Κατανεμημένα Συστήματα

Εξαμηνιαία Εργασία - Χειμερινό εξάμηνο 2019-2020

Αραβαντινός-Σιμωνέτος Ιωάννης ***03115103***
Γιαννούλης Αλέξανδρος ***03115199***
Γεωργόπουλος Παναγιώτης ***03115169***

Noobcash

Το ψηφιακό κρυπτονόμισμα του μέλλοντος

Α μέρος - **Σχεδιασμός συστήματος**

Για την ανάπτυξή του παρόντος συστήματος blockchain , χρησιμοποιήσαμε python. Για την δημιουργία του api , κάναμε χρήση του web framework flask . Η αλληλεπίδραση της εφαρμογής με τον χρήστη γίνεται μέσω μιας εφαρμογής cli μέσω απλών εντολών και υποστήριξης στην χρησιμοποίηση τους

Usage:
Commands:
--> 't <recipient_address> <amount>' --- Choose this command to create a new transaction

--> 'bulk_transactions' --- Choose this command to import a file of transactions

--> 'view' --- Choose this command to view last transactions

--> 'balance' --- Choose this command to show the balance

--> 'help || -h' --- Choose this command to get yourself some help ASAP

--> 'state' --- Choose this command to get yourself a readable view of all data

--> 'benchmark' --- Choose this command to get benchmark data. Better use when mining has stopped

--> 'exit' --- Choose this command to exit client mode

Στο πακέτο με τους κώδικες συμπεριλαμβάνεται αρχείο README.md με τις οδηγίες εγκατάστασης της εφαρμογής. Το αρχείο με το api περιλαμβάνει τόσο το στήσιμο του server , όσο και τα endpoints της εφαρμογής μας. Στην συνέχεια ,εκμεταλευτήκαμε την αντικειμενοστρέφια της python και ορίσαμε την ύπαρξη αντικειμένων τόσο για τα transactions , όσο και για κάθε block και για το blockchain συνολικά, με τους κώδικες των κλάσεων να είναι ασφαλώς σε διαφορετικά αρχεία. Το wallet για τον κάθε κόμβο επίσης υλοποιείται σε αυτοτελές αρχείο. Ο κώδικας συμπληρώνεται με διάφορες λειτουργίες με τα αρχεία setupNetwork (αρχική εγκατάσταση των κόμβων, μετά την σύνδεση του καθένα),utilities (βασικές μετατροπές μεταξύ τύπων δεδομένων , υλοποίηση consensus algorithm κτλ) , mining και data (δήλωση διαφόρων βασικών μεταβλητών του συστήματος, όπως capacity και difficulty).
Χρησιμοποιήσαμε επίσης , το module pipenv , ως package manager, ενώ οι βιβλιοθήκες που χρειάστηκε να εγκαταστήσουμε ήταν οι flask,requests και pycryptodome.

Η ροή του κώδικα ακολουθεί το πρότυπο που περιγράφεται στην εκφώνηση της εργασίας.

☐ Αρχικά λοιπόν, κάθε κόμβος μετά τον πρώτο, δηλώνει την παρουσία τους στο δίκτυο μέσω του end point setup.
☐ Όταν εγγραφούν όλοι οι “participants” κόμβοι, ο admin μοιράζει στον καθένα τις απαραίτητες πληροφορίες για το όλο δίκτυο (urls ,public keys) και τους ξεχωρίζει με ένα κατάλληλο id.
☐ Στην συνέχεια ο πρώτος κόμβος λαμβάνει το αρχικό προκαθορισμένο ποσό στο wallet , το οποίο και μοιράζει στους υπόλοιπος .Το transaction αυτό τοποθετείται στο genesis block. Έτσι ,όταν ολοκληρωθεί αυτή η

αυτόματη διαδικασία, ο κάθε κόμβος καταλλήγει με 100 μονάδες "ψηφιακού χρήματος" στο wallet του .
☐ Από εκεί και πέρα και με την χρήση των εντολών του τερματικού (παρέχεται και εντολή help για καθοδήγηση),δύναται η ολοκλήρωση κάθε συναλλαγής (χειροκίνητα αλλά και αυτόματα από αρχείο εισόδου) με την εκάστοτε κατάλληλη εντολή. Σχετικά με τις ρυθμιζόμενες παραμέτρους του συστήματος,αυτές βρίσκονται στο αρχείο data.py και είναι οι παρακάτω :

numOfParticipants=X , αριθμός συμμετέχων των κόμβων στο δίκτυο ,
capacity=Y , αριθμός transactions που λαμβάνονται προτού ξεκινήσει το mining νέου κόμβου .
difficulty=Z , όπου ορίζεται ώς ο αριθμός των μηδενικών, σε hexadecimal μορφή, με τα οποία πρέπει να ξεκινάει ένα hash κατά τους υπολογισμούς για proof of work .

Β μέρος - **Αποτελέσματα των πειραμάτων**

Μέθοδος:

Δημιουργήσαμε κατάλληλη επιλογή (benchmark) στο cli ώστε να λαμβάνουμε τα ζητούμενα δεδομένα μετά την εκτέλεση των transactions απο τα δοθέντα αρχεία (επίσης με εντολή από το cli , την bulk_transactions)

Για το throughput μετράμε τον χρόνο από την στιγμή που δεχτήκαμε νέο request για δημιουργία transaction μέχρι την στιγμή που εξυπηρετήθηκε το συγκεκριμένο request.Ο τελικός υπολογισμός αποτελεί τον μέσο όρο του throughput μεταξύ των κόμβων.Αντίστοιχα , για το Block time , υπολογίζουμε τον χρόνο από την στιγμή που εκκινείται η διαδικασία εύρεσης proof of work (nonce) μέχρι και την στιγμή που ολοκληρώνεται και η εύρεση του hash του κόμβου (και τελικά τον μέσο όρο για το σύνολο των κόμβων).

Η μετρήσεις για 5 κόμβους έγιναν εκτελώντας ένα back-end participant σε κάθε μηχάνημα πως μας είχε δοθεί στον ωκεανό, ενω για τους 10 κόμβους εκτελούσαμε 2 participants ανά μηχάνημα.

Για 5 κόμβους :

Difficulty:*4*

Capacity	**Throughput**	**Block time**
1	0.53	2.38
5	0.19	6.80
10	0.20	12.60

Difficulty:*5*

Capacity	**Throughput**	**Block time**
1	0.59	33.50
5	0.19	94.23
10	0.20	152.42

Για 10 κόμβους :

Difficulty:*4*

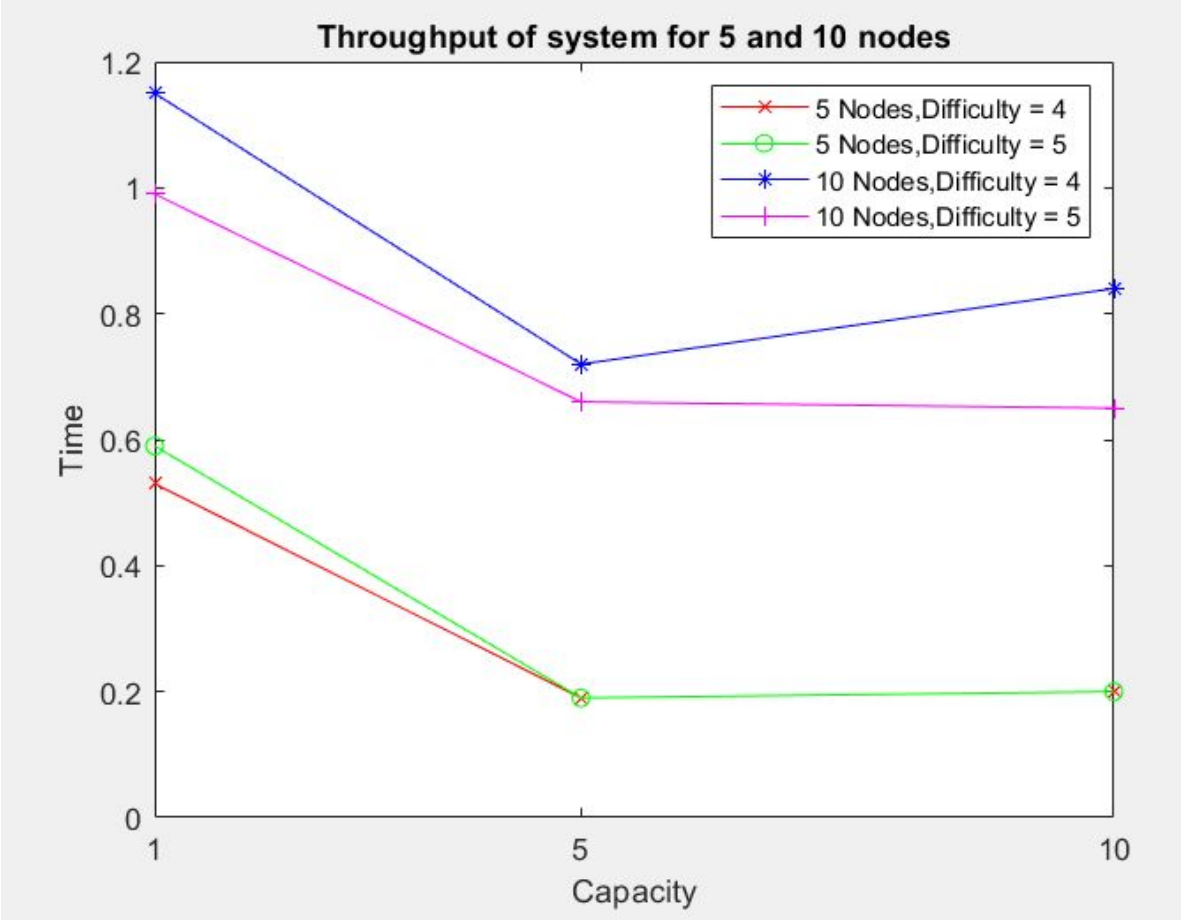
Capacity	**Throughput**	**Block time**
1	1.15	2.51
5	0.72	6.52
10	0.84	12.10

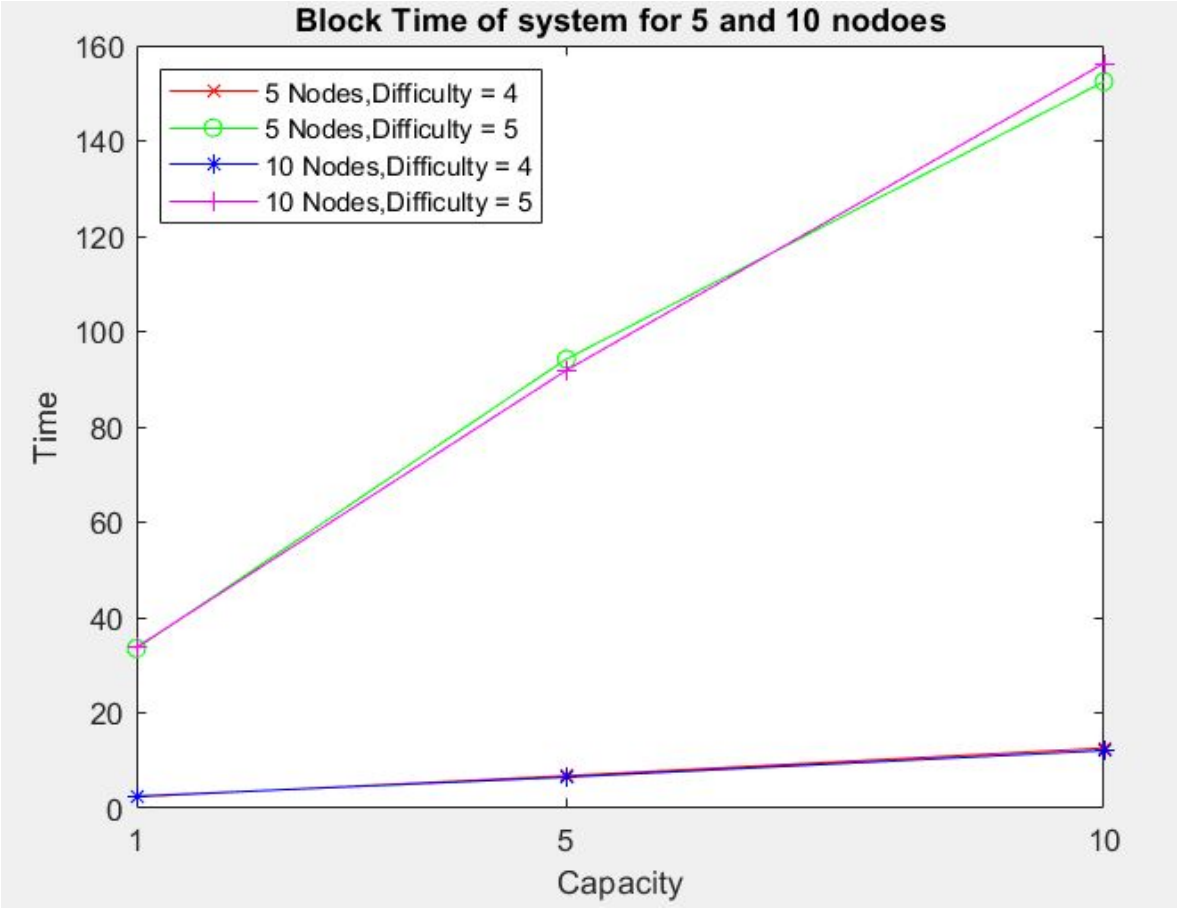
Difficulty:*5*

Capacity	**Throughput**	**Block time**

1	0.99	33.75
5	0.66	91.87
10	0.65	156.25

Γραφικές παραστάσεις :





Σχολιασμός και επεξήγηση:

Για το **Throughput**, αρχικά παρατηρούμε πως το throughout σε κάθε περίπτωση το πολύ ενάμισι δευτερόλεπτο, το οποίο σημαίνει πως πράγματι το σύστημα μας εξυπηρετεί με ικανοποιητικό ρυθμό τις αιτήσεις για νέα transactions.

Επιπροσθέτως βλέπουμε μια εξάρτηση από τον αριθμό των participants, καθώς όσο αυξάνεται ο αριθμός αυτός τόσο περισσότερα πακέτα πρέπει να αποσταλούν στο δίκτυο μας ανά transaction και άρα τόση μεγαλύτερη η συμφόρηση και η καθυστέρηση που παρατηρείται.

Στην συνέχεια βλέπουμε πως για σταθερό αριθμό participant στις περιπτώσεις για block capacity = 5 ή 10 το throughput είναι ίδιο γεγονός, το οποίο και ήταν κάπως αναμενόμενο να μην υπάρχει εξάρτηση, καθώς η αποστολή και επικύρωση των transaction γίνεται παράλληλα με το mining σε ξεχωριστά ανεξάρτητα νήματα. Όταν όμως το capacity γίνει ίσο με 1 τότε το block time γίνεται συγκρίσιμο (για λόγους που θα εξηγηθούν

παρακάτω) με τον χρόνο αποστολής transaction και άρα στο δίκτυο συνυπάρχουν περισσότερα πακέτα και άρα αυξάνεται η συμφόρηση και η καθυστέρηση. Αυτή η συμπεριφορά παρατηρείται και επιβεβαιώνεται και στις τέσσερις περιπτώσεις όπου το capacity είναι ίσο με τη μονάδα.

Τέλος, η μετρική αυτή παρουσιάζει έμμεση εξάρτηση από το difficulty. Επειδή με μικρό difficulty μειώνεται ο χρόνος που χρειάζεται να γίνει ένα block mine, όσο μικραίνει το difficulty και εφόσον το block time γίνεται συγκρίσμο τόσο δυσχεραίνεται το throughput.

Για το **block time**, παρατηρούμε αρχικά το αναμενόμενο. Όσο αυξάνει το difficulty επηρεάζεται ο χρόνος mining ενός block καθώς ο τελευταίος αυξάνει. Στην υλοποίηση του κώδικα μας απαιτούμε το hash ενός block να ξεκινάει με difficulty αριθμό από hexadecimal μηδενικά, άρα κάθε φορά που αυξάνεται το difficulty κατά 1 περιμένουμε αύξηση περίπου 16 φορές. Κοιτάζοντας τις μετρήσεις αυτή η συμπεριφορά επιβεβαιώνεται.

Επίσης, παρατηρείται μια εξάρτηση του χρόνου από τον αριθμό των transactions σε ένα block, δηλαδή το capacity. Αυτό συμβαίνει διότι ο υπολογισμός του hash γίνεται μετατρέποντας ,μεταξύ άλλων ,τα transactions σε hashable μορφή. Αυτό σε συνδυασμό με την σωρεία προσπαθειών που γίνεται κατά το mining οδηγεί σε αισθητή αύξηση του block time.

Τέλος, φαίνεται πως το block time δεν εξαρτάται από τον αριθμό των χρηστών μέσα στο δίκτυο. Αυτό ήταν αναμενόμενο στην περίπτωση μας καθώς μετράμε το block time από την στιγμή που ξεκινήσει το thread που το υλοποιεί, έως ότου βρεί κατάλληλη λύση του proof of work. Συνεπώς δεν εξαρτάται από το broadcast του block και από το αν γίνει αποδεκτό ή όχι. Το mining είναι ένα ξεχωριστό thread το οποίο επηρεάζεται μόνο από τα αρχικά δεδομένα με το οποίο δημιουργείται, ενώ η μετρική του δεν λαμβάνει υπόψιν το δίκτυο.