Πολυτεχνική Σχολή
Τμήμα Μηχανικών Ηλεκτρονικών Υπολογιστών και Πληροφορικής

Πολυδιάστατες Δομές Δεδομένων και Υπολογιστική Γεωμετρία

Chord –

Peer-to Peer Distributed Hash Table

Υλοποίηση και πειραματική μελέτη απόδοσης των βασικών πράξεων

Project Ακαδημαϊκού Έτους 2021 - 2022

Μέλη Ομάδας		
Παπασταύρου	Χριστίνα	1059621
Ρουχωτά	Ειρήνη	1059654
Χατζημιχάλης	Ιωάννης	1059613
Γεωργιάδης	Χρήστος	1059579

Περιεχόμενα

Εισαγωγή

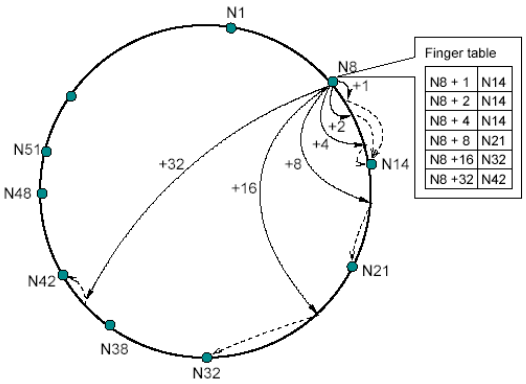
Η εργασία μας αυτή στόχευσε στη υλοποίηση και στην πειραματική μελέτη απόδοσης για τις βασικές λειτουργίες του Chord, ενός πρωτοκόλλου-αλγορίθμου που υλοποιεί ένα peer-to-peer κατανεμημένο πίνακα κατατεμαρτισμού (peer-to-peer Distributed Hash Table ή p2p DHT).

Ένας DHT στόχο αρχικά να αποθηκεύσει ζευγάρια δεδομένων (κλειδί, τιμή) σε διαφορετικούς υπολογιστές του δικτύου που ονομάζονται *κόμβοι*. Το Chord καθορίζει πως κλειδιά (identifiers) ανατίθενται σε κόμβους έτσι ώστε αυτοί να αποθηκεύσουν τα αντίστοιχα ζεύγη για τα οποία λέμε ότι αυτοί είναι οι *αρμόδιοι* κόμβοι. Έτσι αν ένας κόμβος *n* του δικτύου αναζητεί ένα κλειδί με τιμή a, τότε οι κανόνες που υλοποιεί το Chord θα καθορίσουν σε ποιον κόμβο *m* βρίσκεται αποθηκευμένο κλειδί αυτό προσδιορίζοντας με τρόπο αποτελεσματικό και την σχετική τιμή *τιμή_a.*

Το Chord είναι σχεδιασμένο με τέτοιο τρόπο ώστε να μπορεί να λειτουργεί σε ένα δυναμικό δίκτυο κόμβων στο οποίο μπορεί κάθε στιγμή:

1) να συνδεθεί ένας νέος κόμβος (node join)
2) ένας υπάρχων κόμβος να αποσυνδεθεί (node leave)
3) ένας υπάρχων κόμβος να γίνει απροσπέλαστος π.χ. λόγω μιας ζημιάς (node failure)

Οι κόμβοι του δικτύου ταξινομούνται πάνω σε έναν κύκλο modulo 2^m (Chord Ring) όπως στο παρακάτω σχήμα όπου διακρίνονται 9 κόμβοι (N_1, N_8, N_{14}, N_{21}, N_{32}, N_{38}, N_{42}, N_{48}, N_{51}). Κάθε κόμβος όπως και κάθε κλειδί απεικονίζονται πάνω στον Chord Ring μέσα από μία συνάρτηση Consistent Hashing η οποία παράγει τα κατάλληλα αναγνωριστικά (identifiers) τόσο για τους κόμβους όσο και για τα κλειδιά. Κάθε κλειδί με αναγνωριστικό id_{key} ανατίθεται στον πρώτο κόμβο του οποίου το αναγνωριστικό id_{node} είναι $id_{key} <= id_{node}$. Αυτός ο κόμβος ονομάζεται ο «επόμενος» (successor) του κλειδιού key. Κάθε κόμβος n δηλαδή είναι αρμόδιος για τα κλειδιά k για τα οποία ισχύει ότι successor(k)=n



Η πλήρης περιγραφή του Chord βρίσκεται στο [1].

Εκτέλεση της προσομοίωσης από τον χρήστη

Η υλοποίηση μας για την προσομοίωση του Chord σε υπολογιστή έγινε σε γλώσσα Python. Έτσι ο χρήστης δίνοντας :

- python Chord.py -h

εμφανίζονται στην οθόνη του αναλυτικές οδηγίες:

usage: Chord.py [-h] [-n N] [-fn FN] [-d D] [-fr FR] [-fs FS]

Implementing Chord

optional arguments:

-h,	--help show this help message and exit
-n N	N is number of initial numbers of nodes withing Chord ring
-fn FN	FN is the input csv file name to read data from
-d D	D is the number of data records to be loaded from the input csv file
-fr FR	FR is the number of the stored successor failure recovery. If not set no failure recovery action will be used
-fs FS	FS is the name of the file in which statistics will be written

Αν ο χρήστης εκτελέσει το πρόγραμμα προσομοίωσης χωρίς παραμέτρους

- python Chord.py

τότε η προσομοίωση θα λειτουργήσει με προκαθορισμένες τιμές
Initial settings : n=5,d=10,fn=WorldPorts.csv,fs=statistics.csv,fr=False, r=0

όπου:

- Ο αριθμός των κόμβων στο αρχικό Chord Ring θα είναι n=5
- Το αρχείο δεδομένων εισόδου από το οποίο θα διαβαστούν τα αρχικά δεδομένα των κόμβων θα είναι το WordPorts.csv (που θα πρέπει να βρίσκεται στο ίδιο φάκελο με το αρχείο Chord.py)
- Θα φορτωθούν στους κόμβους d=10 εγγραφές από το αρχείο εισόδου
- Δεν θα γίνει Failure Recovery (r=0)
- Το αρχείο καταγραφής των χρόνων εκτέλεσης των βασικών λειτουργιών της προσομοίωσης θα είναι το *statistics.csv* (που θαβρίσκεται στο ίδιο φάκελο με το αρχείο Chord.py)

Το πρόγραμμα προσομοίωσης αφού δημιουργήσει το Chord Ring με τους n κόμβους:

Creating initial configuration of the Chord ring : [====================] 100%

και φορτώσει τα αρχικά δεδομένα από το αρχείο εισόδου στους κόμβους

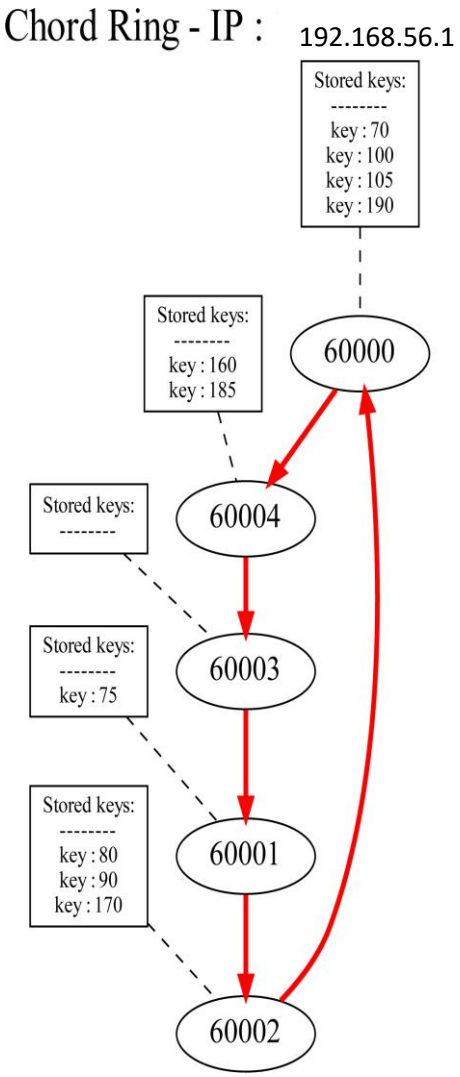
Loading init data from input file to the ring nodes (through the first Node) : [====================] 100%

Τότε εμφανίζει το παρακάτω μενού επιλογών στον χρήστη:

```
================================================================
=
Type [1] to insert a new (key/value) pair in the Chord ring
Type [2] to delete a (key/value) pair from the Chord ring
Type [3] to update value in an existing (key/value) pair
Type [4] to perform an exact match in the network (Lookup)
Type [5] to display current Chord ring configuration
Type [6] to add a new Node (Join) in the Chord ring
Type [7] to delete an existing Node (Leave) from the Chord ring
Type [8] to run a complete benchmark on the current Chord ring
Type [0] to exit
================================================================
==
Select an operation:
```

Για να επιλέξει τη λειτουργία που επιθυμεί. Για παράδειγμα αν επιλέξει το [5] τότε θα εμφανιστεί το παρακάτω γράφημα στην οθόνη του.

Στο γράφημα αυτό, εμφανίζονται συνοπτικές πληροφορίες για το Chord Ring και τα κλειδιά για οποία είναι υπεύθυνος ο κάθε κόμβος. Η διεύθυνση που έχει ο κάθε κόμβος αποτελείται από την πραγματική IP διεύθυνση του υπολογιστή στον οποίο εκτελείται η προσομοίωση και μια αντίστοιχη πόρτα επικοινωνίας. Στο διπλανό γράφημα κάθε κόμβος απεικονίζεται μόνο με την πόρτα επικοινωνίας του.



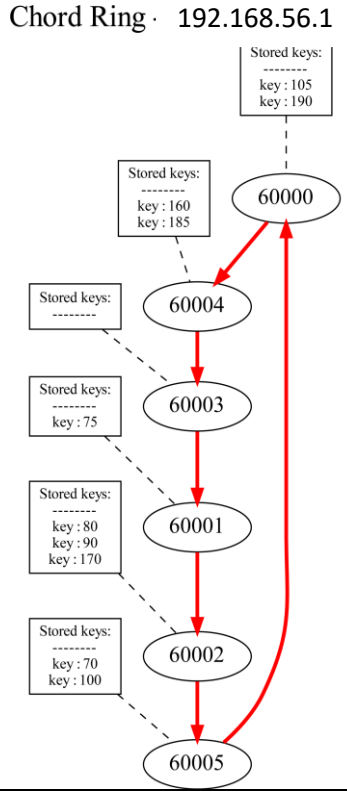
Κάθε φορά που ο χρήστης κάνει μια επιλογή λειτουργίας, ο αντίστοιχος χρόνος που απαιτήθηκε για την εκτέλεση της λειτουργίας αυτής μαζί με ένα αναγνωριστικό της λειτουργίας καταγράφεται αυτόματα από το πρόγραμμα προσομοίωσης στο αρχείο εξόδου (π.χ. στο statistics.csv).

Το αρχείο καταγραφής των χρόνων εκτέλεσης των βασικών λειτουργιών της προσομοίωσης μπορεί να αναλυθεί σε οποιαδήποτε χρονική στιγμή και να γίνουν τα κατάλληλα γραφήματα μέσα από το βοηθητικό πρόγραμμα Python stats.py που φτιάξαμε για το σκοπό αυτό.

Αξίζει, ακόμη, να σημειώσουμε ότι για τον κάθε υπολογιστή που έχει διαφορετική IP παράγονται διαφορετικά αποτελέσματα από τα αυτά των γραφημάτων που παρουσιάζονται στην αναφορά. Για τον κατακερματισμό των κόμβων λαμβάνεται υπόψιν η IP διεύθυνση του υπολογιστή και η πόρτα επικοινωνίας.

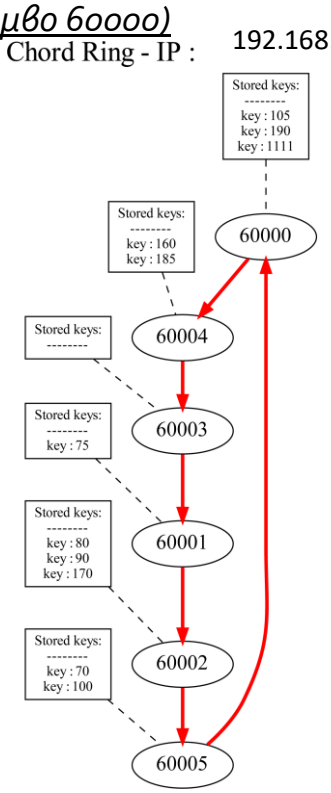
Στον παρακάτω πίνακα φαίνονται αντίστοιχα γραφήματα μετά από σχετικές βασικές λειτουργίες της προσομοίωσης μας. Τα γραφήματα αυτά παράγονται με την επιλογή [5] (βιβλιοθήκη graphviz) μετά από κάθε αντίστοιχη βασική λειτουργία.

Εισαγωγή νέου κόμβου (Join 60005)



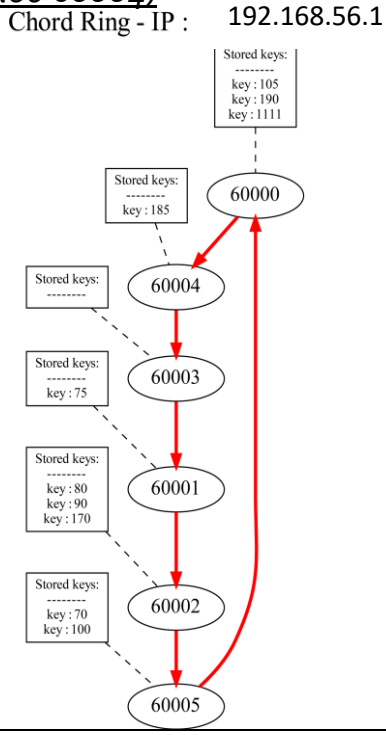
Εισαγωγή ζεύγους με κλειδί : 1111

(στον κόμβο 60000)



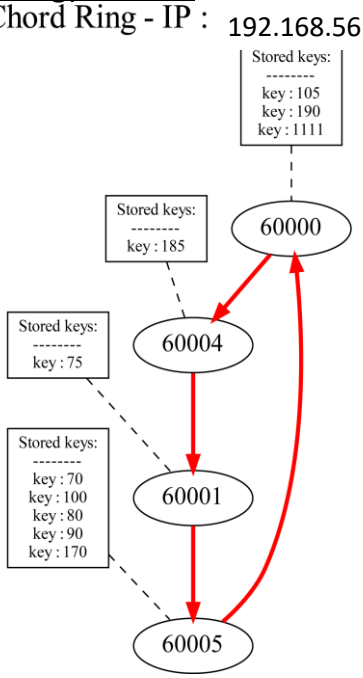
Διαγραφή κόμβου με κλειδί 160

(από το κόμβο 60004)



Αποχώρηση κόμβων

(leave 60003, 60002)



Η επιλογή [8] τρέχει αυτόματα (batch benchmark) όλες τις βασικές λειτουργίες χωρίς την παρέμβαση του χρήστη καταγράφοντας τους χρόνους στο αρχείο καταγραφής. Όπου χρειάζεται κάποιο κλειδί ή τιμή επιλέγεται με τυχαίο τρόπο. Σκοπός της επιλογής αυτής είναι η καταγραφή πολλών δεδομένων με σκοπό τη στατιστική τους επεξεργασία στη συνέχεια.

Για παράδειγμα, για n=5, ένας κόμβος μπορεί από το πρόγραμμα προσομοίωσης μας, να εκτυπώσει κάποιες από τις πληροφορίες του:

==========

I am xxx.xxx.xxx.xxx and I am listening on port 60000

My predecessor is xxx.xxx.xxx.xxx:60002

My successor is xxx.xxx.xxx.xxx:60004

---------- Local keys ----------

Hashed_key:337517828, value : 160->Iceland West Coast -- 60,Bildudalur, ,IS BIL,Iceland,Denmark Strait; North Atlantic Ocean, ,Sailing Directions Pub. 181 (Enroute) - Greenland and Iceland,38030, , ,coa19d, gen19b,...65.683333,-23.6

Hashed_key:2017906878, value : 185->Iceland West Coast -- 60,Bolungavik, IS BOL,Iceland,Denmark Strait; North Atlantic Ocean, ,Sailing Directions Pub. 181 (Enroute) - Greenland and Iceland, 38690...., 66.166667,-23.233333

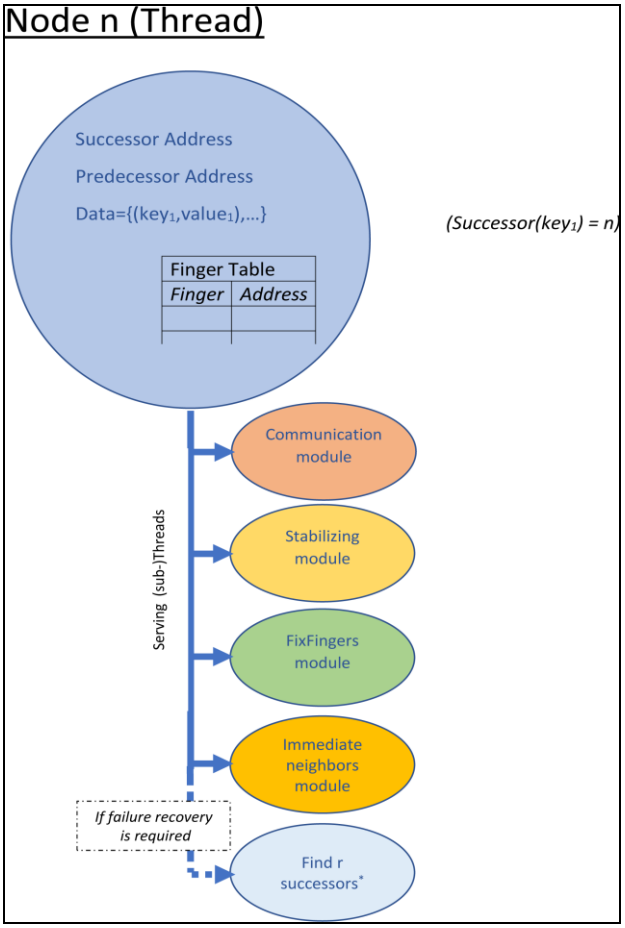
==========

Υλοποίηση

Το πρωτόκολλο όπως περιγράφεται στο [1], υλοποιεί διεργασίες σε ένα δικτύου με ανεξάρτητους κόμβους. Κάθε κόμβος εκτελεί το ίδιο λογισμικό*(decentralization)* χωρίς καμία έννοια δικτυακής ιεραρχίας Είναι γνωστό ότι για την υλοποίηση της επικοινωνίας μεταξύ τέτοιων κόμβων, οι διάφορες γλώσσες προγραμματισμού είναι δυνατόν να χρησιμοποιήσουν **sockets.**

Για την προσομοίωση της παράλληλης λειτουργίας των ανεξάρτητων διαδικτυακών κόμβων σε έναν υπολογιστή, κάθε κόμβος στο πρόγραμμα προσομοίωσής μας υλοποιείται ως μία **υποκλάση (Node)** του αντικειμένου **Thread** της γλώσσας Python. **Σε κάθε τέτοιο κόμβο** όμως θα πρέπει να εκτελούνται σύμφωνα με το Chord, «παράλληλα» διάφορες εργασίες όπως:

- **διεργασία της σταθεροποίησης (Stabilization)**
- **ανταλλαγή μηνυμάτων με άλλους κόμβους**
- **αποκατάσταση διευθύνσεων successor, predecessor, fingers καθώς κόμβοι έρχονται ή φεύγουν**



Εφόσον τα παραπάνω πρέπει να εκτελούνται «παράλληλα» κάθε κόμβος ορίζει δικές ιδιότητες ως νέες υποκλάσεις του threads που αναλαμβάνουν τις παραπάνω εργασίες. Για να είναι αποτελεσματική η υλοποίηση της συνάρτησης κατατεμαρτισμού διευθύνσεων η υλοποίηση μας χρησιμοποιεί για τη μεταβλητή m την τιμή m=32. Συνεπώς ο πίνακας FingerTable έχει 32 θέσεις και τα αναγνωριστικά τοποθετούνται σε Chord Ring modulo 2m.

Παρακάτω βλέπουμε τον constructor της (υπο-) κλάσης Node.

```
class Node(Thread):
  def __init__(self, address, _r, _init=False):
    Thread.__init__(self)
    self.alive = True                  # Is node alive?
    self.lock1 = RLock()               # Node critical point mechanism 1
    self.lock2 = RLock()               # Node critical point mechanism 2
    self.lock3 = RLock()               # Node critical point mechanism 3
    self.localAddress = address        # InetSocketAddress(ip, port) of the node
    self.failure_recovery = _r > 0     # Is failure recovery requested?
    self.r = _r                        # MySuccessors' list length if failure recovery is required
    self.local_id = address.hash()     # node's identifier
    self.print_finger_table = print_finger_table   # Should Finger Table be printed with
                                                   #other node's info
    self.init_phase = _init            # Is it the init phase of the Chord ring
    self.predecessor = None            # predecessor InetSocketAddress of the node
    self.data = dict()                 # dictionary for storing local (key, value) data pairs
    self.FingerTable = dict()          # dictionary to hold Finger Table of the node
    self.MySuccessors = []             # List of my R successors (if failure recovery is required)
    #Initializing all the 32 Finger Tables entries (Nullify fingers)
    for finger_index in range(1, 33):
      self.FingerTable[finger_index] = None  # Finger Table contains Inetsocket-Addresses

# Initializing node's Serving threads
    self.MyPortListener = PortListener(self, self.failure_recovery)
    if self.failure_recovery: self.AskMyRSuccessors = AskMyRSuccessors(self)
    self.MyStabilizer = Stabilizer(self)
    self.MyFixFinger = FixFingers(self)
    self.MyAscPredecessor = AskPredecessor(self)
```

Σημαντικές καθολικές μεταβλητές

Παρακάτω παρατίθενται σημαντικές καθολικές μεταβλητές που χρησιμοποιούνται και επηρεάζουν τη λειτουργία της προσομοίωσης.

```
# Global variables
m = 32                                  # m bits of Chord ring
buffer_size = 1024                      # buffer size for message interchanging
stabilization_frequency = 0.5           # How often stabilization procedure runs (in
secs)
refresh_pointers_frequency = 0.5        # How often pointer refreshing runs (in secs)
socket_time_out = 15                    # Seconds to wait for a socket request (in secs)
socket_response_waiting_time = 0.1 # Waiting time for an answer to be sent (in
secs)
```

Υλοποίηση serving threads για κάθε κόμβο

Ενδεικτικά παραθέτουμε την υλοποίηση ενός από τα serving threads που υλοποιεί την περιοδική ανανέωση του predecessor ενός κόμβου.

```
from threading import Thread
from Commons import sleep_for, sendRequest, refresh_pointers_frequency

#SERVING THREAD of a node
# Periodically inquiries for the node's predecessor. If it fails, it clears node's prede-
cessor
class AskPredecessor(Thread):
  # constructor
  def __init__(self, _node):
    Thread.__init__(self)
    self.local_node = _node
    self.alive = True

# Override main thread run procedure
  def run(self):
    while self.alive:
      predecessor = self.local_node.getPredecessor()
      if predecessor is not None:
        response_from_predecessor = sendRequest(predecessor, "AreYouThere")
        if response_from_predecessor is None or response_from_predecessor != "IAmA-
live":
          self.local_node.clearPredecessor()
```

```
            sleep_for(refresh_pointers_frequency)

    def activate(self):
        self.start()

    def prepare_to_die(self):
        self.alive = False
```

Σχόλια για την επιλογή της συγκεκριμένης μεθοδολογίας

Επιλέξαμε τον συγκεκριμένο τρόπο υλοποίησης για την προσομοίωση λειτουργίας ενός Chord Ring γιατί θεωρούμε ότι είναι αρκετά **ρεαλιστικό**. Λειτουργεί με **πραγματικές διαδικτυακές διευθύνσεις** και sockets. Έτσι μπορεί πολύ εύκολα, με μικρές τροποποιήσεις, η προσομοίωση αυτή να μεταφερθεί σε πολλούς διαδικτυακούς υπολογιστές. Κάθε ένας από αυτούς θα αναλάβει τη λειτουργία **ενός κόμβου (Thread)** δηλώνοντας την **δική του IP** καθώς και τη λειτουργία των επιμέρους serving Threads του.

Υλοποίηση Failure Recovery

Βασικό βήμα για την υλοποίηση της αποκατάσταση βλαβών είναι η διατήρηση των **σωστών δεικτών στον επόμενο (εν ενεργεία) κόμβο** του Chord Ring. Για να γίνει αυτό σωστά και αποτελεσματικά, κάθε κόμβος να αποθηκεύει μία λίστα **r-successors** με τους r επόμενους δικτυακούς κόμβους στο Chord Ring. Το r είναι μια παράμετρος του εν λόγω συστήματος. Έτσι αν ένας κόμβος n διαπιστώσει ότι **ο successor** του είναι **απροσπέλαστος** (π.χ. timeout σε κάποια σύνδεση) τότε τον αντικαθιστά με τον επόμενο εν ενεργεία κόμβο από τη λίστα των r-successors. Οι διαδικασίες stabilize και FixFingers (στα αντίστοιχα serving Threads) εκτελούμενες **περιοδικά** θα επιδιορθώσουν τους δείκτες στους ανενεργούς κόμβους. Η απόδοση του αλγορίθμου προφανώς εξαρτάται από τη συχνότητας Node Join και Leave σε συσχέτιση από τη συχνότητα εκτέλεσης των παραπάνω serving Threads.

Failures και αντιγραφή δεδομένων

Όταν ένας κόμβος n καταστεί ανενεργός (fails/δεν απαντά), οι υπόλοιποι κόμβοι στους οποίους υπάρχει αναφορά στον κόμβο n, θα πρέπει να ανανεώσουν το επόμενο και προηγούμενο τους ενεργό κόμβο. Για να μπορέσουν όμως να απαντήσουν σε αιτήσεις αναζήτησης (lookup) θα πρέπει να έχουν **αντίγραφα των ζευγαριών που ήταν αποθηκευμένα και στον κόμβο n.**

Αν ένας κόμβος καταστεί ανενεργός τότε, οι άμεσα γείτονες τους θα διαπιστώσουν το γεγονός αυτό μέσα από τα serving Threads τους (π.χ. το AskPredecessor). Έτσι θα πρέπει να ακολουθηθεί επίσης μια διαδικασίας ανανέωσης/επιδιόρθωσης των αντίστοιχων finger (από το serving Thread FixFingers).

Η ανοχή σε σφάλματα προϋποθέτει τα διπλότυπα ζεύγη (key, value) σε διαδοχικούς κόμβους. Έτσι κατά την εισαγωγή ενός νέου κόμβου και τη μεταφορά των κατάλληλων ζευγών σε αυτόν, θα πρέπει να **αποσταλούν αντίγραφα των ζευγών αυτών και στους r-successors του**. Το ίδιο προφανώς πρέπει να συμβεί κατά την εισαγωγή ενός νέου ζεύγους σε έναν κόμβο. Αν ένας κόμβος, κατά τη διάρκεια μιας αναζήτησης δεν απαντήσει μέσα σε προκαθορισμένα πλαίσια (π.χ. connection timeout) τότε η αναζήτηση ανακατευθύνεται στο επόμενο στοιχείο της λίστα των r-successors του κόμβου.

Στατιστική ανάλυση υλοποίησης Chord

Πρόγραμμα για την στατιστική ανάλυση

Η πειραματική μελέτη της απόδοσης που ακολουθεί, αναφέρεται στους πραγματικούς χρόνους εκτέλεσης των βασικών πράξεων του Chord. Όπως προαναφέρθηκε κάθε φορά που το πρόγραμμα προσομοίωσης εκτελεί μία βασική πράξη, η ταυτότητα της πράξης και ο χρόνος που απαιτήθηκε καταγράφονται στο τέλος ενός τοπικού αρχείου εξόδου που ο χρήστης μπορεί να καθορίσει ως παράμετρο στη γραμμής εντολών.

Στη συνέχεια παρατίθενται μερικά παραδείγματα από αρχείο καταγραφής το οποίο μπορεί να συγκεντρώσει δεδομένα από πολλαπλές εκτελέσεις του προγράμματος:

Χρονοσήμανση	**Λειτουργία**	**Αριθμός κόμβων**	**Χρόνος εκτέλεσης (σε secs)**
30_01_2022_11_02_22	Insert	4	0.234375
30_01_2022_11_02_22	Lookup	7	2.25
30_01_2022_11_02_22	Update	8	3.671875
30_01_2022_11_02_22	Delete	4	,2.796875

30_01_2022_11_02_22	AddNewNode	3	5.265625
30_01_2022_11_02_22	Leave	4	4.578125

Το πρόγραμμα που δημιουργήσαμε για τη στατιστική επεξεργασία των αποτελεσμάτων (stats.py) διαβάζει το αρχείο καταγραφής και για κάθε μια από τις βασικές λειτουργίες υπολογίζει τον μέσο όρο χρόνου ανά αριθμό κόμβων στο Chord Ring.

Για να λειτουργήσει το παραπάνω πρόγραμμα θα πρέπει ο χρήστης να πληκτρολογήσει :

- python stats.py <input csv file name>

Σχόλια για τη στατιστική ανάλυση

Είναι χρήσιμο θεωρούμε, πριν παρουσιάσουμε τα αποτελέσματα της στατιστικής επεξεργασίας να αναφερθούμε στο πως αυτά επηρεάζονται από την υλοποίηση του προγράμματος μας.

Όπως προαναφέρθηκε, κάθε κόμβος του προγράμματος υλοποιείται από ένα Thread. Για κάθε τέτοιο κόμβο, ενεργοποιούνται 4-5 serving threads **(PortListener, Stabilizer, FixFingers, AskPredecessor, AskMyRSuccessors** (αν έχει ζητηθεί Failure Recovery). Άρα στο παράδειγμα με τους 5 κόμβους στο υπολογιστή που τρέχει το πρόγραμμα προσομοίωσης εκτελούνται 20 έως 25 thread ταυτόχρονα με την ανταλλαγή μηνυμάτων Sockets. Οι απαιτήσεις όλων των παραπάνω σε **υπολογιστικούς πόρους** είναι μεγάλες και σίγουρα επηρεάζουν και τους καταγεγραμμένους χρόνους εκτέλεσης. Σημαντικός παράγοντας επίσης στην μέτρηση των χρόνων είναι και η **συχνότητα εκτέλεσης των serving threads**. Στο σχετικό άρθρο για το Chord, αναφέρει ότι η συχνότητα εκτέλεσης των ενεργειών σταθεροποίησης σε σχέση με τη συχνότητα των node join/leave επηρεάζει άμεσα και την επίδοση του. Στη δική μας υλοποίηση, προφανώς όσο συχνότερα εκτελούνται τα serving threads τόσο μεγαλώνουν και οι χρόνοι εκτέλεσης των επιμέρους εντολών.

Η υλοποίηση μας, θα μπορούσε εύκολα να επεκταθεί και να συμπεριλάβει και άλλους **διαδικτυακούς υπολογιστές** αλλάζοντας την IP διεύθυνση των αντίστοιχων κόμβων και ορίζοντας κατάλληλη **πόρτα επικοινωνίας**. Τότε προφανώς οι χρόνοι θα ήταν πιο ενδεικτικοί. Στην τωρινή υλοποίηση μας, όλοι οι κόμβοι παίρνουν διαφορετική πόρτα επικοινωνίας αλλά την ίδια IP (του υπολογιστή που εκτελεί την προσομοίωση). Έτσι λοιπόν οι αναφερόμενοι χρόνοι μπορεί να επηρεαστούν και από **άλλα προγράμματα ή λειτουργίες που εκτελούνται στον ίδιο υπολογιστή**.

Επίσης όπως γνωρίζουμε, η αναζήτηση ενός κλειδιού για παράδειγμα είναι O(log(n)) σε ένα Chord Ring με n κόμβους. Για να μπορέσουμε να δούμε διαφορές στους χρόνους θα πρέπει ο **αριθμός των κόμβων στα πειράματά μας να διαφέρει κατά δυνάμεις του 10**. Κάτι που στην εκτέλεση του προγράμματος σε έναν υπολογιστή είναι δύσκολο έως αδύνατον για τους λόγους που αναφέραμε στις προηγούμενες παραγράφους.

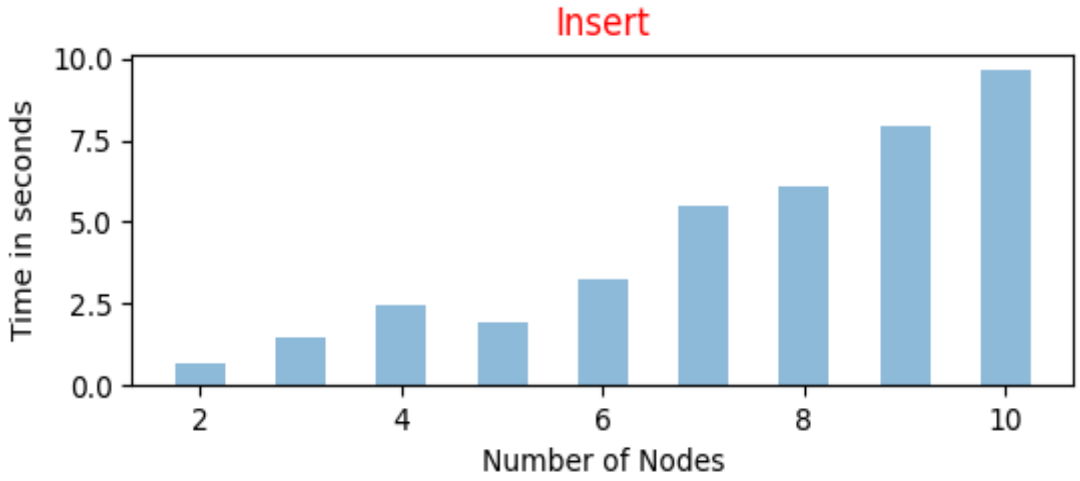
Παρουσίαση γραφημάτων στατιστικής επεξεργασίας

Εκτελώντας το πρόγραμμα στατιστικών

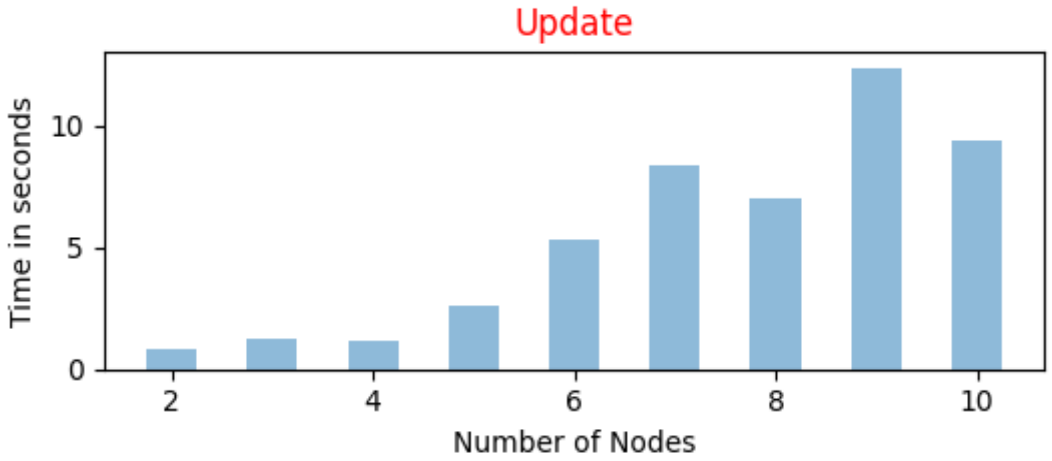
- python stats.py statistics.csv

σχηματίζονται τα παρακάτω γραφήματα για κάθε βασική λειτουργία. Κάθε ένα από τα γραφήματα αυτά δείχνει το μέσο χρόνο εκτέλεσης μιας λειτουργίας (σε διαφορετικές εκτελέσεις του προγράμματος προσομοίωσης) σε σχέση με τον αριθμό των κόμβων που υπάρχουν στο Chord Ring.

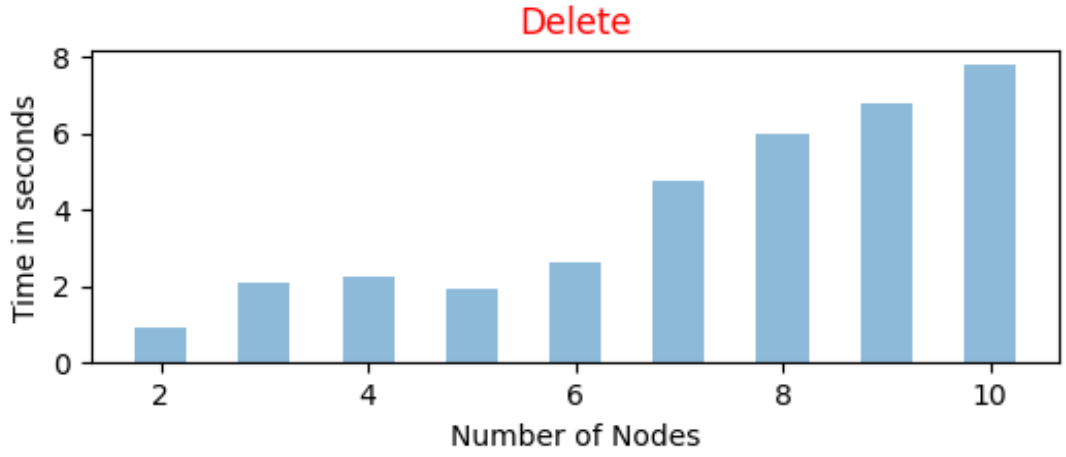
Εισαγωγή νέου ζεύγους (key, value)



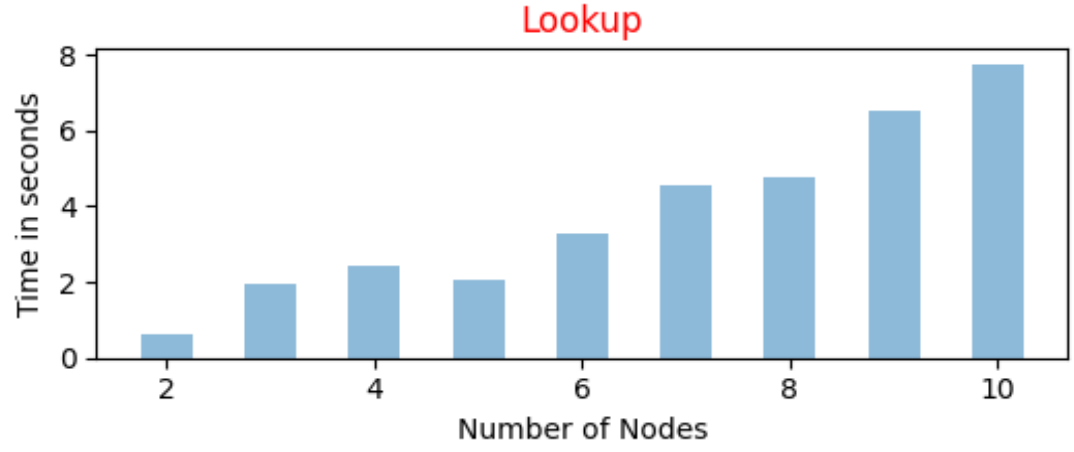
Αλλαγή τιμής σε ζεύγος (key, value)



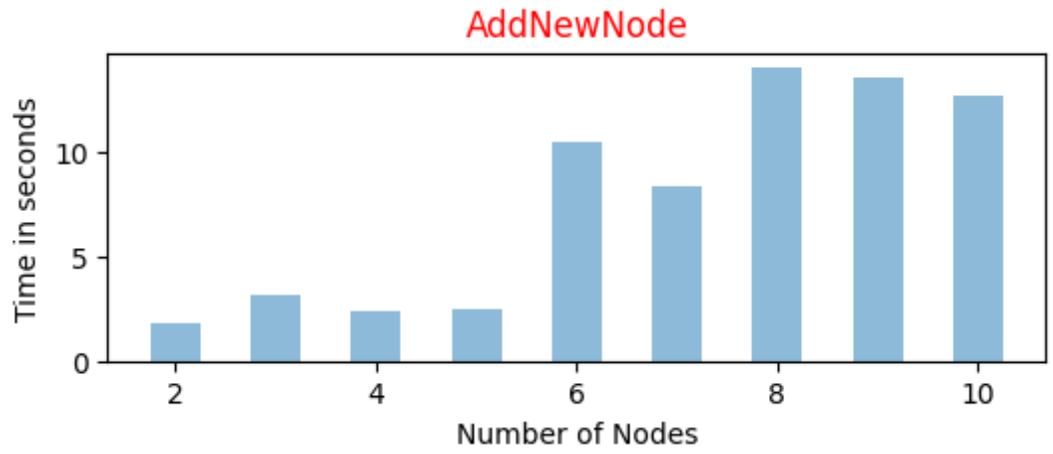
Διαγραφή ζεύγους (key, value)



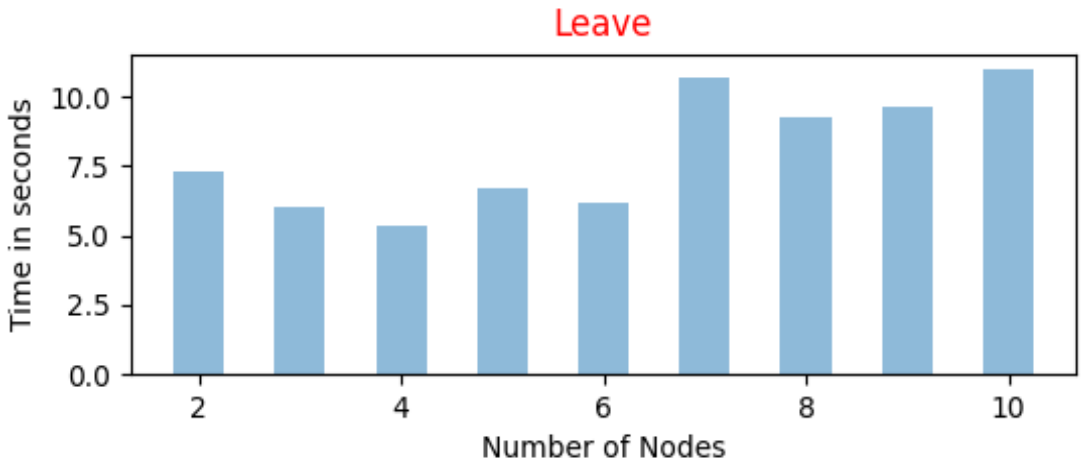
Εύρεση τιμής (Lookup)



Προσθήκη νέου κόμβου

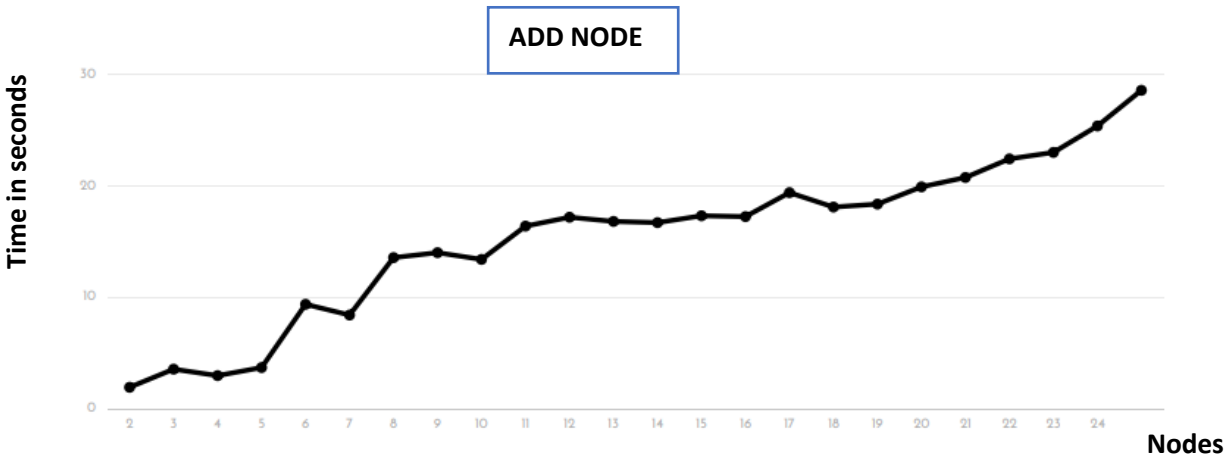


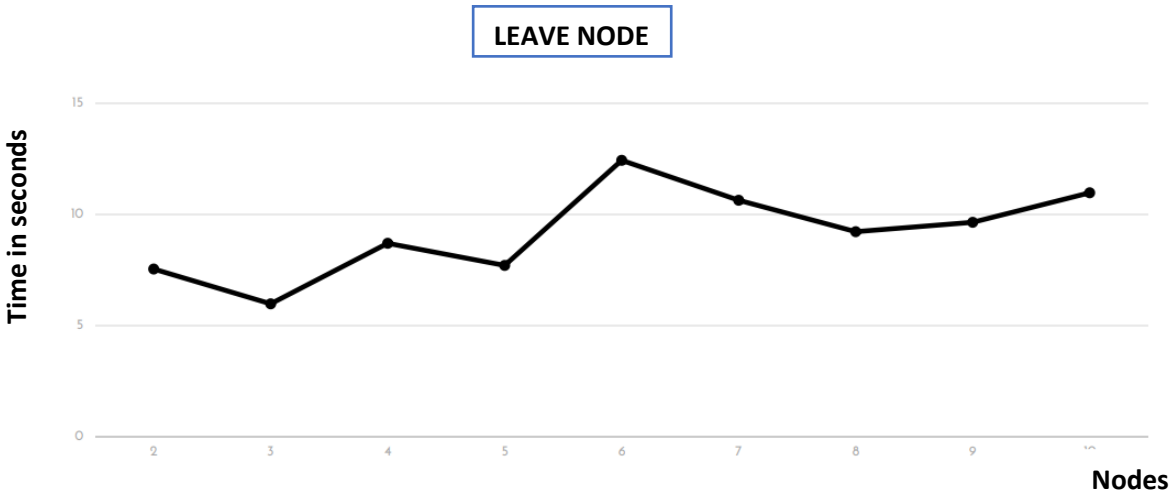
Αποχώρηση ενός κόμβου

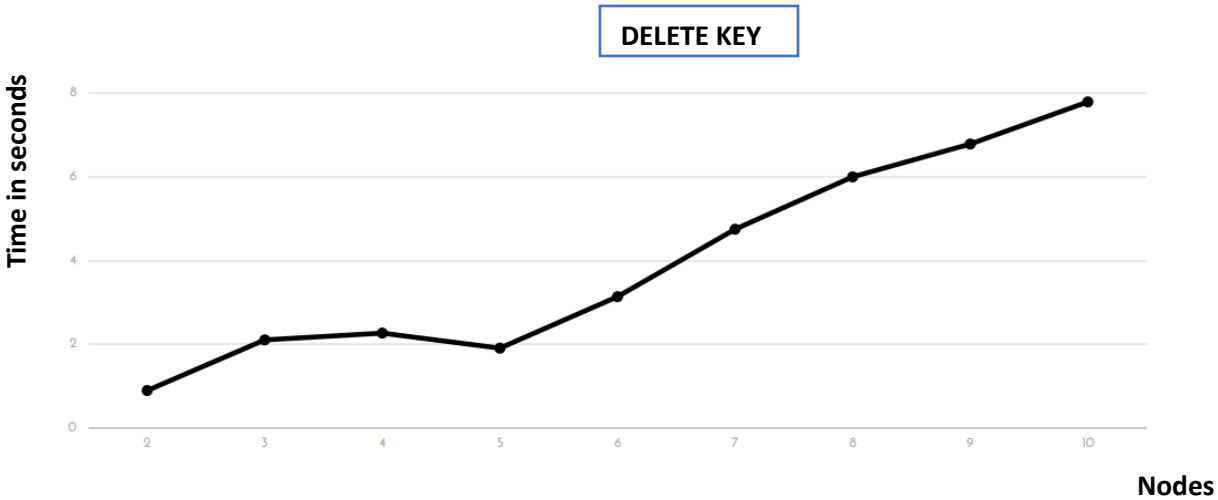


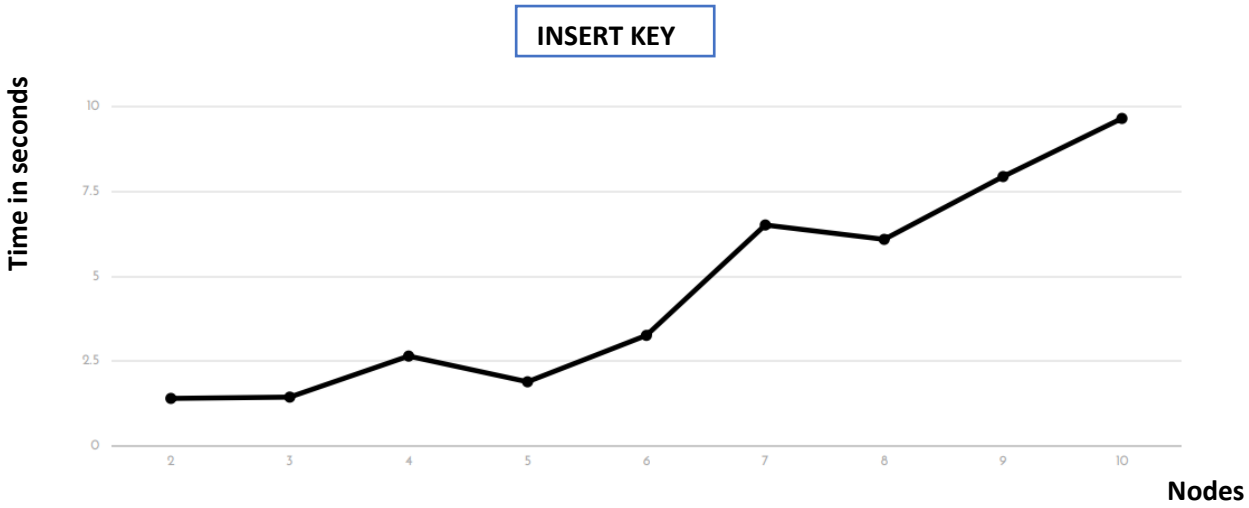
Στην επόμενη ενότητα παρουσιάζονται γραφήματα πειραματικής μελέτης των βασικών πράξεων του chord σε επεξεργαστή Intel Core i9, όπου μπορούσαμε να τεστάρουμε το σύστημά μας για περισσότερους από δέκα κόμβους χωρίς την εμφάνιση μηνυμάτων σφάλματος.

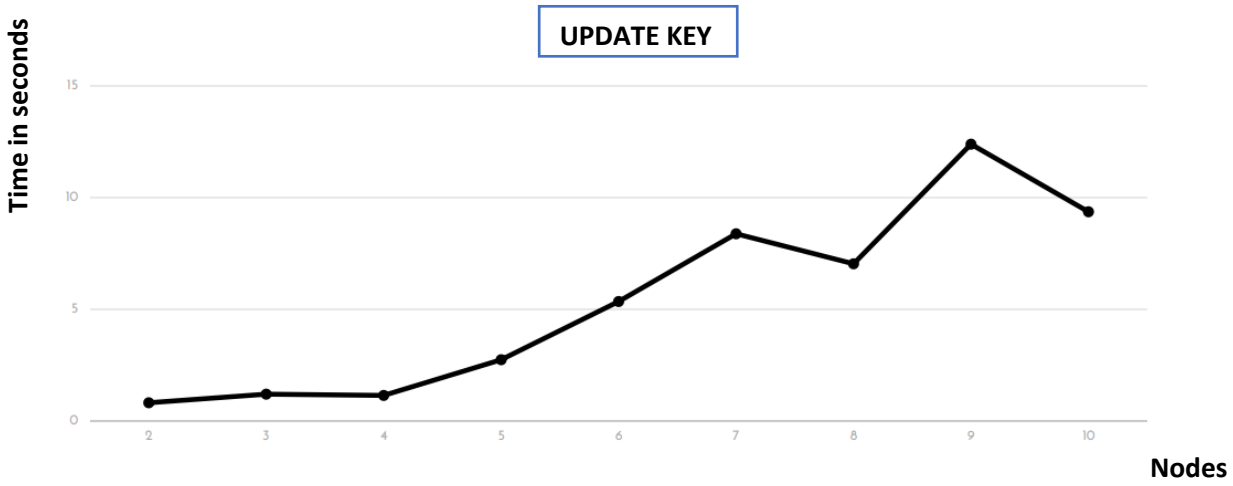
Εκτέλεση πειραμάτων σε επεξεργαστή Intel Core i9

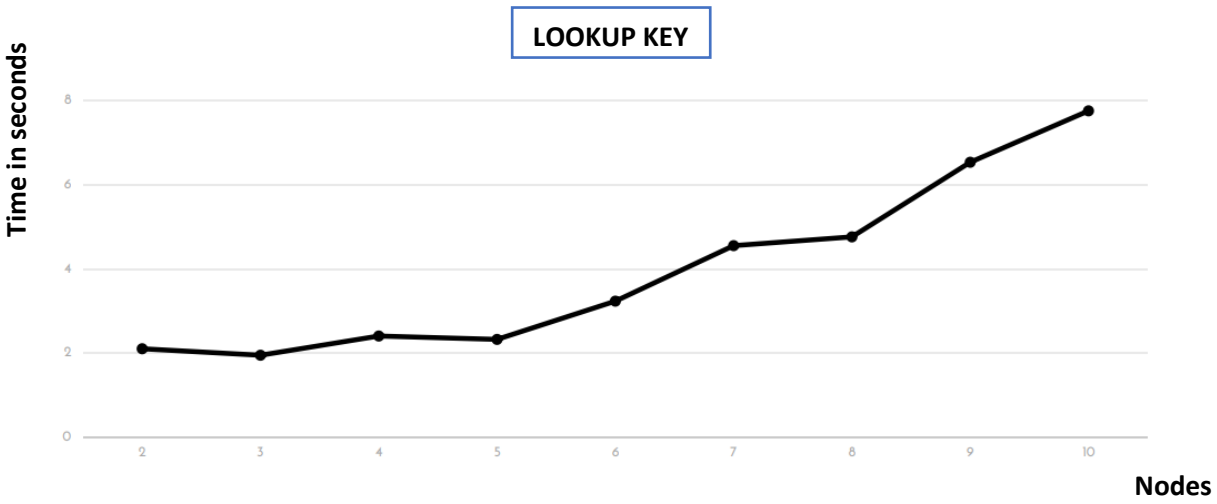







INSERT KEY
Time in seconds
10
7.5
5
2.5
0
2
3
4
5
6
7
8
9
10
Nodes


UPDATE KEY
Time in seconds
15
10
5
0
2
3
4
5
6
7
8
9
10
Nodes


LOOKUP KEY
Time in seconds
8
6
4
2
0
2
3
4
5
6
7
8
9
10
Nodes

Συνοπτικά Κύριες Συναρτήσεις

load_record_to_node(key, value, contact_node):Το ζευγάρι (key, value) μέσω το διαμεσολαβητή κόμβου στέλνεται στον αρμόδιο κόμβο

delete_record_from_node(key, contact_node): Το ζευγάρι (key, value) σβήνεται από τον αρμόδιο κόμβο.

load_file_on_the_ring(node, records_to_load): Φόρτωση αρχικών δεδομένων από αρχείο csv. Χρησιμοποιείται η πρώτη στήλη του αρχείου ως κλειδί και οι υπόλοιπες τις ενώνονται σε string.

fprint(op_type, elapsed_time): Καταγράφονται στο output αρχείο η λειτουργία που επιλέχθηκε και ο χρόνος εκτέλεσής της σε seconds.

InsertNewPair() / insertNewPairCore(key, value): Ο χρήστης εισάγει ένα καινούργιο ζευγάρι (Key, value) και εισάγεται στο chord μέσω ενός τυχαίου κόμβου.

DeletePair() / DeletePairCore(key): Διαγράφεται το κλειδί που θα δώσει ο χρήστης.

UpdatePair() / UpdatePairCore(key, newvalue): Γίνεται Update στο value του κλειδιού που θα δώσει ο χρήστης.

Οι insertNewPairCore(key, value), DeletePairCore(key) και UpdatePairCore(key, newvalue) όταν γίνεται το benchmark τρέχουν με τυχαίες τιμες. Επίσης καταγράφουν τη διάρκεια εκτέλεσης στο output αρχείο.

AddNewNode(init_phase): Εισαγωγή νέου κόμβου στο δίκτυο και ενεργοποίηση - αρχικοποίηση αυτού. Τον προσθέτουμε στο δίκτυο μέσω του κόμβου ο. Μετράμε και καταγράφουμε το χρόνο εκτέλεσης της προσθήκης.

SingleNodeLeave(): Αποχωρεί ένας τυχαία επιλεγμένος κόμβος από το δίκτυο. Ο χρόνος εκτέλεσης της λειτουργίας καταγράφεται.

ExactQuery() / ExactQueryCore(key): Γίνεται εύρεση του κλειδιού που έδωσε ο χρήστης μέσω ενός τυχαίου κόμβου.

autobenchmark(): Αυτόματη λειτουργία βασικών πράξεων και καταγραφή χρόνων εκτέλεσης

DepictNetwork(): Οπτικοποίηση του δικτύου για δεδομένη χρονική στιγμή μέσω της βιβλιοθήκης graphviz.

propagatePairToMyRSuccessors(self, request): Όταν υπάρχει failure recovery ο κόμβος στέλνει όλα τα ζευγάρια (key, value) στους successors του.

fillSuccessor(self): Συμπληρώνει το πεδίο successor του κόμβου. Αν δεν υπάρχει ο successor τον βρίσκουμε μέσα από το πιο ενημερωμένο finger μας. Αν και πάλι δεν τον βρούμε ψάχνουμε στον predecessor.

deleteSuccessor(self): Όταν φεύγει ένας κόμβος, σβήνουμε το successor του και όλες τις αναφορές του στον finger table. Προσπαθούμε παράλληλα να τον αποκαταστήσουμε και να βρούμε τον καινούργιο successor.

move_keys(self, to_address): Μεταφέρει τα κλειδιά που πρέπει στους αρμόδιους κόμβους όταν γίνεται είσοδος νέου κόμβου.

chordjoin(self, contact_node): Προσθέτει έναν νέο κόμβο μέσω του πρώτου κόμβου το chord και καλείται η move_keys για την μεταφορά των κλειδιών.

chordleave(self, AllNodes): είναι η μέθοδος που εκτελείται όταν ένας κόμβος φεύγει. Σταματούν όλες τις ιδιότητες - thread του και ενημερώνει τον επόμενό του ποιος είναι ο νέος του προηγούμενος και αντίστοιχα ενημερώνει τον προηγούμενό του ποιος είναι ο νέος επόμενός του. Μεταφέρει όλα του τα κλειδιά στον επόμενό του.

closest_preceding_finger(self, search_id): Αυτό είναι το βασικό κομμάτι του lookup. Βρίσκω ποιος κόμβος είναι ο πιο αρμόδιος για να κάνει την αναζήτηση του κλειδιού που ψάχνω, ποιος κόμβος δηλαδή είναι πιο κοντά στο id που ψάχνω.

find_predecessor(self, search_id): Ψάχνει μέσω των finger tables να βρει ποιος κόμβος είναι ο κοντινότερος στο search_id και όταν τον βρει επιστέφει ποιος είναι ο προηγούμενός του.

find_successor(self, local_id): Βρίσκει τον επόμενο του local_id ενός κόμβου.

Βιβλιογραφία

[1] ION STOICA AND ROBERT MORRIS AND DAVID LIBEN-NOWELL AND DAVID R. KARGER AND M. FRANS KAASHOEK AND FRANK DABEK AND HARI BALA-KRISHNAN, Chord: A scalable peer-to-peer lookup protocol for Internet applications, IEEE/ACM Transactions on Networking, Vol 11/2003, ISSN: 10636692, Issue1, DOI: 10.1109/TNET.2002.808407,

[2] STOICA, I., MORRIS, R., KARGER, D., KAASHOEK, M. F., AND BALAKRISHNAN, H. Chord: A scalable peer-to-peer lookup service for internet applications. Tech. Rep. TR-819, MIT LCS, March 2001. http://www.pdos.lcs.mit.edu/chord/papers/.