Εργασία στα Κατανεμημένα Συστήματα

Ακαδημαϊκό Έτος 2021-2022

Ζωγράφος Ορφέας	Παπαϊωάννου Κωνσταντίνος	Θεοδωρόπουλος Βασίλειος
el17160	el17005	el17092

Εισαγωγή

Η εργασία αυτή έχει ως σκοπό την δημιουργία ενός απλού συστήματος blockchain, το noobcash. Στο σύστημά μας κάθε χρήστης είναι κάτοχος ενός wallet με το οποίο κάνει συναλλαγές σε NBC. Οι συναλλαγές αυτές, αφού υπογραφούν από τον αποστολέα με το private key του, γίνονται broadcast στο δίκτυο. Στη συνέχεια, κάθε χρήστης/miner αφού κάνει τις κατάλληλες επικυρώσεις στα transactions, τα προσθέτει στο block του. Όταν αυτό γεμίσει οι miners προσπαθούν να βρούν το κατάλληλο nonce (proof-of-work) και μόλις κάποιος το βρεί, έχει κάνει επιτυχώς mine το block και μπορεί να το κάνει broadcast στο υπόλοιπο δίκτυο.

Το δίκτυο αποτελείται από n κόμβους στην γενική του μορφή, οι οποίοι έχουν id από 0 έως n-1. Ο κόμβος με id μηδέν είναι ο bootstrap, ο οποίος δημιουργεί το genesis block. Για την εισαγωγή κάποιου κόμβου στο blockhain πρέπει ο κόμβος να συνδεθεί με τον bootstrap και αυτός να του αναθέσει αντίστοιχο id. Μόλις ολοκληρωθεί η διαδικασία της εισαγωγής ο bootstrap στέλνει σε όλους τους κόμβους όλα τα ζεύγη ip/port και τα public keys των wallet όλων των συμμετεχόντων κόμβων.

Κατά την δημιουργία του genesis block φτιάχνεται ένα transaction με 100*n NBC τα οποία δίνονται στον bootstrap κόμβο. Όταν ένας κόμβος εισάγεται στο δίκτυο ο bootstrap του στέλνει 100 NBC.

Κάθε κόμβος γνωρίζει για τους υπόλοιπους την ip και το port στο οποίο ακούνε προκειμένου να επιτευχθεί επικοινωνία. Στην παρούσα εργασία η επικοινωνία έχει επιτευχθεί με χρήση API χρησιμοποιώντας την βιβλιοθήκη net/http της GO η οποία είναι και η γλώσσα με την οποία φτιάξαμε το σύστημα.

Noobcash Backend

Το σύστημα του noobcash έχει βασιστεί πάνω στην δομή και τις συναρτήσεις της εκφώνησης. Πέραν από τις βασικές λειτουργίες που αφορούν την δημιουργία, υπογραφή, αποστολή και επαλήθευση συναλλαγών και blocks, δώσαμε έμφαση στην δημιουργία επιπλέον λειτουργιών των κόμβων οι οποίες είτε διευκόλυναν την υλοποίηση του συστήματος, είτε το βελτίωσαν αισθητά σε ταχύτητα και ευελιξία. Παρακάτω αναφέρουμε τα κυριότερα features των κόμβων καθώς και του συστήματος συνολικά.

Υποστήριξη Συναλλαγών με Πολλαπλούς Αποδέκτες

Εκτός της δημιουργίας και αποστολής συναλλαγών προς έναν κόμβο, το σύστημά μας υποστηρίζει τις ίδιες λειτουργίες για περισσότερους από έναν παραλήπτες. Ο κύριος λόγος που ενσωματώσαμε αυτό το feature είναι για να κάνουμε ευκολότερα και γρηγορότερα την αρχική διανομή χρημάτων από τον bootstrap κόμβο προς τους υπόλοιπους συμμετέχοντες. Εκτός της πολύ ειδικής χρήσης που κάνουμε με αυτό το feature, πιστεύουμε οτι θα αυξάνει την ταχύτητα και την απόδοση του συστήματος, καθώς αντί να γίνονται μεμονωμένα οι συναλλαγές πλεόν θα γίνονται μαζικά. Για να πετυχουμε την παραπάνω λειτουργικότητα δεν χρησιμοποιούμε το πεδίο receiver_address, ενώ δημιουργούμε παραπάνω από 2 transaction outputs όπου απαιτείται.

Διάσπαση Transaction Outputs σε Πολλαπλά Κομμάτια

Επιλέξαμε τα outputs μιας συναλλαγής να τα χωρίζουμε σε μικρότερα κομματια. Με αυτό τον τρόπο μπορεί το σύστημα να δημιουργεί συναλλαγές δεσμεύοντας μικρότερα κομμάτια του συνολικού balance ενός πορτοφολιού, επιτρέποντας έτσι τη δημιουργία περισσότερων συναλλαγών πρωτού κριθεί απαραίτητη η αναμονή μέχρι την εφαρμογή του επόμενου block. Για παράδειγμα, στην περίπτωση που ένας κόμβος έχει 100 NBC συσσωρευμένα σε 1 UTXO θα μπορεί να εκτελέσει 1 συναλλαγή μέχρι την εφαρμογή του επόμενου block. Αντίθετα, αν αυτά τα 100 NBC είναι μοιρασμένα σε 5 UTXO των 20 NBC θα μπορεί να εκτελέσει 5 συναλλαγές και έτσι η απόδοση του συστήματος θα αυξηθεί.

Δυναμική Διαχείριση Συναλλαγών και Χωρητικότητας Block

Σε κάθε κόμβο κρατάμε μια ουρά με όλες τις συναλλαγές οι οποίες δεν έχουν μπεί ακόμα σε block. Από αυτήν την ουρά γεμίζουμε blocks και όταν ο αριθμός συναλλαγών φτάσει στο block capacity ξεκινάει η διαδικασία του mine. Ωστόσο πολλές φορές μπορεί να περιμένουμε αρκετή ώρα μέχρι να γεμίσει ένα block, ειδικά αν το block capacity γίνει αρκετά μεγάλο. Ο παραπάνω σχεδιασμός μερικές φορές θα είχε ως αποτέλεσμα να μην προστίθενται νέα block στο chain αν δεν συμπληρωθεί το capacity, πράγμα που αντιμετωπίσαμε μειώνοντας προσωρινά το capacity όσο περνάει χρόνος με συναλλαγές σε αναμονή.

API Επικοινωνίας Κόμβων

- **"/accept-nodes"**
  Δέχεται μια λίστα από κόμβους και τις πληροφορίες που χρειάζεται για επικοινωνία μ'αυτούς

- **"/submit-blocks"**
  Δέχεται μια λίστα από blocks και προσπαθεί να τα βάλει στο chain

- **"/submit-txs"**
  Δέχεται μια λίστα από transactions και προσπαθεί να τα βάλει σε blocks

- **"/bootstrap-node"**
  Πληροφορεί τον bootstrap για τα στοιχεία ενός κόμβου και συνδέεται μ'αυτόν

- **"/chain-length"**
  Επιστρέφει το μήκος του τρέχοντος chain

- **"/chain-tail/{length}"**
  Επιστρέφει τα τελευταία 'length' blocks της αλυσίδας

Αρχικοποιηση Κόμβου και Λειτουργίας Συστήματος

Οι παράμετροι λειτουργίας του συστήματος έχουν reasonable defaults τα οποία μπορούν να γίνουν override με κατάλληλα commandline arguments στο εκτελέσιμο του κόμβου.

Η βασική λειτουργία του κόμβου έχει διαχωριστεί σε jobs, κάθε ένα απ' τα οποία αναλαμβάνει διαφορετικές ευθύνες του συστήματος:

- **ServeApiForCli**
  Υπεύθυνο για την παροχή του API προς την cli εφαρμογή

- **ServeApiForNodes**
  Υπεύθυνο για την παροχή του API προς τους υπόλοιπους κόμβους

- **SelectMinedOrIncomingBlock**
  Υπεύθυνο για την διαχείρηση blocks που λήφθηκαν από γειτονικό κόμβο ή που ολοκλήρωσαν την διαδικασία mine. Υλοποιήθηκε ως parallel job που κάνει blocking wait μέχρι να υπάρχει επόμενο block. Έτσι, έχουμε σχεδόν στιγμιαία ανταπόκριση είτε ολοκληρωθεί η διαδικασία mine στον παρόντα κόμβο είτε έρθει από γειτονικό κόμβο νεο block. Σ' αυτό το job γίνεται ο χειρισμός της ακύρωσης της διαδικασίας mine σε περίπτωση εισερχόμενου block, ο έλεγχος για πιθανά conflict κατά την εφαρμογή καθώς και η έναρξη του χειρισμού τους.

- **CheckTxQueueForMining**
  Υπεύθυνο για την δημιουργία νέων blocks και την έναρξη της διαδικασίας mine εφόσον έχει συγκεντρωθεί ο κατάλληλος αριθμός συναλλαγών. Επιπλέον, φροντίζει να μειώνει τη χωρητικότητα ενός block εφόσον το σύστημα είναι σε αναμονή αρκετή ώρα και υπάρχουν συναλλαγές που περιμένουν να εισαχθούν σε νέο block (βλ. Δυναμική Διαχείριση Συναλλαγών και Χωρητικότητας Block).

- **ConnectToBootstrapJob**
  Υπεύθυνο για την εκκίνηση της διαδικασίας σύνδεσης με τον bootstrap σε κόμβους που δεν έχουν όριστεί ως bootstrap. Φροντίζει για την αποστολή των στοιχείων του κόμβου (ip, port κλπ.) στον bootstrap κόμβο και την παραλαβή μοναδικού id.

Noobcash CLI

Υλοποιήσαμε ένα cli με τις παρακάτω εντολές:

- **t <recipient_address> <amount>**
  Δημιουργία transaction προς τον χρήστη με id <recipient_address>

- **view**
  Τυπώνει τα transactions που βρίσκονται στο τελευταίο επικυρωμένο block

- **balance**
  Ο χρήστης βλέπει τα χρήματα που έχει στο wallet

- **utxos**
  Τυπώνει τα UTXOs του χρήστη

- **s <filename>**
  Δημιουργεί πολλαπλά transactions που περιέχονται στο αρχείο <filename>

- **stats**
  Τυπώνει ορισμένα στατιστικά για τον κόμβο, όπως τον χρόνο που πήρε σε ένα block για να δημιουργηθεί, τον μέσο χρόνο δημιουργίας block, τον συνολικό χρόνο δημιουργίας block, το μέγεθος του chain, τον αριθμό των συναλλαγών και το throughput του δικτύου

- **help**
  Επεξήγηση των εντολών

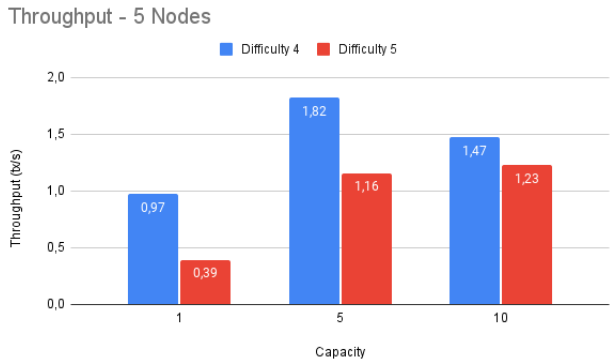
Πειράματα

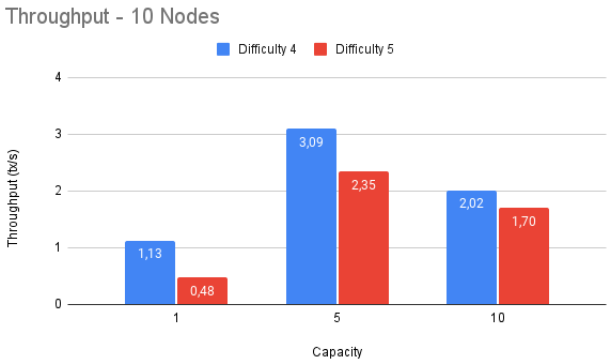
Για την διεξαγωγή των πειραμάτων χρησιμοποιήσαμε τη λειτουργικότητα του συστήματος μας που επιτρέπει ανάγνωση και εκτέλεση transactions από αρχείο (βλ. Noobcash CLI). Επιπλέον, κάναμε χρήση του flag -w που φροντίζει να επαναλμβάνει συναλλαγές που απορρίφθηκαν, λόγω έλλειψης NBC, έως αυτές να γίνουν αποδεκτές. Αυτό βέβαια είχε ως αποτέλεσμα το σύστημα να μπλοκάρει πολύ γρήγορα όταν έβρισκε συναλλαγές που απαιτούσαν περισσότερα NBC από αυτά που βρίσκονται στην κυκλοφρία του Noobcash. Για παράδειγμα, συναλλαγές που απαιτούν περισσότερα από 100*n NBC, σε σύστημα n κόμβων είναι αδύνατες. Έτσι, αποφασίσαμε να τροποποιήσουμε ελαφρώς τα αρχεία των συναλλαγών ώστε να αποφύγουμε αυτό το πρόβλημα. Καταφέραμε έτσι να εκτελούμε 441/500 (88,2%) συναλλαγές στην περίπτωση 5 κόμβων και 947/1000 (94.7%) στην περίπτωση 10 κόμβων.

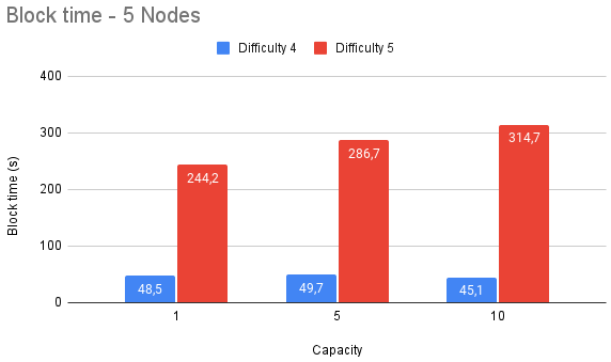
Για τον υπολογισμό του Throughput υπολογίσαμε για κάθε κόμβο το χρονικό διάστημα από την αποδοχή της πρώτης συναλλαγής έως την εφαρμογή της τελευταίας συναλλαγής του τελευταίου block. Στη συνέχεια, βρήκαμε τον Μ.Ο. για όλους τους κόμβους και τον χρησιμοποίσαμε στα διαγράμματα μας.

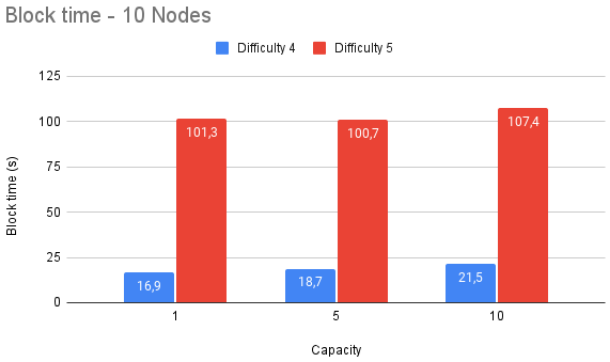
Για τον υπολογισμό του Block time υπολογίσαμε για κάθε κόμβο τον μέσο χρόνο που χρειάζεται ένα block για να γίνει mine. Αυτός ο χρόνος περιλαμβάνει τον χρόνο επικύρωσης των συναλλαγών του block και τον χρόνο επιλογής κατάλληλου nonce.

Απόδοση Συστήματος



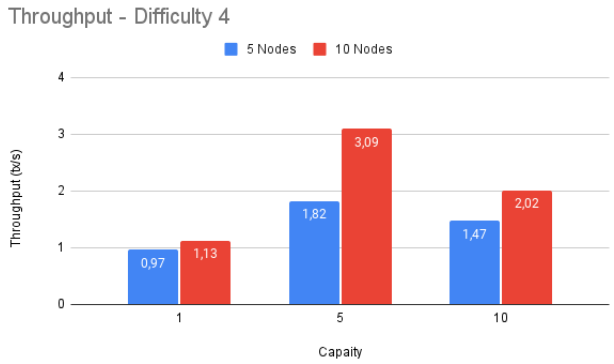


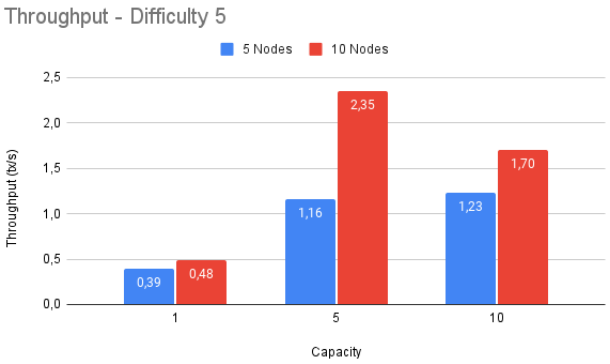


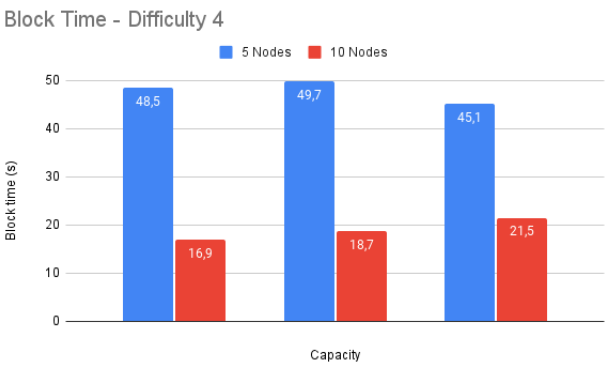


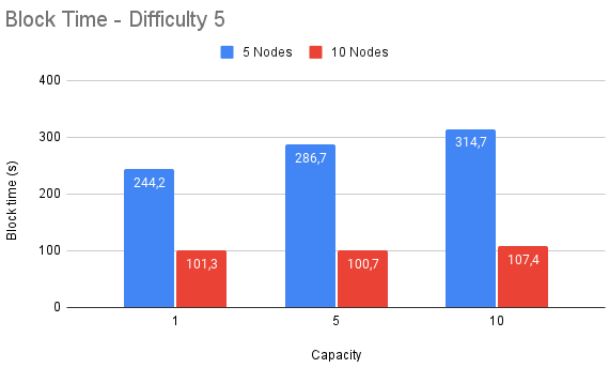
Παρατηρούμε ότι σε κάθε περίπτωση η χρήση μεγαλύτερου difficulty οδηγεί σε μειωμένο Throughput και αυξημένο Block Time. Αυτό είναι αναμενόμενο καθώς ο χρόνος που απαιτείται για την εύρεση του κατάλληλου nonce και κατ' επέκταση το να γίνει mine ένα block και άρα να εξυπηρετηθούν τα transactions είναι μεγαλύτερος όσο αυξάνουμε το difficulty.

Κλιμακωσιμότητα Συστήματος









Συγκρίνοντας τα αποτελέσματα των πειραμάτων μας για τις περιπτώσεις που έχουμε 5 και 10 κόμβους παρατηρούμε ότι το Throughput αυξάνεται και το Block Time μειώνεται όσο οι κόμβοι αυξάνονται. Δεδομένης της καλής σχεδίασης του συστήματος μας σε περιπτώσεις κλιμάκωσης των κόμβων, αυτό είναι αναμενόμενο, αφού στο σύστημα διακινούνται περισσότερες συναλλαγές και τα block γεμίζουν και γίνονται mine πιο συχνά.