wallet είναι ουσιαστικά ένα private key γνωστό μόνο σε αυτόν και ένα public key. Τα transactions είναι συναλλαγές στις οποίες ένας χρήστης δίνει NBC (noobcash) coins σε κάποιον άλλον και στον κώδικα μας υλοποιούνται από την *create_transaction(receiver, amount)*. Ο αποστολέας χρησιμοποιεί το private key του για να υπογράψει την συναλλαγή -συνάρτηση *sign_transaction()*-, ενώ το public key ενός λειτουργεί και σαν την διεύθυνση του wallet του. Έτσι μέσω του transaction θα στείλει τα "λεφτά" στην διεύθυνση του παραλήπτη.

- Μέσω της *broadcast_transaction* κάθε transaction γίνεται broadcast σε όλο το δίκτυο.

- Όταν οι miners λάβουν ένα transaction -συνάρτηση *receive_transaction* μέσω μεθόδου POST του Flask Api- το επικυρώνουν -καλείται η *validate_transaction*-.

- Όταν γεμίσει το τρέχων block, οι χρήστες (που ταυτόχρονα είναι και όλοι miners) ξεκινάνε mining χρησιμοποιώντας την μέθοδο proof of work. Συγκεκριμένα, όταν δημιουργηθεί ή γίνει επικύρωση ενός transaction, αν ο αριθμός των transactions στο block φτάσει στο ορισμένο CAPACITY, καλείται η συνάρτηση *mine_block* του αρχείου state.py που, εάν δεν υπάρχει conflict, καλεί την *mine_block* του αρχείου block. Κατά την διαδικασία του mining, αρχικά, χρησιμοποιούμε την δομή Merkle Tree στην οποία αποθηκευουμε της hash τιμές των transactions. Ύστερα οι miners ψάχνουν το σωστό nonce ώστε η hash τιμή του block να ξεκινάει με όσα μηδενικά έχουν οριστεί με το difficulty. Ο miner που θα βρει πρώτος αποδεκτό nonce καλεί την συνάρτηση *add_block(block)* η οποία προσθέτει το block στον blockchain αφού το επικυρώσει. Για

την επικύρωση εξετάζεται αν το previous_hash πεδίο είναι ίδιο με το hash του προηγούμενου block ενώ καλείται και η *validate_hash()* που επικυρώνει πως η τιμή hash του block είναι η σωστή (και άρα έχει γίνει σωστό mining). Ταυτόχρονα, όταν τρέχει η *add_block* χρησιμοποιούμε την lock ώστε να επιβεβαιώσουμε πως δεν θα τρέχει ταυτόχρονα ο αλγόριθμος consesus. Ύστερα ο miner κάνει broadcast το νέο block μέσω της συνάρτησης *broadcast_block(block).*

- Υπάρχει περίπτωση δύο ή παραπάνω miners να κάνουν ταυτόχρονα mine ένα block. Έτσι οι παραλήπτες προσθέτουν το block που λαμβάνουν αντίστοιχα, κάτι που μπορεί να καταλήξει σε δύο ή παραπάνω διακλαδώσεις της αλυσίδας. Σε τέτοια περίπτωση καλείται η συνάρτηση *resolve_conflict* που υλοποιεί τον αλγόριθμο consensus. Συγκεκριμένα, δεδομένης της μίας αλυσίδας, υπολογίζεται το μήκος της άλλης που στέλνεται μέσω μεθόδου GET του Api, και κρατάμε την μεγαλύτερη αλυσίδα. Στο κρίσιμο σημείο της συνάρτησης, χρησιμοποιώντας την lock σιγουρευόμαστε πως δεν θα θα προστεθούν νέα blocks ενώ τρέχει ο αλγόριθμος consesus.

Η validate_transaction που αναφέρθηκε πάνω περιλαμβάνει αρκετούς διαφορετικού ελέγχους.
Αρχικά επαληθεύεται η υπογραφή του αποστολέα με την *verify_signature()*.
Ύστερα ελέγχονται τα transaction input και outputs.
Τα transaction inputs περιέχει πληροφορία για τα προηγούμενα transactions από όπου προήλθαν τα χρήματα που τώρα μεταφέρονται.
Τα transaction outputs από την άλλη είναι δύο για κάθε transaction· ένα για τον αποστολέα με τα "ρέστα", το ποσό που του έμεινε απο την συναλλαγή, και ένα για τον παραλήπτη, με το ποσό που πήρε. Τα transaction outputs παράγονται κάθε φορά που δημιουργείται / επαληθεύεται ένα transaction και έχουν δομή λεξικού που περιλαμβανουν το id του transaction από όπου προέρχονται, ένα μοναδικό id για το συγκεκριμένο output (id του transaction από όπου προήλθαν + '0' για τον αποστολέα και + '1' για τον παραλήπτη), την διεύθυνση / public key του παραλήπτη και τέλος το μεταφερόμενο ποσό / ρέστα.
Τα transaction inputs αποτελούνται από UTXOs (unspent transaction outputs), δηλαδή από transaction outputs που δεν έχουν ξοδευτεί ακόμη. Το πεδίο των inputs σε κάθε transtaction δημιουργείται στην create_transaction, παίρνοντας αρκετά UTXOs ώστε να συμπληρωθεί το ποσό του transaction, εφόσον υπάρχουν. Τα inputs αποτελούν απλά ids των αντίστοιχων UTXOs.
Έτσι στην *validate_transaction()* ελέγχεται αν τα transaction inputs αντιστοιχούν σε έγκυρα UTXOs. Τα UTXOs κάθε wallet είναι γνωστά σε όλους τους κόμβους ώστε κάθε χρήστης να μπορεί να κάνει την επαλήθευση. Εφόσον επαληθευτεί το transaction, τα UTXOs που χρησιμοποιήθηκαν σαν input που έχουν αποθηκευτεί στην *pending_removed* λίστα αφαιρούνται από τα UTXOs globally με την χρήση της συνάρτησης *remove_utxo(utxo)*.
Τέλος, τόσο στην *validate_transaction* όσο και στην *create_transaction* προσθέτουμε globally στα UTXOs τα νέα outputs (μέσω της *add_utxo(utxo))*, καθώς και το νέο transaction στην λίστα με τα unmined *transactions*.

Δίκτυο

Όσον αφορά το δίκτυο μας, οι συμμετέχοντες συμβολίζονται με κόμβους με μοναδικό id. Κάθε συμμετέχοντας / κόμβος μπορεί να ανταλλάσει μηνύματα με τους υπόλοιπους κόμβους. Η επικοινωνία πραγματοποιείται μέσω REST api όπως είδαμε και παραπάνω.
Ο πρώτος κόμβος στο δίκτυο μας με id 0 είναι ο coordinator. Η αρχικοποίήση του γίνεται με το κάλεσμα της συνάρτησης *start_coordinator()* με ένα POST request, η οποία καλεί την *genesis(n)* που δημιουργεί τον κόμβο και αρχικοποιεί τα απαραίτητα πεδία. Έτσι δημιουργείται το genesis block (το οποίο ως το πρώτο block δεν επαληθεύεται) με το πρώτο transaction που έχει ως ποσό 100 φορές τους κόμβους / χρήστες NBC (noobcash) coins. Τα δεδομένα του coordinator μπορούμε να τα δούμε με ένα GET request με την βοήθεια της *show_coordinator_data()*.
Στη συνέχεια μέσω του πρώτου κόμβου προστίθενται και οι υπόλοιποι. Αρχικά το σύστημα επικοινωνεί με τον κόμβο-coordinator και στέλνει το public key του wallet του. Στη συνέχεια ο κόμβος-coordinator του δίνει το μοναδικό id του. Με την δημιουργία κάθε κόμβου, που πραγματοποιείται με την συνάρτηση *register_new_node()*, δημιουργείται αυτόματα και ένα transaction από τον αρχικό κόμβο σε αυτόν, με ποσό 100 NBC coins.
Γενικά η παραπάνω επικοινωνία πραγματοποιείται με την χρήση της συνάρτησης *connect_to_coordinator*. Για την ακρίβεια γίνεται POST request από τον χρήστη, με αποτέλεσμα να τρέξει η *register_new_node()*. Κατά την διαδικασία ζητείται και το blockchain όπως έχει διαμορφωθεί, το οποίο γίνεται μέσω ενός GET request με την βοήθεια της *request_chain()*. Ακόμη, η αλυσίδα επιβεβαιώνεται μέσω της *validate_chain()*, η οποία τρέχεο μόνο αν έχει “κλειδώσει” και δεν τρέχει ταυτόχρονα ο αλγόριθμος consesus, και εξετάζει αν το previous_hash πεδίο είναι ίδιο με το hash του προηγούμενου block για κάθε block, ενώ καλείται και η *validate_hash()* για κάθε block, επικυρώνοντας έτσι την hash τιμή του.

Αφού εισαχθούν όλοι οι κόμβοι στην αρχή, το δίκτυο παραμένει ίδιο, δηλαδή δεν έχουμε εισαγωγή ή αποχώρηση κόμβων. Ύστερα γίνονται broadcast σε όλους τους κόμβους τα απαραίτητα στοιχεία (ip adress / port υπόλοιπων κόμβων και τα public keys των wallets τους) μέσω της συνάρτησης *broadcast_nodes_info()* στο αρχείο *broadcast.py*.

Έτσι τελικά δημιουργείται ένα αμετάβλητο δίκτυο με όσους κόμβους χρειαζόμαστε (5 ή 10 για τις ανάγκες των πειραμάτων της εργασίας), ο καθένας εκ των οποίων έχει 100 NBC coins.

Noobcash Client

Στα πλαίσια της εργασίας αναπτύχθηκε και ένα cli που λειτουργεί ως client. Η υλοποίηση του φαίνεται στα αρχεία cli.py και python.
Αφού στηθεί ο δίκτυο μπορούμε με ένα POST request να ξεκινήσουμε την λειτουργία του client (με την βοήθεια της *start_client()*). Μέσω του cli μπορούμε να:

- δημιουργήσουμε νέο transaction: ο client μπορεί να δημιουργήσει ένα transaction της μορφής t <recipient_address> <amount>, κάνοντας ένα POST request με την βοήθεια της εντολής cli_*new_transaction*, που παίρνει ως όρισμα την διεύθυνση του recipient (δηλαδή το public key του wallet του) και το ποσό της συναλλαγής.

- δούμε τα τελευταία transactions: με την εντολή view καλείται *view_transactions,* ο client βλέπει τις συναλλαγές του τελευταίου επικυρωμένου block στην αλυσίδα.

- δούμε το υπόλοιπο του wallet: με την εντολή balance καλείται η συνάρτηση *show_balance* (POST request) που καλεί την *wallet_balance*, ο client βλέπει το υπόλοιπο στο wallet του. Το υπόλοιπο προκύπτει ουσιαστικά από όλα τα UTXOs που έχουν ως κάτοχο αυτόν. Τόσο η *wallet_ballance* όσο και η *view_transactions* από πάνω χρησιμοποιούν την *get_node_id* για να πάρουν το id των εμπλεκόμενων χρηστών.

- ζητήσουμε επεξήγηση των εντολών: με την help μπορούμε να δούμε επεξηγήσεις για τις παραπάνω εντολές.