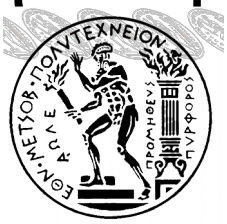
Κατανεμημένα Συστήματα
Εξαμηνιαία Εργασία



Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Μέλη Ομάδας:

ΑΠΟΣΤΟΛΟΥ ΑΘΑΝΑΣΙΟΣ	03112910
ΠΑΝΑΓΙΩΤΟΠΟΥΛΟΣ ΒΑΣΙΛΗΣ	03114303
ΔΟΥΦΑΣ ΕΥΓΕΝΙΟΣ	03115728

NoobCash

Περιγραφή του κώδικα

Έχουμε υλοποιήσει το project σε javascript με περιβάλλον node js. Όταν εκτελείται το πρόγραμμα δημιουργεί ένα instance κλάσης Node η οποία βρίσκεται στο αρχείο Node.js.

Node

Ο κάθε node περιέχει ένα instance κλάσης Wallet από το αρχείο Wallet.js, ένα instance κλάσης Blockchain του αρχείου Blockchain.js, ένα instance κλάσης Rest του αρχείου Rest.js, ένα instance κλάσης Cli του αρχείου Cli.js. Περιέχει επίσης μια λίστα contacts[] με javascript object τύπου contact όπου έχει πληροφορίες για τους υπόλοιπους κόμβους. Ο node έχει ως μεθόδους τις συναρτήσεις που τελούν την λογική της άσκησης.

Wallet

Περιέχει το privatekey και το publickey του συγκεκριμένου node.

Rest

Χρησιμοποιεί την βιβλιοθήκη express για να φτιάξει ένα api στο οποίο ακούει ο κόμβος για την επικοινωνία των άλλων κόμβων. Την χρησιμοποιούμε τόσο για τις λειτουργίες του backend (rest api) όσο και για ένα μινιμαλιστικό fontend με το οποίο ο χρήστης μπορεί να δει κάποιες πληροφορίες του κόμβου καθώς και να εκτελέσει τις λειτουργίες του frontend.
Το backend ακούσει στα url
/backend/receivecontacts
/backend/receiveblockchain
/backend/receivetransaction
/backend/receiveblock
/backend/askedblockchain
/backend/readfile
/backend/transactionended

Το frontend υλοποιείται στο url / που έχει μια απλή html σελίδα για τις λειτουργίες του client (transaction, view, balance, help). Έχουμε προσθέσει κάποιες επιπλέον επιλογές που δείχνουν τα στοιχεία του κόμβου για λόγους debugging και μια επιλογή που δείχνει τα measurements του κόμβου για το throughput και το block_time.

Cli

Επιπλέον του frontend μέσω browser, έχουμε προσθέσει και έναν client γραμμής εντολών ο οποίος δέχεται τις εντολές
t <recipient_adress> <amount>
view
balance
help

Contacts[]
Είναι μια λίστα με n objects, όπου κάθε object έχει τις πληροφορίες για κάθε κόμβο: ip, port, publickey, UTXO. Όπου UTXO είναι λίστα με τα unspent transactions κάθε κόμβου.

Blockchain
Το blockchain περιέχει μια λίστα από instances της κλάσης Block του αρχείου Block.js.

Block
Το κάθε block με τη σειρά του περιέχει μια λίστα από instances της κλάσης Transaction του αρχείου Transaction.js.

Transaction
Η κλάση που ορίζει ένα transaction.

BootstrapNode
Είναι μια κλάση που κάνει extend την κλάση Node και αποτελεί ειδική περίπτωση του 1ου κόμβου που εκτελείται. Περιέχει όλες τις λειτουργίες του Node με μερικές επιπλέον ακόμα.

Περιγραφή της Λειτουργίας του Συστήματος

Αρχικοποίηση

Εκτελούμε κάποιον κόμβο δίνοντάς του τις πληροφορίες
bootstrap_ip, bootstrap_port, ip, id, n, capacity, difficulty
Ο κάθε κόμβος αρχικοποιεί τα μεγέθη του, φτιάχνει το wallet του και εκκινεί το rest controller του.
Σημειώνουμε ότι δίνουμε εμείς το id κάθε κόμβου και δεν το παίρνει από τον bootstrap όπως ζητάει η εκφώνηση. Αυτό γίνεται επειδή θέλουμε να μπορούν όλοι οι κόμβοι να τρέξουν στο ίδιο μηχάνημα για τους σκοπούς των πειραμάτων (πέρα των μηχανημάτων του okeanos) και έχουμε βάλει τον κάθε κόμβο να ακούει στην ip του στο port 3000+id. Έτσι καταφέρνουμε αντιστοιχία port και id, δηλαδή ο node0 (bootstrap) ακούει στο port 3000, ο node1 στο port 3001 κτλπ.

Αρχικά βήματα εκτέλεσης

Εκτελείται αρχικά ο bootstrap node ο οποίος φτιάχνει το genesis block με το 1ο transaction που του δίνει 100*n nbc νομίσματα και περιμένει μέχρι να λάβει τα contact από όλους τους υπόλοιπους κόμβους. Κάθε επόμενος κόμβος που εκκινείται στέλνει την ip του και το publickey του ως contact στον bootstrap node. Όταν ο bootstrap λάβει n συνολικά contacts τότε τα κάνει broadcast σε όλους τους κόμβους. Έπειτα κάνει broadcast το blockchain του σε όλους τους υπόλοιπους κόμβους. Ο κάθε άλλος κόμβος λαμβάνει το blockchain του bootstrap και ενημερώνει το δικό του. Έπειτα ο bootstrap αρχίζει και δημιουργεί ένα transaction για κάθε άλλο κόμβο που του μεταφέρει 100 nbc, χρησιμοποιώντας την create_transactions(). Αφού δημιουργήσει αυτά τα αρχικά transactions τότε λέει σε όλους τους κόμβους (και στον εαυτό του) να αρχίσουν να εκτελούν την read_file() στην οποία κάθε κόμβος διαβάζει το αντίστοιχο αρχείο του
‘data/transactions/<N>nodes/transactions<ID>.txt’
και βάζει τα στοιχεία κάθε transaction κάθε γραμμής στην λίστα transactions_to_create. Έπειτα καλείται η create_transactions() που δημιουργεί ένα transaction για κάθε ένα αντικείμενο στην λίστα. Αφού εκτελεστούν αυτά τα transactions ο κόμβος ενεργοποιεί και το cli του ώστε να μπορεί να δεχτεί εντολές και από εκεί πέρα από τον html client του.

Διαδικασία εκτέλεσης transactions και block mining

Ο κάθε κόμβος στην μέθοδο create_transactions(), αφαιρεί το 1ο αντικείμενο της λίστας transactions_to_create και καλεί την create_transaction() με αυτό. Όταν ένας κόμβος κάνει create_transaction() τότε φτιάχνει ένα νέο instance της κλάσης Transaction, το αρχικοποιεί με τα στοιχεία που του έχουν δοθεί και το κάνει sign. Έπειτα καλεί την broadcast_transaction() η οποία το κάνει broadcast σε όλους τους άλλους κόμβους χτυπώντας το rest api τους στο url /backend/receivetransaction το οποίο ενεργοποιεί την action_receivetransaction() σε αυτούς τους κόμβους, και καλεί την action_receivetransaction() στον εαυτό του.

Όταν σε ένα κόμβο καλείται η action_receivetransaction() τότε αυτός προσθέτει το transaction σε μια λίστα pending_transactions και αν ο κόμβος δεν εκτελεί κάποιο transaction εκείνη την στιγμή τότε καλείται η execute_transactions().

Η execute_transactions() παίρνει το 1ο transaction από την λίστα και το εκτελεί καλώντας την execute_transaction(). Όταν τελειώσει καλεί αναδρομικά τον εαυτό της μέχρι η λίστα με τα pending_transactions να αδειάσει.

Η execute_transaction() επικυρώνει το transaction που της έχει δωθεί ελέγχοντας αν το signature είναι verified, αν το hash της transaction είναι σωστό και αν επαρκούν τα χρήματα του UTXO του sender για το amount που στέλνει στον receiver. Αν οι έλεγχοι επιτύχουν τότε προσθέτει το transaction στην λίστα transactions του τελευταίου block του blockchain. Αν με την πρόσθεση του transaction γεμίσει το block τότε εκκινεί την διαδικασία mine_block().

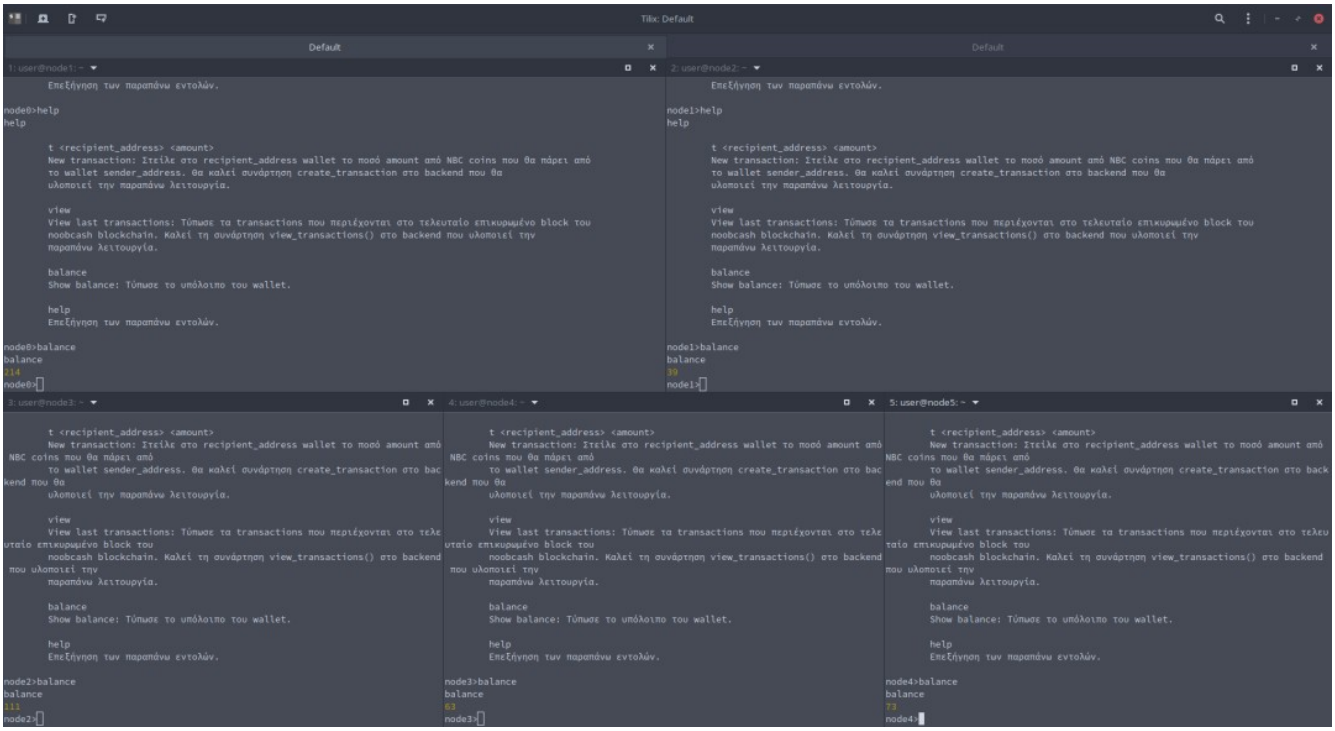
Η mine_block() δημιουργεί ένα νέο instance της κλάσης Block με previous_hash το hash του τελευταίου block της blockchain και καλεί την μέθοδό του Block.mineBlock() η οποία συνεχώς κάνει hash το block μέχρι το hash να έχει τόσα αρχικά μηδενικά όσο το difficulty. Έπειτα ελέγχει ότι το block έχει ως previous hash το hash του τελευταίου block (δηλαδή ότι δεν έχει λάβει κάποιο block μέχρι εκείνη την στιγμή) και αν επιτύχει ο έλεγχος καλεί την broadcast_block() η οποία το στέλνει σε όλους τους κόμβους χτυπώντας το rest api τους στο url /backend/receiveblock που καλεί την action_receiveblock() σε αυτούς τους κόμβους, και μετά καλεί ο ίδιος την action_receiveblock() στον εαυτό του.

Η action_receiveblock() ελέγχει ότι το block που έλαβε έχει index ίσο με το index του τελευταίου block του blockchain του συν 1 (ότι δηλαδή δεν έχει λάβει ήδη κάποιο block σε αυτή την φάση). Αν ο έλεγχος επιτύχει τότε ελέγχει ότι το previous_hash του block ισούται με το hash του τελευταίου block (ότι δηλαδή δεν έχει δημιουργηθεί διακλάδωση στο blockchain) και τότε το προσθέτει στο blockchain. Αν έχει δημιουργηθεί διακλάδωση τότε καλεί την resolve_conflict() η οποία παίρνει το blockchain από όλους τους άλλους κόμβους και αντικαθιστά το δικό του με αυτό με το μεγαλύτερο μέγεθος.

Για λόγους συντονισμού, στο τέλος κάθε executed transaction καλείται η end_transaction(). Εκεί ο κόμβος που εκτέλεσε το transaction στέλνει μήνυμα στο rest api αυτού που του το είχε κάνει broadcast (sender_id) χτυπώντας το url /backend/transactionended. Ο κόμβος που έχει στείλει το transaction περιμένει να λάβει n απαντήσεις (μαζί με την δικιά του) και μόνο τότε καλεί αναδρομικά την create_transactions() για να δημιουργήσει τα υπόλοιπα. Σημειώνουμε ότι μέχρι να λάβει τις απαντήσεις συνεχίζει κανονικά να κάνει execute_transactions() για αυτά που του έχουν μείνει στην λίστα pending_transactions και αυτά που έρχονται από άλλους και μόνο όταν αδειάσει εκείνη η λίστα ξεκινάει να κάνει create το επόμενο transaction.

Εκτέλεση Πειραμάτων

Για την εκτέλεση στο okeanos συνδεόμαστε στο κεντρικό μηχάνημα vm1 με ip 83.212.76.173 με ssh και μέσω αυτού συνδεόμαστε και στα υπόλοιπα vms. Κάνουμε clone το project μας που το έχουμε στο github στην διεύθυνση https://github.com/githubBill/distrib_project_2020.
Και αρχικοποιούμε τα dependencies τρέχοντας
npm install
Εκτελούμε στο vm1 την εντολή
npm run okeanos-node0
για να τρέξουμε τον bootstrap κόμβο και στα υπόλοιπα vm2...vm5
npm run okeanos-node1...node4
για τους υπόλοιπους κόμβους.

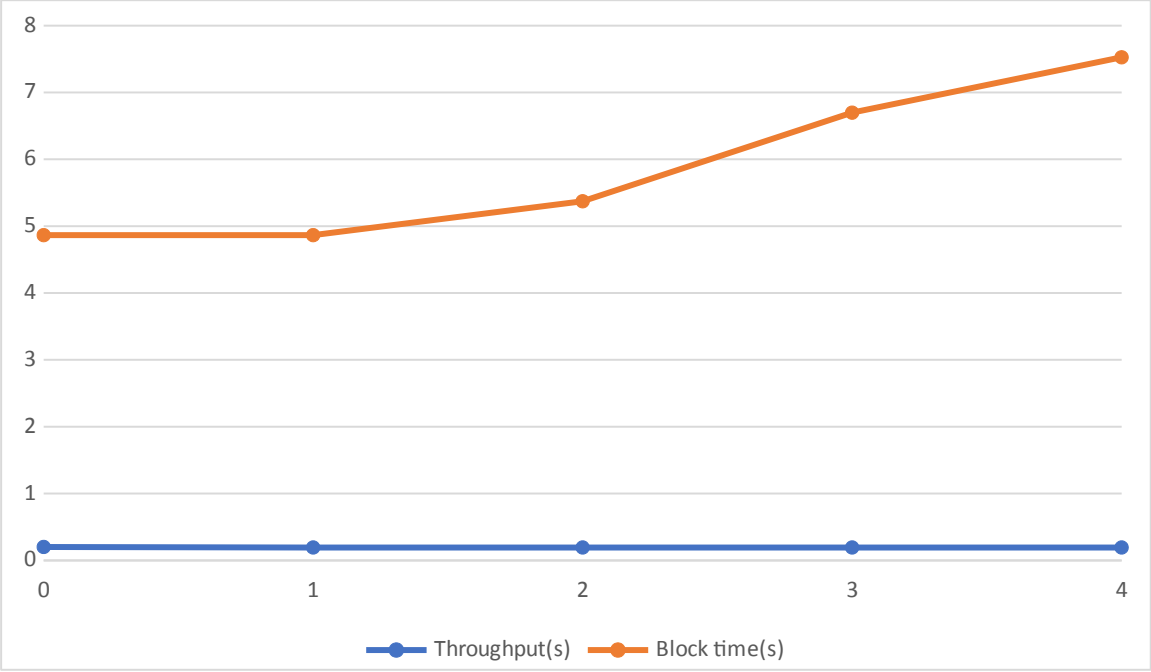


Για να επιτύχουμε ομοιογένεια των αποτελεσμάτων καθώς και για να μπορούμε εύκολα να κάνουμε scale το πείραμα σε 10 κόμβους, εκτελούμε όλα τα τα πειράματα σε έναν τοπικό υπολογιστή και όχι στον okeanos. Για αυτόν τον σκοπό έχουμε φτιάξει ένα script σε javascript το experiment.js το οποίο δέχεται 3 παραμέτρους
n=αριθμός κόμβων, capacity, difficulty
Το script αυτό εκκινεί τον bootstrap κόμβο και μετά τους υπόλοιπους n-1 σε ξεχωριστά processes. Ο κάθε κόμβος ακούει στο localhost και στο port 3000+id, δηλαδή από το port 3000 μέχρι το 3009 στην περίπτωση των 10 κόμβων.

Ο κάθε κόμβος κάνεις τις δικές του μετρήσεις για τα transactions και τον χρόνο εκτέλεσης τους όσο και τους χρόνους προσθήκης κάθε block στο blockchain. Έτσι για κάθε πείραμα βγάζουμε τα αποτελέσματα για το throughput και το block_time .Το throughput του συστήματος είναι το ελάχιστο των τιμών των κόμβων ενώ για το block time υπολογίζουμε τον μέσο όρο.

Εκτελώντας των κώδικα για Nodes=5 ,Capacity=10 ,Difficulty=4 παίρνουμε τις παρακάτω τιμές στους κόμβους. Τα αποτελέσματα είναι εμφανη στην διεύθυνση localhost:300i/measurements , όπου i το id του κόμβου .

Node	0	1	2	3	4
Throughput(s)	0.20207	0.19429	0.19429	0.19429	0.19429
Block time(s)	4.86547	4.86547	5.37267	6.69795	7.52591



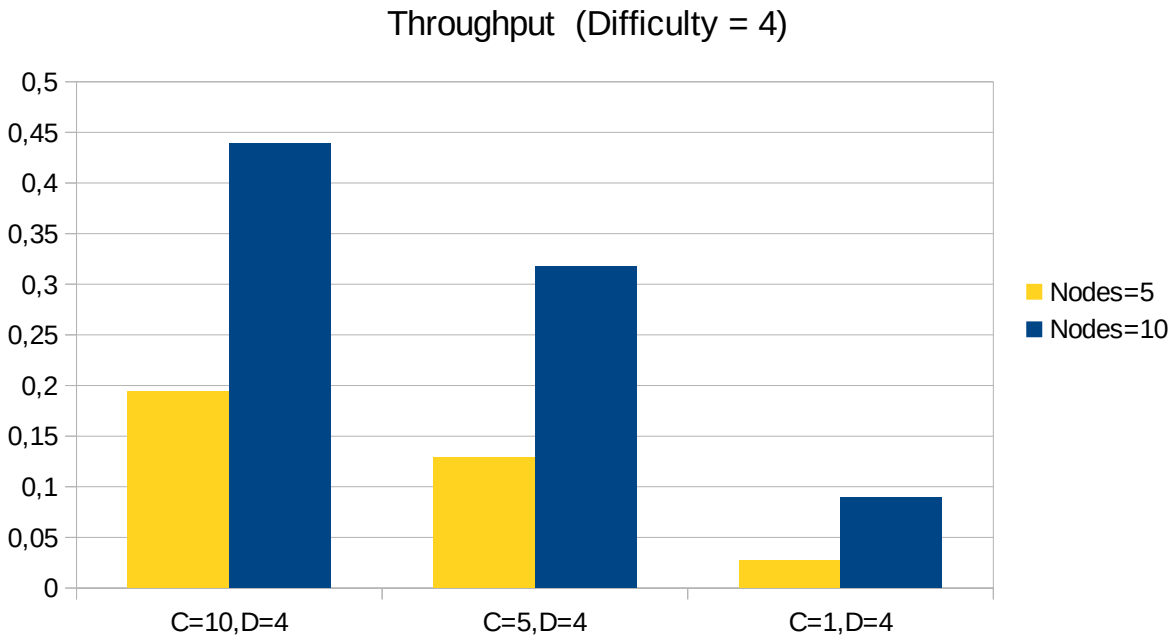
Για το παραπάνω πείραμα κραταμε throughput= 0.19429 και block time=5,86549

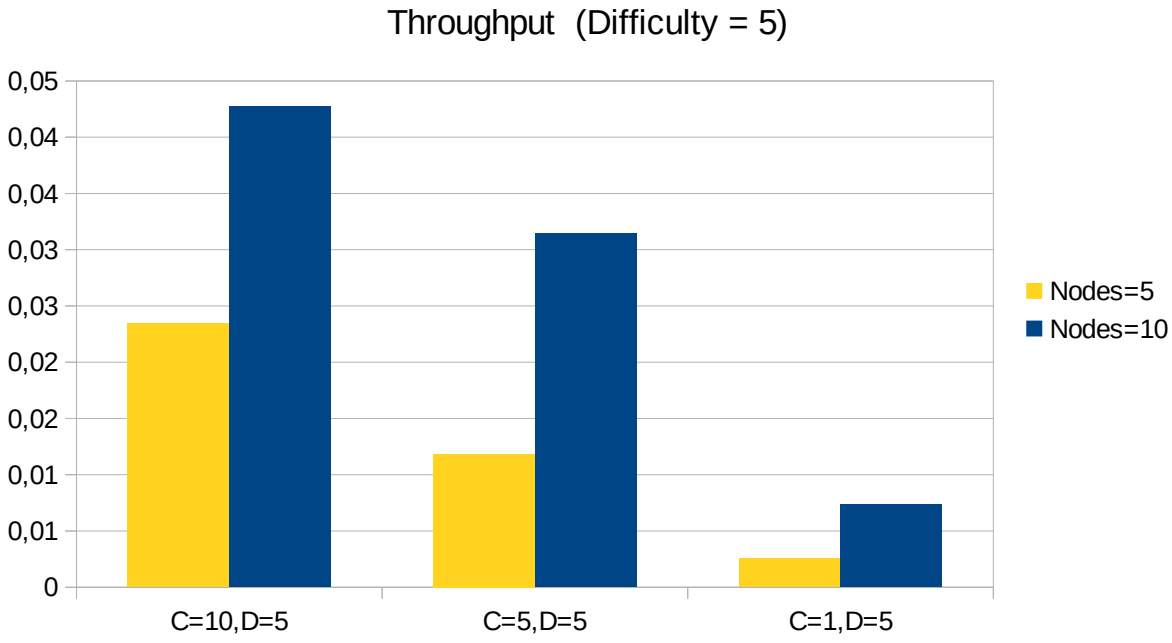
Ομοίως υπολογίσαμε τις τιμές και για τις υπόλοιπες εκτελέσεις.

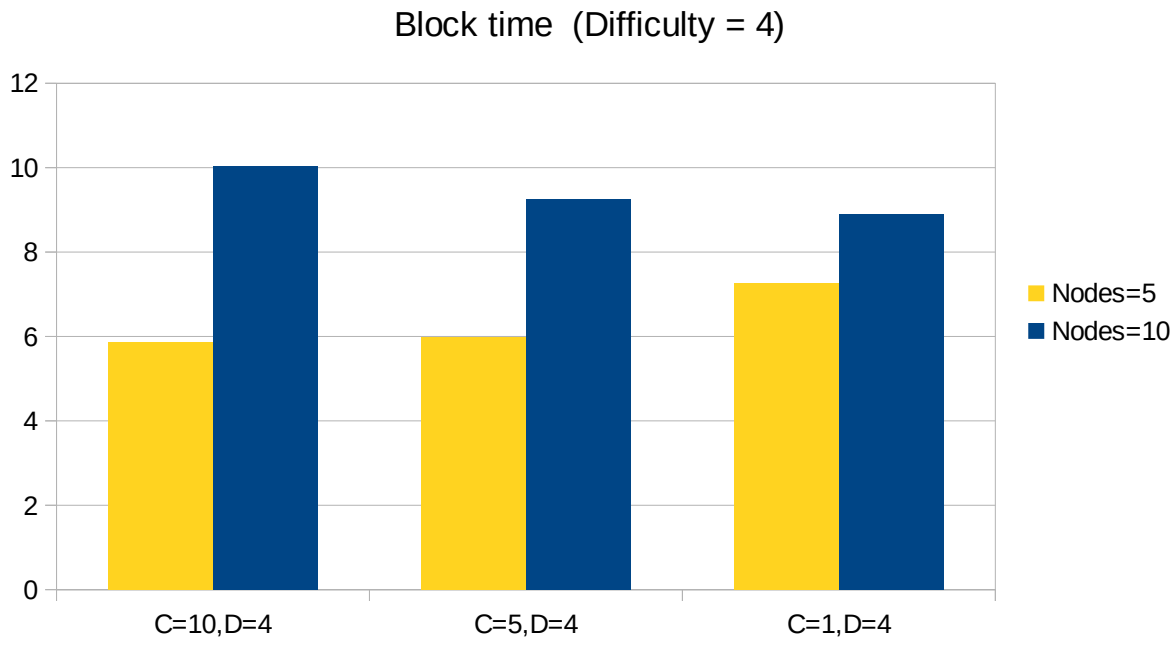
Nodes	Capacity	Difficulty	Throughput	Block time
5	10	4	0,19429	5,86549
5	5	4	0,12847	5,98732
5	1	4	0,02748	7,26534
5	10	5	0,02345	76,07103
5	5	5	0,01184	87,56472
5	1	5	0,00254	71,75643
10	10	4	0,43875	10,03604
10	5	4	0,31756	9,26454
10	1	4	0,08965	8,89485
10	10	5	0,04274	110,56372
10	5	5	0,03146	140,85736
10	1	5	0,00736	124,67462

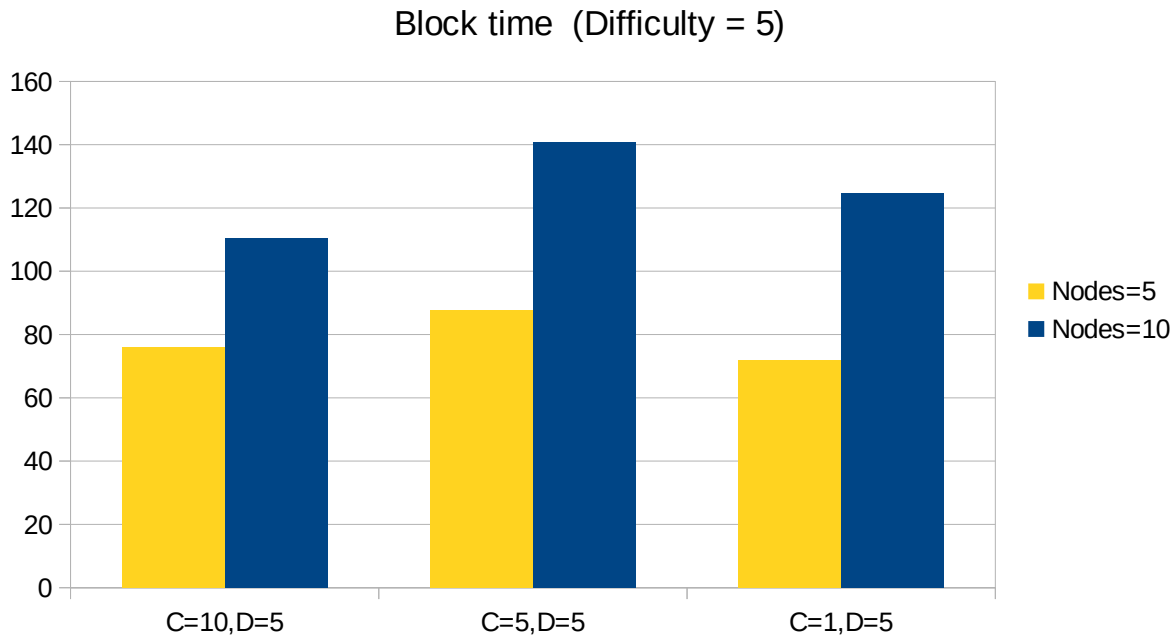
Παρατηρούμε ,με βάση τις τιμές που λάβαμε, πως το Block time εξαρτάται κυρίως από το Difficulty ,είναι τελείως ανεξάρτητο του Capacity και επηρεάζεται αλλά σε μικρότερο βαθμό από τον αριθμό των κόμβων .Το τελευταίο συμβαίνει διότι τα πειράματα εκτελέστηκαν σε τοπικό υπολογιστή και η αύξηση των κόμβων ισως να μειώνει την διαθέσιμη υπολογιστική ισχύ για το κάθε mine.
Σχετικά με το Throughput παρατηρούμε πως είναι ανάλογο του Capacity και του αριθμού των κόμβων ,καθώς όσο μεγαλύτερο το Capacity τόσο λιγότερες θα είναι οι απαιτήσεις για mining.

Κλιμάκωση συστήματος









Με την αύξηση του αριθμού των κόμβων παρατηρούμε αισθητή βελτίωση στο throughput του συστήματος , αυτό συμβαίνει διότι υπάρχουν περισσότεροι miners και πολλά block γίνονται mine πριν ξεκινήσει το mining σε όλους τους κόμβους του συστήματος. Το Block time θεωρητικά θα έπρεπε να παραμένει σταθερό αλλά στην περίπτωσή μας υπάρχει μια μικρή αύξηση του χρόνου διότι όλοι οι κόμβοι εκτελούνται στον ίδιο υπολογιστή.