Κατανεμημένα Συστήματα, NTUA 2019 - 2020
Αναφορά εξαμηνιαίας εργασίας

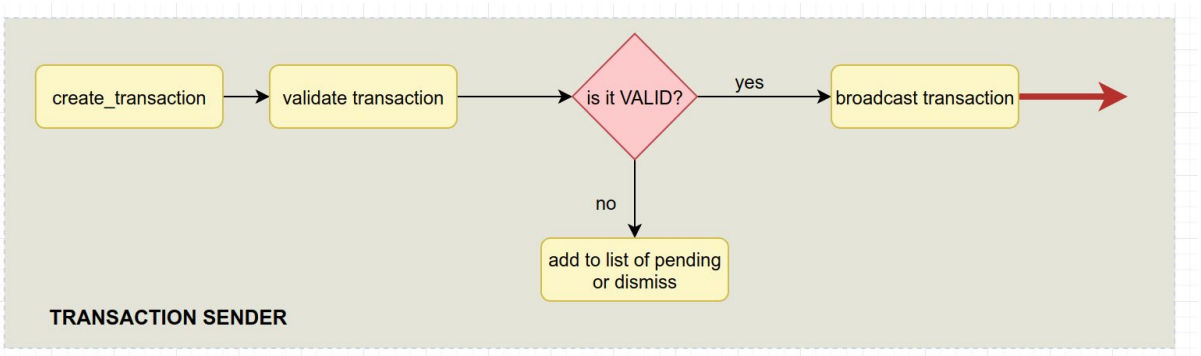
Φωτεινή Δεληγιαννάκη 03115099
Ίλια Διολέτη 03115055
Αθηνά Κυριάκου 03117405

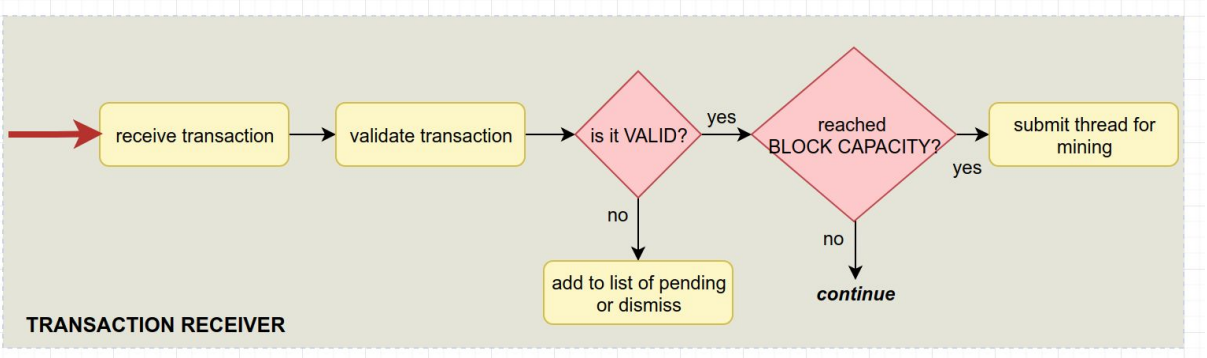
Περιεχόμενα

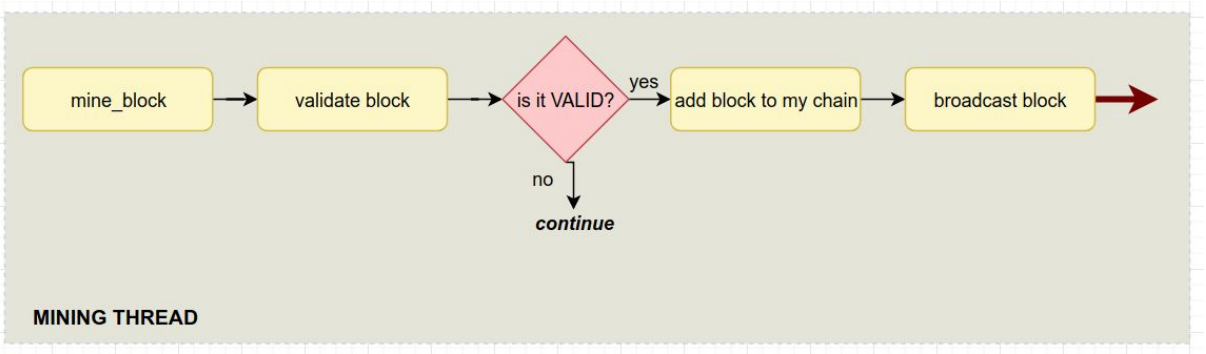
Περιγραφή εφαρμογής:

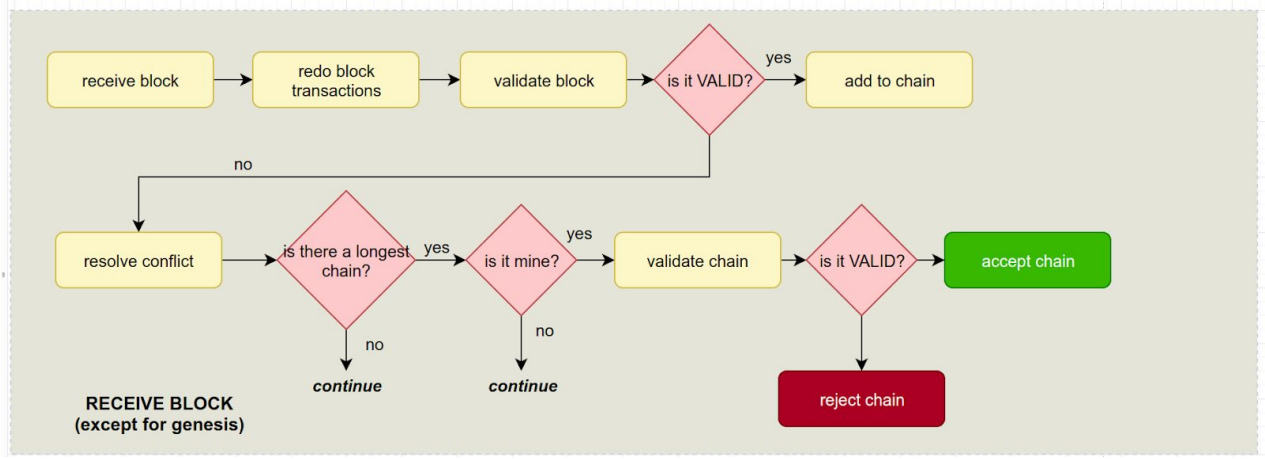
Η εφαρμογή υλοποιεί ένα απλό σύστημα blockchain, όπου καταγράφονται οι δοσοληψίες μεταξύ των συμμετεχόντων και εξασφαλίζεται το consensus με proof-of-work.
Η ανάπτυξη έγινε σε Python3 με χρήση του Flask framework. Το σύστημα αποτελείται από 5 virtual machines (VM) στον okeanos με 2 cores και 2 GB μνήμης το καθένα. Τα VMs είναι συνδεδεμένα σε private δίκτυο και μόνο ένα από αυτά, ο master, είναι συνδεδεμένο στο εξωτερικό public δίκτυο.
Στα πειράματα που διεξήχθησαν, κάθε VM έχει 1 ή 2 κόμβους και η ανταλλαγή μηνυμάτων μεταξύ όλων των κόμβων του cluster έγινε με REST API, χρησιμοποιώντας τη βιβλιοθήκη requests της Python.

Αλληλουχία λειτουργιών συστήματος:









Περιγραφή κλάσεων και λειτουργιών:

Κλάσεις

Node: ο κόμβος του cluster

Αποτελείται από το ID του κόμβου, το wallet και την blockchain του καθώς το ring με τις πληροφορίες για κάθε κόμβο του cluster (ip, port, public key) και τις λίστες:

- valid_trans: τα validated transactions του node
- pending_trans: τα transactions που δεν έγιναν validated επειδή λείπουν τα απαιτούμενα inputs για το validation
- unreceived_trans: transactions που έχει λάβει ο node από block που έγινε broadcast και όχι μεμονωμένα

Wallet: το wallet κάθε κόμβου του cluster

Περιλαμβάνει το private και public key, τα τρέχοντα utxos του κόμβου όπως προκύπτουν από τις κλήσεις της validate transaction, τα utxos του όπως διαμορφώθηκαν μετά την

προσθήκη του τελευταίου block στο blockchain του και τη συνάρτηση balance για εύρεση των NBC του κόμβου.

Block: το block του blockchain

Αποτελείται από το index με τη θέση που θα έχει στο chain του κόμβου και το hash του προηγούμενου block, το hash του, το timestamp δημιουργίας του, τη λίστα των transactions του και τη μεταβλητή nonce με τη λύση του proof-of-work. Η συνάρτηση myHash() υπολογίζει το hash του block συναρτήσει όλων των πεδίων του block χρησιμοποιώντας τη συνάρτηση SHA256() της βιβλιοθήκης Crypto.Hash της Python.

Blockchain:

Περιέχει τη λίστα των blocks του chain και τη συνάρτηση create_blockchain() που καλείται από τον bootstrap κόμβο για τη δημιουργία του genesis block με index 0 και previous hash 1 και την προσθήκη στην αλυσίδα του.

Transaction:

Περιλαμβάνει τα public keys του sender και του receiver κόμβου, καθώς και τα IDs τους στο ring του cluster, το ποσό που μεταφέρεται, το hash του transaction και την υπογραφή του sender. Έχει επίσης λίστα με τα inputs, δηλαδή τα id προηγούμενων συναλλαγών από τα οποία αθροιστικά χρησιμοποιεί funds ο sender ώστε να εκτελέσει την παρούσα συναλλαγή, αλλά και τα outputs του transaction, τα οποία προστίθενται στα utxos του κατάλληλου παραλήπτη (“to_who”), και σίγουρα περιλαμβάνουν τα NBCs προς τον receiver κομβο και στην περίπτωση που υπάρχει υπόλοιπο στη συναλλαγή (inputs-transaction amount), περιλαμβάνουν και τα ρέστα που πάνε προς τον sender . Το hash του transaction προκύπτει από την SHA256() συναρτήσει των public keys αποστολέα και παραλήπτη, του amount και των transaction inputs με τη συνάρτηση hash().

Σημαντικές μέθοδοι της κλάσης είναι το “ζεύγος” sign και verify transaction που χρησιμοποιούν την βιβλιοθήκη PKCS1_v1_5 για κρυπτογραφημένες υπογραφές. Η πρώτη επιτρέπει στον sender να προσθέσει μια μοναδική κρυπτογραφημένη υπογραφή, με βάση το private key του στο οποίο έχει πρόσβαση μόνο ο ίδιος, σε ένα μήνυμα το οποίο στην προκειμένη είναι το hash του συγκεκριμένου transaction. Η verify signature, με χρήση της ίδιας βιβλιοθήκης και μέσω του public key του sender επιβεβαιώνει ότι η υπογραφή της συγκεκριμένης υπογραφής παράχθηκε με χρήση του σωστού private key.

Threadpool:

Με τη χρήση του ThreadPoolExecutor της βιβλιοθήκης concurrent.futures, δημιουργείται ένα poll με threads για το mining. Το πλήθος των διαθέσιμων threads είναι ένα, δεδομένου ότι κάθε VM έχει 2 cores και έτσι επιπλέον παραλληλισμός δε θα οδηγούσε σε βελτιωμένα αποτελέσματα. Υπάρχει μόνο 1 επιπλέον διαθέσιμο thread, καθώς κάθε vm έχει 2 cores και επομένως δεν μπορεί να υποστηρίξει επιπλέον επίπεδο παραλληλισμού με βελτιωμένη απόδοση.

Rest: αποτελείται από τις συναρτήσεις για την επικοινωνία μεταξύ των κόμβων

Λειτουργίες

Δημιουργία cluster:

Για την αρχικοποίηση του cluster, ο bootstrap καλεί τη συνάρτηση init_connection() της κλάσης rest με το πλήθος των αναμενόμενων nodes. Δημιουργείται το blockchain με το αρχικό transaction στο genesis block, όπου ο bootstrap λαμβάνει 100NBC επί το πλήθος των nodes που θα συνδεθούν. Επίσης τίθεται το id του bootstrap node σε 0 και εισάγεται στο ring του cluster. Σημειώνεται ότι η transaction του genesis block δε γίνεται validated.

Το αίτημα σύνδεσης ενός node στο cluster γίνεται με τη συνάρτηση connect_node_request() της rest αποστέλλοντας τα IP, port και public key. Για την αποδοχή στο cluster, ο bootstrap καλεί τη συνάρτηση receive_node_request() του rest όταν λάβει το αίτημα σύνδεσης και στην περίπτωση που δεν έχει ξεπεραστεί ο αναμενόμενος αριθμός nodes στο cluster, στέλνει μήνυμα αποδοχής στον προς σύνδεση κόμβο. Στο μήνυμα περιλαμβάνεται το ID του νεοεισαχθέντα κόμβου στο ring και τα utxos και το blockchain του bootstrap εκείνη τη στιγμή. Τότε δημιουργείται το αρχικό transaction και ο bootstrap στέλνει 100NBCs στον κόμβο.

Μόλις εισαχθεί και ο τελευταίος κόμβος στο cluster, ο bootstrap κάνει broadcast το τελικό ring του cluster με τις απαραίτητες πληροφορίες επικοινωνίας σε όλους τους κόμβους.

Δημιουργία συναλλαγής:

Για τη δημιουργία μιας συναλλαγής, κάθε κόμβος καλεί τη συνάρτηση create_transaction() της κλάσης node μέσω της transaction_new() του rest. Σε αυτή τη συνάρτηση καθορίζονται τα inputs και outputs της συναλλαγής. Για τα inputs προσθέτουμε στη λίστα τα id των utxos (με παραλήπτη φυσικά τον sender) τα οποία αθροιστικά δίνουν (ακόμα και ξεπερνώντας το) το απαραίτητο ποσό για τη συναλλαγή. Για τα outputs, θα έχουμε μια λίστα μήκους 2 που περιλαμβάνει 1 output προς τον receiver με ποσό αυτό της συναλλαγής, και 1 προς τον sender με τα “ρέστα” του, δηλαδή τη διαφορά του αθροίσματος των inputs που χρησιμοποίησε με το ποσό που έστειλε.

Μόλις η συναλλαγή αρχικοποιηθεί, καλείται η validate_transaction() στην ίδια κλάση η οποία ελέγχει αν το transaction είναι έγκυρο. Στην περίπτωση που η υπογραφή του transaction και το μεταφερόμενο ποσό, ο αποστολέας και ο παραλήπτης είναι έγκυροι, το ποσό της συναλλαγής δεν είναι αρνητικό, προχωράμε στον έλεγχο των inputs. Γίνεται δηλαδή έλεγχος αν όλα τα utxo ids που συμπεριλαμβάνονται στα inputs είναι πραγματικά και υπάρχουν με παραλήπτη τον sender στο dictionary με τα utxos του κόμβου και επιπλέον αν επαρκούν τα χρήματά τους για τη συναλλαγή. Αν αυτός ο έλεγχος επιτύχει τα input utxos αφαιρούνται από το dict ώστε να μην γίνει double spending αν όχι, η συνάρτηση επιστρέφει “pending”. Επιπλέον γίνεται έλεγχος αν τα outputs που υπολογίζονται από τα validated πλέον inputs

είναι ίδια με αυτά που έχουν συμπεριληφθεί στην συναλλαγή και αν αυτό ισχύει είμαστε τελικά έτοιμοι να δεχτούμε τη συναλλαγή, να προσθέσουμε τα outputs στα wallet utxos ωστε να είναι διαθέσιμα για επόμενες συναλλαγές και επιστρέφουμε “validated”. Αν υπάρχει κάποιο σφάλμα από αυτά που περιγράφηκαν παραπάνω επιστρέφει “error”.

Αν η συναλλαγή είναι “validated”, θα προστεθεί στη λίστα των validated transactions με την add_transaction_to_validated() και θα γίνει broadcast. Αν είναι “pending” θα προστεθεί στη λίστα των pending transactions.

Λήψη συναλλαγής από broadcast:

Ένας κόμβος στο cluster λαμβάνει μία συναλλαγή που έχει γίνει broadcast με τη συνάρτηση receive_trans() του rest. Αρχικά, ελέγχεται αν η συναλλαγή έχει ήδη ληφθεί από κάποιο block που έλαβε ο κόμβος και έχει προσθέσει στο blockchain του ώστε να μην ελεγχθεί ξανά. Διαφορετικά καλείται η validate_transaction() και ανάλογα με την επιστρεφόμενη τιμή, η συναλλαγή προστίθεται στη λίστα των validated ή των pending. Σημειώνεται πως αν η συναλλαγή είναι valid, μετά την προσθήκη της στη λίστα των validated ελέγχεται αν γίνεται έγκυρη κάποια από τις pending συναλλαγές, δεδομένου ότι προστέθηκε νέο input.

Mining:

Όταν μία συναλλαγή προστίθεται στη λίστα των validated στη συνάρτηση add_transaction_to_validated() του node, ελέγχεται αν έχει γεμίσει η λίστα των validated, όπως ορίζεται από το capacity του block. Αν έχει γεμίσει, το mining thread του thread pool ξεκινάει τη διαδικασία του mining. Επιλέχθηκε η χρήση thread για το mining ώστε η κύρια διεργασία να συνεχίσει να λαμβάνει transactions και blocks που γίνονται broadcast.

Η init_mining() του node δημιουργεί ένα νέο block από το πακέτο των έγκυρων συναλλαγών που έχει λάβει, θέτοντας κατάλληλα το index του στο chain αν τελικά προστεθεί και το previous hash. Στη συνέχεια, κάνει ξανά validate όλα τα transactions του block βάσει των utxos όπως είχαν διαμορφωθεί μετά την προσθήκη του τελευταίου block στον chain. Το βήμα αυτό γίνεται για να ελέγξουμε ότι τα transactions αυτά δεν έχουν μπει ήδη στο blockchain από κάποιο block που παρέλαβε ο κόμβος πριν ολοκληρώσει τη διαδικασία του mining. Το block γίνεται κατόπιν mined από το thread όπου με PoW υπολογίζεται το hash block στη συνάρτηση mine_block() του node.

Ο έλεγχος της εγκυρότητας του block και η προσθήκη του στο chain αν τελικά είναι έγκυρο γίνεται από το thread μέσα σε lock. Ένα block είναι έγκυρο αν το hash του έχει υπολογιστεί σωστά βάσει των δεδομένων του και το previous hash του ισούται με το hash του τελευταίου block στο chain του κόμβου. Η διαδικασία πραγματοποιείται σε lock προκειμένου να αποφευχθεί race condition μεταξύ του process και του mining thread και έτσι μόνο ένας κάθε φορά να μπορεί να διαβάζει το hash του τελευταίου block του chain, να καθορίζει βάσει αυτού αν το block είναι έγκυρο και αν ναι, να το προσθέτει.

Αν τελικά το block είναι έγκυρο και προστεθεί στο chain, ο κόμβος το κάνει broadcast, εκτός του lock, στους υπόλοιπους κόμβους του cluster.

Λήψη block από broadcast:

Ένας κόμβος λαμβάνει block που έγινε broadcast με τη συνάρτηση receive_block() του rest και του node. Με την συνάρτηση block_REDO(), εκτελεί πάλι τις συναλλαγές του block για να βεβαιωθεί ότι είναι έγκυρες, ελέγχει αν τα transactions έχουν προστεθεί ήδη στο chain και στη συνέχεια μέσα σε lock κάνει validate το block και το προσθέτει αν είναι έγκυρο. Ελέγχει τέλος εκτός του lock αν μπορεί να κάνει validate συναλλαγές των pending, χρησιμοποιώντας τα inputs του νέου block. Σημειώνουμε ότι η συνάρτηση block_REDO χρησιμοποιείται και σε άλλες περιπτώσεις όπου χρειάζεται έλεγχος των transactions ενός block.

Αν συναλλαγές του block έχουν προστεθεί νωρίτερα στο chain ή το block δεν είναι έγκυρο, έχει γίνει διακλάδωση στο blockchain και καλείται η resolve conflict().

Resolve conflict:

Η resolve_conflict() της κλάσης node καλείται όταν έχει παρατηρηθεί διακλάδωση στο chain. Ο κόμβος που την καλεί ζητάει από όλους τους άλλους στο cluster να του στείλουν το μήκος του blockchain τους. Αν όλα τα blockchains είναι ισομήκη ή ο αιτούντας κόμβος έχει το μακρύτερο, δεν πραγματοποιείται κάποια αλλαγή. Αν εντοπιστεί κάποιο μακρύτερο blockchain άλλου κόμβου, τότε καλείται η συνάρτηση validate_chain() για να ελεγχθεί.

Μία αλυσίδα είναι έγκυρη αν τα previous hash όλων των blocks στην αλυσίδα συμφωνούν με το hash του προηγούμενου block. Επιπλέον οι συναλλαγές κάθε block επαναλαμβάνονται ώστε να ελεγχθεί η εγκυρότητα τους, και να εξαλειφθεί ο κίνδυνος κάποιος κόμβος να δρα κακόβουλα ή απλώς λάθος και να έχει συμπεριλάβει λανθασμένες συναλλαγές.

Στην περίπτωση που η μακρύτερη αλυσίδα που λήφθηκε είναι έγκυρη, τότε ο κόμβος που κάλεσε τη resolve_conflict() την υιοθετεί και θέτει κατάλληλα τα utxos του βάσει της αλυσίδας που υιοθετήθηκε. Το conflict σε αυτή την περίπτωση επιλύεται!

Χρήση εφαρμογής:

Εντολές χρήσης εφαρμογής:

Εκκίνηση εφαρμογής σε κάθε κόμβο: **./setup.sh run 5000**

Δημιουργία cluster από bootstrap: **./setup.sh init 5000**

Σύνδεση κόμβου σε cluster: **./setup.sh connect <PORT> <IP>**
σύμφωνα με το PORT στο οποίο τρέχει η εφαρμογή για κάθε node.

Ανάγνωση αρχείου transactions σε κάθε κόμβο του cluster:
python testing.py <PORT> <ID> <#Nodes> <START> <END>

όπου διαβάζουμε από τη γραμμή START μέχρι και τη γραμμή END του αρχείου (για ολόκληρο το αρχείο θέτουμε 0 και 99 αντίστοιχα)

Client:

Για τις ανάγκες της εφαρμογής υλοποιήθηκε ένα απλό command line interface στο αρχείο cli.sh. Δίνει στο χρήστη τη δυνατότητα να δημιουργήσει κάποια συναλλαγή, να δει τις πιο πρόσφατες συναλλαγές του συστήματος, να δει το προσωπικό του υπόλοιπο NBCs και τέλος πληκτρολογώντας help μπορεί να δει λεπτομέρειες για τις παραπάνω λειτουργίες.

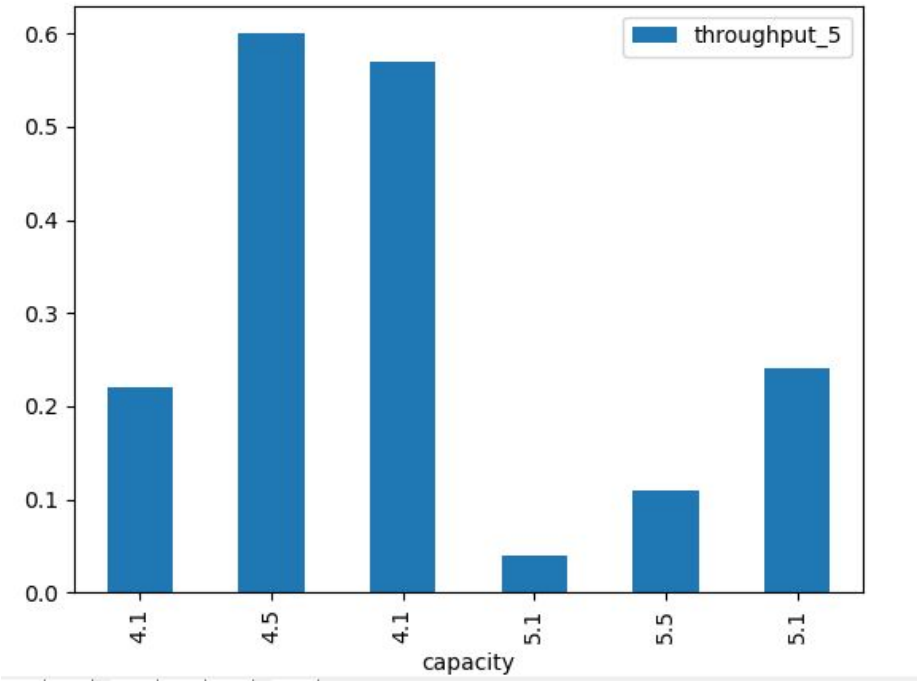
Πειράματα:

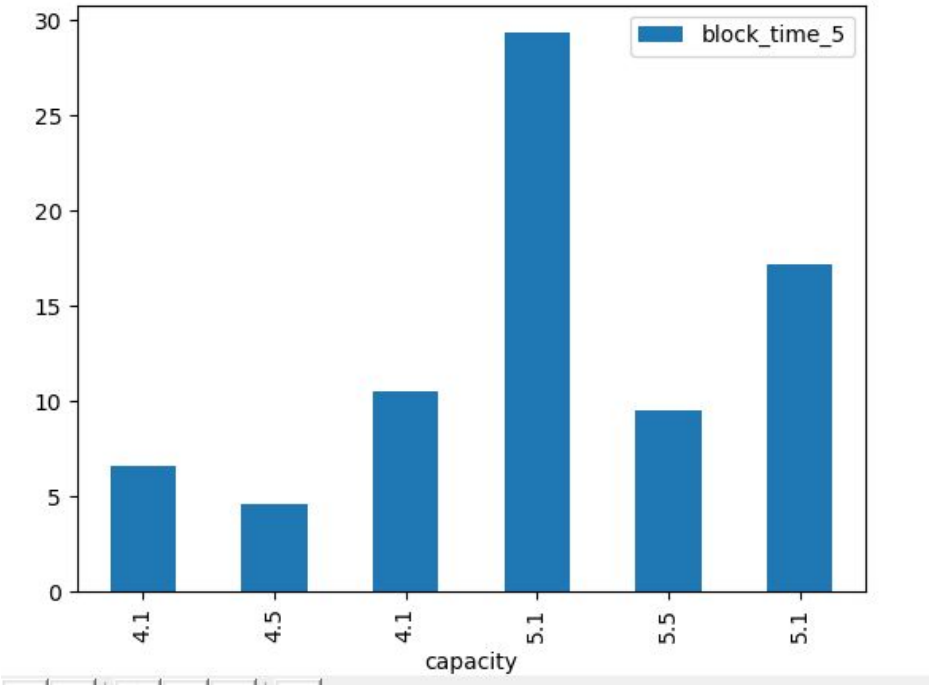
Στα παρακάτω πειράματα, αξιολογήθηκε η εφαρμογή για 5 και 10 συνδεδεμένους κόμβους στο cluster. Συγκεκριμένα, μετρήθηκαν το throughput, δηλαδή ο αριθμός των transactions που εξυπηρετούνται στη μονάδα του χρόνου, και το block time, δηλαδή ο μέσος χρόνος για να προστεθεί ένα block στο chain όπως καθορίζεται κυρίως από τη διαδικασία του mining συναρτήσει της χωρητικότητας σε transactions του block και του difficulty του PoW.

Πραγματοποιήθηκαν δοκιμές για τιμές χωρητικότητας 1, 5 και 10 του block και δυσκολίας 4 και 5 του PoW και κάθε κόμβος διαβάζει από ένα αρχείο και δημιουργεί συναλλαγές. Για τον υπολογισμό του χρόνου χρησιμοποιήθηκε η βιβλιοθήκη time. Πιο συγκεκριμένα, χρησιμοποιήθηκαν οι συναρτήσεις time() και thread_time() για τον υπολογισμό της διάρκειας διαδικασιών και η gmtime() για τη μετατροπή σε UTC ως κοινό σημείο αναφοράς μεταξύ των VMs.

Απόδοση:

Για cluster 5 κόμβων, τα αποτελέσματα συνοψίζονται στο διάγραμμα.





Κλιμακωσιμότητα:

Για cluster 8 κόμβων (επειδή πέθανε ένα machine τελευταία στιγμή), τα αποτελέσματα εμφανίζονται στο διάγραμμα.

Σχολιασμός αποτελεσμάτων:

Αναμένουμε όσο αυξάνει το block capacity, να αυξάνεται το throughput γιατί η διαδικασία του mining γίνεται πιο σπάνια. Επίσης, αύξηση του difficulty οδηγεί σε μικρότερο throughput γιατί θα καθυστερήσει περισσότερο η διαδικασία του mining. Τέλος θα υπάρχει μεγαλύτερη καθυστέρηση όταν αυξηθεί το πλήθος των node ανά machine.