Κατανεμημένα Συστήματα

The NoobCash Blockchain

Όνομα: Ανδρέας
Επώνυμο: Κούλουμος
ΑΜ: 03115712

Όνομα: Σταύρος
Επώνυμο: Σταύρου
ΑΜ: 03115701

Όνομα: Λένος
Επώνυμο: Τσοκκής
ΑΜ: 03115700

Σκοπός

Σκοπός της εργασίας ήταν η δημιουργία ενός Blockchain (με το όνομα Noobcash) μέσω του οποίου ένας αριθμός κόμβων μπορεί να εκτελεί συναλλαγές, χωρίς τη μεσολάβηση κάποιου TTP (Trusted Third Party, πχ κάποια τράπεζα). Ο κάθε κόμβος/χρήστης του συστήματος που θέλει να πληρώσει κάποιον άλλο, στέλνει την συναλλαγή σε όλο το σύστημα. Οι κόμβοι γεμίζουν το block τους και όταν αυτό γεμίσει ξεκινούν μια διαδικασία mining με Proof-of-Work, ψάχνοντας κατάλληλο nonce ώστε τα πρώτα ψηφία του hash του block να είναι 0. Όταν κάποιος κόμβος βρει το κατάλληλο nonce, στέλνει το block τους στους υπόλοιπους και το προσθέτει και ο ίδιος στην αλυσίδα του.

Σχεδιασμός/Υλοποίηση

Για το σχεδιασμό του συστήματος χρησιμοποιήθηκε ο σκελετός που δόθηκε για python και στον οποίο προσθέσαμε άλλα 3 αρχεία, το cli.py στο οποίο περιέχεται η υλοποίηση για τον noobcash_client, το blockchain.py το οποίο περιέχει την υλοποίηση για τη λίστα με τα block του κάθε κόμβου και το constants.py στο οποίο απλά περιέχονται σταθερές για τη λειτουργία του συστήματος (difficulty, capacity και αριθμός κόμβων). Θα περιγράψουμε συνοπτικά τη χρήση και υλοποίηση κάθε κλάσης/αρχείου. Ο κώδικας συνοδεύεται με σχόλια.

block.py

Τα πεδία που περιέχει είναι ακριβώς αυτά που περιγράφονται στην εκφώνηση, ενώ επιπρόσθετα προσθέσαμε το πεδίο creator, το οποίο δηλώνει το δημιουργό του block. Αυτό γίνεται απλά για περεταίρω διαφοροποίηση των block μεταξύ τους (διαφορετικά σε περίπτωση που το σύστημα δεν παρουσίαζε βλάβες και 2 κόμβοι δημιουργούσαν τα ταυτόχρονα το block τότε θα έκαναν mine ακριβώς το ίδιο block). Η κλάση διαθέτει 3 μεθόδους. Η σημαντικότερη από αυτές είναι η to_json(), η οποία μετατρέπει το αντικείμενο σε μια αναπαράσταση σε μήνυμα JSON, η οποία είναι αναγκαία τόσο για να μπορέσουμε να κάνουμε hash το block αλλά και για να μπορούμε ακολούθως να το μεταδώσουμε πάνω από «σύρμα» (με άλλα λόγια σειριακά). Ακόμη, έχουμε τη myHash(), η οποία υπολογίζει το hash του block και την add_transaction(), η οποία προσθέτει μια συναλλαγή στο block και ενημερώνει κατάλληλα την τιμή του πεδίου current_hash.

blockchain.py

Η blockchain περιέχει απλά 2 πεδία, τη λίστα με τα blocks και το μήκος αυτής. Ακόμη κατ'αντιστοιχία με την block.py περιέχει την add_block() για προσθήκη block στην αλυσίδα, καθώς και την to_json() για σειριοποίηση αντικειμένων της κλάσης. Τέλος, η lastBlock() επιστρέφει το τελευταίο block που προστέθηκε στην αλυσίδα.

cli.py

Ο κώδικας του CLI δεν είναι παρά ένα ατέρμονο while loop το οποίο περιμένει να δώσει ο χρήστης κάποια εντολή. Κατά την εκτέλεση του αρχείου ο χρήστης δίνει με χρήση των flags -a και -p τη διεύθυνση ip και τη θύρα στην οποία λειτουργεί ο server του. Το CLI προσφέρει 6 λειτουργίες εντολές, τις 4 που ζητούνται στην εκφώνηση καθώς και τις exit και clear.

transaction.py

Η transaction περιέχει τα πεδία που αναφέρονται στην εκφώνηση. Κατ' αντιστοιχία με τις block και blockchain η to_dict() προσφέρει τη δυνατότητα μετάδοσης αντικειμένων της κλάσης πάνω από σύρμα. Οι υπόλοιπες μέθοδοι που υλοποιούνται είναι η myHash() η οποία υπολογίζει το hash που χρησιμοποιείται σαν id της κάθε συναλλαγής, η outputUTXOS() για παραγωγή των transaction_outputs της συναλλαγής, η add_utxos() για προσθήκη των transaction_inputs και η sign_transaction() για υπογραφή με το privatekey του δημιουργού της συναλλαγής.
Κατά τη δημιουργία του transaction, υπολογίζεται η υπογραφή με το privatekey χωρίς αυτό να φυλάσσεται κάπου, για λόγους ασφαλείας. Ακόμη, το transaction_id υπολογίζεται πριν ορίσουμε όλα τα πεδία της κλάσης, καθώς αυτό που μας ενδιαφέρει είναι η μοναδικότητα του αναγνωριστικού, κάτι που εξασφαλίζεται από το ότι τα transaction_inputs είναι αναπόφευκτα διαφορετικά.

wallet.py

Η wallet περιέχει ακριβώς τα πεδία που δίνονται στην εκφώνηση. Κατά τη δημιουργία ενός αντικειμένου της κλάσης παράγεται ένα κλειδί RSA, το οποίο διανέμεται στους υπόλοιπους. Η μόνη μέθοδος της κλάσης είναι η balance() η οποία αθροίζει τα UTXOS του κόμβου και επιστρέφει το άθροισμα, το οποίο είναι και τα διαθέσιμα χρήματα που έχουμε για ξόδεμα. Για λόγους μετάδοσης πάνω από σύρμα το κλειδί RSA αποθηκεύεται σε μορφή hex, κάτι που μας επιτρέπει να το ξαναδημιουργήσουμε στην «άλλη» πλευρά του σύρματος με την RSA.importKey().

node.py

Η node είναι η καρδία της υλοποίησης μας. Στα πεδία της περιέχονται χρήσιμες πληροφορίες όπως το id του κόμβου, η διεύθυνση ip του, η διεύθυνση του bootstrap, ο αριθμός των κόμβων που συμμετέχουν στο σύστημα, η αλυσίδα του κόμβου, καθώς και πληροφορίες για τον κάθε κόμβο που αποθηκεύονται στο dictionary ring. Ανάλογα με το αν ο κόμβος είναι bootstrap ή όχι αρχικοποιούνται κατάλληλα κάποια πεδία ή ενημερώνεται από τον bootstrap για τις τιμές τους. Οι μέθοδοι που υλοποιούνται είναι οι:

- verify_transaction(): Επιτρέπει την πιστοποίηση συναλλαγών βάση του πεδίου υπογραφής και του public_key.
- create_wallet(), create_new_block(): Δημιουργούν ένα πορτοφόλι και ένα block για το νέο κόμβο. Η $2^{η}$ καλείται κάθε φορά που ο κόμβος χρειάζεται ένα «φρέσκο» block.
- validate_transaction(): Ο κόμβος ελέγχει αν η συναλλαγή που έλαβε είναι έγκυρη, δηλαδή αν ο κόμβος που την δημιούργησε έχει αρκετό υπόλοιπο για υλοποίηση της, αλλά και την εγκυρότητά της με την verify_transaction()
- register_node_to_ring(): Καλείται μόνο από τον bootstrap όταν λάβει έτοιμα για προσθήκη κόμβου στο σύστημα. Ο bootstrap ενημερώνει με τις καινούριες πληροφορίες το ring του και στέλνει το ring, την αλυσίδα του, 100 NBCS και ένα αναγνωριστικό στον κόμβο που έκανε το αίτημα. Ακόμη, ενημερώνει τους υπόλοιπους με τα στοιχεία του νέου κόμβου.
- reconstruct_X(): Επιτρέπουν την αναδημιουργία αντικειμένων transcation, block και blockchain από μηνύματα JSON.
- add_transaction_to_block(): Προσθέτει μια συναλλαγή στο τρέχον block του κόμβου.
- broadcast_block(), broadcast_transaction(): Επιτρέπουν τη μετάδοση block ή/και συναλλαγής σε όλο το δίκτυο.
- create_transaction(): Δημιουργείται μια συναλλαγή. Σαν αποστολέας ορίζεται ο ίδιος ο κόμβος, ενώ το ποσό και ο αποστολέας δίνονται ως ορίσματα

- mine_block(): Όταν γεμίζει το κάθε block σύμφωνα με τη σταθερά CAPACITY, το block γίνεται mine. Αυτό που γίνεται είναι αύξηση του nonce και υπολογισμός του hash μέχρι να βρούμε κάποιο κατάλληλο βάσει της σταθεράς DIFFICULTY.
- valid_proof(): Ελέγχεται η εγκυρότητα ενός ληφθέντος block. Ελέγχεται ότι το previous_hash του καινούριου block ταιριάζει με το current_hash του τελευταίου block στην αλυσίδα μας και ότι ο αριθμός των μηδενικών στην αρχή του hash είναι όντως σωστός.
- resolve_conflicts(): Καλείται όταν ο κόμβος λάβει κάποιο block το οποίο δεν μπορεί να κάνει valid. Ο κόμβος ρωτάει όλους τους υπόλοιπους για τα μήκη των αλυσίδων τους και κρατάει τη μακρύτερη.
- valid_chain(): Ελέγχει αν τα blocks σε ένα chain έχουν σωστές τιμές στο current_hash και previous_hash.

rest.py

Στην κλάση αυτή περιέχονται τα endpoints ενός RESTful API το οποίο οι κόμβοι μας χρησιμοποιούν για επικοινωνία, αλλά και για εξυπηρέτηση αιτημάτων από κάποιο Client, όπως το CLI που περιγράψαμε πιο πάνω. Κατά την εκκίνηση του rest.py ο χρήστης δίνει τα ορίσματα -p [PORT] -a [ADDRESS] -l [BOOTSTRAPADDRESS] -m [BOOTSTRAPPORT] -b, με την προφανή χρήση του καθενός και με τη σημαία -b να δίνεται μόνο αν ο τρέχον κόμβος είναι ο bootstrap κόμβος. Βάσει αυτών δημιουργείται ένα αντικείμενο της κλάσης node και σε αυτό αποθηκεύονται οι πληροφορίες για τον κάθε κόμβο. Ακολούθως «σηκώνουμε» τον server και είμαστε έτοιμοι. Τα endpoints εκτελούν διάφορους μικροελέγχους και επιστρέφουν ανάλογα τη ζητούμενη η απάντηση με κωδικό 200 ή το σφάλμα και τον κατάλληλο κωδικό σφάλματος.

Προβλήματα/Παραδοχές

Κατά την υλοποίηση του συστήματος και την εκτέλεση των πειραμάτων είχαμε τα εξής:

- Για συντονισμό των κόμβων χρησιμοποιήσαμε χρονοκαθυστέρηση και χρήση event της κλάσης threading. Μάλλον μια καλύτερη προσέγγιση θα ήταν και η χρήση locks.
- Προσπαθώντας να εκτελέσουμε τα πειράματα στον Ωκεανό ο κώδικας είχε πάρα πολύ διαφορετική συμπεριφορά από την εκτέλεση του κώδικα στον υπολογιστή που γράφτηκε το μεγαλύτερο κομμάτι του κώδικα. Ως εκ τούτου εκτελέσαμε τα πειράματα τοπικά.

Πειράματα

Όπως αναφέραμε και πιο πριν τα πειράματα εκτελέστηκαν τοπικά και όχι στον ωκεανό, «σηκώνοντας» τους server σε διαφορετικές θύρες και όχι σε διαφορετικά μηχανήματα. Ως εκ τούτου τα αποτελέσματα για τη ρυθμαπόδοση και το χρόνο mine κάθε block είναι εξαιρετικά κακά και δεν μπορούν να ληφθούν πραγματικά υπόψη για αξιολόγηση του συστήματος. Μπορούμε ωστόσο να δούμε τη σύγκριση μεταξύ διαφορετικών παραμέτρων, αλλά και για τη δυνατότητα κλιμάκωσης του συστήματος. Χρησιμοποιήσαμε το script reader.py το οποίο επισυνάπτεται.
Οι μετρήσεις που πήραμε ήταν οι εξής:

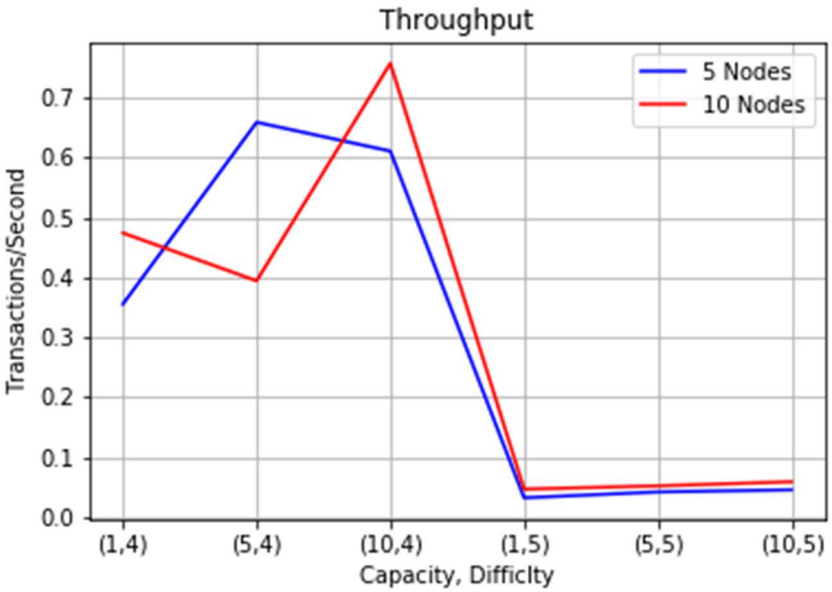
5 NODES

	Throughput				AvgMiningTime			
	CAPACITY = 1	CAPACITY = 5	CAPACITY = 10			CAPACITY = 1	CAPACITY = 5	CAPACITY = 10
DIFFICULTY = 4	0.355449	0.658938	0.61081		DIFFICULTY = 4	2.755696	7.556676	16.439921
DIFFICULTY = 5	0.032047	0.041811	0.045311		DIFFICULTY = 5	31.010899	119.091294	223.047007

10 NODES

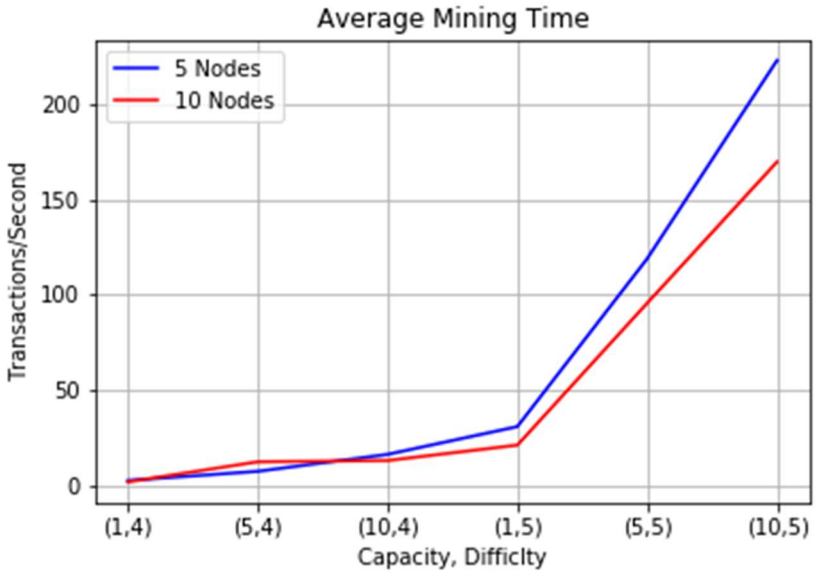
	Throughput				AvgMiningTime			
	CAPACITY = 1	CAPACITY = 5	CAPACITY = 10			CAPACITY = 1	CAPACITY = 5	CAPACITY = 10
DIFFICULTY = 4	0.474104	0.394234	0.757378		DIFFICULTY = 4	2.028596	12.555359	13.216915
DIFFICULTY = 5	0.046512	0.051942	0.058764		DIFFICULTY = 5	21.30562	95.770256	169.828997

Για τη ρυθμαπόδοση του συστήματος έχουμε το εξής:



Παρατηρούμε, τυπικά πως μεγαλύτερες τιμές capacity οδηγούν σε καλύτερη ρυθμαπόδοση. Αυτό οφείλεται στο ότι για τον ίδιο αριθμό συναλλαγών κάνουμε mine λιγότερα block, οπότε αυτό είναι λογικό. Ακόμη, παρατηρούμε πως ο μεγαλύτερος αριθμός κόμβων γενικά επέφερε μεγαλύτερη ρυθμαπόδοση. Αυτό ίσως οφείλεται στο ότι με μεγαλύτερο αριθμό κόμβων έχουμε περισσότερες πιθανότητες κάποιο nonce να μας «κάνει» και άρα έχουμε mine πιο εύκολα.

Για το μέσο χρόνο mine έχουμε:



Παρατηρούμε πως μεγαλύτερες τιμές capacity οδηγούν τυπικά σε μεγαλύτερο χρόνο mine. Αυτό οφείλεται στο ότι ο υπολογισμός του hash παίρνει περισσότερο χρόνο, λόγω του μεγέθους του string που δίνεται σαν είσοδος στον SHA256. Ακόμη, και πάλι παρατηρούμε τυπικά καλύτερη απόδοση στο δίκτυο με 10 κόμβους, δηλαδή χαμηλότερο μέσο χρόνο mining. Αυτό και πάλι ίσως οφείλεται στο μεγαλύτερο αριθμό κόμβων που προσπαθούν να βρουν το κατάλληλο nonce.