Πανεπιστήμιο Πειραιώς – Τμήμα Πληροφορικής
Πρόγραμμα Μεταπτυχιακών Σπουδών
«Προηγμένα Συστήματα Πληροφορικής»

Μεταπτυχιακή Διατριβή

Τίτλος Διατριβής	Κατανεμημένος υπολογισμός πινάκων αφετηρίας-προορισμού με χρήση της τεχνολογίας mapreduce Distributed computation of origin-destination matrices using the mapreduce technology
Ονοματεπώνυμο Φοιτητή	ΙΩΑΝΝΗΣ ΤΣΑΝΤΙΛΗΣ
Πατρώνυμο	ΓΕΩΡΓΙΟΣ
Αριθμός Μητρώου	ΜΠΣΠ/ 13114
Επιβλέπων	Πελέκης Νικόλαος, Επικ. Καθηγητής

Ημερομηνία Παράδοσης Ιανουάριος 2017

Τριμελής Εξεταστική Επιτροπή

(υπογραφή)	(υπογραφή)	(υπογραφή)
Πελέκης Νικόλαος Επικ. Καθηγητής	Θεοδωρίδης Ιωάννης Καθηγητής	Δουληγέρης Χρήστος Καθηγητής

Σύνοψη

Σκοπός της παρούσας μεταπτυχιακής διατριβής είναι η δημιουργία τρόπων επεξεργασίας, μεγάλου όγκου χωροχρονικών δεδομένων. Στόχος είναι η δημιουργία πινάκων αφετηρίας-προορισμού (origin-destination matrices) μέσω της χρήσης του mapreduce προγραμματιστικού παραδείγματος (programming paradigm) και του εργαλείου Hadoop. Τέλος, θα πραγματοποιηθεί η εγκατάσταση και ενεργοποίηση του παραπάνω αλγορίθμου σε συστάδα Η/Υ (cluster) του εργαστηρίου infoLab για την παραγωγή αποτελεσμάτων εκτέλεσης και την αξιολόγησή τους.

Πιο συγκεκριμένα, δίδονται αρχεία δειγμάτων δρομολογίων μέσων μεταφοράς με συντεταγμένες σε πολική μορφή **(geospatial data)** από τους χρήστες, αταξινόμητα με απλή μορφοποίηση. Η πρώτη συμβολή της εργασίας μας έγκειται στην ορθή αναπαράσταση των δρομολογίων με βάση βέλτιστο πλέγμα κατηγοριοποίησης χώρου και την συνεπακόλουθη δημιουργία των κατάλληλων αντίστοιχων μητρώων προέλευσης προορισμού **(O-D matrices)**.

Η δεύτερη συμβολή της εργασίας μας, σχετίζεται με την κατάλληλη εισαγωγή της τεχνολογίας mapreduce στην επεξεργασία των ανωτέρω O-D matrices. Πιο συγκεκριμένα, ακολουθείται μία διαδικασία τριών σταδίων κατά την οποία ξεκινώντας από χάρτες **(mappers)** είτε από διαδρομές, ανά κελί του πλέγματος, η κατάληξη είναι η δημιουργία reduced O-D matrices που περιλαμβάνουν όλη την απαραίτητη πληροφορία προς περαιτέρω επεξεργασία από κατανεμημένο σύστημα χρησιμοποιώντας την τεχνολογία Hadoop.

Abstract

The purpose of the present work is the creation of an experimental distributed algorithm for the process of large volume of space-time data. The main goal is to generate origin-destination matrices through the utilization of the mapreduce programming paradigm and the Hadoop program. The ultimate objective is the installation and activation of the above algorithm in a small computer cluster of the infoLab laboratory so as, to produce performance results and evaluate them.

More specifically transport route coordinates samples, are given in polar form (geospatial data) from users, unsorted in raw format in large-scale text files. The first contribution of our work is the correct representation of the trajectories based on optimum spatiotemporal grid and the subsequent creation of the appropriate origin destination **(O-D)** matrices.

The second contribution of our work is associated with the proper introduction of mapreduce technology to the management of the above O-D matrices. More specifically, a three-step procedure is followed in which, starting from maps (1 mapper per cell) or trajectories (1 trajectory per cell) of the grid; the outcome is the creation of reduced O-D matrices that include all the necessary information for further processing by a distributed system using Hadoop technology.

ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ

Εισαγωγή

Εκατομμύρια ενεργοποιημένες GPS συσκευές χρησιμοποιούνται καθημερινά παράγοντας ένα τεράστιο ποσό δεδομένων και ειδικότερα δεδομένων που αφορούν την κινητικότητα. Εξαιτίας αυτού του γεγονότος, αρκετές προκλήσεις προκύπτουν σχετικά με τη διαχείριση και αξιοποίηση αυτού του τύπου των δεδομένων. Αυτό αποκτά ιδιαίτερη σημασία στην εποχή μας, όπου λέξεις όπως “big data” και ”cloud computing” έχουν εισχωρήσει στην καθημερινότητα των επιχειρήσεων. Η χρήση των δεδομένων αυτών στην πρόβλεψη κίνησης μέσων μεταφοράς (**Transportation forecasting**) κάτω από την αιγίδα των big data έχει συγκεντρώσει αρκετό ερευνητικό ενδιαφέρον ιδιαίτερα στο κομμάτι της βελτιστοποίησης.

1.1 Μοντέλο πρόβλεψης κίνησης μέσων μεταφοράς (Transportation Forecasting)

Το Transportation forecasting αποτελεί μία προσέγγιση της εκτίμησης του αριθμού των οχημάτων (ή ανθρώπων) τα οποία θα χρησιμοποιήσουν ένα από τα υπάρχοντα συστήματα μεταφοράς στη διάρκεια του χρόνου. Για παράδειγμα, μια πρόβλεψη μπορεί να προϋπολογίσει τον αριθμό των οχημάτων που θα περάσουν από ένα υπό-κατασκευή δρόμο, ή τον αριθμό των επιβατών που επισκέπτονται ένα αεροδρόμιο, ή τον αριθμό των πλοίων που καταπλέουν σε ένα λιμάνι.

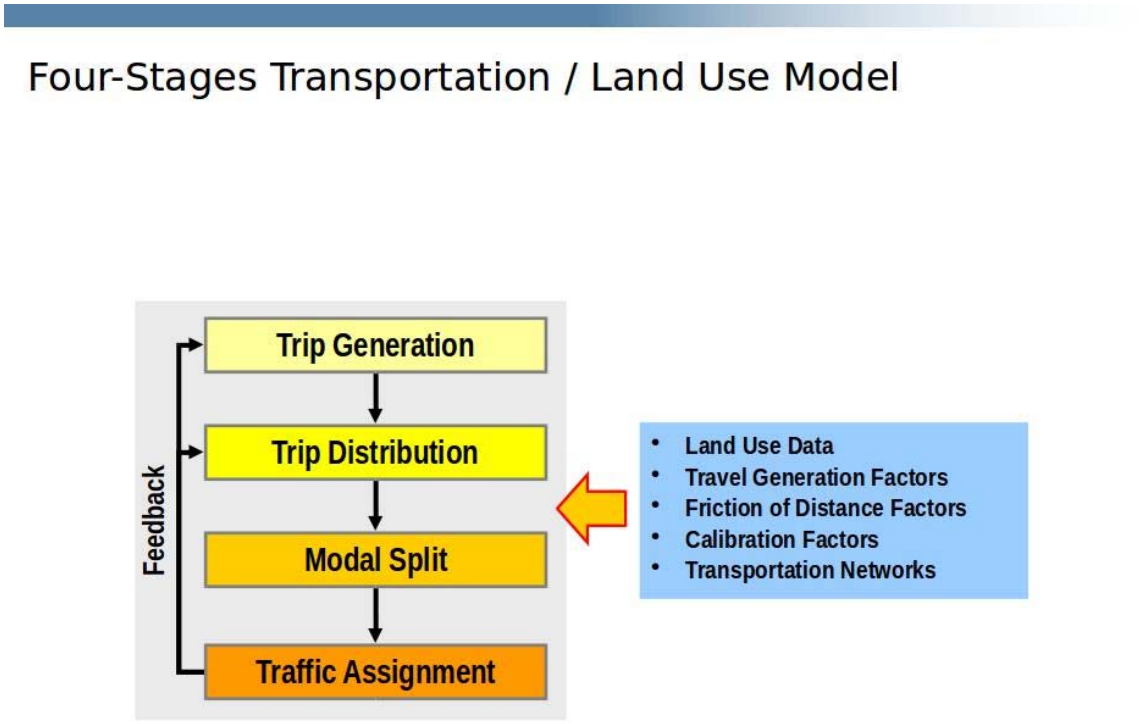
Η πρόβλεψη της κυκλοφορίας ξεκινά με τη συλλογή δεδομένων σχετικά με την τρέχουσα κατάσταση. Αυτά τα δεδομένα κίνησης συνδυάζονται με άλλα γνωστά στοιχεία, όπως είναι ο πληθυσμός, η μετακίνηση για εργασία, τα ποσοστά κίνησης μεταξύ περιοχών κλπ. Στόχος, είναι να αναπτυχθεί ένα μοντέλο ζήτησης κυκλοφορίας (**Traffic demand model**) για την τρέχουσα κατάσταση. Το μοντέλο αυτό στη συνέχεια τροφοδοτείται με τα δεδομένα πρόβλεψης (υποθετικά μελλοντικά δεδομένα) αναφορικά με τον πληθυσμό, την μετακίνηση για εργασία, κλπ. για κάθε τμήμα της υπό ανάλυσης υποδομής. Οι προβλέψεις αυτές, χρησιμοποιούνται σε αρκετούς βασικούς σκοπούς στον τομέα των μεταφορών όπως είναι: ο σχεδιασμός, ο υπολογισμός της χωρητικότητας της υποδομής και ο υπολογισμός των περιβαλλοντικών επιπτώσεων.

Εντός του ορθολογικού πλαισίου σχεδιασμού (Rational planning framework), το traffic forecasting παραδοσιακά ακολουθεί το σειριακό μοντέλο τεσσάρων βημάτων (**Sequential four-step model**) ή τη διαδικασία πολεοδομικού σχεδιασμού μεταφορών (**Urban Transportation Planning - UTP**).

1.2 Μοντέλο τεσσάρων βημάτων (Four-step model)

Τα βήματα του κλασικού four-step model του συστήματος UTP είναι:

- Παραγωγή ταξιδιού (**Trip generation**) κατά το οποίο καθορίζεται η συχνότητα της αρχής ή του τέλους των διαφόρων προορισμών σε κάθε ζώνη. Ο καθορισμός αυτός γίνεται βάση της αιτίας για την οποία πραγματοποιήθηκε το ταξίδι, τα δημογραφικά στοιχεία καθώς και άλλους κοινωνικό-οικονομικούς παράγοντες.
- Κατανομή ταξιδιού (**Trip distribution**) κατά το οποίο γίνεται η σύνδεση των αφετηριών (origins) με τους προορισμούς (destinations), συχνά χρησιμοποιώντας στατιστικά μοντέλα , τα οποία μεγιστοποιούν την εντροπία.
- Επιλογή μέσου (**Mode choice**) κατά το οποίο υπολογίζεται η αναλογία των ταξιδιών μεταξύ προέλευσης και προορισμού που χρησιμοποιούν ένα συγκεκριμένο μέσο μεταφοράς.
- Ανάθεση διαδρομής (**Route assignment**) κατά την οποία κατανέμονται τα ταξίδια μεταξύ προέλευσης και προορισμού, με συγκεκριμένο μέσο, σε μια διαδρομή.



Εικόνα 1. Τα 4 βήματα του κλασσικού four-stages μοντέλου για τη δημιουργία traffic forecasting [1].

1.2.1 Κατανομή ταξιδιού (Trip distribution)

Η Κατανομή ταξιδιού (**Trip distribution**) αποτελεί τη δεύτερη συνιστώσα του παραδοσιακού four-step μοντέλου πρόβλεψης. Κατά τη διάρκεια του βήματος αυτού, όλες οι αφετηρίες και οι προορισμοί ταιριάζονται έτσι ώστε να δημιουργηθεί ένας ενιαίος “πίνακας ταξιδιού”, μια μήτρα που εμφανίζει τον αριθμό των ταξιδιών από την αφετηρία τους προς τον προορισμός τους. Ο πίνακας αυτός ονομάζεται πίνακας αφετηρίας-προορισμού (Origin-Destination matrix).

Ορίζεται:

- Ως S ένα σύνολο περιοχών, με $S \in [1, N]$,
- Ως L ένα σύνολο χρονικών διαστημάτων, με $L \in [1, M]$
- Ως T ένα σύνολο τροχιών, με $T = \{T_1, T_2, .., T_L\}$.

Για κάθε ζεύγος i και j, με $[i, j] \in S \times S$ και $t \in L$ με "i" ≠ "j", θέλουμε να προσδιορίσουν ποιες διαδρομές (και, ως εκ τούτου, τον αριθμό τους) ξεκίνησαν από i και κατέληξαν στο j. Με τον τρόπο αυτό δημιουργείται ο ακόλουθος OD(t) (N x M) πίνακας.

From\To →	1	2	3	4	5
1	—	30	35	40	15
2	10	—	15	12	10
3	50	40	—	35	20
4	25	30	35	—	40
5	45	30	35	40	—

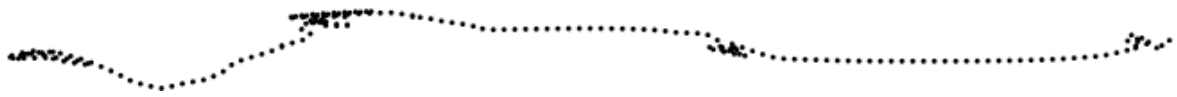
Εικόνα 2. Origin/Destination matrix, όπου: T_{ij} = πλήθος διαδρομών από τον τόπο προέλευσης *i* στον τόπο προορισμού *j*[2].

1.3 Ορισμός του Προβλήματος (Problem Definition)

Όπως προαναφέρθηκε, τα OD matrices χρησιμοποιούνται από εμπειρογνώμονες και επιστήμονες του κλάδου των μεταφορών & υποδομών, προκειμένου να βοηθήσουν στην καλύτερη πολεοδομική (και μη) σχεδίαση των διαδρομών και υποδομών.
Η δημιουργία OD matrix είναι μία δύσκολη (και συχνά δαπανηρή διεργασία) καθώς ακόμα και σήμερα, παράγονται από άμεσες μετρήσεις/συνεντεύξεις ή επιτόπιες έρευνες και χρειάζονται αρκετό χρόνο συλλογής δεδομένων, έτσι ώστε να θεωρηθούν αξιόπιστες. Παρόλα αυτά, συχνά παύουν να είναι ενήμερες (ή και χρήσιμες) καθώς τα μοτίβα κίνησης αλλάζουν γρήγορα λόγω τυχαίων γεγονότων.
Λόγω των παραπάνω, ένας εναλλακτικός τρόπος αρχίζει να έρχεται στην επιφάνεια, χάρις στον οποίο μπορεί κανείς να αποκτήσει μια πραγματική εκτίμηση. Αυτό περιλαμβάνει τη χρήση των παραγόμενων δεδομένων τροχιάς από τις εκατομμύρια GPS (**Global Positioning System**) συσκευές οι οποίες βρίσκονται ενσωματωμένες είτε σε κινητά τηλέφωνα είτε στα χρησιμοποιούμενα οχήματα.
Εκατομμύρια ενεργοποιημένες GPS συσκευές χρησιμοποιούνται καθημερινά, παράγοντας ένα τεράστιο ποσό δεδομένων κίνησης. Αρκετές προκλήσεις προκύπτουν σχετικά με τη διαχείριση και αξιοποίηση αυτού του τύπου των δεδομένων, στην εποχή των μεγάλων δεδομένων (**Big Data**), προκειμένου να χρησιμοποιηθούν στον υπολογισμό των OD matrices (άλλα και άλλων πολλών στατιστικών στοιχείων που θα μπορούσαν να βοηθήσουν τον σχεδιασμό των μεταφορών).

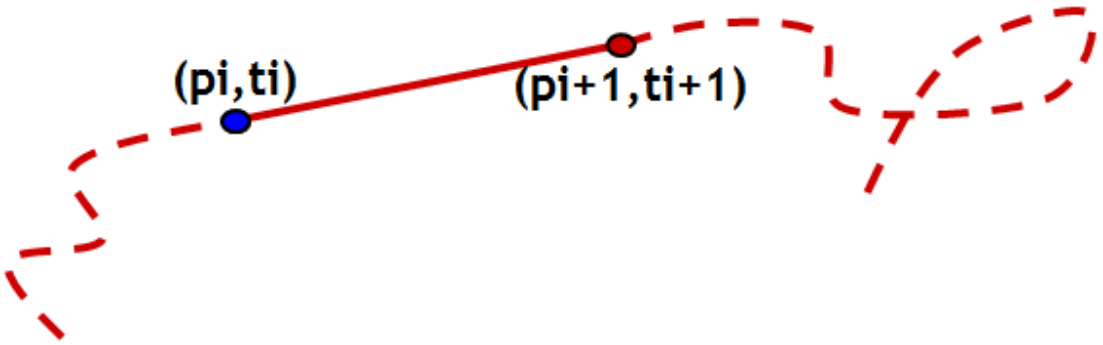
1.3.1 Βασικοί ορισμοί

Μια τροχιά (trajectory) (ή ίχνος μονοπατιού) είναι η διαδρομή που ακολουθεί ένα κινούμενο αντικείμενο μέσα στο χώρο ως μια συνάρτηση του χρόνου. Τα ανεπεξέργαστα δεδομένα τροχιάς, είναι μια ακολουθία δεδομένων χαμηλού επιπέδου (x_i, y_i, t_i σημείων) διατεταγμένα στο χρόνο.



Εικόνα 3. Σημεία μίας τροχιάς στο χώρο και στο χρόνο.

Λόγω του ότι πάντα υπάρχει δειγματοληψία μεταξύ των σημείων, χρησιμοποιείται η προσέγγιση της γραμμικής παρεμβολής (π. χ. μεταξύ των σημείων (p_i, t_i) και (p_i+1, t_i+1) – βλ. Εικόνα 4).



Εικόνα 4. Παράδειγμα γραμμικής παρεμβολής.

Ένας οποιοδήποτε OD πίνακας θα μπορούσε να εσωκλείει:

- Μόνο τα σημεία εκκίνησης και τερματισμού
- Όλες τις περιοχές που μια χωροχρονική τροχιά επικάλυψε κατά τη διάρκεια της πορείας της
- Τα αντικείμενα που ξεκίνησαν από μια συγκεκριμένη περιοχή και κατέληξαν σε άλλη (ή απλά το πλήθος τους)

Τα mobility δεδομένα πρέπει να συλλεχθούν και να κατανεμηθούν κατάλληλα, έτσι ώστε να δημιουργηθούν τα επιθυμητά OD matrices. Έτσι, έχουμε δύο βασικά στάδια, αυτό της συλλογής το λεγόμενο input και αυτό της επεξεργασμένης εξόδου output.

1.3.1.1 Βασική προσέγγιση δημιουργίας μητρώου OD (Trajectory count)

Η βασική προσέγγιση θα ήταν να προσδιοριστεί, ποιες τροχιές έχουν την εκκίνηση τους και τον τερματισμό τους για κάθε χωροχρονική περιοχή. Με τον τρόπο αυτό, θα δημιουργηθούν 2 διανύσματα για κάθε χωροχρονική περιοχή:

- Ένα το οποίο θα περιέχει τις ταυτότητες των τροχιών που έχουν ως αφετηρία τους το χώρο *i*, και
- Ένα το οποίο θα περιέχει τις ταυτότητες των τροχιών που έχουν ως σημείο τερματισμού το χώρο *j*.

Τέλος, με την επιμέρους άθροιση μπορούμε να κατασκευάσουμε την πρώτη μορφή, OD trajectory count:

Πίνακας 1. $N_{obj}[I,j]$: Πόσες τροχιές ξεκίνησαν από την περιοχή X και τέλειωσαν στην περιοχή Y.

	A	B	C	D
A		$N_{obj}[A,B]$	$N_{obj}[A,C]$	
B				
C				
D				

1.3.1.2 1η Βελτιωμένη προσέγγιση δημιουργίας διαδρομής OD (OD trajectory IDs)

Μία πρώτη βελτίωση θα ήταν να συμπεριλάβουμε εκτός από τον συνολικό αριθμό και το ποιες είναι αυτές οι τροχιές για κάθε χωροχρονική περιοχή:

Πίνακας 2. $N_{obj}[I,j]$: Πόσες και ποιες τροχιές ξεκίνησαν από την περιοχή I και τελείωσαν στην περιοχή J.

	A		B	C	D
A			$N_{obj}[A,B]$ (1, 2, 3, 4)		
B				$N_{obj}[B,C]$ (6, 8, 10, 12)	
C					
D					

1.3.1.3 2η Βελτιωμένη προσέγγιση δημιουργίας διαδρομών (All regions trajectory count)

Μία δεύτερη βελτίωση θα ήταν να συμπεριλάβουμε τα επιμέρους τμήματα, από τα οποία πέρασαν οι τροχιές (δηλαδή όλα τα ενδιάμεσα κελιά):

Πίνακας 3. $N_{obj}[I,j]$: Πόσες τροχιές έφτασαν στην περιοχή J από την περιοχή I μέσω της περιοχής Z.

	A	B	C	D
A		$N_{obj}[A,B]$		
B			$N_{obj}[B,C]$	
C				$N_{obj}[C,D]$
D				

1.3.1.4 3η Βελτιωμένη προσέγγιση δημιουργίας διαδρομών (All regions trajectory IDs)

Μία τρίτη βελτίωση θα ήταν να συμπεριλάβουμε τα επιμέρους τμήματα, από τα οποία πέρασαν οι τροχιές που ξεκίνησαν από μία περιοχή i και έφτασαν σε μία περιοχή j, ποιες είναι οι τροχιές αυτές και πόσες είναι:

Πίνακας 4. $N_{obj}[I,j]$: Πόσες και ποιες τροχιές έφτασαν στην περιοχή J από την περιοχή I .

	A	B	C	D
A		$N_{obj}[A,B]$ (1, 2, 3, 4)		
B			$N_{obj}[B,C]$ (2, 3, 4)	
C	$N_{obj}[C,A]$ (2, 4)			
D				

Στόχος της παρούσας εργασίας, είναι ο υπολογισμός και παραγωγή των προαναφερθέντων OD matrices με τη χρήση του MapReduce programming paradigm. Τα κεφάλαια που ακολουθούν, περιέχουν την εξής δομή:

- Κεφάλαιο 2: Στο κεφάλαιο αυτό πραγματοποιείται η ανάλυση του κατανεμημένου αλγορίθμου που δημιουργήθηκε για την παραγωγή των OD matrices.
- Κεφάλαιο 3: Στο κεφάλαιο αυτό γίνεται μία σύντομη βιβλιογραφική ανασκόπηση στο Hadoop framework και κατ' επέκταση στο Mapreduce.
- Κεφάλαιο 4: Στο κεφάλαιο γίνεται παρουσίαση και ανάλυση των αποτελεσμάτων για την εκτέλεση του αλγορίθμου σε πειραματικό cluster, το οποίο συστάθηκε για το σκοπό αυτό.

Κεφάλαιο 2ο

Απόδοση Αλγορίθμου MapReduce “ODMatrixCalc”

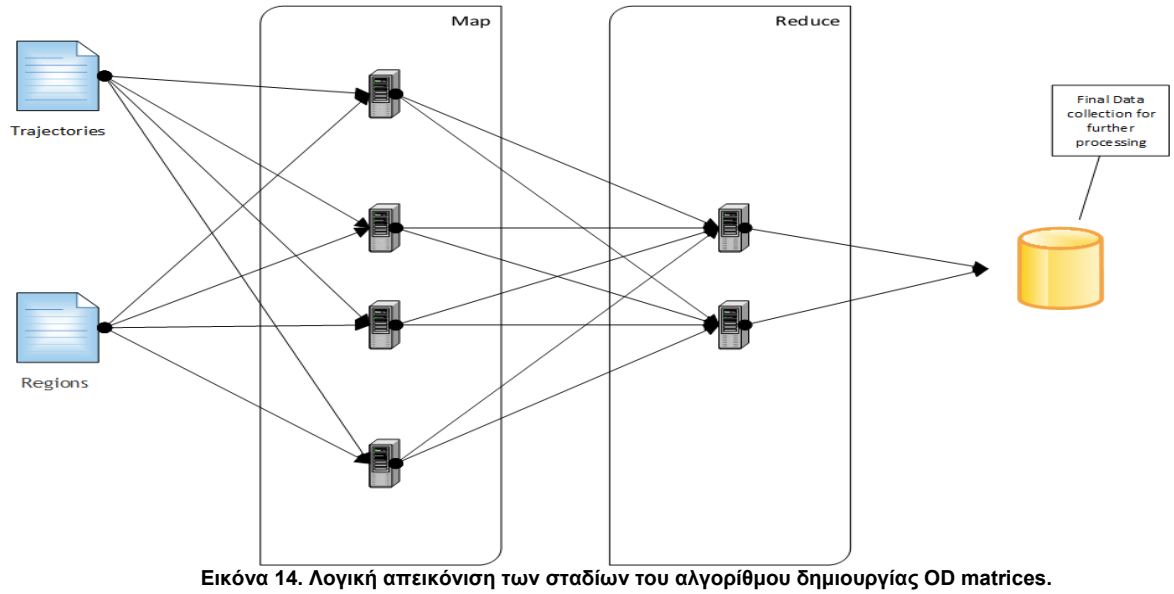
Όπως αναφέρθηκε στο προηγούμενο κεφάλαιο, λόγω του τεράστιου όγκου των δεδομένων κίνησης (που είναι διαθέσιμα από τις GPS συσκευές), γεννήθηκε η ιδέα του να χρησιμοποιηθούν (τα δεδομένα αυτά) στο στάδιο παραγωγής των OD matrices. Σε αντίθεση με την παλαιά μέθοδο των συνεντεύξεων και της επιτόπιας έρευνας, τα δεδομένα αυτά είναι πάντα επίκαιρα και με την κατάλληλη υποδομή μπορούν να χρησιμοποιηθούν για την συνεχή δημιουργία OD matrices οι οποίοι, με τη σειρά τους, θα χρησιμοποιηθούν από το 4-step μοντέλο πρόβλεψης έτσι ώστε τα τελικά στοιχεία να αντικατοπτρίζουν πάντα την “πραγματική εικόνα”.

Το Hadoop αποτελεί το εργαλείο που επιλέχτηκε για την επεξεργασία των αρχικών δεδομένων και την τελική μορφοποίηση τους σε OD matrices, λόγω του ότι:

- Το MapReduce framework, το οποίο αποτελεί τμήμα του Hadoop, παρέχει τα κατάλληλα εργαλεία για τη εκμετάλλευση του τεράστιου αυτού όγκου δεδομένων κινητικότητας (ικανοποίηση των συνθηκών volume & variety) και
- Για το γεγονός ότι το παράθυρο μεταβολής των παραπάνω δεδομένων (data velocity) είναι σχετικά μεγάλο.

Ο αλγόριθμος αποτελείται από τρία διακριτά στάδια:

1. Είσοδος (Input)
2. Επεξεργασία (Processing)
3. Έξοδος (Output)

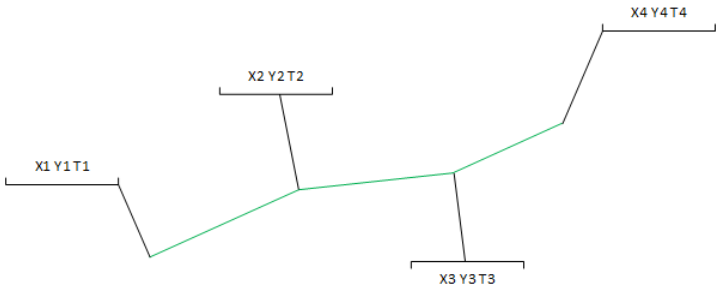


Εικόνα 14. Λογική απεικόνιση των σταδίων του αλγορίθμου δημιουργίας OD matrices.

Στις επόμενες παραγράφους γίνεται αναλυτική επισκόπηση του αλγορίθμου βάσει του λογικού διαχωρισμού των τμημάτων του αλγορίθμου που προαναφέρθηκαν.

2.1 Είσοδος (Input)

Η είσοδος αποτελεί το πρώτο λογικό τμήμα του αλγορίθμου και είναι σχεδιασμένη έτσι ώστε να δέχεται ένα σύνολο από χωροχρονικά δεδομένα [3D trajectories - xi, yi, ti] (**Εικόνα 15**) και αυτόματα να δημιουργεί τον ελάχιστο "κύβο" που τα περικλείει.



Εικόνα 15. Απεικόνιση ενός trajectory στο χρόνο.

Η εισαγωγή των trajectories πραγματοποιείται από την *DBInputCreator* και μπορεί να γίνει:

- Από txt αρχείο προς μία σχεσιακή βάση δεδομένων (**RDBMS**, π. χ. mySQL) και στην συνέχεια να δοθούν ως είσοδο στο πρόγραμμα ή
- Απευθείας από txt αρχείο

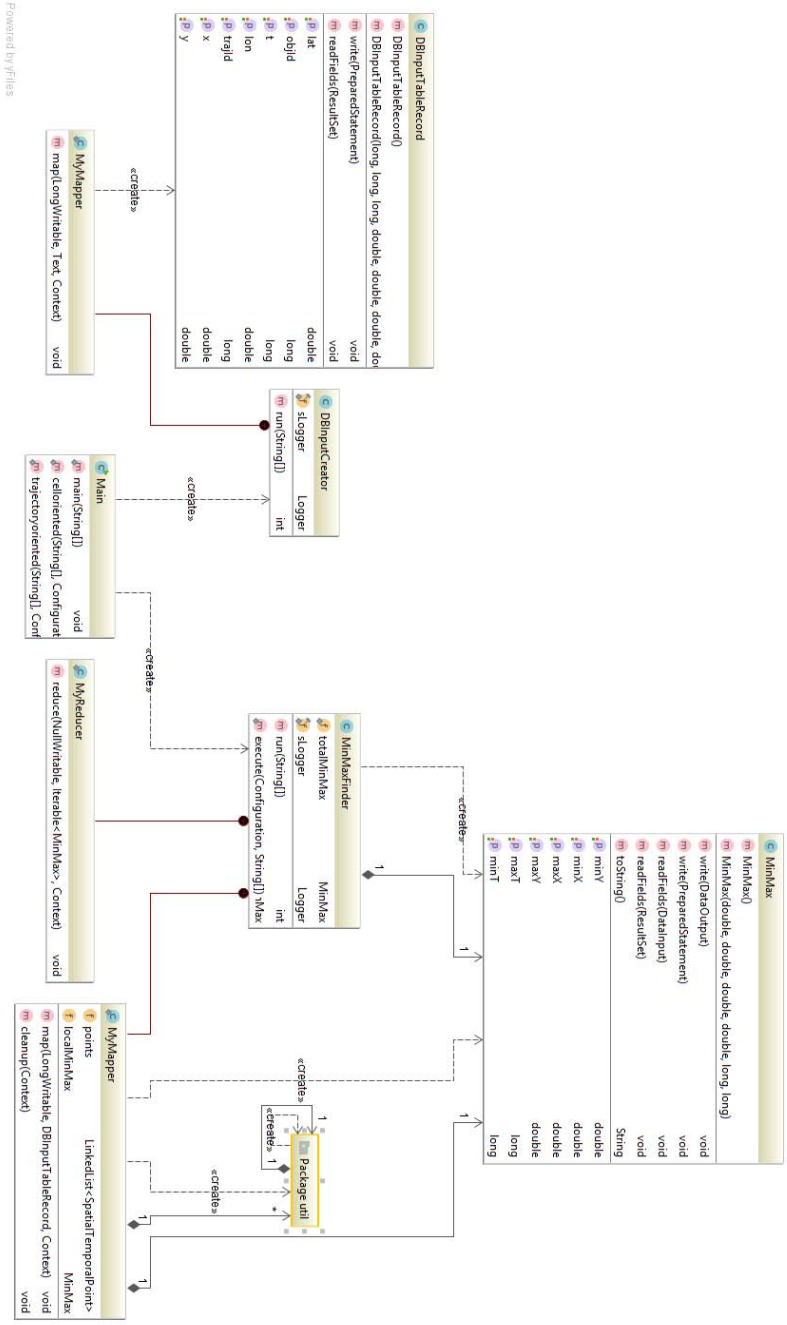
Η κλάση αυτή χρησιμοποιεί την *DBInputTableRecord* ως αντικείμενο το οποίο αντικατοπτρίζει ένα οποιοδήποτε trajectory record μέσα στο RDBMS.
Η *DBInputTableRecord* αποτελείται από τις παρακάτω μεταβλητές:

- **objId**, από το object id: Το μοναδικό αναγνωριστικό του κάθε κινούμενου αντικειμένου (χρησιμοποιείται από το RDBMS).
- **trajId**, από το trajectory id: Το μοναδικό αναγνωριστικό της κάθε τροχιάς.
- **t**, από το time: Το αναγνωριστικό που αφορά το χρόνο που πάρθηκε η συγκεκριμένη μέτρηση.
- **lon**, από το longitude: Το αναγνωριστικό που αφορά το γεωγραφικό μήκος που πάρθηκε η συγκεκριμένη μέτρηση.
- **lat**, από το latitude: Το αναγνωριστικό που αφορά το γεωγραφικό πλάτος που πάρθηκε η συγκεκριμένη μέτρηση.
- **x**, από το καρτεσιανό x: Προβολή του lon σε καρτεσιανό σύστημα.
- **y**, από το καρτεσιανό y: Προβολή του lat σε καρτεσιανό σύστημα.

Στη συνέχεια, όλα τα trajectories, περνάνε μέσα από την *MinMaxFinder* η οποία είναι υπεύθυνη να δημιουργήσει ένα αντικείμενο τύπου *MinMax* το οποίο περιέχει τη μέγιστη και ελάχιστη τιμή από όλα τα χρόνο-σημεία των trajectories που καταχωρήθηκαν στη βάση. Η MinMax με τη σειρά της, χρησιμοποιείται στη δημιουργία του ελάχιστου κύβου ο οποίος περικλείει όλα τα spatiotemporal δεδομένα.

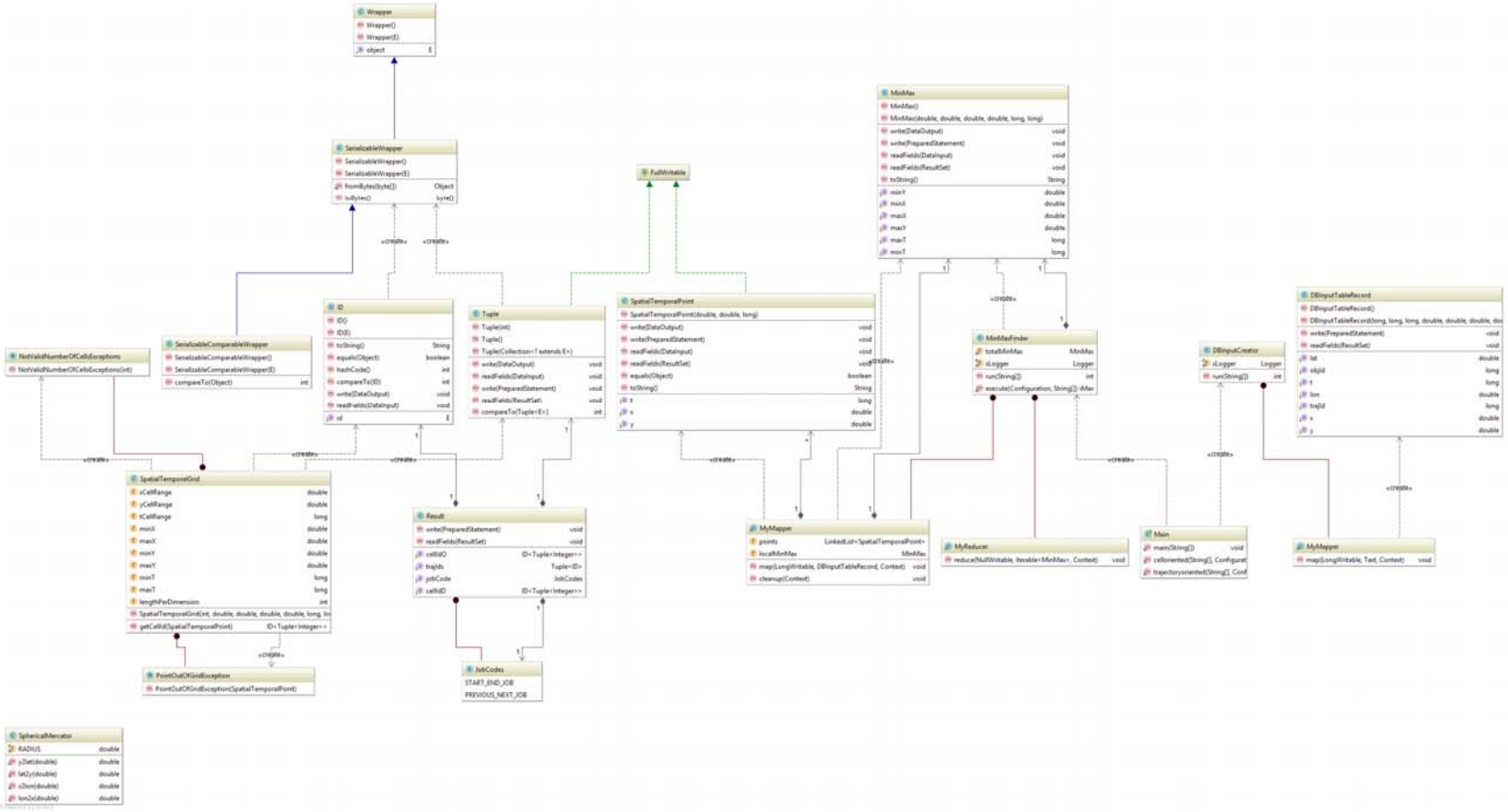
Ακολουθούν οι εικόνες 16_a & 16_b οι οποίες απεικονίζουν το UML διάγραμμα του λογικού τμήματος της εισόδου που χρησιμοποιούνται κατά τη δημιουργία του κύβου και των διαφόρων util τύπου κλάσεων οι οποίες χρησιμοποιούνται από όλες τις υπόλοιπες κλάσεις (σε όλα τα στάδια του αλγορίθμου):

Η Εικόνα 16_a περιέχει τις συνδέσεις που υπάρχουν μεταξύ της DBInputTableRecord και της MinMaxFinder για την δημιουργία του ελάχιστου κύβου ο οποίος περικλείει όλα τα δεδομένα.



Εικόνα 16_a. UML διάγραμμα περιγραφής της εισόδου των δεδομένων και εύρεσης μέγιστων & ελαχίστων.

Η Εικόνα 16_b περιέχει τις συνδέσεις που υπάρχουν μεταξύ των κλάσεων που βρίσκονται μέσα στο util package. Οι κλάσεις αυτές χρησιμοποιούνται από όλες τις υπόλοιπες κλάσεις (και κατ' επέκταση τα στάδια του προγράμματος) και περιέχουν μεθόδους και τύπους για την παραγωγή του τελικού αποτελέσματος.



Εικόνα 16_b. UML διάγραμμα των κλάσεων στο util package.

2.2 Αλγόριθμοι υπολογισμού πινάκων αφετηρίας/προορισμού

Αφού ολοκληρωθεί η είσοδος των spatiotemporal δεδομένων και παραχθεί το MinMax αντικείμενο, το επόμενο λογικό βήμα που ακολουθεί είναι αυτό της επεξεργασίας των δεδομένων. Το τμήμα αυτό αποτελείται από δύο προσεγγίσεις, αναφορικά με τον τρόπο αναπαράστασης/απόδοσης των δεδομένων στους mappers:

- **Cell oriented** προσέγγιση: κατά την οποία όλες οι τροχιές αντιμετωπίζονται ως τμήματα (segments) μέσα σε μία περιοχή (cell).
    - Σε αυτή τη περίπτωση κάθε cell περιέχει πολλά segments, τα οποία μπορεί να μην ανήκουν στο ίδιο αντικείμενο (να έχουν το ίδιο trajectory id). Ο κάθε mapper αναλαμβάνει τουλάχιστον ένα cell και κατ' επέκταση τα segments που περιέχονται μέσα σε αυτό.
- **Trajectory oriented** προσέγγιση: κατά την οποία όλες οι τροχιές ομαδοποιούνται σε διανύσματα (**vectors**) του τύπου (x_i, y_i, t_i).
    - Σε αυτή τη περίπτωση κάθε cell περιέχει τμήμα του διανύσματος μίας τροχιάς (ή και πολλών), το οποίο ανήκει στο ίδιο object ή και trajectory id. Ο κάθε mapper αναλαμβάνει τουλάχιστον ένα trajectory.

Ο χρήστης κατά την εκκίνηση του προγράμματος (Main κλάση) έχει την δυνατότητα επιλογής της μεθόδου επεξεργασίας που θέλει να ακολουθηθεί από τον αλγόριθμο.

2.2.1 Προσέγγιση με βάση το κελί (Cell oriented)

Σε περίπτωση επιλογής του Cell oriented τρόπου, επιλέγονται, αρχικοποιούνται και τέλος εκτελούνται όλες οι κλάσεις που βρίσκονται στο org.lumi.odmatrixcalc.celloriented package. Κάθε μία από αυτές περιέχει τις αναγκαίες υποκλάσεις mapper & reducer στις οποίες δίνονται οι οδηγίες για την αλληλουχία τον υπολογισμών από το MapReduce framework.

Συνολικά πραγματοποιούνται τρία Mapreduce:

- ***Η κλάση FirstMapReduce***, η οποία πραγματοποιεί το πρώτο Mapreduce**:**
    a. Κατά την αρχικοποίηση της, χρησιμοποιούνται σαν ορίσματα τα αποτελέσματα του πρώτου σταδίου (δηλαδή οι μεταβλητές της *MinMax*) και ο επιθυμητός αριθμός των cell, ο οποίος ζητείται από το χρήστη στην αρχή του προγράμματος. Η κλάση *SpatialTemporalGrid* αρχικοποιείται από τις πληροφορίες αυτές και δημιουργεί τον ελάχιστο κύβο (διότι έχουμε τρεις διαστάσεις) ο οποίος περικλείει όλα τα δεδομένα.
    b. Στη συνέχεια πραγματοποιείται η map για κάθε cell, κατά την οποία κάθε *DBInputTableRecord* εγγραφή μεταφράζεται σε *SpatialTemporalPoint* object. Στη συνέχεια το SpatialTemporalPoint (είναι οι συντεταγμένες ενός σημείου μίας διαδρομής μαζί με τον χρόνο καταγραφής) μαζί με το TrajectoryId (το αναγνωριστικό διαδρομής) στο οποίο ανήκει, αντιστοιχίζονται στο cell (cell id) στο οποίο εμπεριέχεται. Με λίγα λόγια, η έξοδος του Mapper είναι <key; value> = <cellId; (trajId, point)>.
    c. Τέλος πραγματοποιείται η reduce, η οποία συλλέγει όλα τα ζεύγη (trajectoryID, point) που βρίσκονται στο ίδιο cell και δημιουργεί το τελικό (ολικό) key-value pair για κάθε cell και τα segments που πέρασαν από αυτό, δηλαδή ως Reducer παράγει ως έξοδο <key; value> = <cellId; (trajId, list(point))>.

Ακολουθεί ο ψευδοκώδικας της *FirstMapReduce*:

```
Map
//Pre-mapping stage
setup(Context) {
        insert values of MinMax(x, y, t) and numberOfCells to be used;
}

map(LongWritable, DBInputTableRecord, ID, Tuple) {
```

for each local spatio-temporal point find each trajectoryID and match it to cellID;
}

Reduce
reduce(ID, Tuple, ID, Tuple) {
for each spatio-temporal point match trajectoryID and cellID;
}

- ***Η κλάση SecondMapReduce***, η οποία πραγματοποιεί το δεύτερο Mapreduce:
  a. Η *SecondMapReduce* λαμβάνει ως είσοδο τα αποτελέσματα του reduce της *FirstMapReduce*.
  b. Στη συνέχεια εκτελείται το map όπου το <key value> = <cellId; (trajId, list(point))> μετατρέπεται σε <key value> = < trajId; (cellId, list(point))>. Σκοπός αυτής της μετατροπής είναι η συλλογή, στο reducer(), όλων των σημείων ενός trajectoryID, έχοντας προσδιορίσει σε κάθε σημείο το αναγνωριστικό του κελιού στο οποίο ανήκει (cellID).
  c. Τέλος εκτελείται το reduce όπου παράγει ως έξοδο ζευγάρια (cellO, cellD, jobCode) -> trajID που υποδηλώνουν ότι το trajectory με το συγκεκριμένο id, ανάλογα με τις τιμές που έχει πάρει το JobCode αντικείμενο, είτε μεταβεί από το κελί cellO στο κελί cellD, είτε έχει ξεκινήσει από το cellO και καταλήξει στο cellD.
     1. Το αντικείμενο JobCode είναι ένα τύπου enum αντικείμενο το οποίο παίρνει δύο διακριτές τιμές: “START_END_JOB” και “PREVIOUS_NEXT_JOB”. Αποτελεί βοηθητική μεταβλητή της κλάσης *Result* η οποία περιέχει τα τελικά αποτελέσματα.

Ακολουθεί ο ψευδοκώδικας της *SecondMapReduce*:

Map
map(ID, Tuple, ID, Tuple) {
emit (local trajectoryID, local List<cellID&Points>);
}

Reduce
reduce(ID, Tuple, Tuple, ID) {
Create the list of points of the local trajID and sort them by time;

Iterate over the list and calculate all the pairs (cellOriginID, CellDestinationID), where (cellOriginID, CellDestinationID) refers either to the begin-end of a trajectory or to a crossover between two cells. For a clear separation between the types of pairs use a code named JobCode;

Add cellOriginID, CellDestinationID & JobCode in tuple;
emit (cellOCellDAndJC, trajId);
}

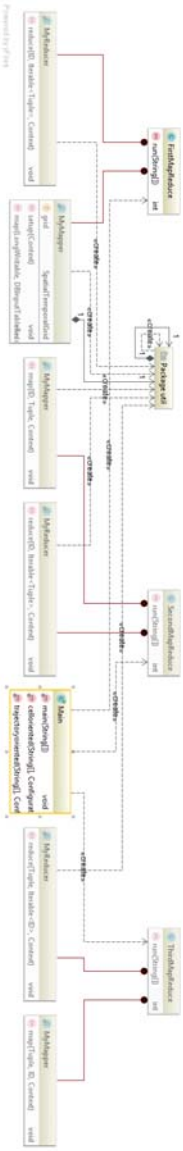
- ***Η κλάση ThirdMapReduce***, η οποία πραγματοποιεί το τρίτο Mapreduce:
  a. Η *ThirdMapReduce* λαμβάνει ως είσοδο τα αποτελέσματα του reduce της *SecondMapReduce.*
  b. Στη συνέχεια εκτελείται το map όπου το <key value> = <cellOCellDAndJC; trajId> μετατρέπεται σε <key value> = < trajId; (cellOCellDAndJC, trajId)>.
  c. Τέλος εκτελείται το reduce όπου παράγει ως έξοδο αντικείμενα τύπου Result που περιέχουν τα τελικά αποτελέσματα (δηλαδή τα OD matrices). Αξίζει να σημειωθεί ότι για κάθε trajectory δημιουργούνται δύο αντικείμενα τύπου Result ένα για κάθε JobCode.

Ακολουθεί ο ψευδοκώδικας της *ThirdMapReduce*:

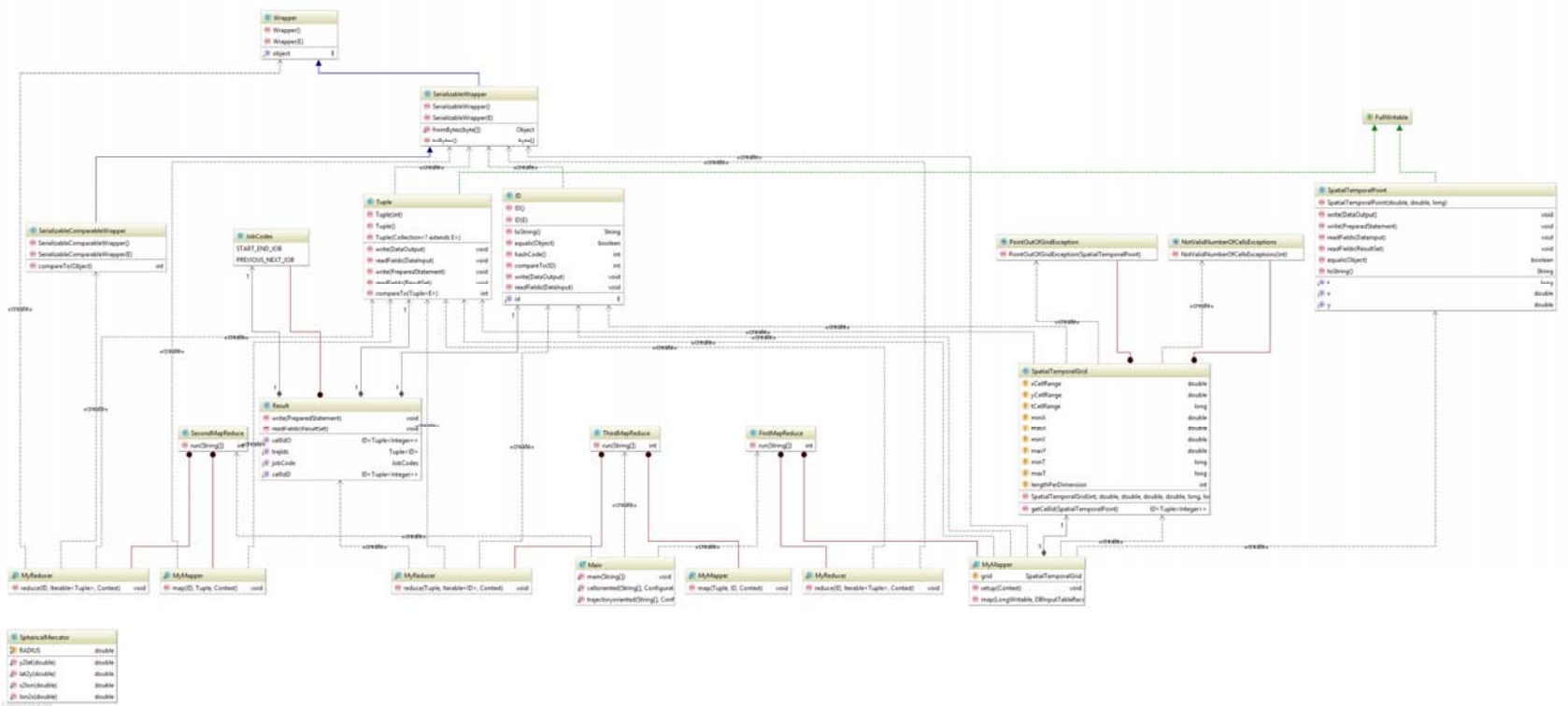
```
Map
map(Tuple, ID, Tuple, ID) {
        emit (local cellOCellIDAndJC, local trajId);
}

Reduce
reduce(Tuple, ID, Result, NullWritable) {
        emit (result);
}
```

Ακολουθούν οι εικόνες 17_a & 17_b οι οποίες απεικονίζουν το UML διάγραμμα του λογικού τμήματος της επεξεργασίας βάσει της Cell oriented προσέγγισης:



Εικόνα 17_a. UML διάγραμμα περιγραφής της επεξεργασίας των δεδομένων για την cell oriented προσέγγιση.



Εικόνα 17_b. UML διάγραμμα περιγραφής της επεξεργασίας των δεδομένων για την cell oriented προσέγγιση.

2.2.2 Προσέγγιση με βάση τη διαδρομή (Trajectory oriented)

Σε περίπτωση επιλογής του trajectory oriented τρόπου, επιλέγονται, αρχικοποιούνται και τέλος εκτελούνται όλες οι κλάσεις που βρίσκονται στο org.lumi.odmatrixcalc.vectororiented package. Κάθε μία από αυτές περιέχει τις αναγκαίες υποκλάσεις mapper & reducer στις οποίες δίνονται οι οδηγίες για την αλληλουχία τον υπολογισμών από το MapReduce framework.

Συνολικά πραγματοποιούνται δύο Mapreduce:

- ***Η κλάση FirstMapReduce***, η οποία πραγματοποιεί το πρώτο Mapreduce**:**
  A. Κατά την αρχικοποίηση της, περνιούνται σαν ορίσματα τα αποτελέσματα του πρώτου σταδίου (δηλαδή οι μεταβλητές της *MinMax*) και ο επιθυμητός αριθμός των cell. Η κλάση *SpatialTemporalGrid* αρχικοποιείται από τις πληροφορίες αυτές και δημιουργεί τον ελάχιστο κύβο ο οποίος περικλείει όλα τα δεδομένα.
  B. Στη συνέχεια πραγματοποιείται η map για κάθε trajectory, και βρίσκονται τα points τα οποία ανήκουν σε αυτό (μαζί με το cell στο οποίο ανήκει το κάθε point). Δημιουργούνται δηλαδή αντιστοιχίσεις της μορφής <key value> = < trajId; (cellId, point) >.
  C. Τέλος εκτελείται το reduce όπου παράγει ως έξοδο ζευγάρια <key value> = < cellOCellDAndJC; trajId > που υποδηλώνουν ότι το trajectory με το συγκεκριμένο id έχει ,ανάλογα με το jobCode, είτε μεταβεί από το κελί cellO στο κελί cellD, είτε ξεκίνησε από το cellO και κατέληξε στο cellD.

Ακολουθεί ο ψευδοκώδικας της *FirstMapReduce*:

Map

```
//Pre-mapping stage
setup(Context) {
        insert values of MinMax(x, y, t) and numberOfCells to be used;
}

map(LongWritable, DBInputTableRecord, ID, Tuple) {
        for each local spacial-Temporal point find its cellID and match it to trajectoryID;
}
```

Reduce

```
reduce(ID, Tuple, ID, Tuple) {
        for each spacial-Temporal point match trajectoryID and cellID;
}
```

- ***Η κλάση SecondMapReduce***, λαμβάνει ως είσοδο τα αποτελέσματα του reduce της *FirstMapReduce και πραγματοποιεί* ένωση (join) αυτών (βάσει του *tuple* (cellO, cellD, jobCode)). Ως απότοκο αυτού, για κάθε ζεύγος cellO/cellD βρίσκονται όλα τα trajectories τα οποία είτε μετέβησαν από το cellO στο cellD είτε ξεκίνησαν από το cellO και κατέληξαν στο cellD. Η διαφοροποίηση μεταξύ των δύο αυτών τύπων αποτελεσμάτων πραγματοποιείται βάσει του jobCode. Τα αποτελέσματα κάθε reduce() αποθηκεύονται σε δύο (για κάθε ένα JobCode) ξεχωριστά αντικείμενα τύπου Result.

Ακολουθεί ο ψευδοκώδικας της *SecondMapReduce*:

Map

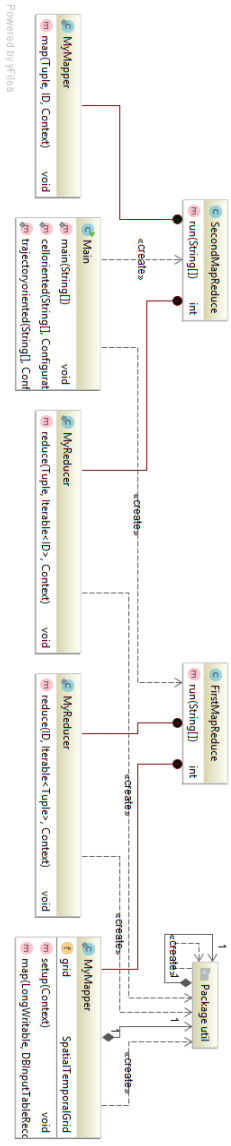
```
map (Tuple, ID, Tuple, ID) {
        emit (local trajectoryID, local List<cellID&Points>);
}
```

Reduce

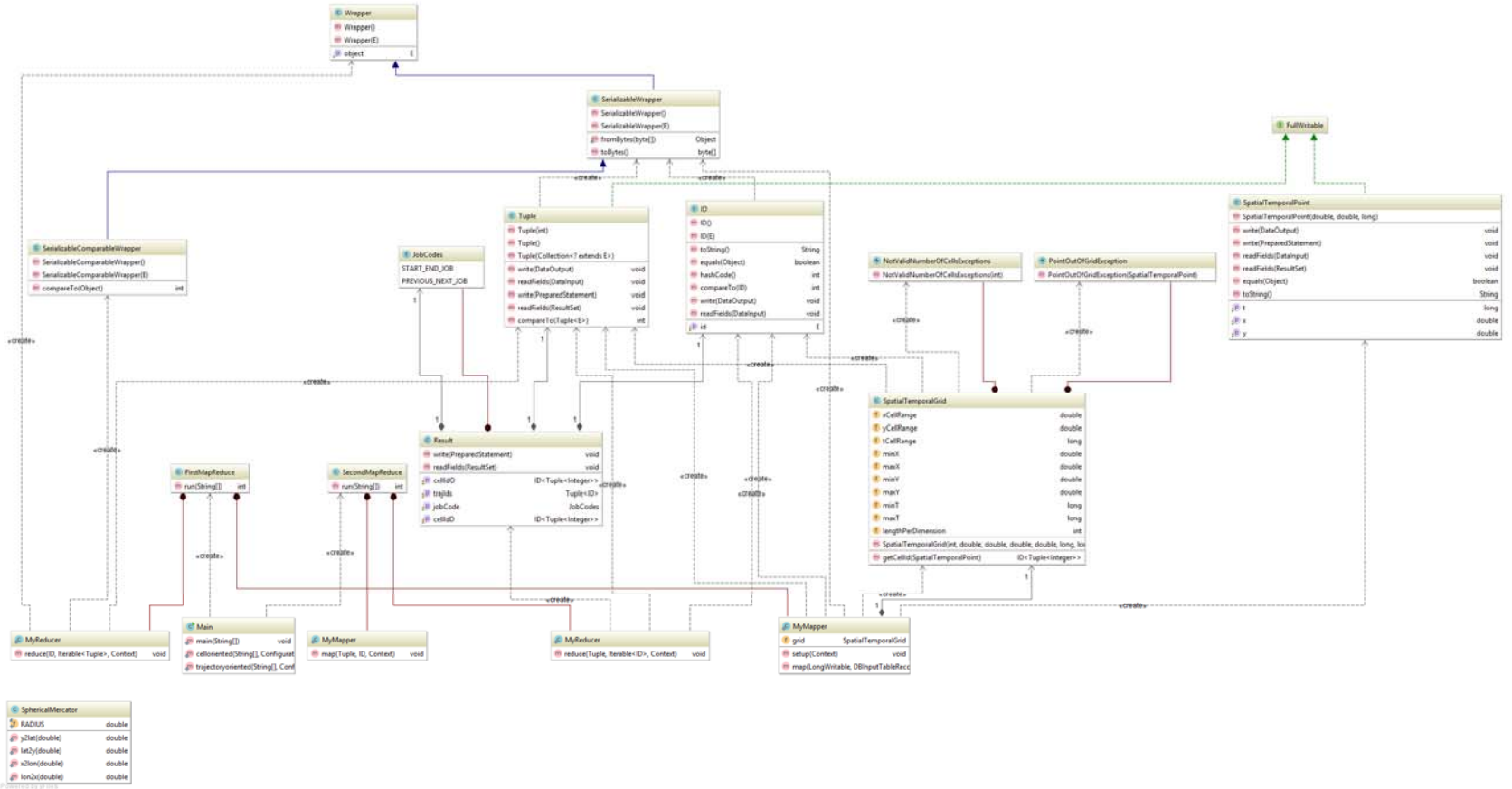
```
reduce(Tuple, ID, Result, NullWritable) {
```

```
            Add cellOriginID, CellDestinationID & JobCode in tuple;
            emit (cellOCellDAndJC, trajId);
    }
```

Ακολουθούν οι εικόνες 18_a & 1_b οι οποίες απεικονίζουν το UML διάγραμμα του λογικού τμήματος της επεξεργασίας βάση της trajectory oriented προσέγγισης:



Εικόνα 18_a. UML διάγραμμα περιγραφής της επεξεργασίας των δεδομένων για την trajectory oriented προσέγγιση.



Εικόνα 18_b. UML διάγραμμα περιγραφής της επεξεργασίας των δεδομένων για την trajectory oriented προσέγγιση.

Το προγραμματιστικό παράδειγμα του MapReduce

Το MapReduce αποτελεί ένα προγραμματιστικό παράδειγμα (**programming paradigm**) που σχεδιάστηκε για να επιτρέπει την παράλληλη, κατανεμημένη επεξεργασία μεγάλων συνόλων δεδομένων:

- Αρχικά, τη μετατροπή τους σε σύνολα από πλειάδες, και
- Στη συνέχεια συνδυάζοντας και μειώνοντας τις πλειάδες σε μικρότερα σύνολα.

Με άλλα λόγια, το MapReduce σχεδιάστηκε για να λάβει μεγάλους όγκους δεδομένων και χρησιμοποιώντας παράλληλη επεξεργασία, να μετατρέψει τα "μεγάλα δεδομένα" σε κανονικού μεγέθους δεδομένα.
Το MapReduce αποτελεί την καρδιά του Hadoop, καθώς είναι αυτό που επιτρέπει τη μαζική επεξεργασία δεδομένων σε εκατοντάδες ή χιλιάδες servers σε ένα σύμπλεγμα (cluster).

3.1 Διεργασίες απεικόνισης και μείωσης των δεδομένων (Map & Reduce tasks)

Οι MapReduce διεργασίες λειτουργούν μέσω των επιμέρους map & reduce υποδιεργασιών, σε ένα κατανεμημένο σύνολο από servers. Κατά τη διάρκεια του **map**, τα δεδομένα μετατρέπονται σε key-value pairs, φιλτράρονται και τέλος, εκχωρούνται στους κόμβους για επεξεργασία. Κατά τη διάρκεια του **reduce**, τα δεδομένα αυτά οδηγούνται σε μικρότερου μεγέθους σύνολα. Τα δεδομένα στο βήμα αυτό, μετασχηματίζονται σε μια τυποποιημένη key-value pair μορφή, όπου το key δρα ως το αναγνωριστικό εγγραφής και το value αποτελεί τη τιμή η οποία προσδιορίζεται από το key. Τέλος, οι υπολογιστικοί κόμβοι του συμπλέγματος διακομιστών, επεξεργάζονται τις map και reduce διεργασίες που έχουν οριστεί από το χρήστη.

Η όλη διεργασία πραγματοποιείται σύμφωνα με τα ακόλουθα δύο στάδια:

3.1.1 Απεικόνιση των δεδομένων (Map the data)

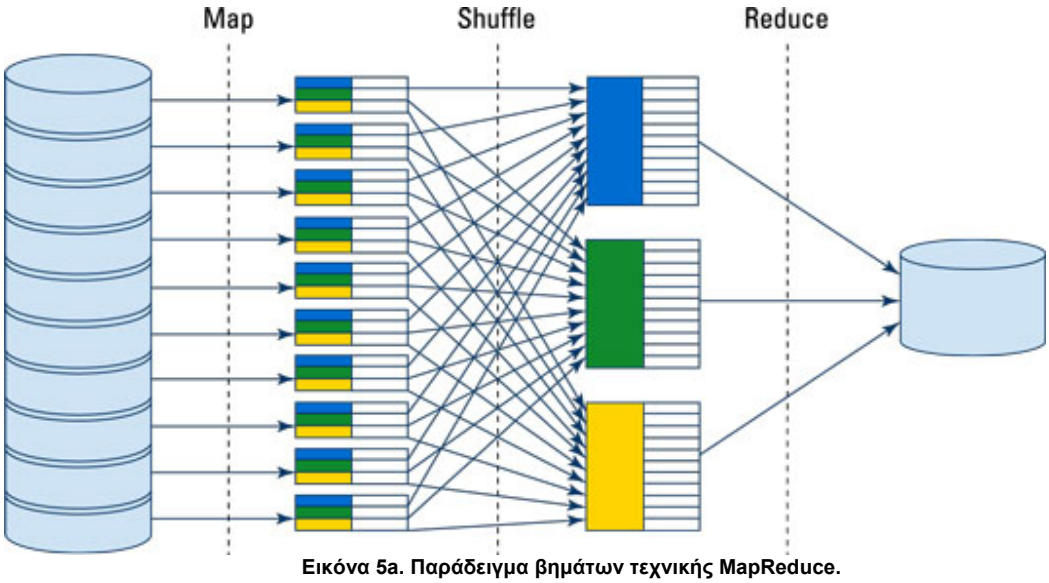
Τα εισερχόμενα δεδομένα πρέπει πρώτα να ανατεθούν σε key-value pairs και να διαχωριστούν σε τμήματα (**fragments**), τα οποία στη συνέχεια αποδίδονται σε map tasks. Σε κάθε συστοιχία υπολογιστών (**cluster**) εκχωρείται ένας αριθμός από maps, τα οποία κατανέμονται στη συνέχεια μεταξύ των κόμβων του δικτύου.

Κατά την επεξεργασία των αρχικών key-value pairs, δημιουργούνται ενδιάμεσα key-value pairs. Αυτά ταξινομούνται βάση των key values τους και στη συνέχεια, η καινούρια αυτή λίστα χωρίζεται σε ένα νέο σύνολο από fragments.

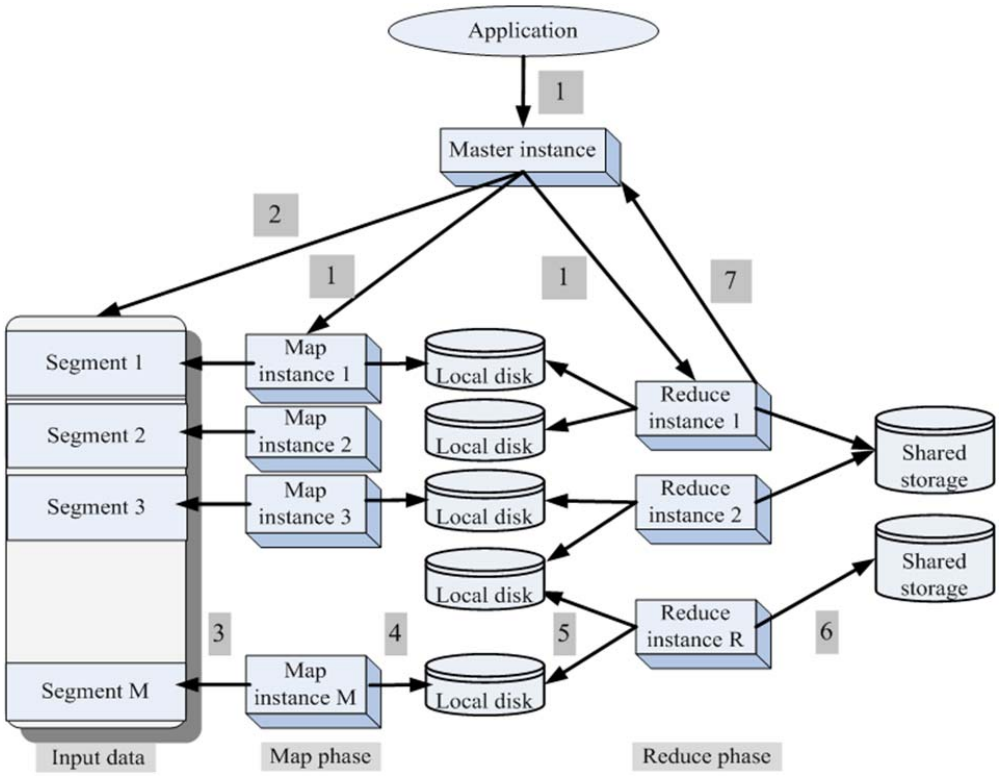
3.1.2 Μείωση των δεδομένων (Reduce the data)

Κάθε reduce διεργασία έχει ένα κομμάτι που της έχει ανατεθεί. Επί της ουσίας, το reduce task επεξεργάζεται το fragment και παράγει μία έξοδο, η οποία είναι επίσης ένα ζεύγος key-value. Τα reduce tasks κατανέμονται και αυτά με τη σειρά τους μεταξύ των διαφόρων κόμβων του cluster. Αφού ολοκληρωθεί η όλη διεργασία, η τελική έξοδος γράφεται επάνω στο σύστημα αρχείων του λειτουργικού

Ακολουθούν κάποια διαγράμματα των βημάτων της τεχνικής MapReduce:



Εικόνα 5a. Παράδειγμα βημάτων τεχνικής MapReduce.



Εικόνα 5b. Παράδειγμα βημάτων τεχνικής MapReduce.

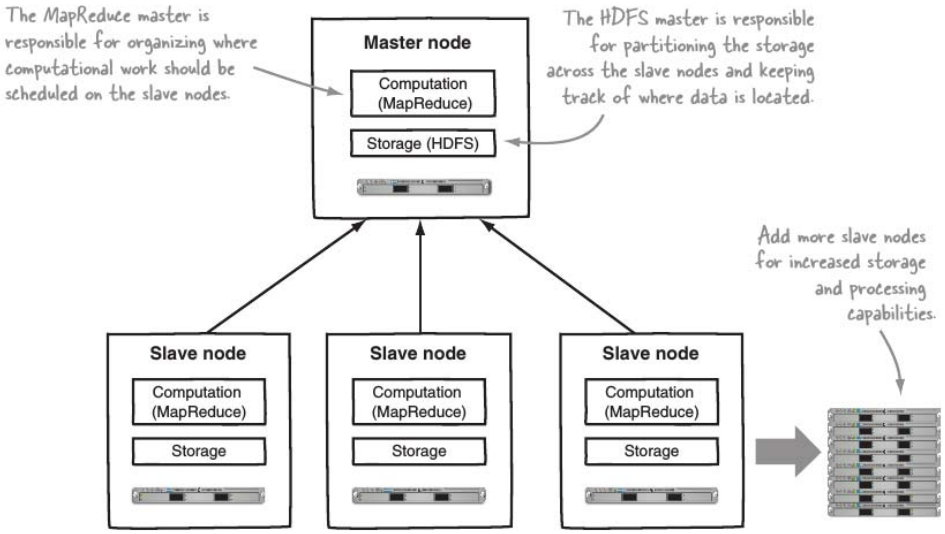
3.2 Τεχνολογία Hadoop

Το Hadoop είναι μια πλατφόρμα που παρέχει κατανεμημένη αποθήκευση και δυνατότητες επεξεργασίας. Το Hadoop αρχικά σχεδιάστηκε για να διορθώσει ένα πρόβλημα επεκτασιμότητας που

υπήρχε στο Nutch, ανοικτού κώδικα (**open source**) crawler και μηχανή αναζήτησης. Εκείνη τη χρονική στιγμή η Google είχε δημοσιεύσει τα έγγραφα που περιέγραφαν ένα καινοτόμο σύστημα αρχείων που χρησιμοποιούσε, το Google File System (**GFS**), και το MapReduce, ένα υπολογιστικό πλαίσιο δηλαδή για την παράλληλη επεξεργασία μεγάλων όγκων δεδομένων. Η επιτυχημένη εφαρμογή των παραπάνω εννοιών στο Nutch οδήγησε στη διάσπαση του σε δύο επιμέρους έργα, από τα οποία το δεύτερο έγινε γνωστό ως Hadoop.
Το Hadoop (όπως φαίνεται στην εικόνα 6), αποτελείται από μια κατανεμημένη αρχιτεκτονική τύπου master-slave η οποία περιέχει:

- Το Hadoop Distributed File System (**HDF**) για την αποθήκευση και
- Το MapReduce framework για την επεξεργασία δεδομένων

Χαρακτηριστικά που προσδιορίζουν το Hadoop είναι ο διαχωρισμός και η παράλληλη επεξεργασία μεγάλων συνόλων δεδομένων. Η αποθηκευτική και υπολογιστική του δυνατότητα μπορεί να κλιμακωθεί με την προσθήκη επιπλέον κόμβων σε ένα cluster.

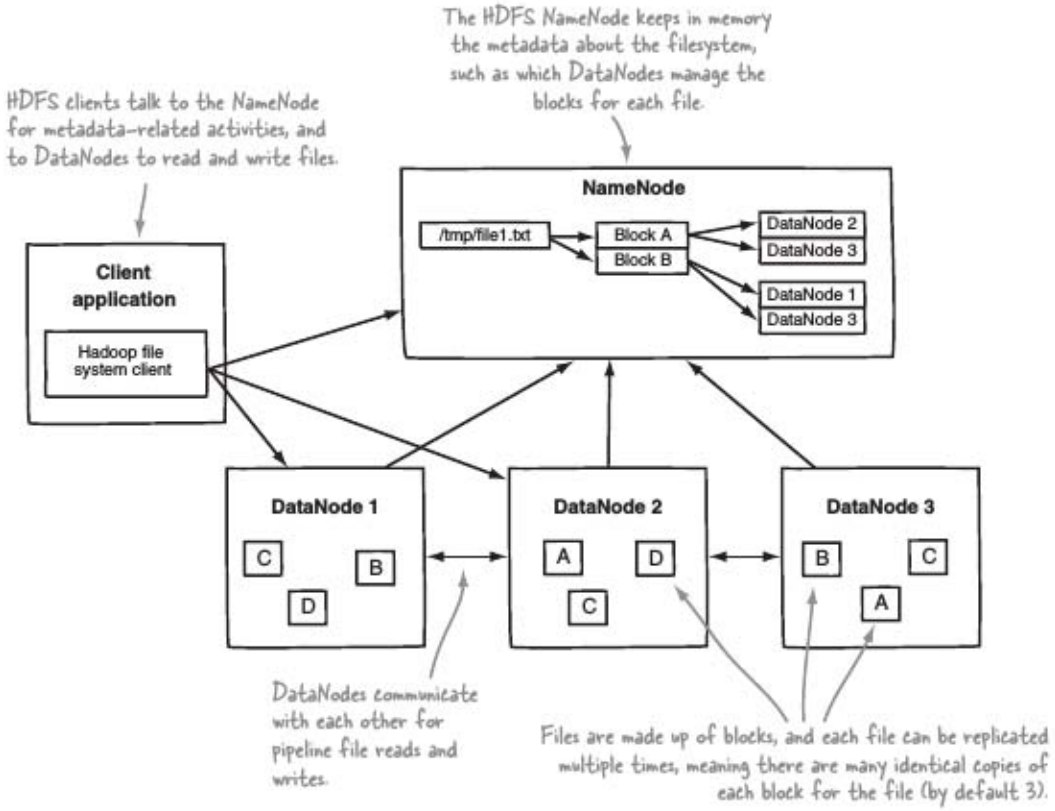


Εικόνα 6. Η Αρχιτεκτονική του Hadoop.[10]

3.2.1 Κατανεμημένο σύστημα αρχείων Hadoop (Hadoop Distributed File System - (HDFS))

Το HDFS αποτελεί το τμήμα αποθήκευσης του Hadoop. Είναι ένα κατανεμημένο σύστημα αρχείων το οποίο μοντελοποιήθηκε βάση του GFS [14][15]. Το HDFS είναι βελτιστοποιημένο για υψηλής απόδοσης δικτυακή λειτουργία και λόγω αυτού έχει καλύτερη συμπεριφορά κατά την ανάγνωση και γραφή μεγάλων αρχείων (από gigabytes και άνω). Για την υποστήριξη αυτή της διακομιστικής ικανότητας, το HDFS, αξιοποιεί έναν ασυνήθιστα μεγάλο (για ένα σύστημα αρχείων) μέγεθος μπλοκ (**block size**) όπως και βελτιστοποιήσεις αναφορικά με την τοπικότητα των δεδομένων (**data locality optimizations**) για τη μείωση της δικτυακής κίνησης εισόδου/εξόδου (I/O).
Η επεκτασιμότητα και η διαθεσιμότητα είναι επίσης βασικά χαρακτηριστικά του HDFS, τα οποία επιτεύχθηκαν εν μέρει λόγω της αναπαραγωγής δεδομένων (**data replication**) αλλά και της ανοχής σε σφάλματα (**fault tolerance**). Το HDFS αναπαράγει τα αποθηκευμένα αρχεία για ένα καθορισμένο αριθμό φορών, και είναι ανεκτικό τόσο στην αποτυχία λογισμικού όσο και υλικού (σε περίπτωση σφάλματος, το HDFS αυτόματα αναπαράγει το μπλοκ δεδομένων στους κόμβους που έχουν αποτύχει).
Η εικόνα 7, απεικονίζει την λογική αναπαράσταση των συστατικών του HDFS:

- Το **NameNode** και
- Το **DataNode**



Εικόνα 7. Αρχιτεκτονική του HDFS.[10]

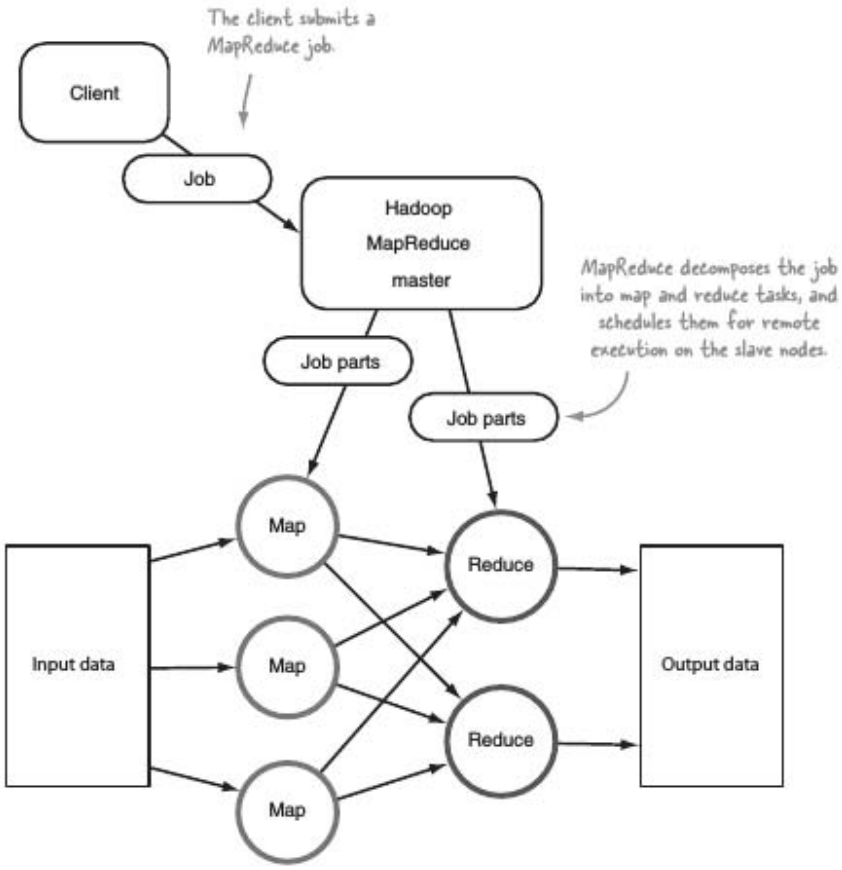
3.2.2 Τεχνολογία Hadoop απεικόνισης/μείωσης δεδομένων (Hadoop mapreduce)

Το MapReduce είναι ένα, βασισμένο σε δέσμη (**batch based**), κατανεμημένο υπολογιστικό πλαίσιο (**distributed computing framework**) το οποίο διαμορφώθηκε βάσει της δημοσίευσης του 2004 [14], της Google.
Επιτρέπει την παραλληλοποίηση της επεξεργασίας ενός μεγάλου ποσού δεδομένων (πχ. όπως ο συνδυασμός των αρχείων καταγραφής ιστού με τα σχεσιακά δεδομένα από μια βάση δεδομένων OLTP για τη διαμόρφωση μοντέλου του τρόπου με τον οποίο οι χρήστες αλληλοεπιδρούν με την ιστοσελίδα). Αυτό το είδος επεξεργασίας, το οποίο θα μπορούσε να απαιτεί μέρες ή και περισσότερο χρόνο με τη χρήση συμβατικών τεχνικών σειριακού προγραμματισμού, μπορεί να μειωθεί σε λεπτά χρησιμοποιώντας MapReduce σε ένα σύμπλεγμα από servers οι οποίοι εκτελούν το Hadoop.
Το μοντέλο του MapReduce απλοποιεί την παράλληλη επεξεργασία με το να αφαιρεί τις περιπλοκές που εμπλέκονται στην εργασία με κατανεμημένα συστήματα, όπως είναι:

1. Η παράλληλη επεξεργασία,
2. Ο καταμερισμός εργασίας και
3. Η ασχολία με αναξιοπιστίες υλικού και λογισμικού.

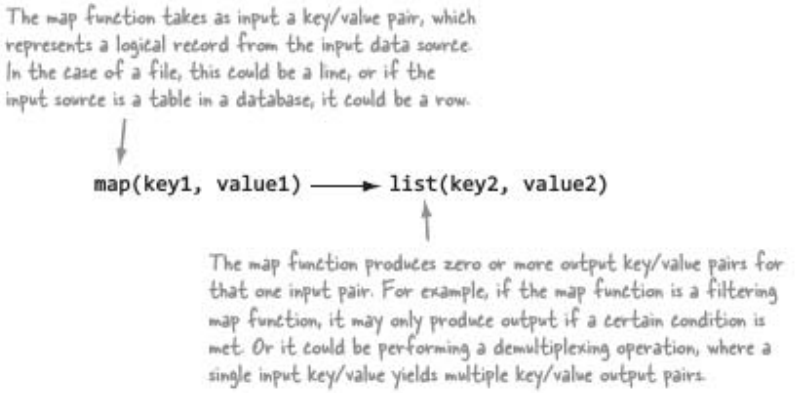
Χάρις σε αυτή την αφαίρεση, το MapReduce επιτρέπει στον προγραμματιστή να αναλωθεί στην αντιμετώπιση των πραγματικών αναγκών, χωρίς να εμπλακεί στις επιπλοκές ενός κατανεμημένου συστήματος.

Όπως αναφέρθηκε και στο προηγούμενο κεφάλαιο, το MapReduce “σπάει” τις εργασίες, που του υποβάλλονται, σε μικρότερα παραλληλοποιημένα τμήματα τα οποία ονομάζονται mappers και reducers (εικόνα 8). Τα map & reduce που χρησιμοποιούνται στο MapReduce είναι δανεισμένα από τη Lisp και χρησιμοποιούν ένα μοντέλο τύπου “shared-nothing” για να απομακρύνουν τυχόν παράλληλες αλληλεξαρτήσεις κατά την εκτέλεση που θα μπορούσαν να προσθέσουν ανεπιθύμητα σημεία συγχρονισμού ή διαμοιρασμού κατάστασης (state sharing).



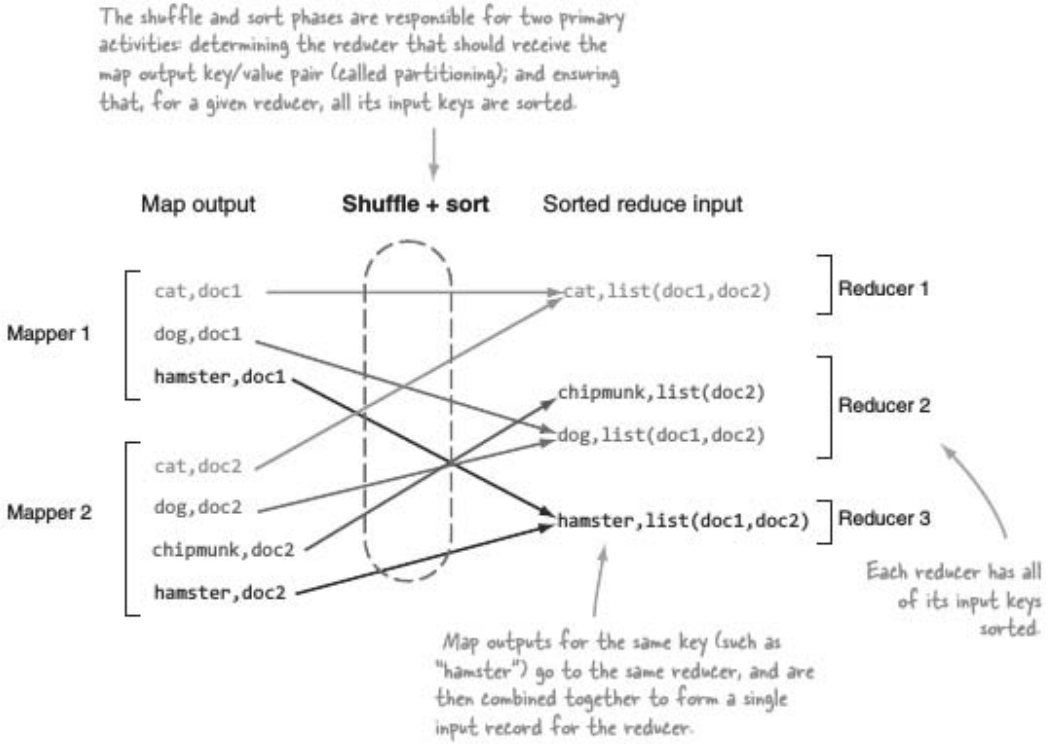
Εικόνα 8. Ένας πελάτης δίνει μία εργασία στο MapReduce.[10]

Ο ρόλος του προγραμματιστή είναι να καθορίσει τις συναρτήσεις map & reduce, κατά τις οποίες τα maps εξάγουν key/value pair tuples, τα οποία στη συνέχεια επεξεργάζονται από τα reduce για να παραχθεί το τελικό αποτέλεσμα. Η εικόνα 9, δείχνει τον ψευδοκώδικα ενός map σε σχέση με την είσοδο και την έξοδο του.

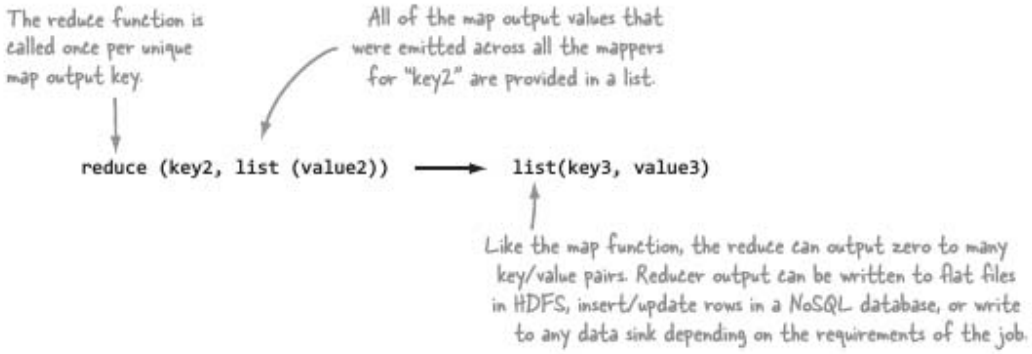


Εικόνα 9. Η map συνάρτηση.[10]

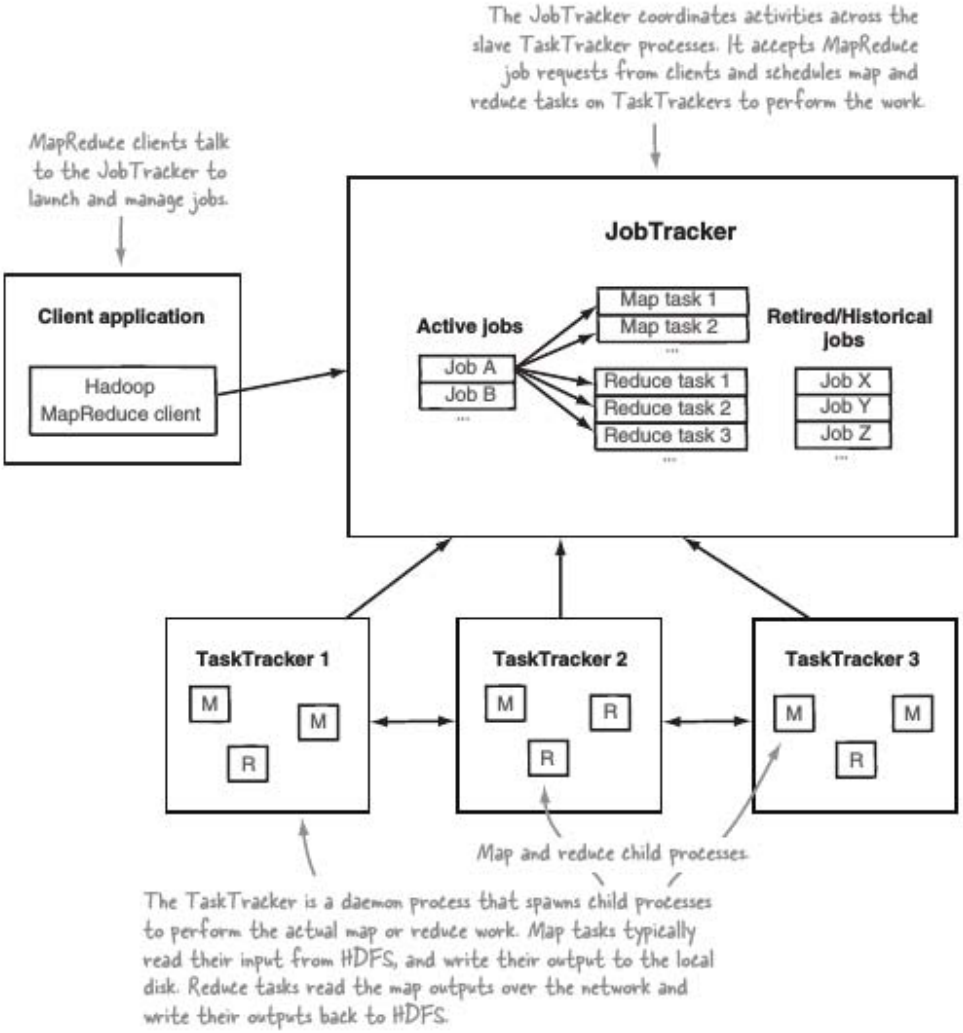
Η πραγματική δύναμη του MapReduce απαντάται μεταξύ της εξόδου του map και της εισόδου του reduce, στις φάσεις ανακατανομής (**shuffle**) και ταξινόμησης (**sort**), όπως φαίνεται στην εικόνα 10. Η εικόνα 11 δείχνει ένα ψευδοκώδικα μιας reduce συνάρτησης. Η αρχιτεκτονική του Hadoop MapReduce είναι παρόμοια με το μοντέλο master-slave του HDFS. Τα κύρια συστατικά της αρχιτεκτονικής του MapReduce απεικονίζονται στην εικόνα 12.



Εικόνα 10. MapReduce shuffle και sort. [10]



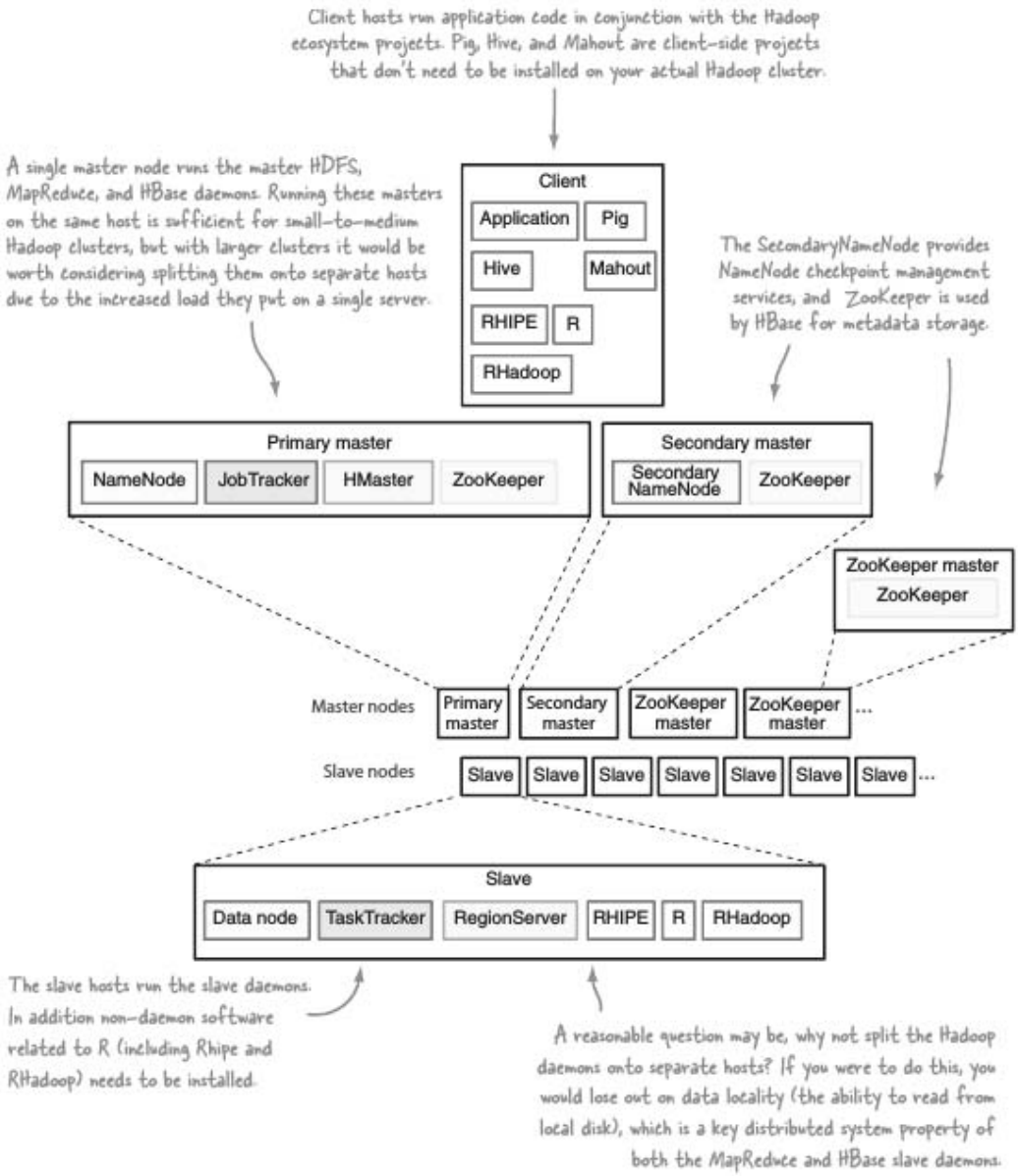
Εικόνα 11. MapReduce reduce λειτουργία. [10]



Εικόνα 12. Η αρχιτεκτονική του Hadoop MapReduce.[10]

3.2.3 Η Φυσική αρχιτεκτονική υποδομής του Hadoop

Η εικόνα 13, δείχνει ένα παράδειγμα φυσικής αρχιτεκτονικής που αφορά το Hadoop και τα επιμέρους στοιχεία που το απαρτίζουν, και τον τρόπο με τον οποίο διανέμεται σε όλους τους φυσικούς ξενιστές (hosts). Το Zookeeper απαιτεί ένα περιττό αριθμό (π.χ. 7). Κατά συνέπεια η συνιστώμενη πρακτική είναι να υπάρχουν τουλάχιστον τρεις από αυτούς σε κάθε ευλόγου μεγέθους σύμπλεγμα (cluster).



Εικόνα 13. Η Φυσική αρχιτεκτονική υποδομής του Hadoop.[10]

Ο όρος commodity hardware χρησιμοποιείται συχνά για να περιγράψει τις απαιτήσεις σε υλικό για το Hadoop. Είναι αλήθεια ότι το Hadoop μπορεί να τρέξει σε οποιοδήποτε παλιό server, αλλά εάν θέλουμε το cluster να αποδίδει τα μέγιστα, τότε με τον όρο αυτό αναφερόμαστε σε mid-level διακομιστές τύπου rack, με τουλάχιστον διπλή υποδοχή για CPU και όση μνήμη RAM είναι δυνατό να

έχουμε. Αναφορικά με την αποθήκευση, η τεχνολογία SATA με οδηγούς συστήματος για τύπου RAID αποθήκευση συνίσταται έντονα για NameNodes (για πρόσθετη αξιοπιστία). Παρόλα αυτά, το RAID δεν συνίσταται για DataNodes, καθώς το HDFS έχει ήδη replication και error-checking ενσωματωμένα σε αυτό.

Από τη σκοπιά του δικτύου σε σχέση με τους διακόπτες και τα τείχη προστασίας, το σύνολο των master και slave κόμβων πρέπει να είναι σε θέση να επικοινωνήσουν μεταξύ τους (καθώς χρησιμοποιούν το πρωτόκολλο SSH για ενδοεπικοινωνία και μεταφορά δεδομένων). Για μικρά cluster, αρκούν κάρτες δικτύου του 1 GB, οι οποίες συνδέονται σε ένα ενιαίο, switch καλής ποιότητας. Για μεγαλύτερα cluster, επιβάλλονται switches των 10 GB.

Παρουσίαση Πειραματικών Αποτελεσμάτων

Σε αυτή την ενότητα παρουσιάζεται η οργάνωση των πειραμάτων για την αξιολόγηση των αναπτυχθέντων αλγορίθμων κατασκευής κατάλληλων OD matrices με βάση την υπολογιστική πολυπλοκότητα, μετρούμενη σε μονάδες υπολογιστικού χρόνου του κατανεμημένου συστήματος.

4.1 Πειραματική Διάταξη

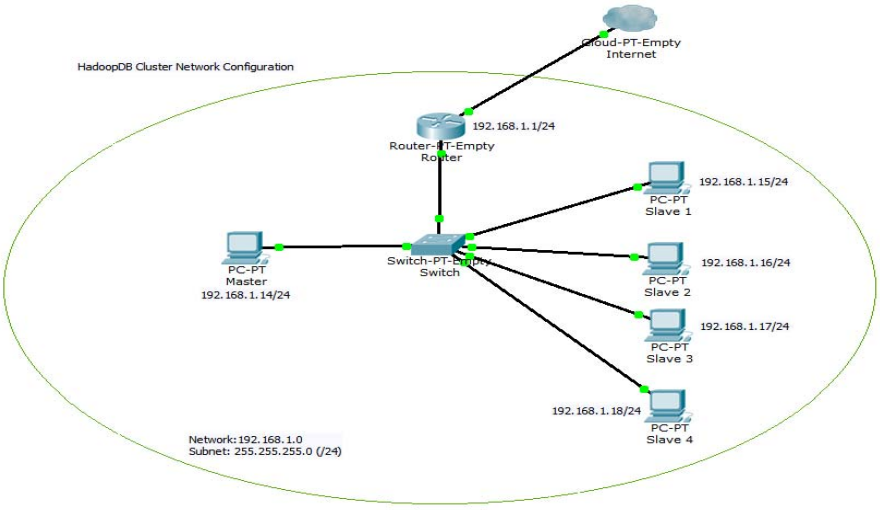
Στην παρούσα εργασία θα χρησιμοποιηθούν ένα Master instance και από τρία έως οκτώ Slave instances βασισμένα στο small Openstack flavor με τα εξής γενικά χαρακτηριστικά:

- 1 VCPUs
- 2 GB RAM
- HDD 20 GB

Κάθε ένα instance περιέχει το εξής λογισμικό:

- Ubuntu Linux OS (x64-86), version 14.04 (codename Trusty)-SSH server enabled
- Oracle OpenJDK, package, version 1.8.0_1011
- Apache Hadoop, version 2.7.3

Όλα τα instance επικοινωνούν μεταξύ τους μέσω Virtual Private Net του Openstack για την μεταφορά μηνυμάτων, χρονοπρογραμματισμό εργασιών και διαμοιρασμό δεδομένων.



Εικόνα 19. Αναπαράσταση του δικτύου του cluster.

4.2 Παρουσίαση Αποτελεσμάτων

Σε αυτό το σημείο γίνεται η εκτέλεση του αλγορίθμου, με έναν αυξανόμενο αριθμό slave μηχανημάτων και αριθμού κελιών (grid). Ποιο συγκεκριμένα, στην αρχή ο αλγόριθμος εκτελείται με 3 slaves και για έναν αυξανόμενο αριθμό κελιών. Τα κελιά, καθώς τα δεδομένα είναι spatio-temporal, αντιπροσωπεύουν ένα κύβο, πράγμα που σημαίνει ότι αποτελούν δύναμη του 3. Άρα ο αλγόριθμος θα εκτελεστεί:

- Για 3 slave μηχανήματα με
  - 8 ,27, 64, 125, 1000, 10648 κελιά (κύβοι) αντίστοιχα
- Για 6 slave μηχανήματα με
  - 8 ,27, 64, 125, 1000, 10648 κελιά (κύβοι) αντίστοιχα και
- Για 8 slave μηχανήματα με
  - 8 ,27, 64, 125, 1000, 10648 κελιά (κύβοι) αντίστοιχα

Τελικός στόχος είναι η εκτέλεση του αλγορίθμου, 36 φορές βάσει του αριθμού των μέγιστων slave μηχανημάτων (που είναι 8) και του μέγιστου αριθμού κελιών που είναι 10648, έτσι ώστε να δημιουργηθεί ένα σταθερό προφίλ αναφορικά με την επεξεργαστική δυνατότητα που προσφέρουν οι επιμέρους λύσεις που προσφέρει. Σε κάθε επιμέρους εκτέλεση του αλγορίθμου, καταγράφεται:

- ο χρόνος περάτωσης της map() μεθόδου
- ο χρόνος περάτωσης της reduce() μεθόδου και
- ο συνολικός χρόνος περάτωσης ανά προσέγγιση (Cell ή Trajectory Oriented) για όλο τον αλγόριθμο

Ο παρακάτω πίνακας περιέχει αναλυτικά τα αποτελέσματα της εκτέλεσης στα Opensack instances:

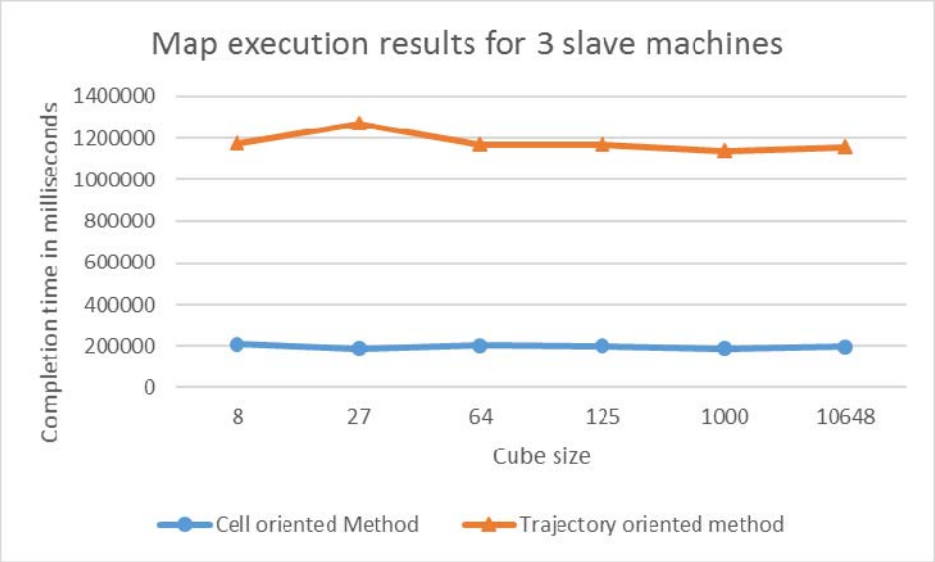
Πίνακας 5. Συνολικά αποτελέσματα εκτέλεσης του αλγορίθμου.

	Αριθμός εξαρτημένων Η/Υ (slaves)	Μέγεθος κύβου	Συνολικός χρόνος εκτέλεσης map() (milliseconds)	Συνολικός χρόνος εκτέλεσης reduce() (milliseconds)	Συνολικός χρόνος για την ολοκλήρωση (milliseconds)	Αντιστοιχία σε γραφήματα του κειμένου
Προσέγγιση με βάση το κελί	3	8	209761	365108	1069767	Εικόνες 20 έως 25
	3	27	188081	357351	1035060	Εικόνες 20 έως 25
	3	64	201818	361020	1107005	Εικόνες 20 έως 25
	3	125	199591	376662	1124714	Εικόνες 20 έως 25
	3	1000	188081	357502	1097778	Εικόνες 20 έως 25
	3	10648	197960	389748	1124471	Εικόνες 20 έως 25
	6	8	208748	376831	1072798	Εικόνες 26 έως 31

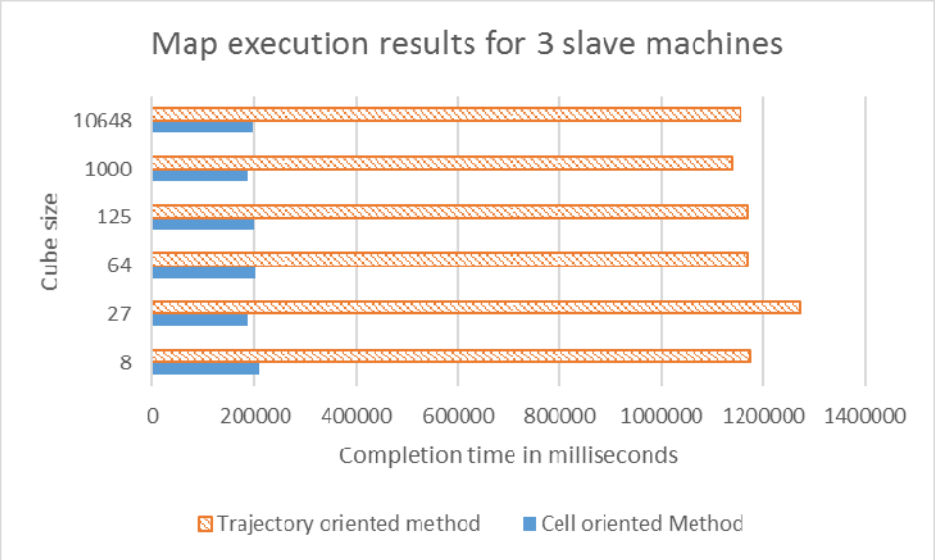
	6	27	192517	369769	1069641	Εικόνες 26 έως 31
	6	64	191791	367104	1117859	Εικόνες 26 έως 31
	6	125	190988	369967	1090510	Εικόνες 26 έως 31
	6	1000	195703	385807	1166693	Εικόνες 26 έως 31
	6	10648	187367	374736	1069457	Εικόνες 26 έως 31
	8	8	207922	368469	1081776	Εικόνες 32 έως 37
	8	27	195180	377164	1098390	Εικόνες 32 έως 37
	8	64	191910	369575	1129318	Εικόνες 32 έως 37
	8	125	191161	371361	1086368	Εικόνες 32 έως 37
	8	1000	185467	366751	1098416	Εικόνες 32 έως 37
	8	10648	194177	385746	1121407	Εικόνες 32 έως 37
Προσέγγιση με βάση τη διαδρομή	3	8	1175209	366863	2214627	Εικόνες 20 έως 25
	3	27	1272120	383791	2345869	Εικόνες 20 έως 25
	3	64	1170492	356975	2188927	Εικόνες 20 έως 25
	3	125	1170570	377990	2227924	Εικόνες 20 έως 25
	3	1000	1138252	343718	2147648	Εικόνες 20 έως 25
	3	10648	1156425	362316	2172252	Εικόνες 20 έως 25
	6	8	1166954	360529	2183534	Εικόνες 26 έως 31
	6	27	1228149	381868	2292159	Εικόνες 26 έως 31
	6	64	1187835	356057	2211168	Εικόνες 26 έως 31
	6	125	1183554	374770	2258434	Εικόνες 26 έως 31
	6	1000	1152038	362992	2187618	Εικόνες 26 έως 31

	6	10648	1141460	338995	2114051	Εικόνες 26 έως 31
	8	8	1152511	359805	2159656	Εικόνες 32 έως 37
	8	27	1230496	367034	2279776	Εικόνες 32 έως 37
	8	64	1143480	352796	2139479	Εικόνες 32 έως 37
	8	125	1133815	356575	2137992	Εικόνες 32 έως 37
	8	1000	1161422	361139	2163968	Εικόνες 32 έως 37
	8	10648	1149411	361343	2157855	Εικόνες 32 έως 37

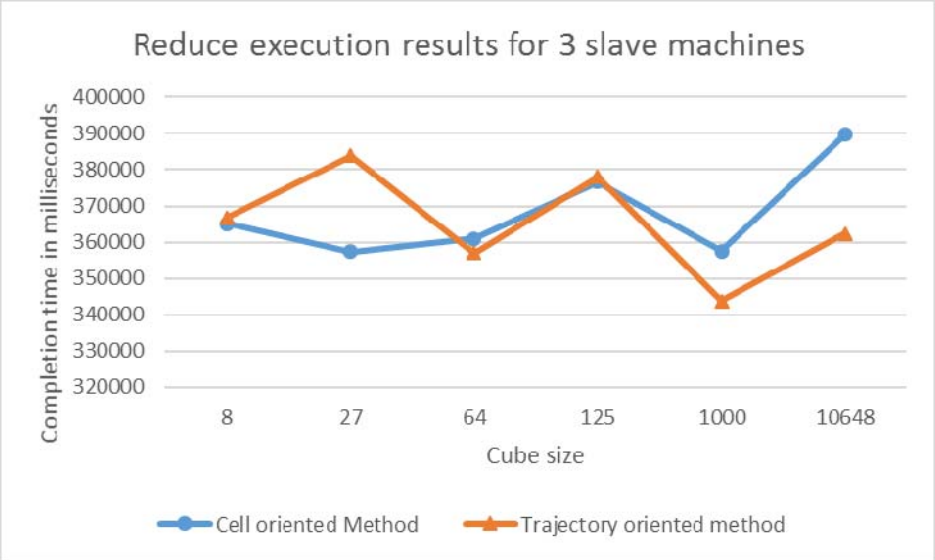
Στη συνέχεια παρατίθενται οι εικόνες με τα σχετικά διαγράμματα:



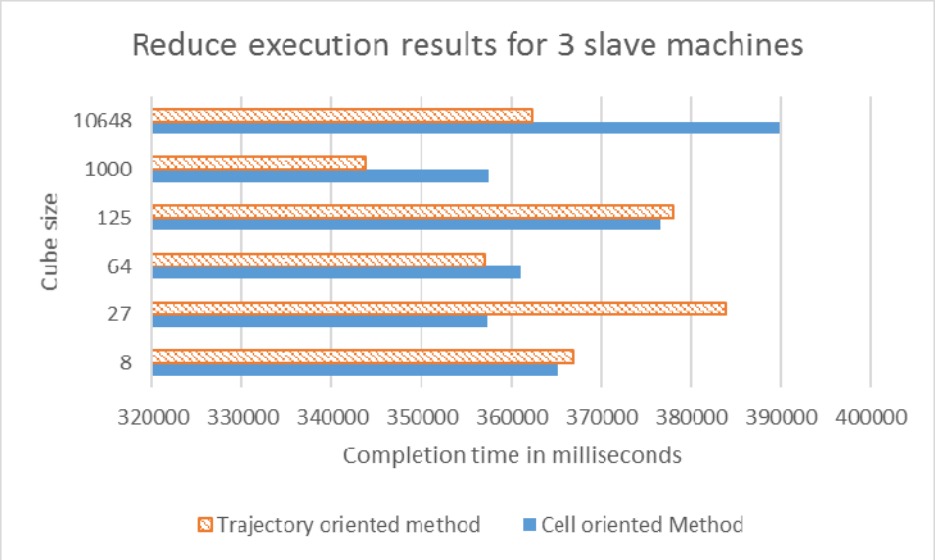
Εικόνα 20. Γραφική παράσταση του χρόνου εκτέλεσης του Map τμήματος του αλγορίθμου για 3 slave μηχανές.



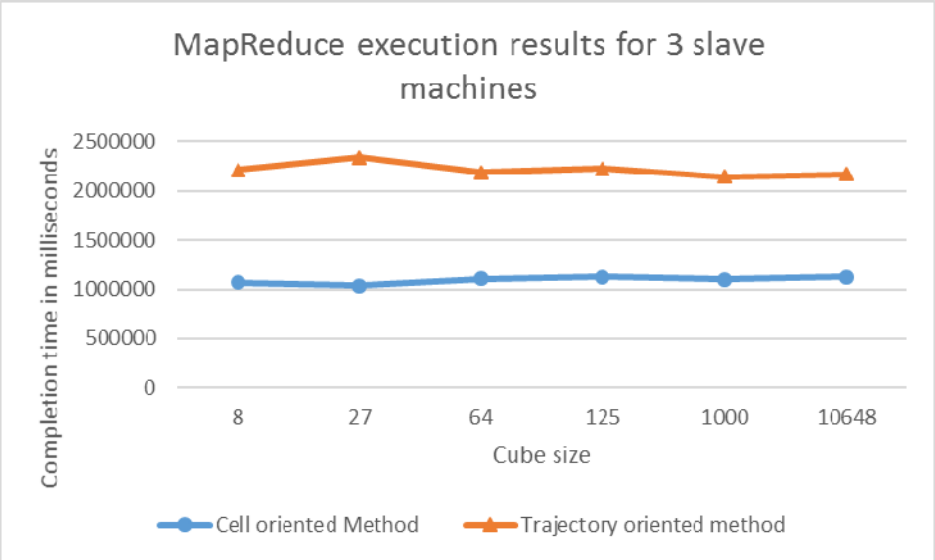
Εικόνα 21. Διάγραμμα μπάρας του χρόνου εκτέλεσης του Map τμήματος του αλγορίθμου για 3 slave μηχανές.



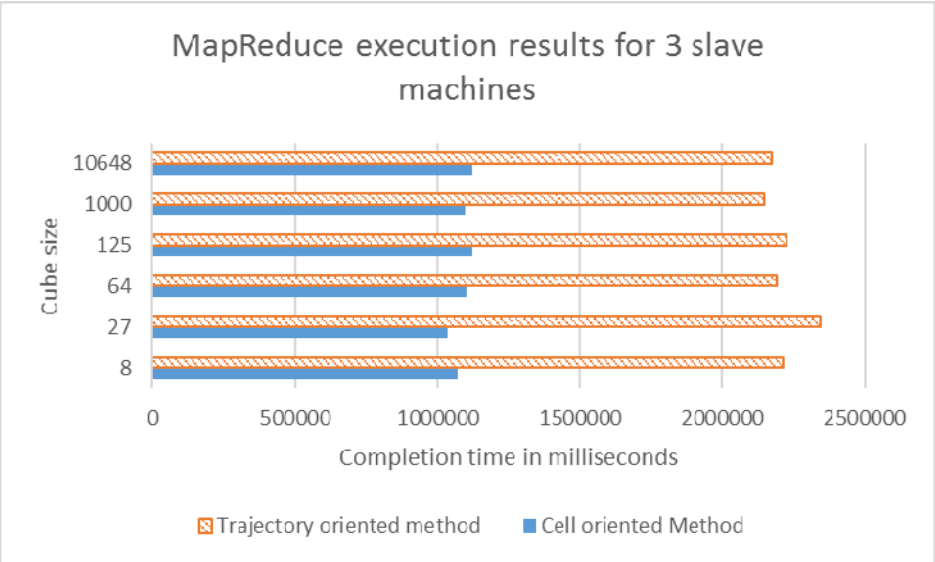
Εικόνα 22. Γραφική παράσταση του χρόνου εκτέλεσης του Reduce τμήματος του αλγορίθμου για 3 slave μηχανές.



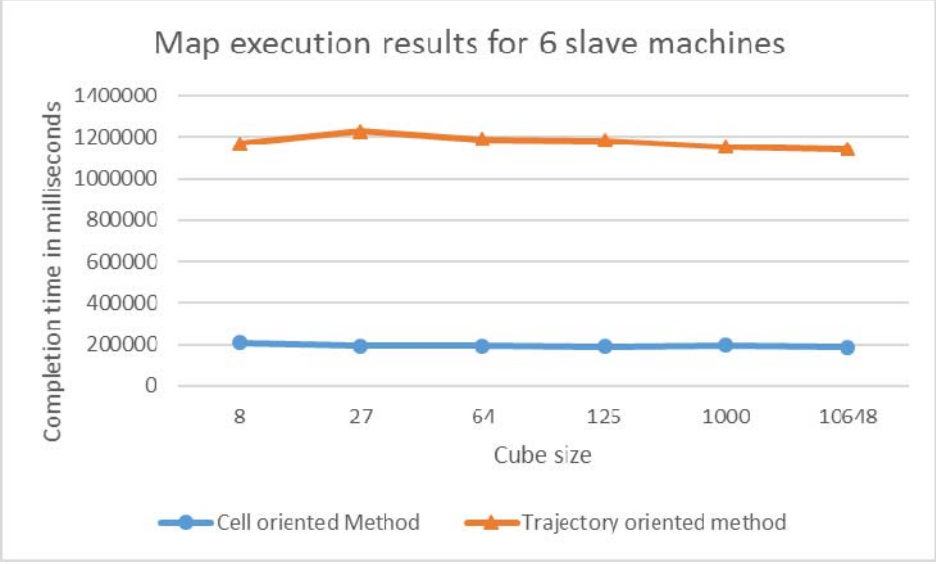
Εικόνα 23. Διάγραμμα μπάρας του χρόνου εκτέλεσης του Reduce τμήματος του αλγορίθμου για 3 slave μηχανές.



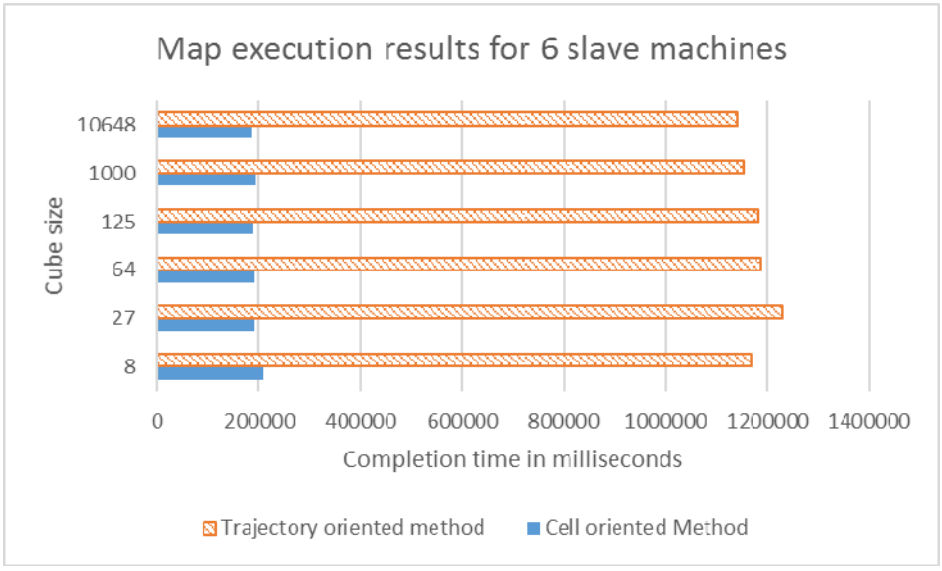
Εικόνα 24. Γραφική παράσταση του χρόνου εκτέλεσης όλου του αλγορίθμου για 3 slave μηχανές.



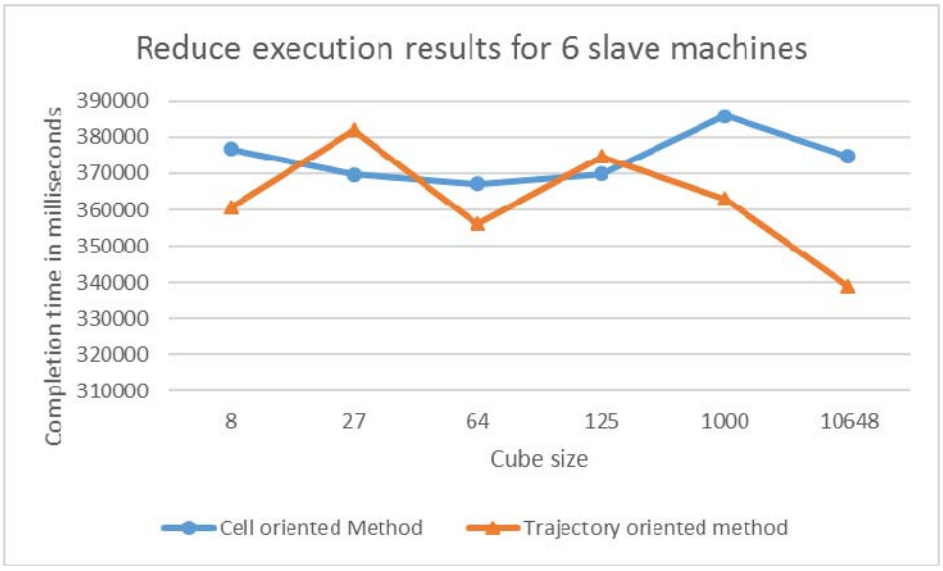
Εικόνα 25. Διάγραμμα μπάρας του χρόνου εκτέλεσης όλου του αλγορίθμου για 3 slave μηχανές.



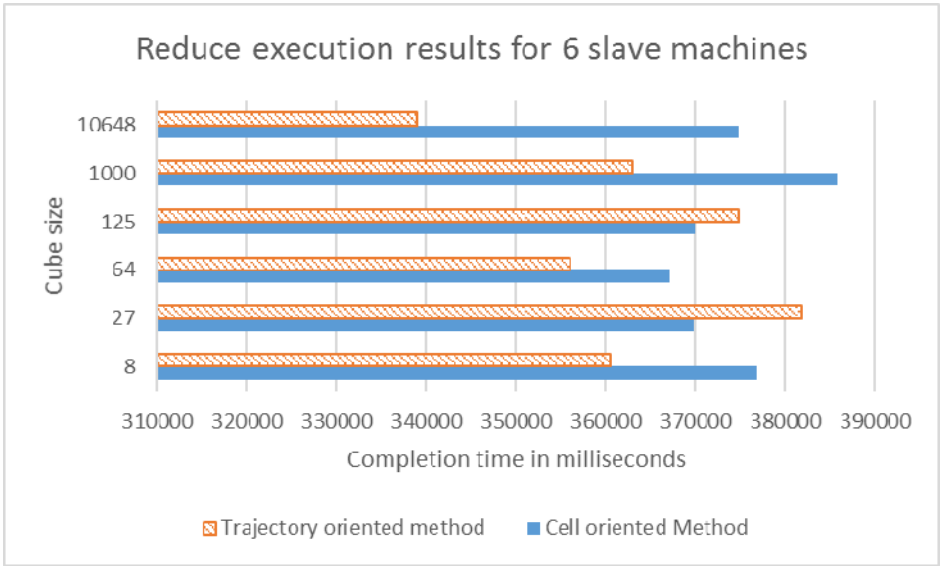
Εικόνα 26. Γραφική παράσταση του χρόνου εκτέλεσης του Map τμήματος του αλγορίθμου για 6 slave μηχανές.



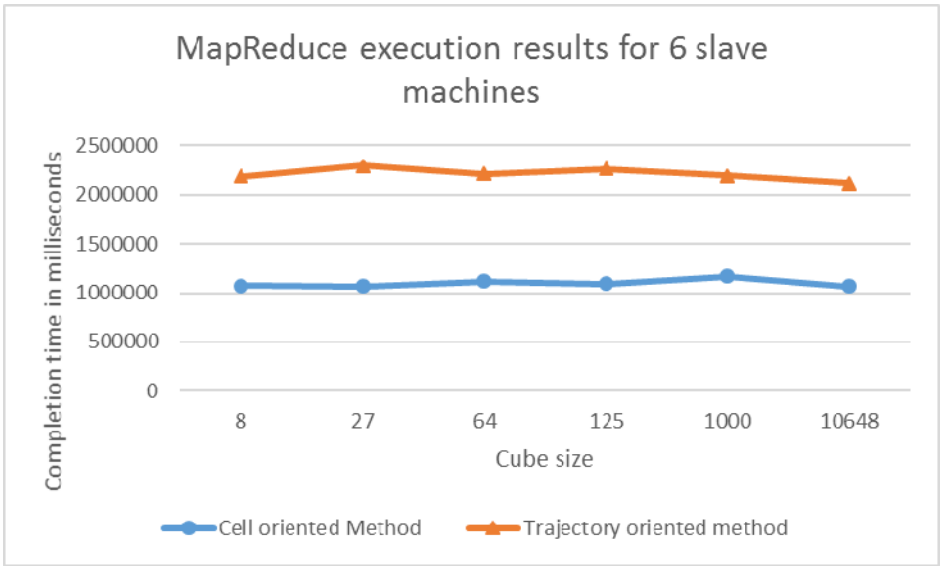
Εικόνα 27. Διάγραμμα μπάρας του χρόνου εκτέλεσης του Map τμήματος του αλγορίθμου για 6 slave μηχανές.



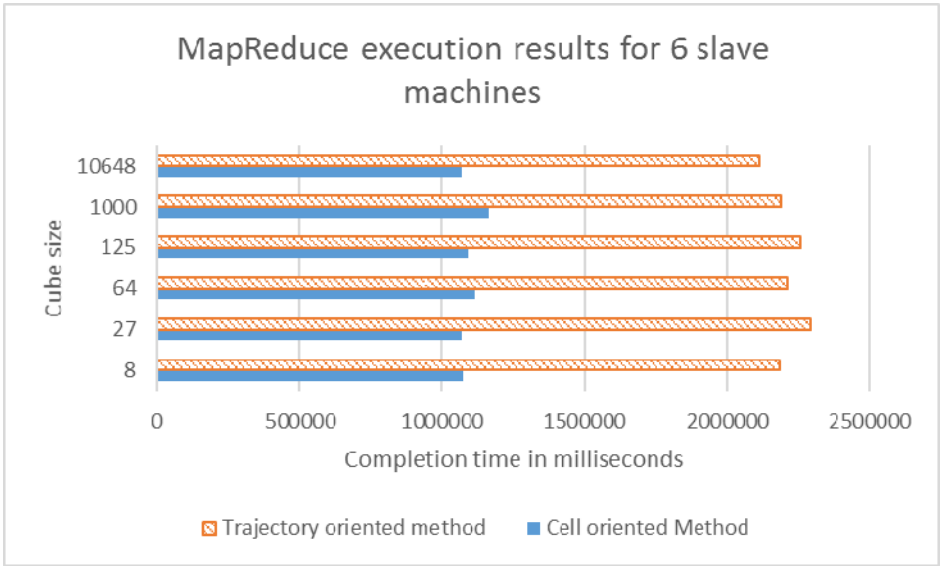
Εικόνα 28. Γραφική παράσταση του χρόνου εκτέλεσης του Reduce τμήματος του αλγορίθμου για 6 slave μηχανές.



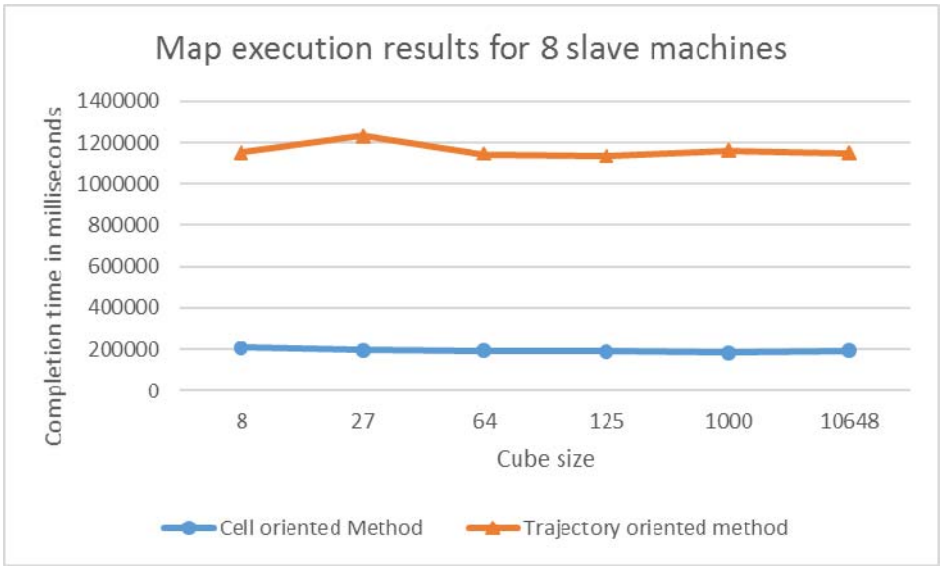
Εικόνα 29. Διάγραμμα μπάρας του χρόνου εκτέλεσης του Reduce τμήματος του αλγορίθμου για 6 slave μηχανές.



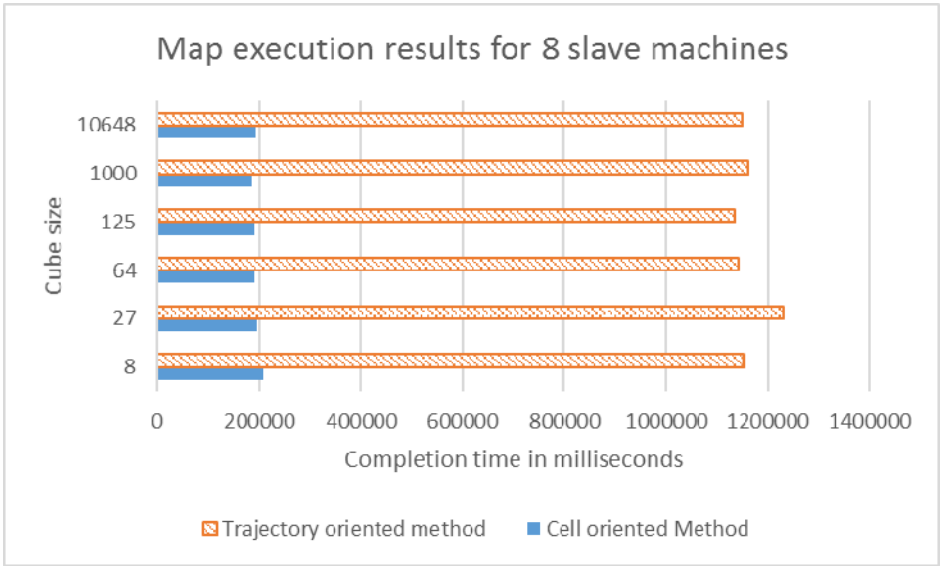
Εικόνα 30. Γραφική παράσταση του χρόνου εκτέλεσης όλου του αλγορίθμου για 6 slave μηχανές.



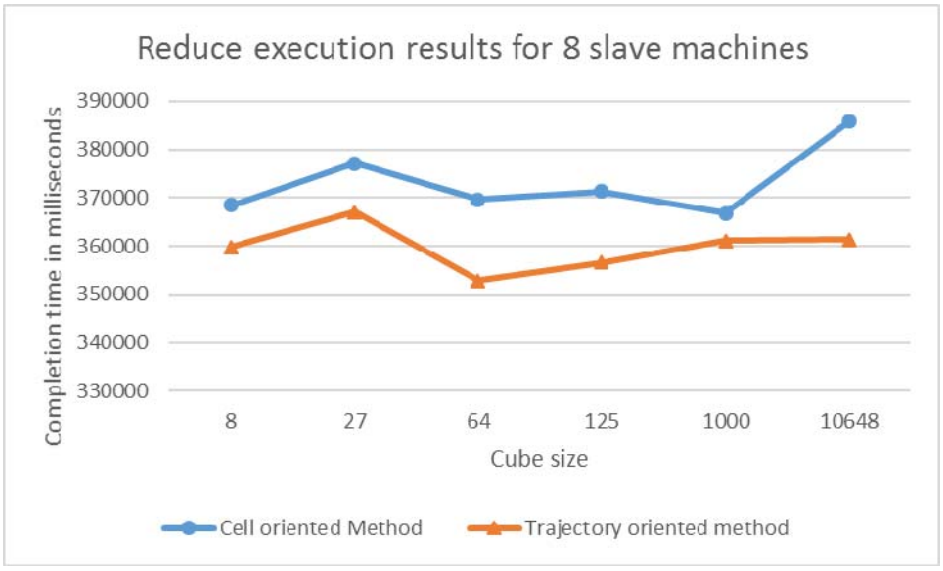
Εικόνα 31. Διάγραμμα μπάρας του χρόνου εκτέλεσης όλου του αλγορίθμου για 6 slave μηχανές.



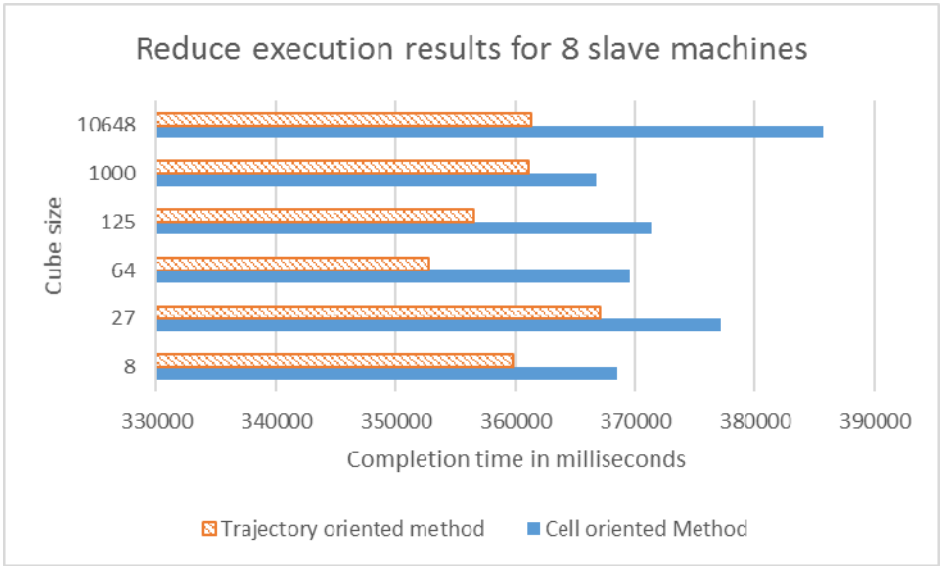
Εικόνα 32. Γραφική παράσταση του χρόνου εκτέλεσης του Map τμήματος του αλγορίθμου για 8 slave μηχανές.



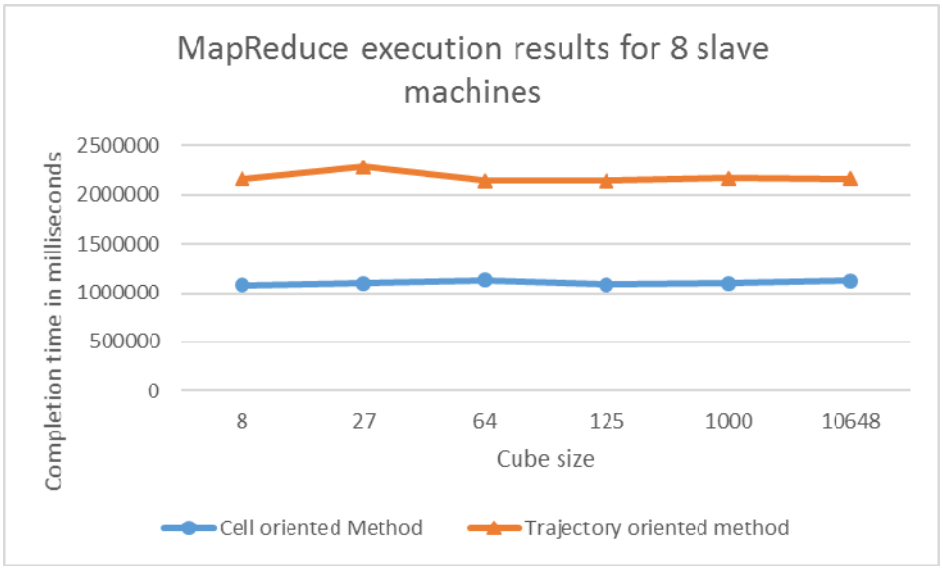
Εικόνα 33. Διάγραμμα μπάρας του χρόνου εκτέλεσης του Map τμήματος του αλγορίθμου για 8 slave μηχανές.



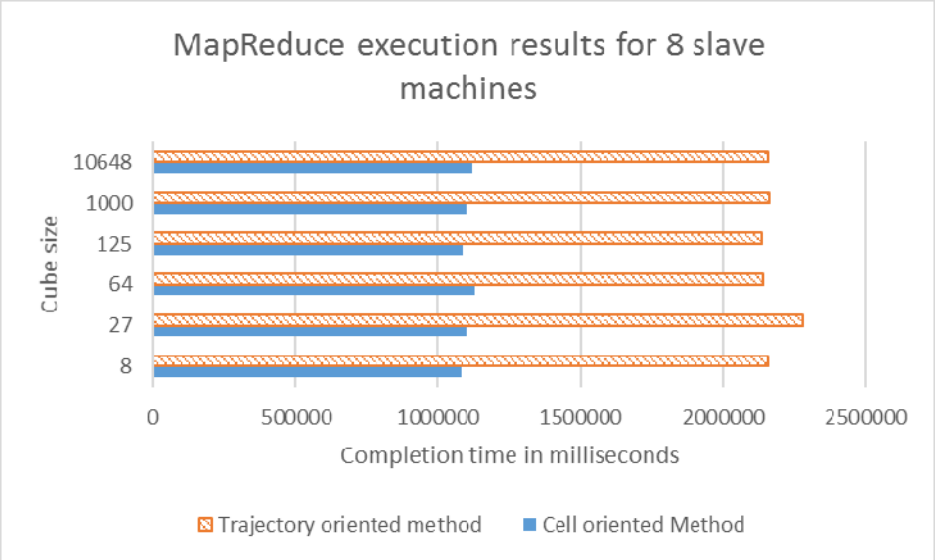
Εικόνα 34. Γραφική παράσταση του χρόνου εκτέλεσης του Reduce τμήματος του αλγορίθμου για 8 slave μηχανές.



Εικόνα 35. Διάγραμμα μπάρας του χρόνου εκτέλεσης του Reduce τμήματος του αλγορίθμου για 8 slave μηχανές.



Εικόνα 36. Γραφική παράσταση του χρόνου εκτέλεσης όλου του αλγορίθμου για 8 slave μηχανές.



Εικόνα 37. Διάγραμμα μπάρας του χρόνου εκτέλεσης όλου του αλγορίθμου για 8 slave μηχανές.

4.3 Συμπεράσματα και μελλοντικές προσδοκίες

1 Από τα γραφήματα 20 έως 37 είναι προφανές, ότι όσον αφορά την εκτέλεση του map τμήματος του αλγορίθμου, η απόσταση σε ό,τι αφορά την απόδοση των δύο προσεγγίσεων είναι σημαντική, με κυρίαρχη την προσέγγιση κελιού, για όλους του αριθμούς εξαρτημένων Η/Υ που δοκιμάστηκαν.

2 Όσον αφορά την εκτέλεση του reduce τμήματος του αλγορίθμου η προσέγγιση κελιού δίνει κατά τμήματα καλύτερη απόδοση αλλά όχι δραματικά διαφορετική για μικρό αριθμό εξαρτημένων Η/Υ (3 και 6 slaves – Εικόνα 22, 28). Αντίθετα, στην περίπτωση των 8 slaves (Εικόνα 34), είναι φανερό ότι η κυριαρχία της προσέγγισης κελιού είναι πλήρης και όχι οριακή.

3 Σε ό,τι αφορά το συνολικό αλγόριθμο, η καλύτερη απόδοσή της προσέγγισης κελιού είναι προφανής σε όλα τα γραφήματα 20 έως 37.

Από τη μελέτη που παρουσιάστηκε διαπιστώθηκαν ενδιαφέροντα συμπεράσματα για τις προσεγγίσεις κελιού και διαδρομής, τα οποία αφορούν τη δημιουργία spatiotemporal κυβικού πλέγματος το οποίο είναι στατικό. Θα είχε μεγάλη σημασία η εξέλιξη στο χρόνο των μοντέλων OD που δοκιμάστηκαν, προκειμένου να γίνει πραγματική πρόβλεψη των μοντέλων κίνησης μεταφορικών μέσων σε μακροπρόθεσμο επίπεδο και ιδιαίτερα η δημιουργία μοντέλων λήψης απόφασης με βάση αυτή τη μοντελοποίηση για την βελτίωση της κίνησης των μεταφορικών μέσων.

Βιβλιογραφικές Πηγές

Origin-Destination matrices

[1] Rodrigue, Comtois and Slack, in The Geography of Transport Systems, 3rd ed. Abingdon, Oxon, UK: Routledge, 2013.
[2] "O-D Matrix .ppt", presentation created by InfoLab, University of Piraeus.
[3] Kwon and Varaiya, "Real-Time Estimation of Origin-Destination Matrices with Partial Trajectories from Electronic Toll Collection Tag Data", in the 84th Annual Meeting Transportation Research Board, Washington, 2005.
[4] Abrahamsson, "Estimation of Origin-Destination Matrices Using Traffic Counts – A Literature Survey", in International Institute for Applied Systems Analysis, Laxenburg, Austria, 1996.
[5] Peterson, "The Origin–Destination Matrix Estimation Problem — Analysis and Computations", Linkoping Studies in Science and Technology, dissertations
No. 1102, Sweden, 2007.

Hadoop & MapReduce

[6] deRoos, in Hadoop For Dummies, 1st ed. New Jersey, 2014.
[7] Turkington, in Hadoop Beginner's Guide, 1st ed. Birmingham, UK, 2013.
[8] Miner and Shook, in MapReduce Design Patterns: Building Effective Algorithms and Analytics for Hadoop and Other Systems, 1st ed. Sebastopol, 2012.
[9] White, in Hadoop: The Definitive Guide, 4th ed. Sebastopol, 2015.
[10] Holmes, in Hadoop in Practice 1st ed. New York, 2012.
[11] Wadkar and Siddalingaiah, in Pro Apache Hadoop 2nd ed. New York, 2014.
[12] Owens, Lentz and Femiano, in Hadoop Real World Solutions Cookbook, 1st ed. Birmingham, UK, 2013.
[13] Doulkeridis and Nørvag, "A Survey of Large-Scale Analytical Query Processing in MapReduce", in VLDB Journal, Volume 23, Issue 3, June, 2014, Pages 355-380.
[14] Dean and Ghemawat, "MapReduce: simplified data processing on large clusters", in Communications of the ACM, Volume 51, Issue 1, January, 2008, Pages 107-113.
[15] Shvachko, Kuang, Radia and Chansler, "The Hadoop Distributed File System", in IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST), 2010.