ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Αναφορά εξαμηνιαίας εργασίας για το μάθημα
«Κατανεμημένα Συστήματα»

Ομάδα 49

Νικόλαος Γιαννακάκης 03110191
Ιωάννης Γιαννούκος 03118918
Δημήτριος Βασιλείου 03119830

Αρχιτεκτονική Συστήματος

Το κεντρικό στοιχείο της υλοποίησης της εφαρμογής Blockchat είναι η ενσωμάτωση ενός Flask Server σε κάθε κόμβο του συστήματος, καθώς και ενός CLI μέσω του οποίου οι χρήστες μπορούν να χρησιμοποιήσουν το Blockchat. Τα δεδομένα του blockchain βρίσκονται μέσα στο runtime των Server, όπου πραγματοποιούνται και οι λειτουργίες/μέθοδοι της εφαρμογής. Αυτό σημαίνει ότι τα δεδομένα δεν αποθηκεύονται κάπου εξωτερικά και χάνονται όταν τερματίζεται το runtime των Server.

Ο Server παρέχει REST API endpoints δύο κατηγοριών:

- Internal endpoints: Τα οποία καταναλώνονται από το “εσωτερικό” CLI της κάθε κόμβου.
- External endpoints: στα οποία πραγματοποιείται η επικοινωνία μεταξύ των διαφορετικών κόμβων.

Σημειώνεται ότι τόσο η επικοινωνία κάθε κόμβο με το CLI του, όσο και η επικοινωνία μεταξύ των κόμβων, πραγματοποιείται μέσω του πρωτοκόλλου HTTP, χρησιμοποιώντας το μορφότυπο JSON για τη μεταφορά δεδομένων.

Data modeling

Για την οργάνωση και διαχείριση των δεδομένων της εφαρμογής χρησιμοποιήθηκε αντικειμενοστραφής προσέγγιση, τα βασικότερα μοντέλα/κλάσεις της οποίας παρουσιάζονται παρακάτω:

Transaction

Η κλάση αυτή μοντελοποιεί τα transactions και έχει τις εξής μεθόδους:

- `create_transaction_string(string)` που δημιουργεί ένα string που αντιπροσωπεύει το transaction παραθέτοντας όλα τα πεδία ενός transaction, ώστε να χρησιμοποιηθεί για τη δημιουργία της υπογραφής.
- `compute_fees()` που υπολογίζει τα fees ανάλογα με τον τύπο του transaction.

Wallet

Υπάρχουν δύο είδη wallets, τα public wallets και τα private wallets.

PublicWallet

Ένα PublicWallet περιέχει το node id του κόμβου, τη διεύθυνσή του, το public key του, το amount που περιέχει, καθώς και το stake του κόμβου. Το amount και το stake βέβαια διαχωρίζονται σε `hard_amount`, `hard_stake` και `soft_amount`, `soft_stake`. Τα hard αναφέρονται σε amount και stake μετά τη λήψη, επικύρωση και προσθήκη block στην αλυσίδα, ενώ τα soft σε amount και stake μετά τη λήψη transaction πριν ακόμα «μετατραπούν» σε block. Για να έλθουν σε consensus οι κόμβοι, τα hard amount και stake θα ενημερώνονται μετά από validation ενός block, όπως θα αναλυθεί στη συνέχεια.

PrivateWallet

Ένα PrivateWallet περιέχει όλα τα παραπάνω αλλά και το private key του κόμβου. Επίσης η κλάση έχει τη μέθοδο `create_transaction()` που δημιουργεί και υπογράφει ένα transaction, με τη λογική ότι μόνο ένα PrivateWalelt μπορεί να δημιουργεί transactions.

Block

Η κλάση αυτή μοντελοποιεί τα blocks και έχει τις εξής μεθόδους:

- `add_transaction(transaction)` που προσθέτει ένα transaction στη λίστα από transactions του block.
- `create_block_hash()` που παραθέτει όλα τα πεδία ενός block δημιουργώντας ένα string το οποίο περνάει μέσα από μια hash function και επιστρέφει το τελικό, κατακερματισμένο string.

Blockchain

Η κλάση αυτή μοντελοποιεί το blockchain. Ένα blockchain περιέχει μια λίστα από blocks, ένα λεξικό `transaction_inbox`, στο οποίο εισάγονται όλα τα transactions που λαμβάνει ο τρέχον κόμβος πρωτού

ληφθεί ένα validated block, καθώς και ένα λεξικό `blockchain_transactions` που περιέχει όλα τα transactions που ανήκουν στα validated blocks του blockchain. Η κλάση έχει τις εξής μεθόδους:

- `add_block(block)` που προσθέτει ένα block στη λίστα από blocks του blockchain.

Σε αυτό το σημείο σημειώνουμε ότι όλες οι παραπάνω κλάσεις περιέχουν τις μεθόδους `to_dict()` και `from_dict()` οι οποίες μετατρέπουν ένα αντικείμενο σε λεξικό ώστε να γίνει στη συνέχεια JSON και, από ένα λεξικό κατασκευάζουν ένα αντικείμενο. Οι μέθοδοι αυτές είναι χρήσιμες, διότι παρέχουν έναν γρήγορο τρόπο μετατροπής μεταξύ JSON και αντικειμένων που είναι απαραίτητος για την επικοινωνία με HTTP.

State

Η κλάση αυτή αντιπροσωπεύει την κατάσταση που αντιλαμβάνεται ο τρέχον κόμβος, δηλαδή την εικόνα που έχει για το blockchain, τα wallets και τα stakes. Σημαντικό είναι το λεξικό `block_waiting_room` που περιέχει όσα block είναι εκτός σειράς (περισσότερες πληροφορίες αναφέρονται παρακάτω). Στην κλάση αυτή υλοποιούνται ως μέθοδοι αρκετές από τις βασικές συναρτήσεις που χρειάζεται η εφαρμογή. Μερικές από αυτές είναι οι εξής:

- `validate_transaction(transaction)` που επαληθεύει την εγκυρότητα ενός transaction με βάση την υπογραφή και το `soft_amount` των wallets. Αν το transcation είναι έγκυρο τότε αλλάζουν κατάλληλα τα `soft_amount` και `soft_stake` άμα χρειάζεται, ενώ το έγκυρο transaction προστίθεται στο `transaction_inbox` που αναφέρθηκε παραπάνω. Στο τέλος επίσης, ελέγχεται αν ο τρέχον κόμβος δεν αναμένει κάποιο block. Ο έλεγχος αυτός πραγματοποιείται για να μην ενεργοποιηθεί η διαδικασία δημιουργίας ενός νέου block χωρίς να έχει παραληφθεί το block που αναμένεται.
- `block_val_process(transaction)`: Αν το μήκος του inbox ξεπεράσει την χωρητικότητα των block, τότε ο τρέχον κόμβος υπολογίζει το index του νέου μπλοκ και ξεκινάει τη διαδικασία validation εκτελώντας το πρωτόκολλο Proof of stake, καλώντας τη συνάρτηση `proof_of_stake(stakes)`. Αν ο τρέχον κόμβος είναι validator τότε δημιουργεί το νέο block καλώντας τη `mint_block()` και το προσθέτει στο blockchain του. Έπειτα, το κάνει broadcast σε όλους τους κόμβους και ενημερώνει το state του καλώντας τη συνάρτηση `update_state()` που θα περιγραφεί παρακάτω. Αν δεν είναι validator, τότε θέτει το flag `waiting_for_block` να είναι ίσο με το index που υπολόγισε, ώστε να δηλωθεί ότι περιμένει το συγκεκριμένο block.
- `validate_block(block)` : Ελέγχεται η εγκυρότητα του block εκτελώντας ξανά το Proof of stake με ίδιο seed και ελέγχοντας ότι το hash του προηγούμενου block είναι ίδιο με το previous_hash του νέου block. Ωστόσο, προτού γίνουν αυτοί οι έλεγχοι είναι σημαντικό να ελεγχθεί αν το index του block που περιμένει ο τρέχον κόμβος είναι ίσο με το index του block που έλαβε. Αν όχι, τότε το block δεν προστίθεται ακόμα στο blockchain διότι η σειρά δεν θα είναι σωστή, αλλά εισέρχεται στο `block_waiting_room`. Αν ωστόσο τα δύο indices είναι ίδια και το block αποδειχθεί έγκυρο τότε προστίθεται στο blockchain και ο τρέχον κόμβος ενημερώνει το state του καλώντας την `update_state()`.
- `update_state()` : Μόλις ληφθεί και επαληθευτεί ένα block πρέπει να ενημερωθούν τα `hard_stake` και `hard_amount` του τρέχοντος κόμβου, καθώς και να επικαιροποιηθούν τα πεδία `blockchain_transactions` και `transaction_inbox`. Αυτό εκτελεί η συγκεκριμένη μέθοδος, διατρέχοντας όλα τα transactions του ληφθέντος block. Όσα transactions βρίσκονται στο block και στο `transaction_inbox` διαγράφονται από το τελευταίο. Επίσης, όσα transactions έχει λάβει αυτός ο κόμβος και δεν ανήκουν στο νέο block, γίνονται ξανά validate με τα νέα `soft_amount` τα οποία είναι πλέον έγκυρα καθώς έχουν λάβει την τιμή των ενημερωμένων `hard_amount` . Τέλος, όσα transactions είναι στο νέο block και δεν ανήκουν στο `transaction_inbox` τοποθετούνται στο `blockchain_transactions` για να μη ληφθούν υπόψιν όταν μελλοντικά τα λάβει ο κόμβος.

Περιγραφή Endpoints

Όπως αναφέρθηκε προηγουμένως υπάρχουν δύο κατηγορίες endpoints.

Internal endpoints

- `/balance`

Επιστρέφει το περιεχόμενο όλων των wallets.

- `/conversations`

Επιστρέφει σε μια λίστα όλα τα μηνύματα που έχει ανταλλάξει ο τρέχων κόμβος με τους υπόλοιπους κόμβους.

- `/exp_signal`

Ξεκινάει η διαδικασία εκτέλεσης των πειραμάτων με εντολή του χρήστη.

- `/home`

Επιστρέφει ότι ο server του τρέχοντος κόμβος λειτουργεί.

- `/send_transaction`

Δημιουργεί ένα transaction και το στέλνει σε όλους τους κόμβους.

- `/view`

Επιστρέφει το τελευταίο block του blockchain που έχει ο τρέχων κόμβος.

External endpoints

- `/endExp`

Όταν γίνει κλήση σε αυτό το endpoint και ο μετρητής `node_count` ισούται με τον αριθμό των κόμβων, δηλαδή όταν όλοι οι κόμβοι καλέσουν αυτό το endpoint, σημαίνει ότι όλοι εκτέλεσαν τα πειράματα, συνεπώς επιστρέφονται τα αποτελέσματα των πειραμάτων. Σημειώνεται ότι όλοι καλούν το `/endExp` του bootstrap κόμβου.

- `/receiveInitFromBootstrap`

Ο bootstrap καλεί αυτό το endpoint σε κάθε κόμβο μόλις συνδεθούν όλοι οι κόμβοι, στέλνοντάς του το τρέχον blockchain και τα wallets όλων των συνδεδεμένων κόμβων.

- `/talkToBootstrap`

Όταν ένας νέος κόμβος εισέρχεται στο σύστημα, καλεί αυτό το endpoint του bootstrap λαμβάνοντας από αυτόν το μοναδικό id του. Επίσης δημιουργείται ένα transaction στο οποίο μεταφέρει στον κόμβο 1000 BCC.

- `/runExp`

Ο bootstrap καλεί αυτό το endpoint σε κάθε κόμβο, σηματοδοτώντας την έναρξη των πειραμάτων από κάθε κόμβο.

- `/validateBlock`

Ο validator κάνει broadcast το block σε όλους τους κόμβους καλώντας αυτό το endpoint που επιστρέφει “success” ή “failed” ανάλογα με το αν επέτυχε το validation.

- `/validateTransaction`

Ο δημιουργός ενός transaction το κάνει broadcast σε όλους τους κόμβους καλώντας αυτό το endpoint.

Λειτουργικότητες

Αρχικοποίηση εφαρμογής

Αρχικά δημιουργείται ο κόμβος bootstrap και το genesis block που περιέχει ένα transaction μεταφοράς $1000 \cdot n$ BCC στον bootstrap όπου n, ο αριθμός των κόμβων. Όταν ένας νέος κόμβος εισέρχεται στο σύστημα, πρώτα καλεί το endpoint `/talkToBootstrap` λαμβάνοντας από τον bootstrap το μοναδικό id του. Επίσης δημιουργεί ένα transaction στο οποίο μεταφέρει στον κόμβο1000 BCC. Όταν εισαχθούν όλοι οι κόμβοι, ο bootstrap καλεί το endpoint `/receiveInitFromBootstrap` κάθε κόμβου στέλνοντας το τρέχον blockchain και όλα τα wallets όλων των κόμβων. Έτσι, κάθε κόμβος λαμβάνει το blockchain και τα wallets, δημιουργεί το δικό του state και η αρχικοποίηση ολοκληρώνεται.

Αποστολή,παραλαβή transaction

Τα transactions στέλνονται απο το CLI κάθε κόμβου με τις εντολές:

- `t <node_id> <amount>`: για “coin” transactions
- `m <node_id> <message>`: για “message” transactions
- `stake <amount>`: για transaction που αλλάζει το stake του κόμβου

Ο αποστολέας ελέγχει αν είναι έγκυρο το transaction και αν έχει αρκετα BCC για να το πραγματοποιήσει και στην συνέχεια προσθέτει το transaction στο `transaction_inbox` του πριν το κάνει broadcast στους υπόλοιπους κόμβους.

Όταν ένας κόμβος δέχεται ένα transaction αρχικά ελέγχει αν το συγκεκριμένο transaction βρίσκεται ήδη μέσα στο blockchain του τρέχοντος κόμβου ώστε να μην ληφθεί υπόψη. Ο έλεγχος αυτός πραγματοποιείται γιατί οι κόμβοι υπάρχει περίπτωση να επεξεργάζονται τα transactions με διαφορετική σειρά και έτσι είναι πιθανό ένας κόμβος να λάβει ένα block που περιέχει κάποια transactions που δεν έχει ακόμα λάβει. Στην συνέχεια ο κόμβος αυτός ελέγχει την εγκυρότητα του transaction αυτού και το προσθέτει στο `transaction_inbox`.

Κατά την διαδικασία αυτή οι κόμβοι ελέγχουν την εγκυρότητα των transactions με βάση τα soft stakes και soft amounts και ανανεώνουν τα ποσά αυτά αναλόγως.

Κλείσιμο Block/Proof of stake

Κάθε φορά που κάποιος κόμβος κάνει validate ένα transaction και το εισάγει στο `transaction_inbox` πρέπει να ελεγχθεί αν υπάρχουν αρκετά transactions στο inbox για την δημιουργία ενός νέου block. Όπως αναφέρθηκε και παραπάνω, αυτός ο έλεγχος πραγματοποιείται μόνο αν όλα τα block για τα οποία έχει γίνει proof of stake έχουν ληφθεί από τον εκάστοτε validator και έχουν τοποθετηθεί στην σωστή θέση τους στο blockchain. Για την παραπάνω λειτουργεία προγραμματίστηκε ένας μηχανισμός που κάνει χρήση ενός flag

`waiting_for_block` που περιέχει την τιμή του επόμενου block που αναμένει ο κάθε κόμβος (στην περίπτωση που δεν είναι αυτός ο validator).

Αν το `wating_for_block` είναι απενεργοποιημένο τότε ο κάθε κόμβος τρέχει τον αλγόριθμο επιλογής του validator που βρίσκεται στην συνάρτηση `proof_of_stake(stakes)`. Η συνάρτηση αυτή παίρνει ως είσοδο τα hard_stakes όλων των κόμβων, δηλαδή τα stakes των οποίων τα transaction βρίσκονται ήδη στο blockchain. Αν η λοταρία βγάλει ως validator τον ίδιο τον κόμβο τότε ο κόμβος αυτός δημιουργεί το block και το κάνει broadcast στους υπόλοιπους. Στην αντίθετη περίπτωση,

Όταν ένας κόμβος λάβει ένα block μέσω του `/validateBlock` ελέγχει αν το block αυτό είναι το επόμενο block στο blockcahain. Αν είναι, το κάνει validate,το τοποθετεί στο τέλος της αλυσίδας και «απενεργοποιεί» το `waiting_for_block`. Στην συνέχεια ελέγχει αν υπάρχουν κάποια άλλα block στο `block_waiting_room` που τώρα μπορούν να τοποθετηθούν στο blockchain.

Εδώ είναι σημαντικό να σημειωθεί, λόγω του ότι το Flask είναι multithreaded έπρεπε να εισαχθούν κλειδώματα (locks) σε κάποια σημεία του κώδικα γιατί αλλιώς δύο εισερχόμενα transactions όταν μπορούσαν να δημιουργήσουν και να κάνουν broadcast δυο blocks με το ίδιο index.

Λοιπά Αρχεία/Εργαλεία

- Οι βοηθητικές συναρτήσεις `broadcast()`, `send_http_request()` που υλοποιούνται στα αντίστοιχα αρχεία, εκτελούν απαραίτητες λειτουργίες όπως η αποστολή δεδομένων σε όλους τους κόμβους και η αποστολή HTTP requests αντίστοιχα.
- Οι συναρτήσεις `init_bootstrap()` και `init_node()` που βρίσκονται στο init_utils.py ξεκινούν τη διαδικασία αρχικοποίησης του bootstrap και ενός κόμβου αντίστοιχα.
- Η συνάρτηση `proof_of_stake(stakes, seed)` που βρίσκεται στο αντίστοιχο αρχείο υλοποιεί την λοταρία και επιστρέφει το id του κόμβου που είναι validator.
- Το αρχείο crypto.py λειτουργεί ως διεπαφή που προσφέρει απαραίτητες κρυπτογραφικές λειτουργίες για την εφαρμογή, συγκεκριμένα τη δημιουργία ζεύγους RSA κλειδιών, την υπογραφή μηνύματος και την επαλήθευση μιας υπογραφής. Σημειώνεται εδώ ότι χρησιμοποιήθηκε το σχήμα υπογραφών RSA – Full Domain Hash, και ότι τα public, private keys είναι tuples της μορφής (n, e), (d, e) όπου n, d, e όπως περιγράφονται στο RSA πρωτόκολλο.

Βιβλιοθήκες

Η εργασία πραγματοποιήθηκε σε Python και η διαχείρηση των βιβλιοθηκών έγινε με χρήση του miniconda. Χρησιμοποιήθηκαν οι εξής βιβλιοθήκες:

- *Flask*: REST API
- *pycryptodome*: δημιουργία των RSA κλειδιών
- *cmd2:* CLI

Εκτέλεση Πειραμάτων

Για την εκτέλεση των πειραμάτων, χρησιμοποιήθηκαν 5 virtual machines από την υπηρεσία *okeanos*. Στα πειράματα που απαιτούν 10 clients, κάθε vm προσομοιώνει 2 nodes τρέχοντας δύο instances της εφαρμογής, σε διαφορετικά ports. Τα αποτελέσματα παρουσιάζονται στον παρακάτω πίνακα:

Clients	**Capacity**	**Staking**	**Throughput ↑** *(transactions/second)*	**Block Time ↓** *(seconds/block)*
5	5	Uniform	38.146	0.126
5	10	Uniform	41.118	0.243
5	20	Uniform	43.917	0.494

5	5	Stake Disparity	40.853	0.151
10	5	Uniform	41.255	**0.119**
10	10	Uniform	46.155	0.210
10	20	Uniform	**50.251**	0.414

Παρατηρούμε τα εξής:

- Στο πειράματα 1 και 2 η αύξηση του capacity αύξησε το throughput αλλά και το block time. Έτσι, η αύξηση του capacity φαίνεται να έχει θετικό αποτέλεσμα στην απόδοση του συστήματος.
- Ο διπλασιασμός των κόμβων αύξησε το throughput και μείωσε το block time. Αυτό είναι μια θετική ένδειξη για την κλιμακωσιμότητα του συστήματος.
- Στο πείραμα 3 ο κόμβος 0 ο οποίος είχε staking 100 BCC, έγινε validator σε 52 blocks, την ώρα που οι υπόλοιποι κόμβοι σε 4, 8, 7 και 9 blocks. Εξαιτίας αυτού, ο κόμβος 0 έχει πολύ μεγαλύτερο ποσό στο wallet του μετά την εκτέλεση των πειραμάτων από τους υπόλοιπους κόμβους, καθώς πιστώθηκε τα περισσότερα fees μια και έγινε πολύ περισσότερες φορές validator. Άρα λοιπόν, το proof of stake λειτουργεί σωστά και το σύστημά μας είναι δίκαιο, δίνοντας μεγαλύτερη ανταμοιβή στον κόμβο που έδωσε το μεγαλύτερο stake.

Παρατίθενται και σχετικά διαγράμματα για οπτικοποίηση των αποτελεσμάτων:

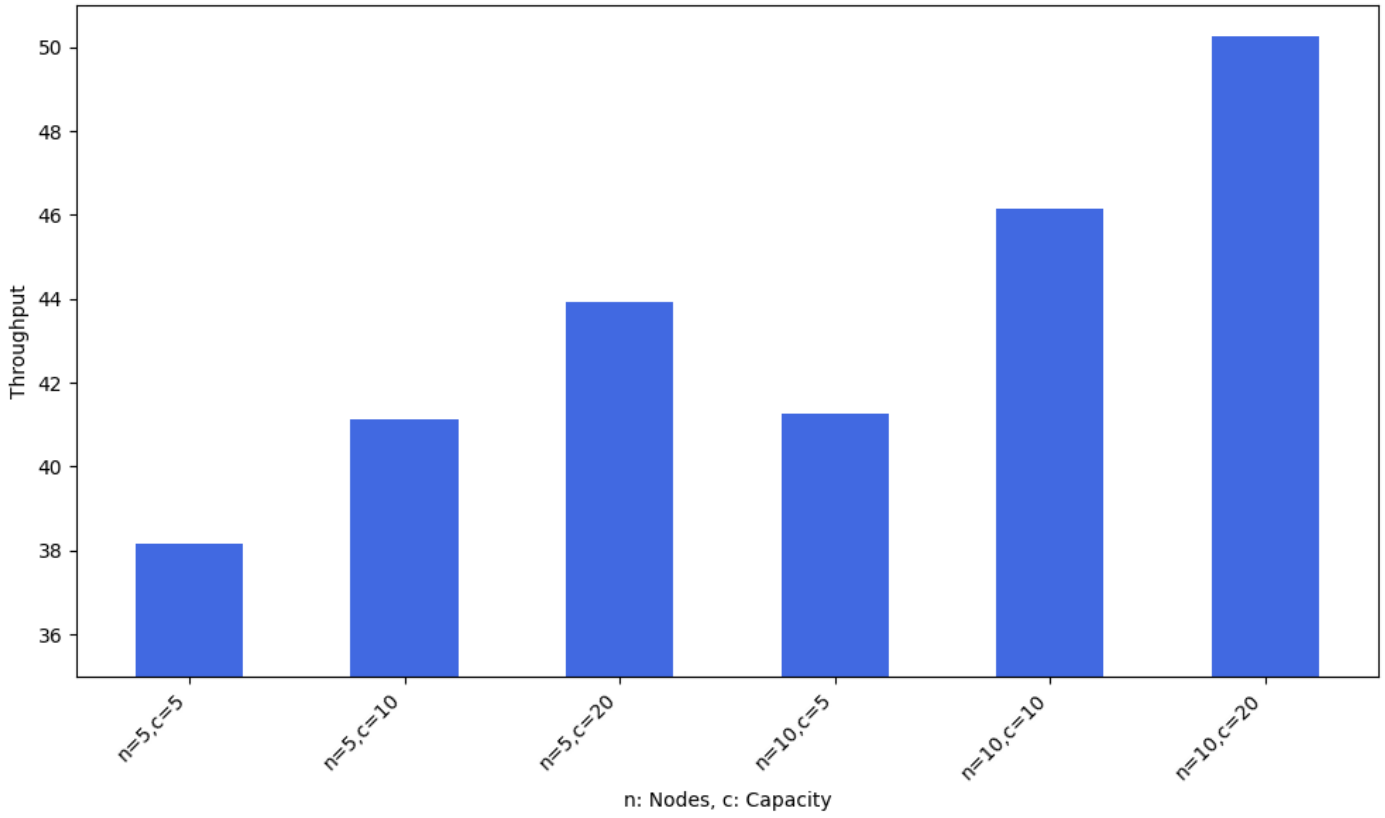

Figure 1Througput για τα Πειράματα 1 & 2

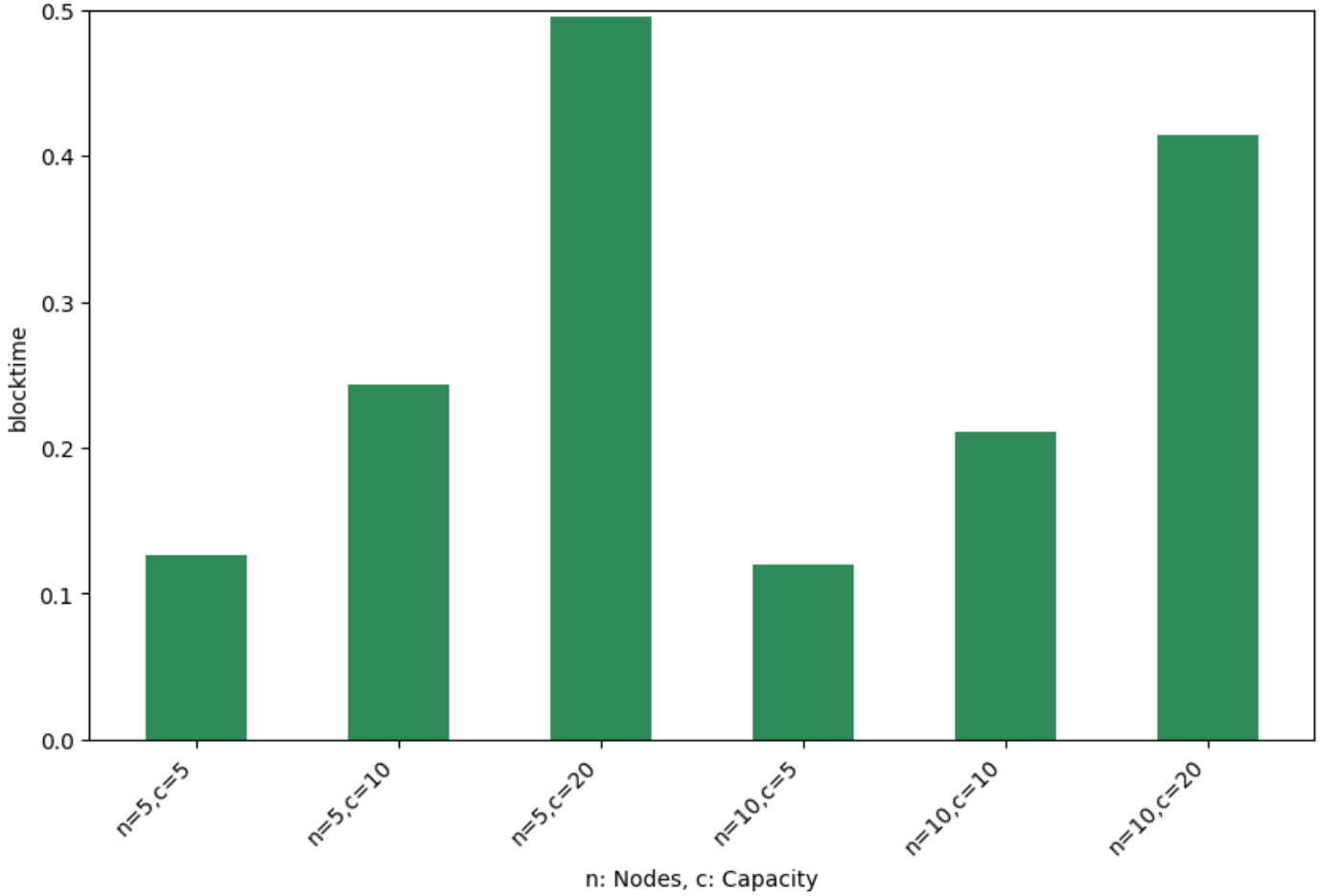

Figure 2 Block Time για τα Πειράματα 1 & 2

Github Repo

Στο παρακάτω link βρίσκεται ο κώδικας της εργασίας στο github:

https://github.com/urcodeboijorto574/distributed-systems-ntua