**ΠΡΟΧΩΡΗΜΕΝΑ ΘΕΜΑΤΑ ΒΑΣΕΩΝ ΔΕΔΟΜΕΝΩΝ**

Ακ. έτος 2013-2014, 9ο Εξάμηνο ΗΜ&ΜΥ

Εξαμηνιαία Εργασία

*Εισαγωγή στο MapReduce και στις βάσεις NoSQL*

# 1. Εισαγωγή

Σκοπός της συγκεκριμένης εργασίας είναι η εισαγωγή και η εξοικείωση με το προγραμματιστικό μοντέλο MapReduce και τις βάσεις NoSQL. Συγκεκριμένα, το MapReduce θα μελετηθεί με τη βοήθεια του framework Hadoop, ενώ οι NoSQL βάσεις μέσω τις HBase. Και τα 2 αυτά εργαλεία τρέχουν σε ένα κατανεμημένο υπολογιστικό σύστημα το οποίο μπορεί να αυξομειώνεται ανάλογα με τις ανάγκες τις εφαρμογής. Για τις ανάγκες τις άσκησης χρησιμοποιήθηκα 2 virtual machines από τον ~okeanos τα οποία έτρεχαν Debian Wheezy.

Σαν αντικείμενο μελέτης είχαμε στη διάθεσή μας 2 διαφορετικά datasets, ένα από την America On Line με 20 εκατομμύρια ερωτήματα 650.000 χρηστών και ένα από την Wikipedia με τίτλους άρθρων.

Τα datasets αυτά ανέβηκαν αρχικά στο HDFS (κατανεμημένο σύστημα αρχείων που χρησιμοποιεί το Hadoop) ώστε να μπορεί να γίνει η κατανεμημένη επεξεργασία.

# 2. Εισαγωγή στο MapReduce

**2.1 Yπολογισμός αριθμού αναζητήσεων ανά ημέρα**

Στο ερώτημα αυτό πρέπει να εξάγουμε ένα διάγραμμα από το dataset της AOL το οποίο θα δείχνει το πλήθος των αναζητήσεων που έγιναν ανά μέρα.

Από το αρχείο των αναζητήσεων οι mappers για κάθε γραμμή θα κάνουν emit ένα keyvalue της μορφής <data, 1>. Στη συνέχεια οι reducers θα μαζεύουν τα key-values ανά μέρα. Συνοπτικά:

MAP(key, value) // value is the file line

date = getDate(line)

emit(date,1)

REDUCE(key, value\_list) //

sum = 0

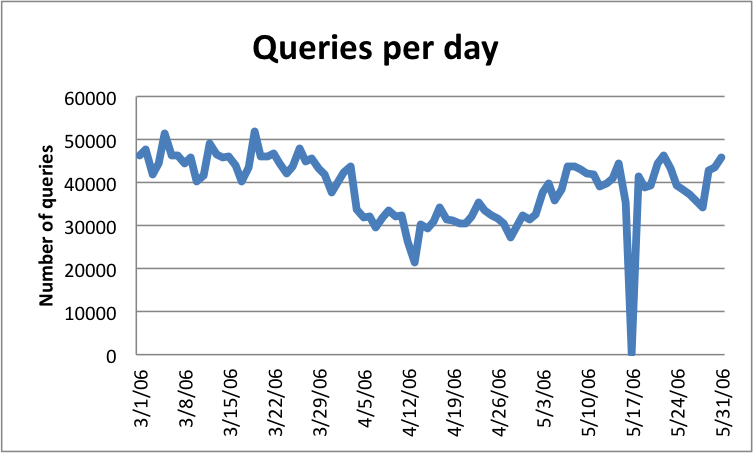
for v in value\_list

sum += v

emit(key, sum)

Μπορούμε να χρησιμοποιήσουμε μόνο έναν reduce ή και περισσότερους. Στην περίπτωση των περισσότερων θα πρέπει να μοιράσουμε τα key-values ανάλογα διατηρώντας τη σειρά. Επιλέξαμε 2 reducers και γνωρίζοντας το εύρος των ημερομηνιών κατασκευάσαμε έναν custom partitioner που μοιράζει τα key-values στα 2.

Παρακάτω βλέπουμε το διάγραμμα του αποτελέσματος.



*Διαγραμμα 1: Ερωτήματα ΑΟL ανά μέρα*

## 2.2 Υπολογισμός ποσοστού «επιτυχών» και «ανεπιτυχών» αναζητήσεων

Στο ερώτημα αυτό ζητείται να υπολογιστούν τα ποσοστά επιτυχών και ανεπιτυχών αναζητήσεων. Επιτυχής αναζήτηση θεωρείται αυτή στην οποία ο χρήστης επέλεξε να μεταβεί σε κάποιο URL, δηλαδή εκείνα τα entries του dataset τα οποία έχουν 4ο και 5ο πεδίο.

Οι mappers διαβάζουν γραμμή-γραμμή το dataset και αν η γραμμή έχει επιτυχή αναζήτηση κάνουν emit keyvalue της μορφής <1,success>, διαφορετικά <1, failure>. Όπως φαίνεται θα χρησιμοποιήσουμε έναν reducer στον οποίο θα καταλήξουν όλα τα αποτελέσματα αφού θέλουμε ποσοστά και χρειαζόμαστε το συνολικό πλήθος αναζητήσεων. O reducer θα μετρήσει τις συνολικές αναζητήσεις και θα υπολογίσει τα ποσοστά επιτυχών και ανεπιτυχών αναζητήσεων. Συνοπτικά:

MAP(key, value):

if check\_success(value):

emit(1,success)

else:

emit(1, failure)

REDUCE(key, value\_list):

success\_count = 0

failure\_count = 0

count = 0

for v in value\_list:

if v==success:

success\_count++

else:

failure\_count++

count++

emit(“Success”, success\_count/count)

emit(“Failure”, failure\_count/count)

Τα αποτελέσματα είναι

Successful searches (%): 53.13

Unsuccessful searches (%): 46.87

## 2.3 Ιστοσελίδων που επισκέφτηκαν πάνω από 10 διαφορετικοί χρήστες

Στο ερώτημα αυτό με βάση το AOL dataset θα πρέπει να να βρούμε τις ιστοσελίδες με πάνω από 10 unique visits και για κάθε μια από αυτές να τυπώσουμε τον αριθμό των επισκέψεων.

Oι mappers κάνουν emit keyvalues της μορφής <url, userId>. Μπορούμε να χρησιμοποιήσουμε πολλούς reducers χωρίς κάποια αλλαγή στο job. Ωστόσο χρησιμοποιήθηκε ένας reducer. Ο reducer παίρνει τα key-values ανά url και τοποθετεί τα userIds μέσα σε ένα HashSet για να υπολογίσουμε τα unique visits. Αν στο τέλος το μέγεθος του set είναι παραπάνω από 10 τότε γίνεται emit το <url, set\_size>. Συνοπτικά:

ΜAP(key, value): // value is the dataset line

url = getUrl(value)

userId = getUser(value)

if (url):

emit(url, userId)

REDUCE(key, value\_list):

Set set;

for id in value\_list:

set.add(id)

if (set.size>10):

emit(key, set.size)

## 2.4 Εύρεση δημοφιλών λέξεων κλειδιών των ερωτημάτων της AOL

Με βάση τα ερωτήματα των χρηστών καλούμαστε να υπολογίσουμε τις πιο δημοφιλείς λέξεις για τις οποίες γίνεται αναζήτηση. Επειδή πρέπει να ταξινομήσουμε το αποτέλεσμα σε φθίνουσα σειρά φορών αναζήτησης δεν μπορούμε να πραγματοποιήσουμε το συγκεκριμένο task με ένα MapReduce job. Στο πρώτο job γίνεται υπολογισμός του πλήθους των αναζητήσεων για κάθε keyword και στο δεύτερο γίνεται η ταξινόμηση.

Ο πρώτος mapper διαβάζει τις γραμμές του dataset και για κάθε λέξη κάνει emit <keyword, 1>. Αντίστοιχα, οι reducers, για κάθε keyword υπολογίζουν το πόσες φορές έγινε αναζήτηση για αυτό και τα ενδιάμεσα αποτελέσματα αποθηκεύονται στο hdfs. Να σημειώσουμε ότι θέλοντας να απαλλάξουμε τα αποτελέσματά μας από «θόρυβο», λέξεις δηλαδή των αγγλικών που χρησιμοποιούνται συχνά όπως άρθα, χρησιμοποιήθηκε μια λίστα από τέτοιες λέξεις, η οποία δινόταν, και οι λέξεις γίνοταν emit στο map σταδιο μόνο στην περίπτωση που δεν υπήρχαν στη συγκεκριμένη λίστα. Η λίστα αυτή έφτασε σε κάθε mapper με τη βοήθεια της DistributedCache class. Τα περιεχόμενα αυτής μπήκαν σε ένα set στο οποίο γινόταν έλεγχος αν υπάρχει ή όχι η εκάστοτε λέξη. Συνοπτικά:

MAP(key, value):

keywords = getKeyWords(value)

word\_list = keywords.split(“ ”)

for w in word\_list:

if (not inStopList(w))

emit(w,1)

REDUCE(key, value\_list):

count=0

for v in value\_list:

count += v

emit(count, key)

Επομένως, τώρα υπάρχουν στο hdfs αποθηκευμά αρχεία που περιέχουν την πληροφορία σχετικά με το πόσες φορές έγινε αναζήτηση για κάποιο keyword. Τα αρχεία αυτά είναι τόσα όσοι και οι reducers του πρώτου jobs, στο οποίο μπορούμε να χρησιμοποιήσουμε όσους θέλουμε αφού γίνεται χρήση του HashPartitioner.

Στο δεύτερο job έχοντας το πλήθος αναζητήσεων για την κάθε λέξη πρέπει να κάνουμε μια ταξινόμηση με βάση το πλήθος. Χρειαζόμαστε identity mappers οι οποίοι θα κάνουν emit ό,τι key-value διαβάσουν. Για το λόγο αυτό στο προηγούμενο job χρησιμοποιήσαμε SequenceFileOutputFormat. Οι reducers θα πρέπει να κάνουν emit αρχεία τα οποία απλά εμείς θα πρέπει να κάνουμε append ώστε να έχουμε την τελική ταξινόμηση. Δεν μας κάνει ο HashPartitioner στην περίπτωση περισσότερων από εναν reducers γιατί θέλουμε ταξινόμηση. Χρειαζόμαστε έναν total-order partitioner. Δεδομένου ότι έχουμε 2 vms στη διάθεσή μας δώσαμε στο 2ο job 2 reducers αλλά μετά από κάποιες δοκιμές διαπιστώθηκε ότι τα δεδομένα δεν ήταν ισοκατανεμημένα και υπήρχε τεράστιος αριθμός keywords με πολύ λίγες αναζητήσεις, δηλαδή κάτω από 10. Επομένως, φτιάξαμε και έναν custom total order partitioner ο οποίος έστελνε keys με τιμή κάτω από 10 στον ένα reducer και τα υπόλοιπα στον άλλον. Ωστόσο, θα μπορούσε να χρησιμοποιηθεί απλά ένας reducer και να μην γίνει όλη η διαδικασία. Τέλος, δεδομένου ότι θέλουμε φθίνουσα ταξινόμηση χρειάστηκε να κάνουμε overwrite τον Comparator. Συνοπτικά για το reduce έχουμε:

REDUCE(key, value\_list): //key=count, value\_list = keywords

for v in value\_list:

emit(v, count)

Η λίστα με τα 50 πιο δημοφιλή keywords είναι και τη συχνότητά τους είναι:

- 104052

free 45149

google 34970

http 24394

yahoo 23491

county 22435

pictures 21416

lyrics 18476

school 18091

myspace 16549

florida 15779

ebay 15603

sale 14612

american 14479

city 14331

home 13794

state 13622

www 12171

music 12161

pics 11704

.com 11691

games 11639

york 11556

girls 10980

beach 10887

bank 10707

texas 10666

online 10664

black 10567

high 10506

nude 10457

aol 10083

yahoo.com 10007

map 9791

news 9614

myspace.com 9577

college 9510

car 9111

mapquest 8895

jobs 8881

2006 8854

homes 8798

ohio 8706

google.com 8682

real 8483

university 8449

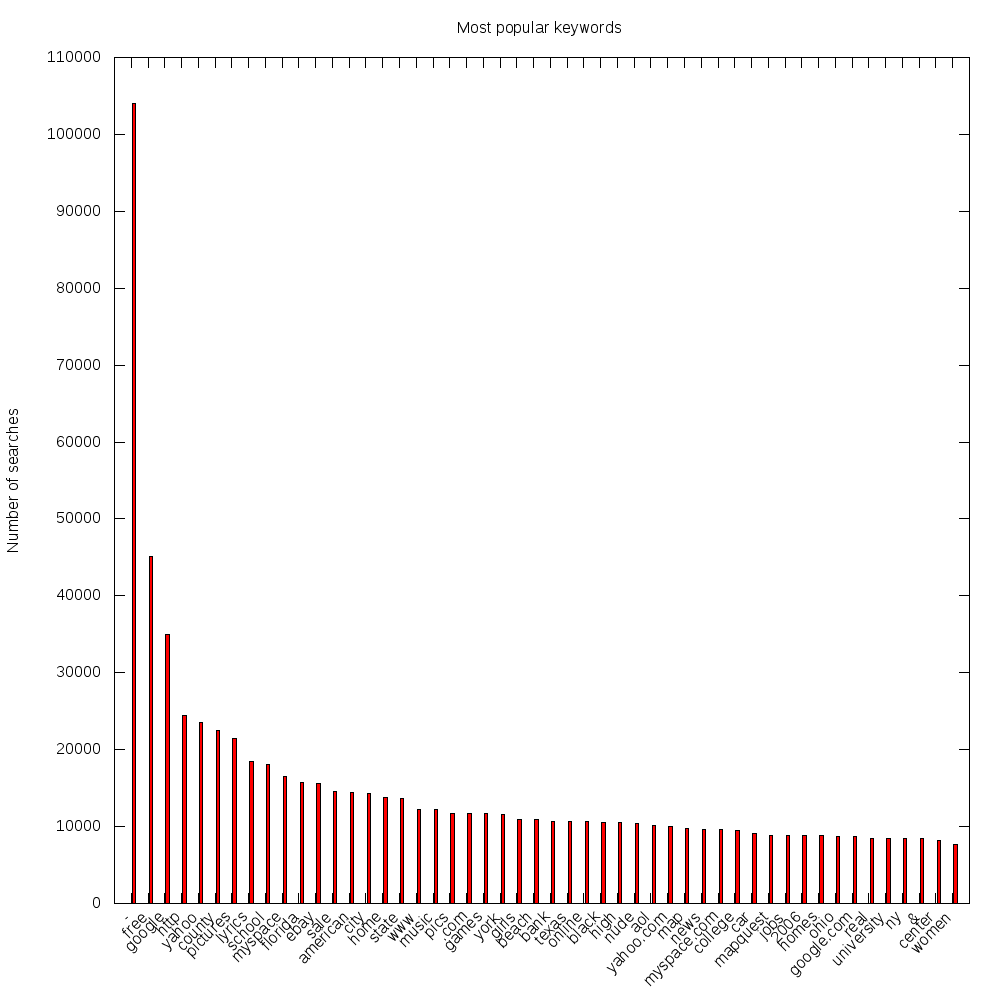
ny 8421

& 8388

center 8169

women 7713

Και σχηματικά,



Αν μετρήσουμε τις γραμμές των αρχείων αποτελεσμάτων βρίσκουμε το πλήθος των μοναδικών λέξεων αναζήτησης οι οποίες είναι συνολικά 120703+459234 = 579937

## 2.5 Υπολογισμός ιστογράμματος της λεξικογραφικής κατανομής των λέξεων

## κλειδιών των άρθρων των τίτλων της wikipedia

### 2.5.1 Κατανομή λέξεων

Με βάση το dataset των τίτλων της Wikipedia, στο ερώτημα αυτό καλούμαστε να κατασκευάσουμε έναν πίνακα ο οποίος για κάθε κατηγορία (έχουμε συνολικά 28 κατηγορίες, μια για κάθε γράμμα του αγγλικού αλφαβήτου, μια για σύμβολα και μια νούμερα) θα περιέχει το ποσοστό εμφάνισης των λέξεων που ανήκουν στην κατηγορία αυτή συνολικά στο αρχείο των τίτλων τηνς wikipedia. Για να το πετύχουμε αυτό θα χρειαστούμε 2 MapReduce jοbs. Το πρώτο θα υπολογίσει το πλήθος των λέξεων ανά κατηγορία και το δεύτερο θα υπολογίσει τα ποσοστά και θα παρουσιάσει ταξινομημένα τα αποτελέσματα. Και πάλι θα γίνει χρήση του αρχείου με τις stop words.

Για το πρώτο job, οι mappers διαβάζουν τους τίτλους και για κάθε λέξη του τίτλου που δεν ανήκει στις stop words κάνουν emit ένα key-value της μορφής <category, 1>. Στη συνέχεια οι reducers αθροίζουν τους άσσους και βρίσκουν το πλήθος λέξεων για τη συγκεκριμένη κατηγορία. Κάνουν emit ένα pair της μορφής <null, category\_count > αφού θέλουμε όλα τα pairs να καταλήξουν στον δεύτερο reducer, χρησιμοποιώντας ένα sequence file. Συνοπτικά,

MAP(key, value): // value ο τίτλος

words = value.split(“\_”)

for w in words:

if (not inStopWords(w)):

emit(category(w), 1)

To category είναι μια συνάρτηση που υπολογίζει σε ποια κατηγορία βρίσκεται η κάθε λέξη και επιστρέφει έναν ακέραιο αριθμό, το id της κατηγορίας.

REDUCE(key, value\_list):

count = 0

for v in value\_list:

count++

emit(null, key+”\_”+count)

Στο δεύτερο job χρησιμοποιούμε έναν identity mapper και όλα τα key-values που παράχθηκαν από το πρώτο job καταλήγουν στον ίδιο reducer. Χρησιμοποιήσαμε το null σαν κλειδί ώστε να έχουμε και τα 28 pairs διαθέσιμα μαζί. Έτσι μπορούμε να υπολογίσουμε αρχικά το συνολικό άθροισμα των λέξεων που μετρήσαμε και μετά το ποσοστό για κάθε κατηγορία. Συνοπτικά,

REDUCE(key, value\_list):

count = 0;

for v in value\_list:

c = v.split(“\_”)[1]

count += c

for v in value\_list:

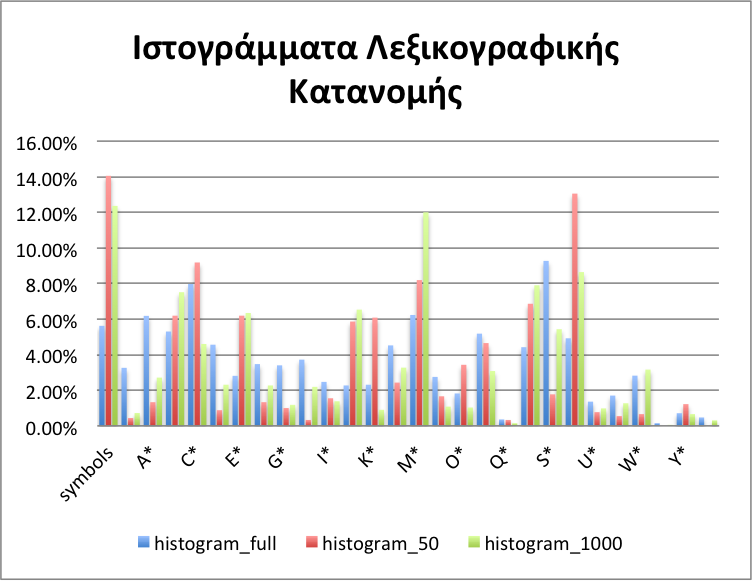
id = v.split(“\_”)[0]

c = v.split(“\_”)[1]

emit(getCategory(id), c/count )

όπου το getCategory μας επιστρέφει τα ονόματα των κατηγοριών, Α\*, Β\* κλπ.

Η παραπάνω διαδικασία πραγματοποιήθηκε 3 φορές. Προσπαθώντας να μειώσουμε το βάρος του υπολογισμού κάναμε δειγματοληψία στον αριθμό των key-values που κάνουν process οι reducers. Για να το καταφέρουμε αυτό κάναμε overwrite τη συνάρτηση run του mapper. Αρχικά εκτελέσαμε το job χωρίς δειγματοληψία. Στη συνέχεια αφήνοντας τους πρώτους mapper να κάνουν process μόνο 50 key-values και τέλος αφήνοντας τους mappers να κάνουν process 1000 key-values. Τα αποτελέσματα παρουσιάζονται στο ακόλουθο διάγραμμα.



time για full

real 4m42.884s

user 0m9.385s

sys 0m0.940s

time για 50

real 3m27.364s

user 0m10.093s

sys 0m0.952s

time για 1000

real 3m25.779s

user 0m9.137s

sys 0m0.852s