Distributed dictionary using a Java-RMI implementation of the Chord protocol

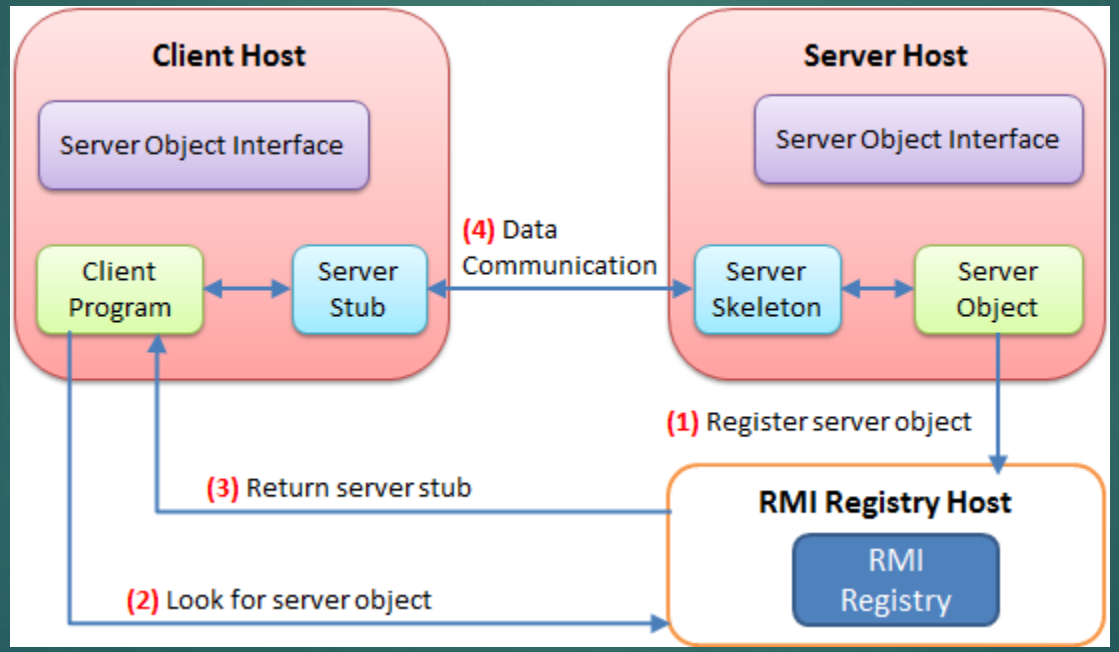
ΕΡΓΑΣΊΑ ΜΑΘΉΜΑΤΟΣ ΠΛΗ414

ΓΙΩΡΓΟΣ ΣΤΑΜΑΤΑΚΗΣ 2013 030 154

Στόχοι και απαιτήσεις

- Στόχος της εργασίας ήταν η δημιουργία ενός p2p δικτύου με τη χρήση του πρωτόκολλου Chord και τη βοήθεια του Java RMI (Remote Method Invocation). Το δίκτυο αυτό αποτελείται από ένα BootStrap node που αποτελεί ένα γνωστό σημείο εισόδου και πολλούς Peer nodes που αποθηκεύουν και επεξεργάζονται τη πληροφορία του δικτύου.
- Κάθε κόμβος μπορεί να κάνει ενέργειες όπως insert/search/update/delete πάνω σε key – value pairs. Στην περίπτωσή μας τα KV pairs είναι συνήθως λέξη – Επεξήγηση λέξεις (αλλά όχι πάντα).

Java RMI (1/2)


Client Host
Server Object Interface
Client Program
Server Stub
(4) Data Communication
Server Host
Server Object Interface
Server Skeleton
Server Object
(1) Register server object
(3) Return server stub
RMI Registry Host
RMI Registry
(2) Look for server object

Java RMI (2/ 2)

- Στην υλοποίηση που έγινε υπάρχουν 2 βασικά Interfaces που καθορίζουν τη συμπεριφορά του δικτύου.
  - Το BootStrapNode interface καθορίζει τη λειτουργία του βασικού node που αποτελεί σημείο εισόδου για το δίκτυο. Έχει γνωστή IP και είναι ο πρώτος κόμβος που έρχεται σε επαφή με ένα νέο κόμβο. Περιέχει μεθόδους join και στατιστικών για το δίκτυο.
  - Το ChordNode interface καθορίζει την αλληλεπίδραση του κάθε κόμβου με τους άλλους peers. Περιέχει μεθόδους insert/get/delete αλλά και μεθόδους για να βρίσκει successor/predecessor. Περισσότερα στο ψευδοκώδικα που ακολουθεί.

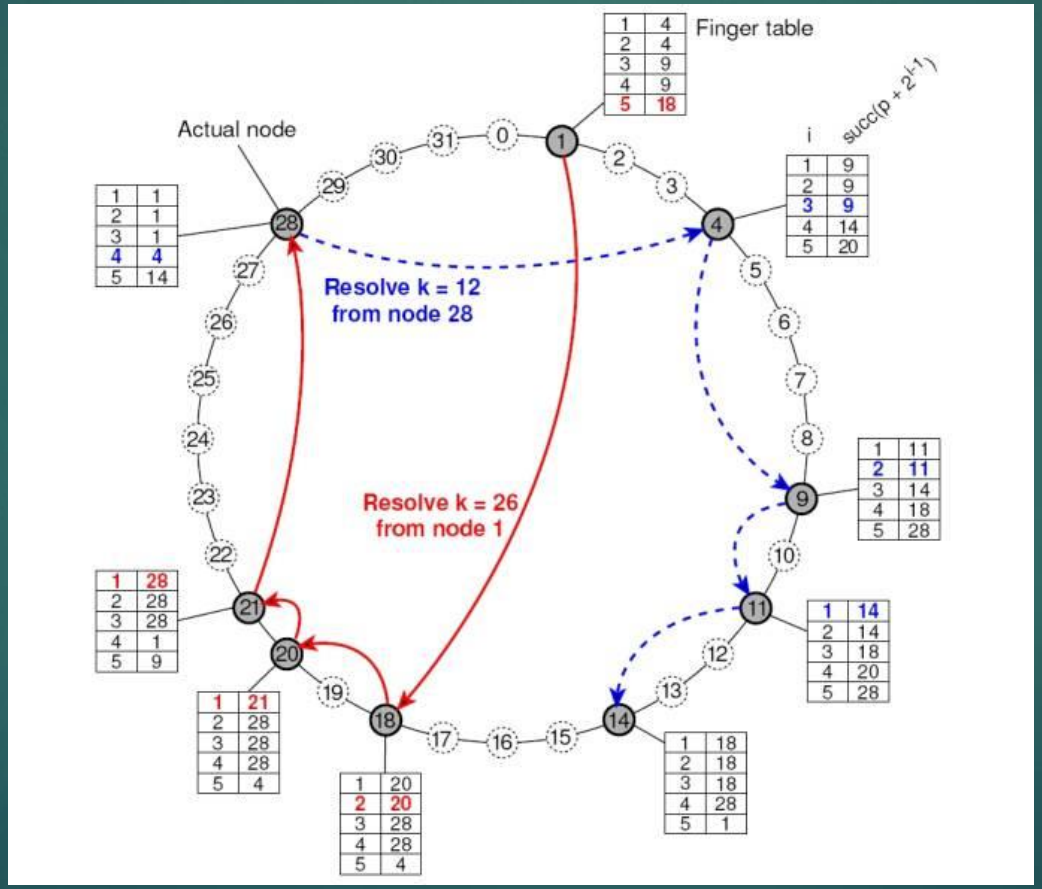
Κώδικας βασικών μεθόδων

```
// ask node n to find id's successor
n.find_successor(id)
    n' = find_predecessor(id);
    return n'.successor;

// ask node n to find id's predecessor
n.find_predecessor(id)
    n' = n;
    while (id ∉ (n', n'.successor])
        n' = n'.closest_preceding_finger(id);
    return n';

// return closest finger preceding id
n.closest_preceding_finger(id)
    for i = m downto 1
        if (finger[i].node ∈ (n, id))
            return finger[i].node;
    return n;
```

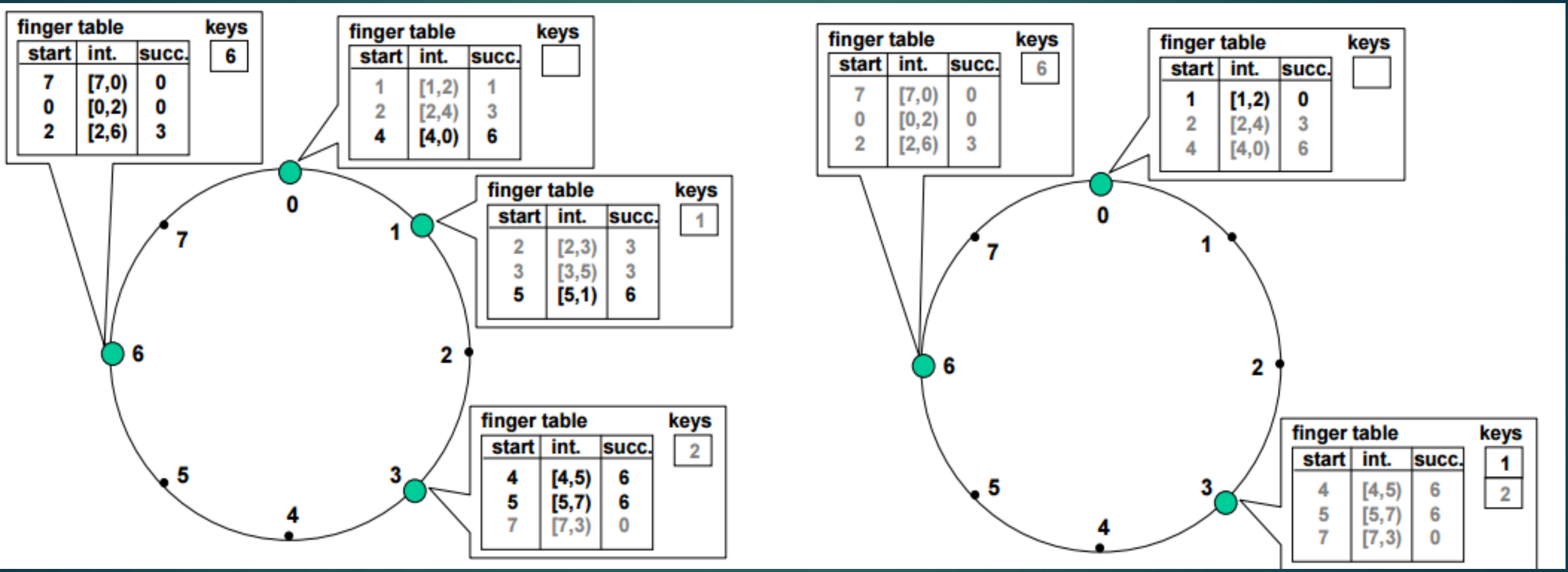
```
// update all nodes whose finger
// tables should refer to n
n.update_others()
    for i = 1 to m
        // find last node p whose i^th finger might be n
        p = find_predecessor(n − 2^(i−1));
        p.update_finger_table(n, i);

// if s is i^th finger of n, update n's finger table with s
n.update_finger_table(s, i)
    if (s ∈ [n,finger[i].node))
        finger[i].node = s;
        p = predecessor; // get first node preceding n
        p.update_finger_table(s, i);
```

```
// periodically verify n's immediate successor,
// and tell the successor about n.
n.stabilize()
    x = successor.predecessor;
    if (x ∈ (n, successor))
        successor = x;
    successor.notify(n);

// n' thinks it might be our predecessor.
n.notify(n')
    if (predecessor is nil or n' ∈ (predecessor, n))
        predecessor = n';

// periodically refresh finger table entries.
n.fix_fingers()
    i = random index > 1 into finger[];
    finger[i].node = find_successor(finger[i].start);
```

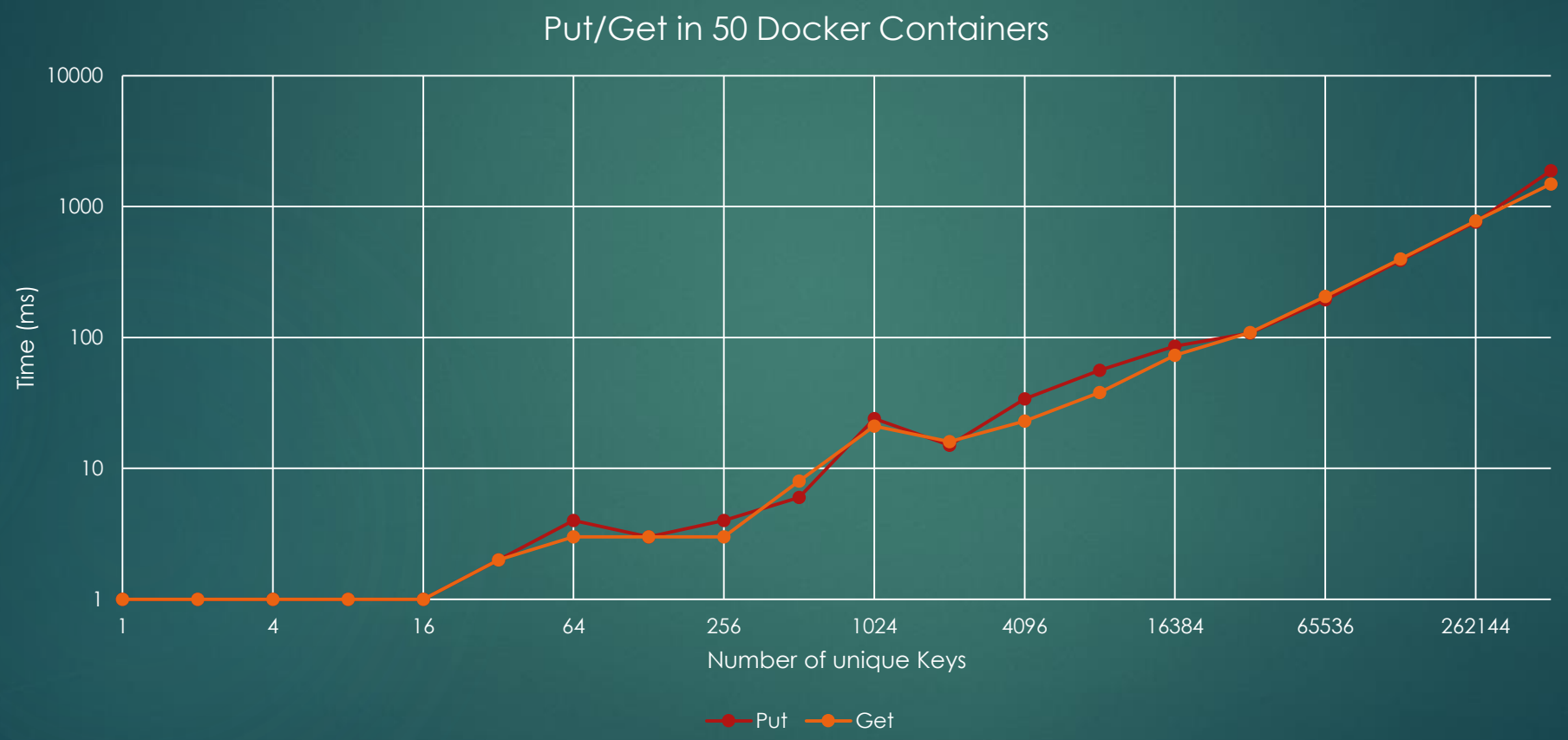
Chord ring lookup example



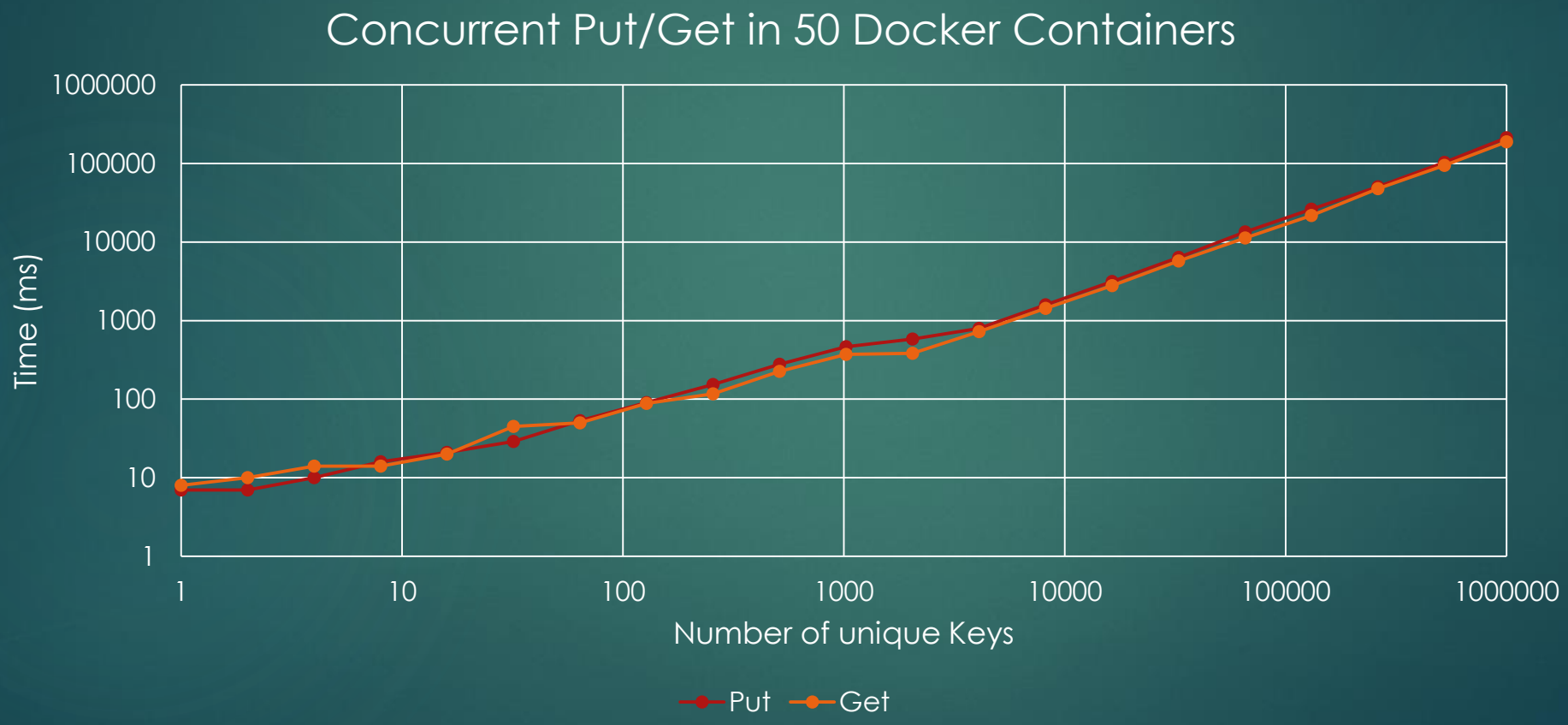
Chord ring joins/departures



Experiments (1/3)



Experiments (2/3)



Experiments(3/3)

- Παρατηρήσεις
  - Παρατηρούμε ότι ο χρόνος εισαγωγής και αναζήτηση αυξάνεται λογαριθμικά του πλήθους των κλειδιών.
  - Από τη στιγμή που το δίκτυο μπορεί να υποστηρίξει την αποθήκευση key-value pairs, μπορούμε να αποθηκεύσουμε σε αυτό οποιαδήποτε μορφής αρχείο απλά κάνοντας σωστό serialization (βλ. εισαγωγή εικόνας από το CLI).

Docker deployment

- Τα Docker Containers είναι πολύ μικρά images από διάφορα services και λειτουργικά συστήματα (μπορεί να τα φανταστεί κανείς σαν πολύ μικρά Virtual Machines). Ο χρήστης φτιάχνει ένα αρχείο που περιγράφει τα services που θέλει και στη συνέχεια φτιάχνει πολλές εικόνες (αντίγραφα) από αυτά.
- Τα πειράματα έτρεξαν σε ένα σύνολο από 50 Docker Containers τα οποία έκανα αρχικά insert κάποια στοιχεία και στη συνέχεια τα έκαναν retrieve.

Διαφορές από το αρχικό proposal

- Η βασική αλλαγή που έγινε ήταν η χρήση java RMI αντί Thrift όπως είχα αρχικά αναφέρει. Ο βασικός λόγος ήταν ότι υπήρχαν πάρα πολλές έτοιμες εργασίες (κάποιος καθηγητής στην California το έχει σαν assignment κάθε χρόνο). Επίσης προτίμησα java RMI γιατί τρέχει out of the box χωρίς επιπλέον πακέτα.
- Όσον αφορά το χρονοδιάγραμμα έγινε χρήση περισσότερων εβδομάδων (3 αντί για 1) για να καταφέρω να κάνω σωστά deploy σε docker containers την εφαρμογή μιας και έπρεπε να αλλάξω μέρος της λογικής της υλοποίησής μου.

Προβλήματα και μελλοντικοί στόχοι

- Αντιμετωπίστηκαν πολλά προβλήματα κατά τη διάρκεια του project με τα περισσότερα να στρέφονται γύρω από τη μεταφορά γνωστού κώδικα σε java RMI.
- Τα docker containers είχαν επίσης αρκετά προβλήματα στην αρχή μιας τόσο αυτά όσο και το Maven δεν είναι ευέλικτα με αποτέλεσμα να χρειαστούν αρκετά ενδιάμεσα scripts για να «δέσουν» σωστά.
- Το debugging με Java RMI είναι αρκετά δυσκολότερο μιας και πρέπει να καταφύγει κανείς συχνά σε logging.
- Μελλοντικά θα ήθελα να βάλω μια μορφή cache στους nodes ώστε να θυμούνται τα queries που έχουν κάνει σε προηγούμενες χρονικές στιγμές. Κάτι τέτοιο θα έριχνε αρκετά το κόστος του lookup.

Πηγές και υλικό

- Όλες οι πληροφορίες εγκατάστασης, οι πηγές, ο κώδικας και τα πειράματα βρίσκονται στο παρακάτω repository.

- https://github.com/gstamatakis/ChordDHT.git