Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών & Μηχανικών Υπολογιστών
Κατανεμημένα Συστήματα - 9ο Εξάμηνο

Αναφορά Εξαμηνιαίας Εργασίας

Ομάδα 57

Χρήστος Παπαδημητρίου, 03118017

Ζαχαρίας Αναστασιάδης, 03118161

Βαρθολομαίος Βαμβακάρης, 03118071

Σύντομη Περιγραφή του σχεδιασμού του συστήματος

Για την ανάπτυξη του συστήματος χρησιμοποιήσαμε την γλώσσα προγραμματισμού Python. Τα βασικά αρχεία του Project είναι:

- **rest.py**: περιέχει όλες τις συναρτήσεις που επιτρέπουν σε έναν κόμβο να επικοινωνεί με τους υπόλοιπους (με λογική Rest API). Επίσης είναι το entry point κάθε κόμβου (το αρχείο που τρέχουμε για να ξεκινήσει)
- **blockchain.py** : περιέχει την κλάση blockchain, το οποίο υλοποιείται σαν μια λίστα από αντικείμενα τύπου Block.
- **block.py**: περιέχει την κλάση Block, η οποία υλοποιείται σαν μια λίστα από αντικείμενα τύπου Transaction
- **transaction.py** : περιέχει την κλάση Transaction, η οποία μας παρέχει λειτουργικότητα για να φτιάχνουμε και να υπογράφουμε Transactions.
- **wallet.py** : Περιέχει την κλάση Wallet, την οποίας η βασική λειτουργικότητα είναι η παραγωγή του δημόσιου και του ιδιωτικού κλειδιού για έναν κόμβο.
- **client.py**: είναι το αρχείο που υλοποιεί τις ζητούμενες CLI λειτουργίες.
- **node.py:** περιλαμβάνει την κλάση node, η οποία περιέχει την πλειονότητα του κώδικα που υλοποιεί την λογική του backend για κάθε κόμβο. Κάνει import τις υπόλοιπες κλάσεις (Blockchain, Transaction, Wallet κλπ) και είναι υπεύθυνη να διατηρεί και να ανανεώνει την κατάσταση ενός κόμβου συνεχώς. Κάποιες βασικές μέθοδοι που παρέχει αυτή η κλάση αφορούν
  - Την αρχικοποίηση του συστήματος.
  - Δημιουργία και αποστολή (broadcast) Transactions
  - Επαλήθευση Ξένου Transaction
  - Δημιουργία νέου Block
  - Πραγματοποίηση Mining για ένα block αποστολή (broadcast) του Block
  - Επαλήθευση Ξένου Block
  - Πραγματοποίηση αλγορίθμου Consensus αν διαπιστωθεί ότι ένα αφηχθέν Block είναι invalid. (Αυτό γίνεται ζητώντας τα μήκη όλων των αλυσίδων των άλλων κόμβων και υιοθετώντας την μεγαλύτερη)

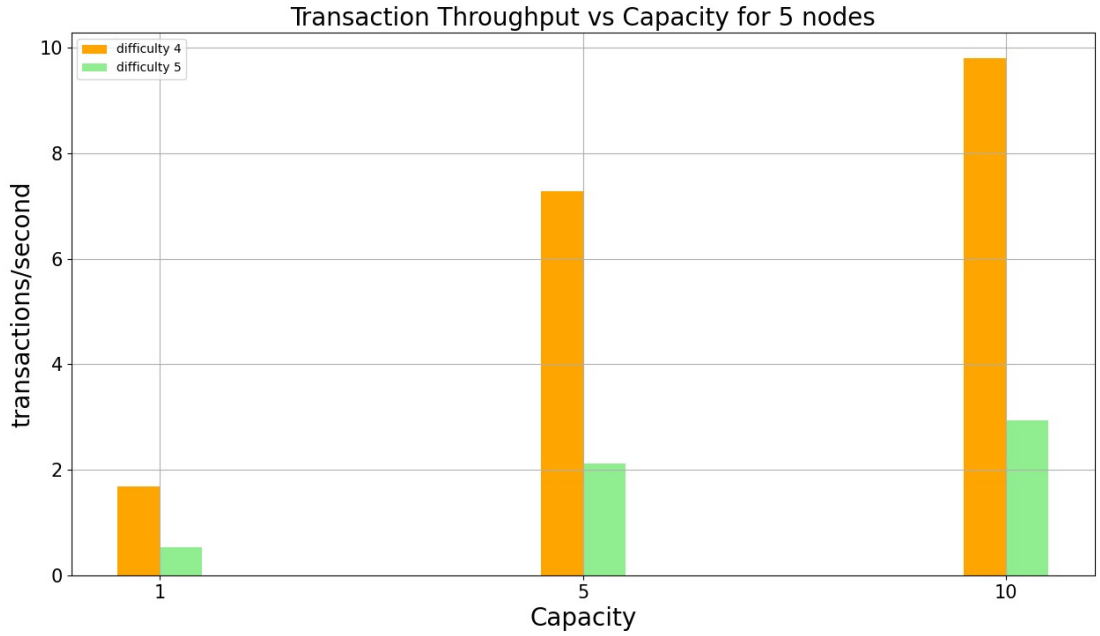
Όσον αφορά τις λεπτομέρειες τις υλοποίησης ξεχωρίζουμε τα εξής “ιδιαίτερα σημεία”:

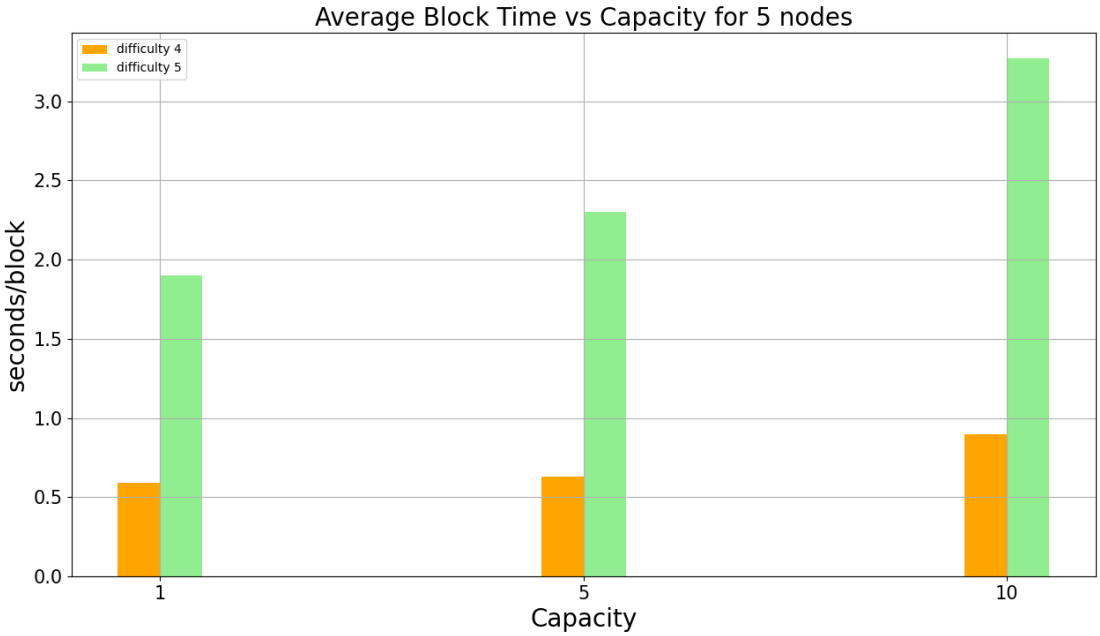
- Χρησιμοποιούμε διάφορα locks για να πετύχουμε συγχρονισμό. Αυτό είναι απαραίτητο, καθώς κάθε κόμβος δέχεται πολλαπλά αιτήματα που ζητούν να αλλάξουν διάφορες δομές (την λίστα με τα UTXOs, το Blockchain κλπ) και για να πετύχουμε συνεπή αποτελέσματα πρέπει το πολύ ένα Thread να έχει πρόσβαση σε αυτές τις δομές κάθε χρονική στιγμή.
- Για το mining επιλέξαμε να έχουμε ένα εξειδικευμένο Thread σε κάθε node (είναι μάλιστα attibute της κλάσης). Αυτό είναι υπεύθυνο να γεμίζει κάθε φορά ένα νεό block και να κάνει mine, ενώ ελεγχει περιοδικά, αν έχει έρθει κάποιο ξένο block. Τότε αυτοκτονεί για αν επιτρέψει την πρόσβαση στις κρίσιμες δομές που χρειάζονται στο validation του αφηχθέντος block.
- Κατά την άφιξη ενός ξένου block και αφού λάβουμε τα κατάλληλα locks, αρχικά κάνουμε validate το block. Αν το validation επιτύχει, βάζουμε το block στο blockchain και συνεχίζουμε. Αν αποτύχει τρέχουμε τον αλγόριθμο consensus. Συγκρικριμένα, αρχικά ζητάμε από όλους τους κόμβους τα μήκη των αλυσίδων τους. Στη συνέχεια βρίσκουμε το μεγαλύτερο μήκος. Μόνο, αν αυτό το μήκος είναι μεγαλύτερο από το μήκος της δικής μας αλυσίδας ζητάμε από τον ανίστοιχο κόμβο να μας στείλει την αλυσίδα του. Έτσι ελαχιστοποιούμε την κυκλοφορία μηνυμάτων με μεγάλο payload στο δίκτυο. Μετά από αυτό ελέγχουμε αν η αλυσίδα είναι valid και αν είναι την υιοθετούμε.
- Σημειώνουμε επίσης ότι χρήση locks γίνεται και κατά την διαδικασία δημιουργίας ή επαλήθευσης transactions για να εξασφαλίσουμε την σωστή ανανέωση των UTXOs καθώς και του pool από transactions που είναι σε αναμονή.

Πειραματικά Αποτελέσματα

Παρακάτω παρουσιάζουμε τα αποτελέσματα των πειραμάτων που κάναμε. Πρόκειται για μετρήσεις Transaction Throughput και Average Block Time για το σύστημα μας όταν το τρέχουμε με 5 ή 10 nodes.

1. Απόδοση του Συστήματος



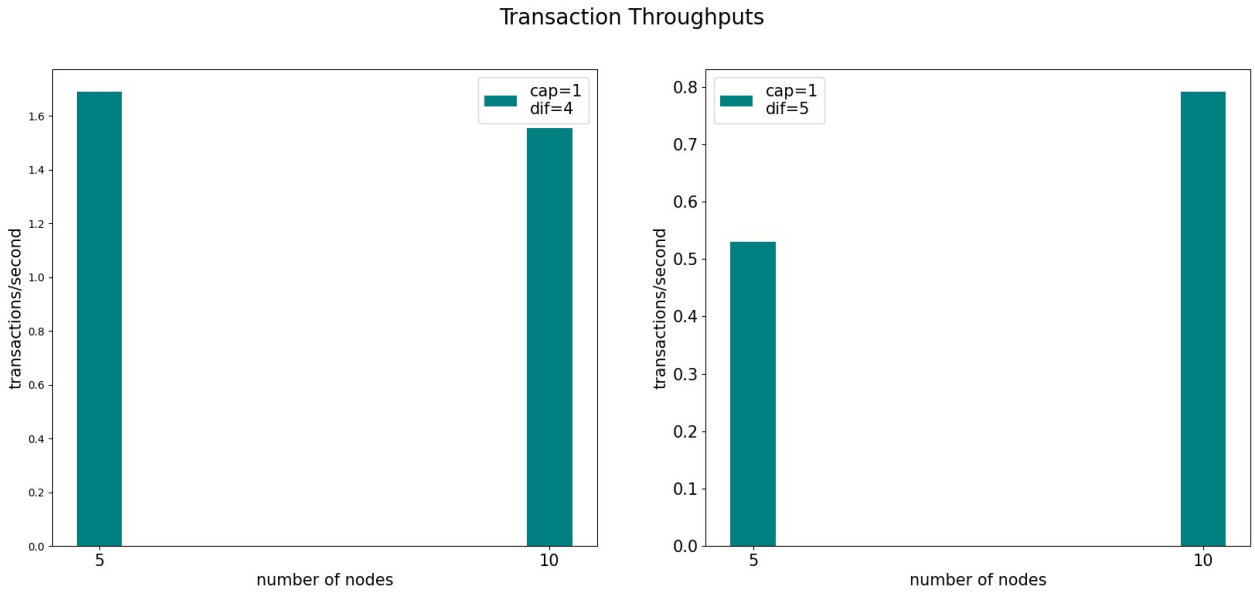


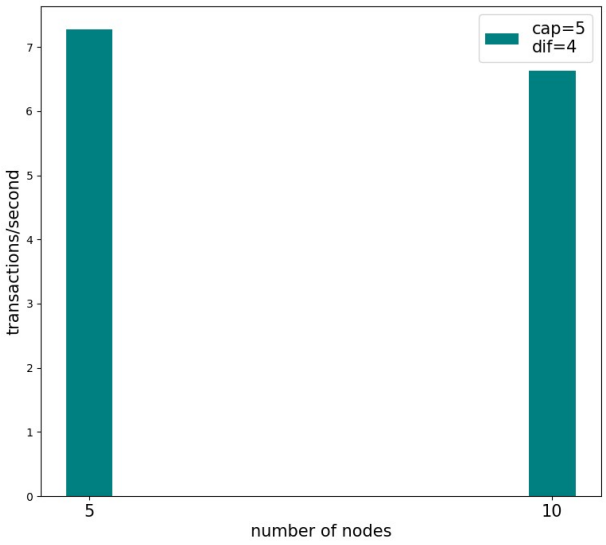
Όσον αφορά το throughput, παρατηρούμε ότι το σύστημα αποδίδει καλύτερα για μεγαλύτερα μεγέθη blocks. Αυτό είναι λογικό, αφού το να κάνουμε mine για κάθε transaction που εισέρχεται στο σύστημα ξεχωριστά (block size = 1) προσθέτουμε πολύ επιπλέον υπολογιστικό βάρος και συνεπώς καθυστέρηση. Αντίθετα, αν κάνουμε mine για μεγαλύτερα πλήθη transactions ομαδικά μειώνουμε αυτό το υπολογιστικό κόστος.

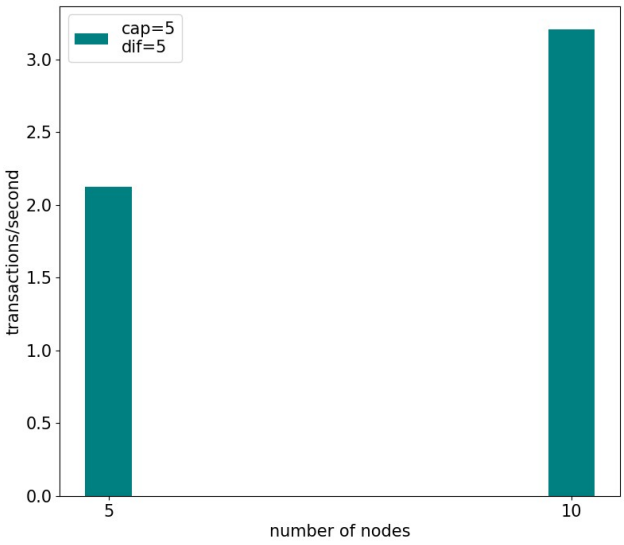
Από την άλλη το block time αυξάνεται καθώς αυξάνουμε το capacity των blocks. Αυτό οφείλεται στον παραπάνω χρόνο που χρειάζεται για να “μαζέψουμε” τα transactions που θα συμπληρώσουν το block, καθώς και στο μεγαλύτερο input που δίνουμε στην hash function. Ωστόσο, η αύξηση στο block time είναι λιγότερο έντονη από την αύξηση στο throughput, οπότε μπορούμε να πούμε ότι το σύστημα συμπεριφέρεται καλύτερα για μεγάλύτερο block size.

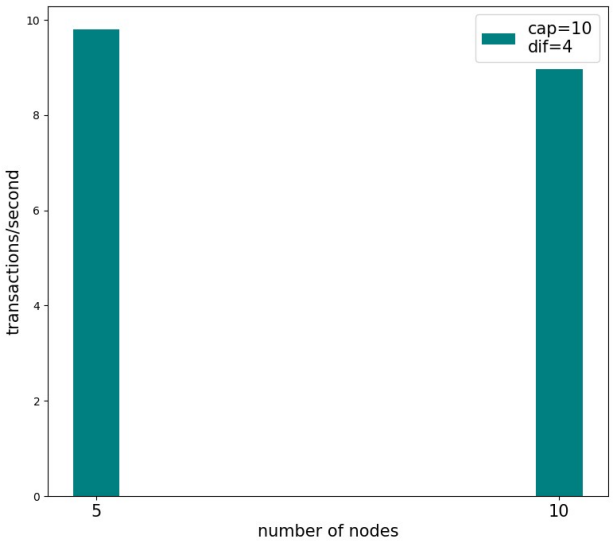
Οι παραπάνω παρατηρήσεις αφορούν τόσο την περίπτωση με mining difficulty = 4 όσο και την περίπτωση με mining difficulty = 5. Όπως είναι αναμενόμενο βέβαια, όταν κρατάμε τις άλλες παραμέτρους σταθερές, ένα σύστημα με difficulty=5 είναι πάντα λιγότερο αποδοτικό (χαμηλήτερο throughput και μεγαλύτερο block time) σε σχέση με το αντίστοιχο σύστημα με difficulty=4, λόγω τις μεγαλύτερης πολυπλοκότητας στους υπολογισμούς.

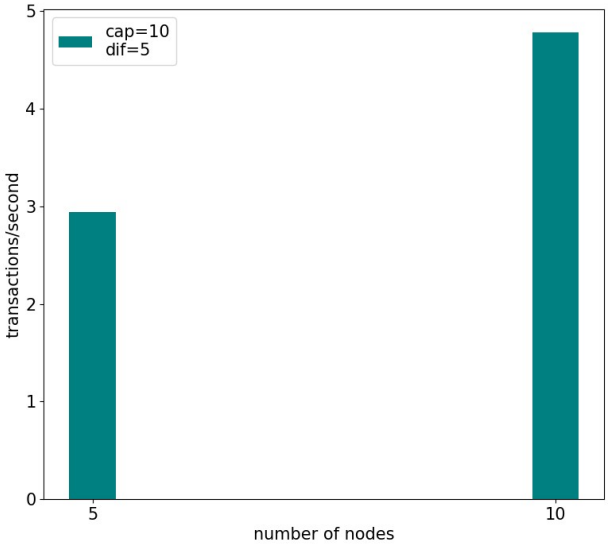
2. Κλιμακωσιμότητα



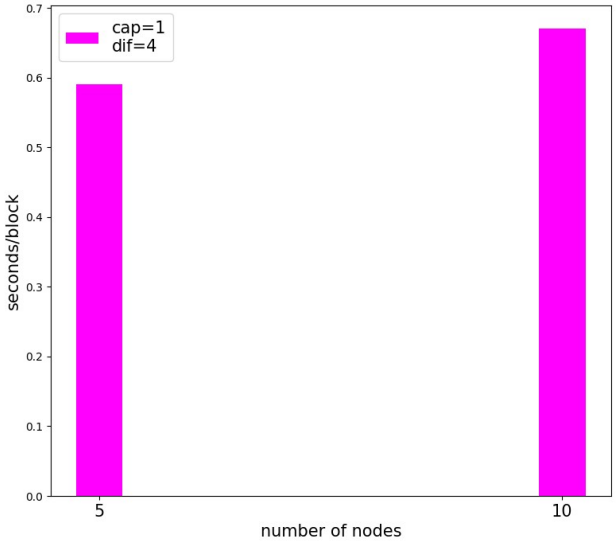

cap=5
dif=4
transactions/second
number of nodes
0
1
2
3
4
5
6
7
5
10

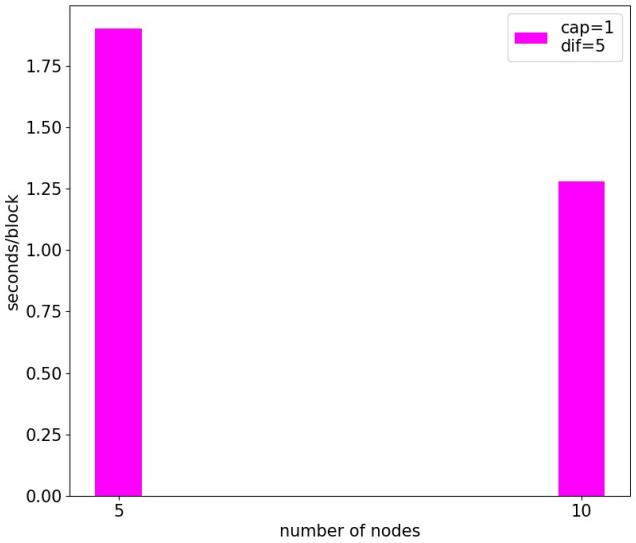

cap=5
dif=5
transactions/second
number of nodes
0.0
0.5
1.0
1.5
2.0
2.5
3.0
5
10

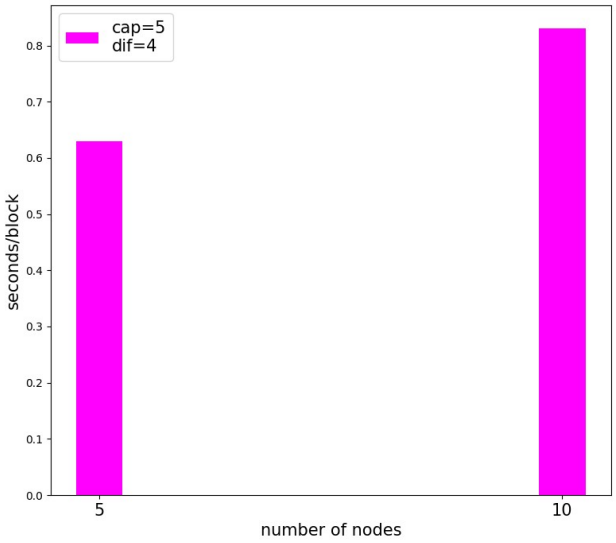

cap=10
dif=4
transactions/second
number of nodes
0
2
4
6
8
10
5
10

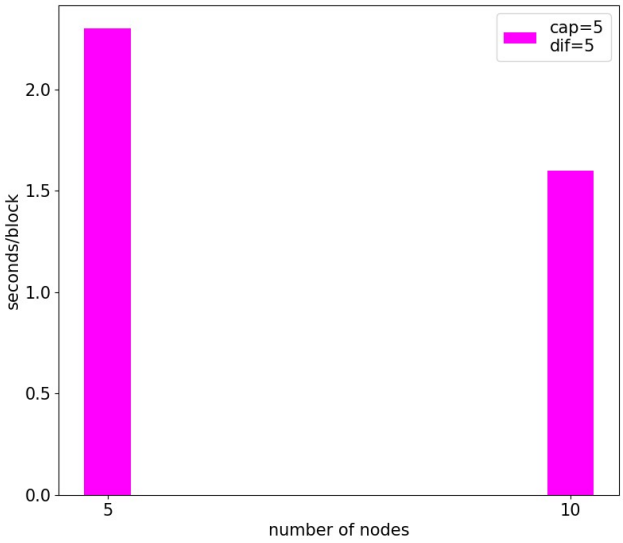

cap=10
dif=5
transactions/second
number of nodes
0
1
2
3
4
5
5
10

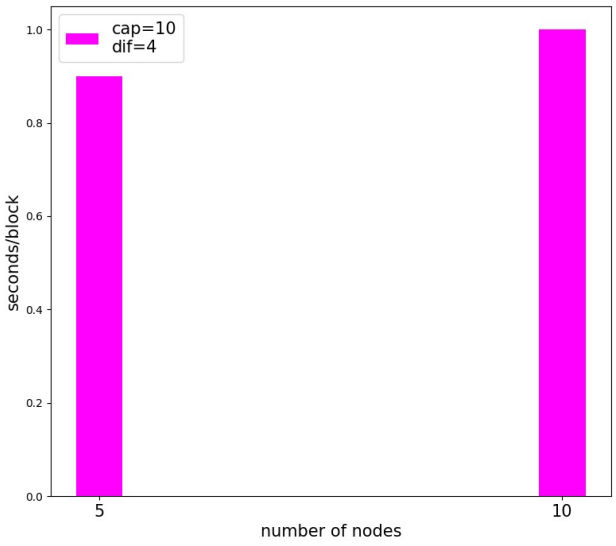
Average Block Time

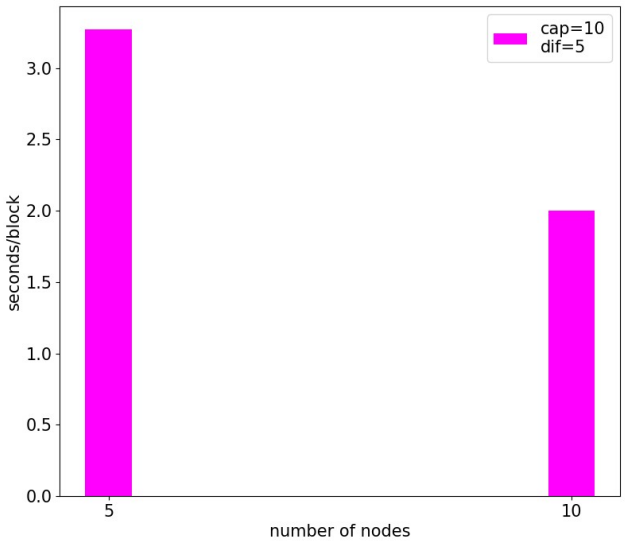












Όσον αφορά την κλικακωσιμότητα κάνουμε μια βασική παρατήρηση: Στις περιπτώσεις difficulty=4 τα πειράματα με 5 κόμβους δίνουν καλύτερα αποτελέσματα (μεγαλύτερο throughput και μικρότερο Block Time) ενώ στις περιπτώσεις με difficulty=5 αποδίδουν καλύτερα τα πειράματα με 10 κόμβους. Μία ερμηνεία που μπορεί να σκεφτεί κανείς για αυτό το φαινόμενο είναι η εξής: στις περιπτώσεις με difficulty=4, οι υπολογισμοί του mining είναι σχετικά φτηνοί υπολογιστικά και συνεπώς το overhead της επικοινωνίας παίζει καθοριστικό ρόλο στην απόδοση του συστήματος. Συνεπώς, τα πειράματα με 5 κόμβους, που απαιτούν λιγότερη επικοινωνία υπερτερούν. Από την άλλη, για difficulty=5, οι υπολογισμοί απαιτούν σημαντικά παραπάνω χρόνο. Συνεπώς η ύπαρξη 5 επιπλέον ομότιμων κόμβων που συμμετέχουν στους υπολογισμούς παράλληλα με τους υπόλοιπους ενισχύει την επίδοση του συστήματος περισσότερο από το κόστος που προσθέτει η επιπλέον επικοινωνία. Έτσι σε αυτές τις περιπτώσεις είναι καλύτερα τα αποτελέσματα για 10 κόμβους.

Εκτέλεση του κώδικα

Για την εκτέλεση του bootstrap κόμβου:

```
python3 rest.py -bc <capacity> -df <difficulty> -n <number_of_nodes>\
-b 1
```

Για την εκτέλεση των άλλων κόμβων:

```
python3 rest.py -bc <capacity> -df <difficulty> -p <port>
```

Για την εκτέλεση σε vms προσθέτουμε τα flags: `-vm 1 -i <ip>`

Εκτέλεση του cli:

```
python3 client.py -a balance -n <id>
```

```
python3 client.py -a view -n <id>
```

```
python3 client.py -a t -n <from_id> -r <to_id> -v <value>
```