



**ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ**

ΔΠΜΣ Επιστήμη Δεδομένων και Μηχανική Μάθηση


Διαχείριση Δεδομένων Μεγάλης Κλίμακας

Ακαδημαϊκό έτος 2023-2024, Εαρινό Εξάμηνο

***Εξαμηνιαία Εργασία***

<b>Ονοματεπώνυμο</b>	<b>ΑΜ</b>
Ιωάννης Βόγκας	03400206
Κωνσταντίνος Πριμέτης	03400231

Ιούνιος 2024

	<b>ΔΙΑΧΕΙΡΙΣΗ ΔΕΔΟΜΕΝΩΝ ΜΕΓΑΛΗΣ ΚΛΙΜΑΚΑΣ</b>  <b>Εξαμηνιαία Εργασία</b>  <b>ΣΥΝΟΔΕΥΤΙΚΗ ΑΝΑΦΟΡΑ</b>	ΙΟΥΝΙΟΣ 2024
		Σελίδα 1 / 20

## ΠΕΡΙΕΧΟΜΕΝΑ


<b>ΕΙΣΑΓΩΓΗ .....</b>	<b>3</b>
<b>ΖΗΤΟΥΜΕΝΟ 1 .....</b>	<b>4</b>
<b>ΖΗΤΟΥΜΕΝΟ 2 .....</b>	<b>6</b>
<b>ΖΗΤΟΥΜΕΝΟ 3 .....</b>	<b>9</b>
<b>ΖΗΤΟΥΜΕΝΟ 4 .....</b>	<b>12</b>
<b>ΖΗΤΟΥΜΕΝΟ 5 .....</b>	<b>14</b>
<b>ΖΗΤΟΥΜΕΝΟ 6 .....</b>	<b>18</b>
<b>ΖΗΤΟΥΜΕΝΟ 7 .....</b>	<b>20</b>

## ΛΙΣΤΑ ΕΙΚΟΝΩΝ

<i>Εικόνα 1. Εικονικές μηχανές στο περιβάλλον ~okeanos-knossos .....</i>	<i>4</i>
<i>Εικόνα 2. Web εφαρμογή Spark Job History Server .....</i>	<i>5</i>
<i>Εικόνα 3. Hadoop Web UI (α): Κατάσταση του συστήματος, (β): Διαθέσιμα δεδομένα .....</i>	<i>7</i>


## ΛΙΣΤΑ ΠΙΝΑΚΩΝ

<i>Πίνακας 1. Αποτέλεσμα Query 1.....</i>	<i>10</i>
<i>Πίνακας 2. Καταγεγραμμένοι χρόνοι για τους ζητούμενους συνδυασμούς προγραμματιστικών APIs και φορμάτ αρχείων εισόδου .....</i>	<i>10</i>
<i>Πίνακας 3. Αποτέλεσμα Query 2 (Φθίνουσα ταξινόμηση ως προς τις καταγραφές εγκλημάτων στον δρόμο).....</i>	<i>12</i>
<i>Πίνακας 4. Καταγεγραμμένοι χρόνοι ανά περίπτωση επίλυσης Query 2.....</i>	<i>12</i>
<i>Πίνακας 5. Αποτέλεσμα Query 3. Περιοχές με το υψηλότερο εισόδημα .....</i>	<i>14</i>
<i>Πίνακας 6. Αποτέλεσμα Query 3. Περιοχές με το χαμηλότερο εισόδημα.....</i>	<i>15</i>
<i>Πίνακας 7. Χρόνοι εκτέλεσης Query 3 ανά στρατηγική join .....</i>	<i>16</i>
<i>Πίνακας 8. Αποτέλεσμα Query 4.....</i>	<i>20</i>

	<p><b>ΔΙΑΧΕΙΡΙΣΗ ΔΕΔΟΜΕΝΩΝ ΜΕΓΑΛΗΣ ΚΛΙΜΑΚΑΣ</b></p> <p><b>Εξαμηνιαία Εργασία</b></p> <p><b>ΣΥΝΟΔΕΥΤΙΚΗ ΑΝΑΦΟΡΑ</b></p>	<p>ΙΟΥΝΙΟΣ 2024</p> <p>Σελίδα 2 / 20</p>
--	--	--

## ΛΙΣΤΑ ΑΠΟΣΠΑΣΜΑΤΩΝ ΚΩΔΙΚΑ

Απόσπασμα Κώδικα 1: Broadcast join .....	18
Απόσπασμα Κώδικα 2: Repartition join .....	19


	<b>ΔΙΑΧΕΙΡΙΣΗ ΔΕΔΟΜΕΝΩΝ ΜΕΓΑΛΗΣ ΚΛΙΜΑΚΑΣ</b>  <b>Εξαμηνιαία Εργασία</b>  <b>ΣΥΝΟΔΕΥΤΙΚΗ ΑΝΑΦΟΡΑ</b>	ΙΟΥΝΙΟΣ 2024
		Σελίδα 3 / 20

## ΕΙΣΑΓΩΓΗ

Η παρούσα εργασία αποτελεί την Εξαμηνιαία Εργασία που εκπονήθηκε στο πλαίσιο του Μαθήματος «Διαχείριση Δεδομένων Μεγάλης Κλίμακας» του ΔΠΜΣ «Επιστήμη Δεδομένων και Μηχανική Μάθηση».

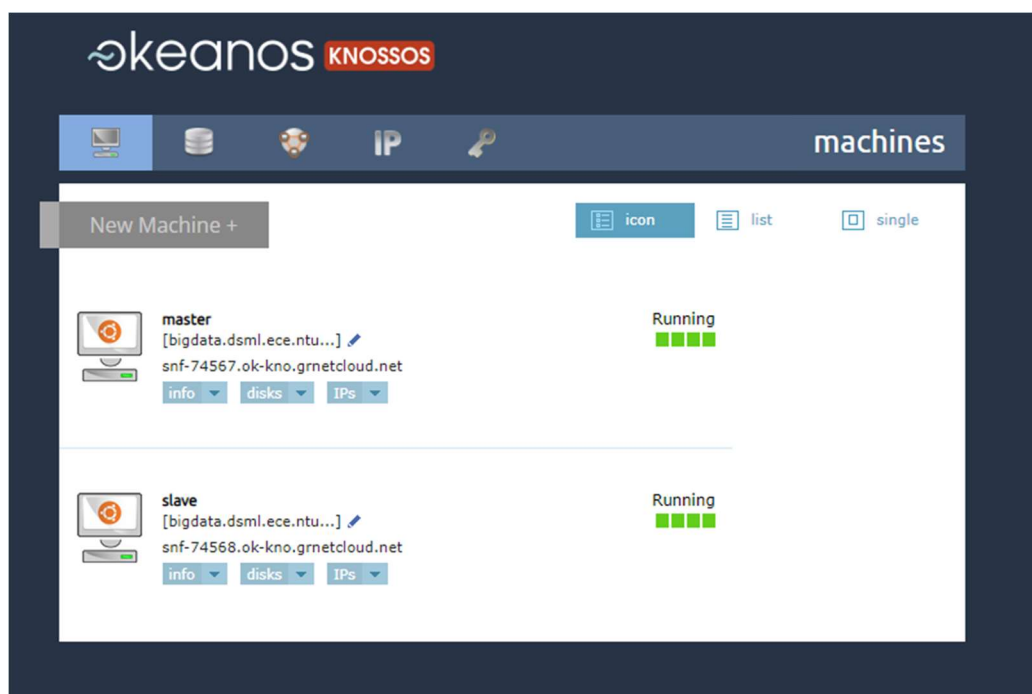
Μέσα από τη συγκεκριμένη εργασία πραγματοποιείται ανάλυση σε μεγάλα σύνολα δεδομένων, με την εφαρμογή κατάλληλων τεχνικών. Στο πλαίσιο της εργασίας γίνεται χρήση των εργαλείων Apache Hadoop και Apache Spark, ενώ για την εγκατάσταση και διαμόρφωση του κατάλληλου περιβάλλοντος εργασίας χρησιμοποιήθηκαν εικονικές μηχανές από το public cloud ~oceanos-knossos.

Η παρούσα έκθεση αποτελεί αναπόσπαστο τμήμα των αρχείων κώδικα Python, τα οποία είναι διαθέσιμα στο ακόλουθο link του αποθετηρίου github: "<https://github.com/ivogkas/Big-Data-Project>". Ακολούθως, διακριτοποιείται και παρουσιάζεται σε επτά (7) ενότητες, βάσει της δομής των ζητούμενων στην εκφώνηση της εργασίας. Στόχος της αναφοράς είναι να παρουσιάσει αποκλειστικά και μόνο (όπως ρητά ζητείται) τις απαντήσεις στα ζητούμενα της εργασίας.


	<p><b>ΔΙΑΧΕΙΡΙΣΗ ΔΕΔΟΜΕΝΩΝ ΜΕΓΑΛΗΣ ΚΛΙΜΑΚΑΣ</b></p> <p><b>Εξαμηνιαία Εργασία</b></p> <p><b>ΣΥΝΟΔΕΥΤΙΚΗ ΑΝΑΦΟΡΑ</b></p>	<p>ΙΟΥΝΙΟΣ 2024</p>
		<p>Σελίδα 4 / 20</p>

## ΖΗΤΟΥΜΕΝΟ 1

Αρχικά, εγκαταστάθηκε και διαμορφώθηκε κατάλληλα το σύστημα κατανεμημένης επεξεργασίας δεδομένων Apache Spark και το κατανεμημένο σύστημα αρχείων HDFS. Το περιβάλλον εργασίας ορίστηκε με τρόπο τέτοιο, ώστε να είναι πλήρως κατανεμημένο με δύο (2) κόμβους. Ακολούθως, παρουσιάζονται στιγμιότυπα οθόνης που αφορούν στην παραπάνω διαδικασία και συγκεκριμένα από το περιβάλλον της ιστοσελίδας ~okeanos-knossos (Εικόνα 1), όσο και από τη web εφαρμογή Spark Job History Server (Εικόνα 2).



Εικόνα 1. Εικονικές μηχανές στο περιβάλλον ~okeanos-knossos

 **History Server**  
3.5.1

Event log directory: file:/tmp/spark-events  
Last updated: 2024-05-31 15:03:44  
Client local time zone: Europe/Athens

Show  entries


Search:

Version	App ID	App Name	Started	Completed	Duration	Spark User	Last Updated	Event Log
3.5.1	app-20240513232437-0007	wordcount example	2024-05-13 23:24:35	2024-05-13 23:24:48	13 s	user	2024-05-13 23:24:48	<a href="#">Download</a>
3.5.1	app-20240516110250-0008	wordcount example	2024-05-16 11:02:48	2024-05-16 11:03:09	20 s	user	2024-05-16 11:03:09	<a href="#">Download</a>
3.5.1	app-20240516111231-0009	wordcount example	2024-05-16 11:12:29	2024-05-16 11:12:40	12 s	user	2024-05-16 11:12:40	<a href="#">Download</a>
3.5.1	app-20240516133647-0010	Convert to parquet	2024-05-16 13:36:40	2024-05-16 13:37:48	1.1 min	user	2024-05-16 13:37:48	<a href="#">Download</a>
3.5.1	app-20240516134134-0011	Convert to parquet	2024-05-16 13:41:32	2024-05-16 13:42:27	55 s	user	2024-05-16 13:42:27	<a href="#">Download</a>
3.5.1	app-20240516134534-0012	PySparkShell	2024-05-16 13:45:32	2024-05-16 13:48:54	3.4 min	user	2024-05-16 13:48:55	<a href="#">Download</a>
3.5.1	app-20240516141358-0013	Query 1 DataFrame	2024-05-16 14:13:56	2024-05-16 14:14:04	8 s	user	2024-05-16 14:14:04	<a href="#">Download</a>
3.5.1	app-20240516141939-0014	Query 1 DataFrame	2024-05-16 14:19:37	2024-05-16 14:19:44	8 s	user	2024-05-16 14:19:44	<a href="#">Download</a>
3.5.1	app-20240516142314-0015	Query 1 DataFrame	2024-05-16 14:23:12	2024-05-16 14:23:19	7 s	user	2024-05-16 14:23:19	<a href="#">Download</a>
3.5.1	app-20240516142613-0016	Query 1 DataFrame	2024-05-16 14:26:12	2024-05-16 14:26:32	21 s	user	2024-05-16 14:26:32	<a href="#">Download</a>
3.5.1	app-20240516144029-0017	Query 1 DataFrame	2024-05-16 14:40:27	2024-05-16 14:40:44	17 s	user	2024-05-16 14:40:44	<a href="#">Download</a>
3.5.1	app-20240518162843-0018	Q2_RDD_csv	2024-05-18 16:28:41	2024-05-18 16:29:00	19 s	user	2024-05-18 16:29:00	<a href="#">Download</a>

**Εικόνα 2. Web εφαρμογή Spark Job History Server**

Οι web εφαρμογές των HDFS και Spark Job History Server είναι προσβάσιμες μέσω των ακόλουθων συνδέσμων:

- Hadoop Web UI: <http://83.212.81.249:9870/>
- Spark Master Web UI: <http://83.212.81.249:8080/>
- Spark Worker Web UI: <http://83.212.81.249:8081/>
- Spark Job History Server Web UI: <http://83.212.81.249:18080/>

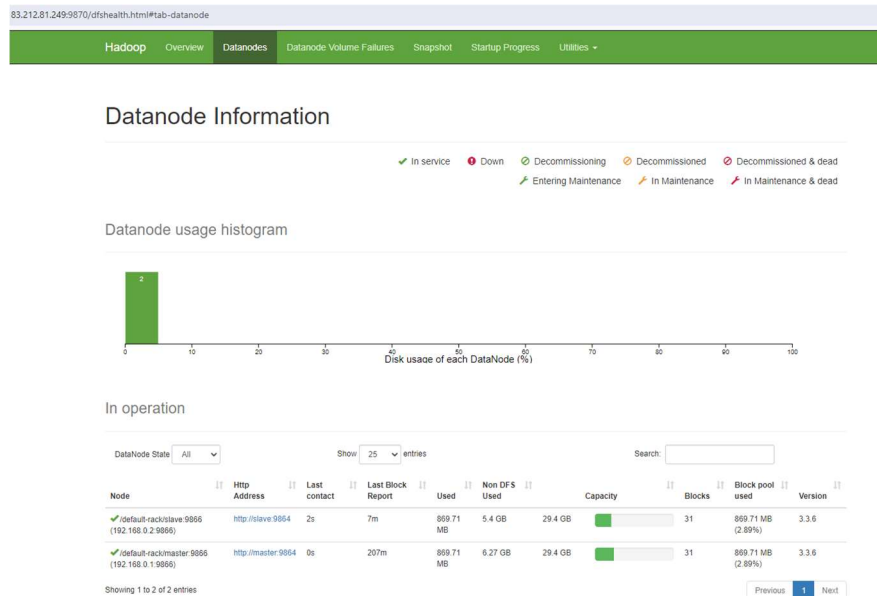
	<p><b>ΔΙΑΧΕΙΡΙΣΗ ΔΕΔΟΜΕΝΩΝ ΜΕΓΑΛΗΣ ΚΛΙΜΑΚΑΣ</b></p> <p><b>Εξαμηνιαία Εργασία</b></p> <p>ΣΥΝΟΔΕΥΤΙΚΗ ΑΝΑΦΟΡΑ</p>	<p>ΙΟΥΝΙΟΣ 2024</p>
		<p>Σελίδα 6 / 20</p>

## ΖΗΤΟΥΜΕΝΟ 2

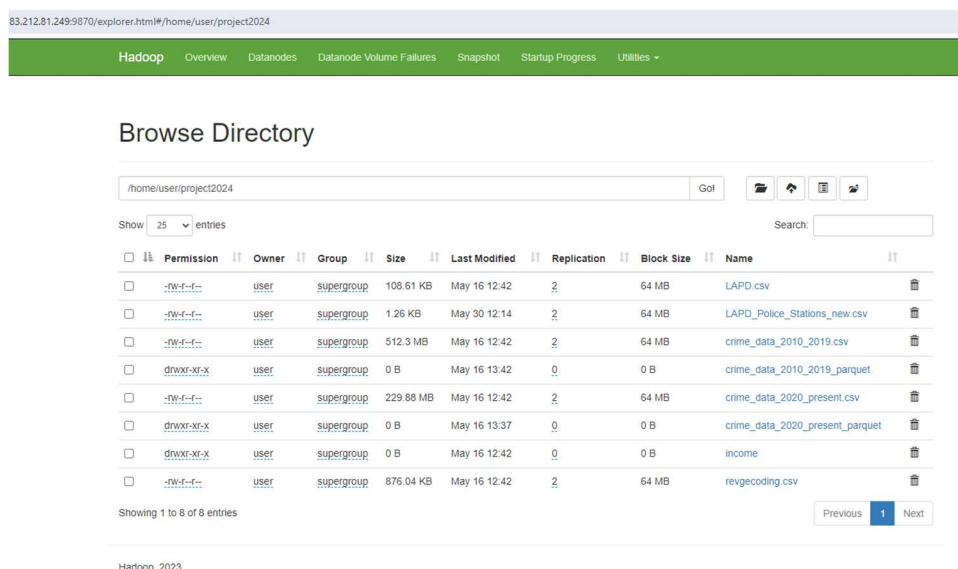
Αρχικά, δημιουργήθηκε ένα directory στο HDSF, όπου στη συνέχεια αποθηκεύτηκαν τα σύνολα δεδομένων σε μορφή .csv. Η διαδικασία που ακολουθήθηκε για αυτό τον σκοπό έχει ως εξής:

- i. Δημιουργία directory στον master, όπου θα αποθηκευτούν τα αρχεία δεδομένων: `mkdir data_project_2024`
- ii. Εκτέλεση της ακόλουθης εντολής, ώστε να γίνει εφικτή η λήψη αρχείων από links και λήψη των δεδομένων μέσω του master, π.χ.: `wget --no-check-certificate https://data.lacity.org/Public-Safety/Crime-Data-from-2010-to-2019/63jg-8b9z`
- iii. Μετονομασία των αρχείων στο επιθυμητό όνομα, π.χ.: `mv initial_filename crime_data_2010_2019.csv`
- iv. Στην περίπτωση συμπιεσμένου αρχείου, εξάγουμε το περιεχόμενο, π.χ.: `tar -xzf data.tar.gz`
- v. Δημιουργία HDFS directory, όπου θα ανέβουν τα αρχεία: `hadoop fs -mkdir -p ~/project2024`
- vi. Μεταφόρτωση των αρχείων στο HDFS directory που δημιουργήθηκε: `hadoop fs -put *.csv ~/project2024`

Στην Εικόνα 3 που ακολουθεί, αποτυπώνεται η κατάσταση του συστήματος αρχείων με τα δεδομένα διαθέσιμα.




(α)



(β)

Εικόνα 3. Hadoop Web UI (α): Κατάσταση του συστήματος, (β): Διαθέσιμα δεδομένα



	<p><b>ΔΙΑΧΕΙΡΙΣΗ ΔΕΔΟΜΕΝΩΝ ΜΕΓΑΛΗΣ ΚΛΙΜΑΚΑΣ</b></p> <p><b>Εξαμηνιαία Εργασία</b></p> <p><b>ΣΥΝΟΔΕΥΤΙΚΗ ΑΝΑΦΟΡΑ</b></p>	<p>ΙΟΥΝΙΟΣ 2024</p>
		<p>Σελίδα 8 / 20</p>

Το ζητούμενο αρχείο κώδικα spark για τη μετατροπή του κυρίως συνόλου δεδομένων σε parquet file format και την αποθήκευση των παραγόμενων αρχείων στο HDFS, μπορεί να βρεθεί στο αποθετήριο της εργασίας.

### ΖΗΤΟΥΜΕΝΟ 3

Το αποτέλεσμα της υλοποίησης του Query 1 παρατίθεται στον ακόλουθο πίνακα (Πίνακας 1):

year	month	crime_total	ranking
2010	1	19520	1
2010	3	18131	2
2010	7	17857	3
2011	1	18141	1
2011	7	17283	2
2011	10	17034	3
2012	1	17954	1
2012	8	17661	2
2012	5	17502	3
2013	8	17441	1
2013	1	16828	2
2013	7	16645	3
2014	10	17331	1
2014	7	17258	2
2014	12	17198	3
2015	10	19221	1
2015	8	19011	2
2015	7	18709	3
2016	10	19660	1
2016	8	19496	2
2016	7	19450	3
2017	10	20437	1
2017	7	20199	2
2017	1	19849	3
2018	5	19976	1
2018	7	19879	2
2018	8	19765	3
2019	7	19126	1
2019	8	18987	2
2019	3	18865	3
2020	1	18542	1
2020	2	17272	2
2020	5	17219	3
2021	10	19326	1
2021	7	18672	2
2021	8	18387	3
2022	5	20450	1

year	month	crime_total	ranking
2022	10	20313	2
2022	6	20255	3
2023	10	20029	1
2023	8	20024	2
2023	1	19902	3
2024	1	18762	1
2024	2	17214	2
2024	3	16009	3

**Πίνακας 1. Αποτέλεσμα Query 1**


Στον ακόλουθο πίνακα (Πίνακας 2) παρατίθενται οι καταγεγραμμένοι χρόνοι για καθέναν από τους τέσσερις (4) συνδυασμούς μεταξύ ζητούμενων προγραμματιστικών APIs και φορμάτ αρχείων εισόδου:

	DataFrame API	SQL API
csv	42 s	46 s
parquet	35 s	34 s

**Πίνακας 2. Καταγεγραμμένοι χρόνοι για τους ζητούμενους συνδυασμούς προγραμματιστικών APIs και φορμάτ αρχείων εισόδου**

Η εκτέλεση με χρήση του DataFrame API είναι κατά τέσσερα (4) δευτερόλεπτα ταχύτερη από τη χρήση του SQL API στην περίπτωση που χρησιμοποιούνται αρχεία μορφής csv. Το γεγονός αυτό δύναται να υποδεικνύει καλύτερη απόδοση του DataFrame API κατά την επεξεργασία δεδομένων σε μορφή csv. Ωστόσο, η διαφορά στους χρόνους είναι μικρή (<10%), ώστε να εξαχθεί κάποιο γενικό συμπέρασμα.

Η εκτέλεση με χρήση του SQL API μετρήθηκε ως κατά ένα (1) δευτερόλεπτο ταχύτερη από την αντίστοιχη του DataFrame API όταν χρησιμοποιούνται αρχεία σε μορφή parquet. Εδώ, η πολύ

	<p><b>ΔΙΑΧΕΙΡΙΣΗ ΔΕΔΟΜΕΝΩΝ ΜΕΓΑΛΗΣ ΚΛΙΜΑΚΑΣ</b></p> <p><b>Εξαμηνιαία Εργασία</b></p> <p><b>ΣΥΝΟΔΕΥΤΙΚΗ ΑΝΑΦΟΡΑ</b></p>	<p>ΙΟΥΝΙΟΣ 2024</p>
		<p>Σελίδα 11 / 20</p>

μικρή διαφορά (υπέρ του SQL API) υποδεικνύει ότι στην περίπτωση των parquet αρχείων, τα δύο APIs φαίνεται να λειτουργούν εξίσου αποδοτικά.

Σε κάθε περίπτωση, είναι εμφανές ότι η χρήση αρχείων τύπου parquet είναι αποδοτικότερη από την αντίστοιχη με csv αρχεία. Η μορφή των αρχείων parquet επιτρέπει την εύκολη διαίρεση των δεδομένων σε μπλοκ, γεγονός που διευκολύνει την παράλληλη επεξεργασία από κατανεμημένα συστήματα επεξεργασίας, όπως το Hadoop και το Spark.

Όλοι οι παραπάνω χρόνοι αφορούν στην επεξεργασία των δεδομένων εξ' αρχής (όπως παρασχέθηκαν).

**ΖΗΤΟΥΜΕΝΟ 4**

Η υλοποίηση του Query 2 πραγματοποιήθηκε με τη χρήση DataFrame και RDD APIs. Το αποτέλεσμα της υλοποίησης του Query 2 παρατίθεται στον ακόλουθο πίνακα (Πίνακας 3):

A/A	Τμήμα ημέρας
1	Νύχτα: 9.00μμ – 4.59πμ
2	Βράδυ: 5.00μμ – 8.59μμ
3	Απόγευμα: 12.00μμ – 4.59μμ
4	Πρωί: 5.00πμ – 11.59πμ

**Πίνακας 3. Αποτέλεσμα Query 2 (Φθίνουσα ταξινόμηση ως προς τις καταγραφές εγκλημάτων στον δρόμο)**


Οι χρόνοι εκτέλεσης για καθεμιά από τις περιπτώσεις επίλυσης του ερωτήματος παρουσιάζονται ακολούθως (Πίνακας 4):

	Χρόνος εκτέλεσης
<b>DataFrame API</b>	33 s
<b>RDD API</b>	35 s

**Πίνακας 4. Καταγεγραμμένοι χρόνοι ανά περίπτωση επίλυσης Query 2**

Παρατηρώντας τους χρόνους που προκύπτουν στο πλαίσιο της άσκησης, γίνεται αντιληπτό ότι το ερώτημα υλοποιείται ταχύτερα με χρήση του DataFrame API (έστω και με μικρή διαφορά).

Το αποτέλεσμα αυτό θεωρείται αναμενόμενο, καθώς το DataFrame API παρουσιάζει πλεονεκτήματα για αναλυτικές εργασίες και επεξεργασία δεδομένων μεγάλης κλίμακας, λόγω των

	<b>ΔΙΑΧΕΙΡΙΣΗ ΔΕΔΟΜΕΝΩΝ ΜΕΓΑΛΗΣ ΚΛΙΜΑΚΑΣ</b>  <b>Εξαμηνιαία Εργασία</b>  <b>ΣΥΝΟΔΕΥΤΙΚΗ ΑΝΑΦΟΡΑ</b>	ΙΟΥΝΙΟΣ 2024
		Σελίδα 13 / 20

βελτιστοποιήσεων του Catalyst Optimizer αλλά και των τεχνικών βελτιστοποίησης μνήμης που διαθέτει.

Το RDD API παρά τα διάφορα πλεονεκτήματα που διαθέτει ως επιλογή (παράλληλη επεξεργασία, ανάκτηση δεδομένων σε περίπτωση αποτυχίας, δυνατότητα μεγαλύτερου ελέγχου από τον προγραμματιστή), στη συγκεκριμένη περίπτωση υστερεί ελαφρώς ως προς την απόδοση. Πιθανός λόγος για τον οποίο συμβαίνει αυτό είναι το γεγονός ότι δεν έχει πραγματοποιηθεί από τους προγραμματιστές η μέγιστη δυνατή βελτιστοποίηση κι έτσι δεν εκμεταλλεύονται πλήρως οι δυνατότητές του API.

**ΖΗΤΟΥΜΕΝΟ 5**

Στους πίνακες που ακολουθούν παρουσιάζονται τα αποτελέσματα του Query 3 για τις τρεις (3) περιοχές με το υψηλότερο (Πίνακας 5) και τις τρεις (3) περιοχές με το χαμηλότερο (Πίνακας 6) εισόδημα ανά νοικοκυριό.

victim descent	total victims
White	347
Other	110
Hispanic/Latin/Mexican	55
Unknown	32
Black	18
Other Asian	16

**Πίνακας 5. Αποτέλεσμα Query 3. Περιοχές με το υψηλότερο εισόδημα**

victim descent	total victims
Hispanic/Latin/Mexican	1550
Black	1093
White	705
Other	400
Other Asian	104
Unknown	65
Korean	9
American Indian/Alaskan Native	3
Japanese	3

victim descent	total victims
Chinese	2
Filipino	2


**Πίνακας 6. Αποτέλεσμα Query 3. Περιοχές με το χαμηλότερο εισόδημα**

Το DataFrame που χρησιμοποιήθηκε στο παρόν ζητούμενο κάνει χρήση του Catalyst Optimizer. Η στρατηγική που ακολουθεί ο συγκεκριμένος βελτιστοποιητής πραγματοποιεί αυτοματοποιημένες βελτιστοποιήσεις στα ερωτήματα που εκτελούνται με το DataFrame API, που οδηγούν σε καλύτερη απόδοση. Στην προκειμένη περίπτωση, κάνοντας χρήση κατάλληλου κώδικα (explain), τυπώνεται ότι ο Catalyst Optimizer πραγματοποιεί sort merge join στην περίπτωση του (left) semi join και broadcast join στις υπόλοιπες. Αυτό είναι λογικό, γιατί ο συγκεκριμένος τύπος join (broadcast) είναι κατάλληλος για περιπτώσεις που ενώνονται ένα μεγάλο κι ένα μικρό σύνολο δεδομένων, όπως ακριβώς ισχύει και στην προκειμένη περίπτωση. Αντίθετα, εκτιμάται ότι στην περίπτωση του left semi join, επιλέγεται η sort merge, λόγω του ότι το “left” στην περίπτωσή μας αφορά στον μικρό πίνακα (πίνακα εισοδημάτων).

Στη συνέχεια (Πίνακας 7), παρουσιάζονται οι χρόνοι εκτέλεσης για καθεμιά από τις διαφορετικές υλοποιήσεις join μέσω της μεθόδου hint. Ύστερα από αξιολόγησή τους σε συνδυασμό με την παρατήρηση στοιχείων, όπως προέκυψαν στο web interface του Spark Job History Server, ακολουθεί ο σχολιασμός της καταλληλότητας των διαθέσιμων στρατηγικών join.

Στρατηγική join	Χρόνος εκτέλεσης
Catalyst Optimizer	1.5 min
Broadcast	1.7 min
Merge	1.4 min
Shuffle Hash	1.3 min




	<b>ΔΙΑΧΕΙΡΙΣΗ ΔΕΔΟΜΕΝΩΝ ΜΕΓΑΛΗΣ ΚΛΙΜΑΚΑΣ</b>  <b>Εξαμηνιαία Εργασία</b>  <b>ΣΥΝΟΔΕΥΤΙΚΗ ΑΝΑΦΟΡΑ</b>	ΙΟΥΝΙΟΣ 2024
		Σελίδα 16 / 20

Στρατηγική join	Χρόνος εκτέλεσης
Shuffle Replicate NL	56 min

**Πίνακας 7. Χρόνοι εκτέλεσης Query 3 ανά στρατηγική join**


Παρατηρώντας τους παραπάνω χρόνους μπορούν να εξαχθούν κάποιες παρατηρήσεις σχετικά με την εφαρμογή καθεμιάς από τις στρατηγικές join:

- Η διάρκεια εκτέλεσης στις τέσσερις (4) πρώτες περιπτώσεις είναι παρόμοια, ενώ ο τύπος join **Shuffle Replicate NL** απαίτησε σημαντικά περισσότερο χρόνο. Αυτό κρίνεται αναμενόμενο, καθώς με τη συγκεκριμένη στρατηγική τα δύο σύνολα δεδομένων καταμερίζονται (shuffled) σε όλο το σύμπλεγμα (cluster) και κάθε τμήμα του ενός συγκρίνεται με κάθε κομμάτι του άλλου με χρήση nested loops (NL). Λόγω της εξαντλητικής σύγκρισης που εφαρμόζει, ο χρόνος εκτέλεσης προκύπτει πολύ μεγάλος. Στην περίπτωση μας, που οι άλλες στρατηγικές δύνανται να εφαρμοστούν, η συγκεκριμένη κρίνεται ως μη κατάλληλη.
- Όπως προαναφέρθηκε, ο Catalyst optimizer στο συγκεκριμένο Query, εφάρμοσε τη στρατηγική **Broadcast** για το inner join και Sort **Merge** για το left semi join. Αυτός είναι ένας πιθανός λόγος που εξηγεί τη μικρή διαφορά στον χρόνο εκτέλεσης μεταξύ της αυτόματης (Catalyst optimizer) και της «χειροκίνητης» εφαρμογής (μέσω hint) της στρατηγικής, η οποία είναι της τάξης των περίπου 12 s. Ακόμη, η διαφορά στον χρόνο είναι τόσο μικρή που μπορεί να αποδοθεί και σε τυχαίους παράγοντες των μεμονωμένων εκτελέσεων. Σε κάθε περίπτωση, η στρατηγική **Broadcast** κρίνεται κατάλληλη, τόσο λόγω του αποτελέσματος όσο και επειδή στην προκειμένη περίπτωση συνενώνεται ένα μικρό σύνολο δεδομένων (που χωρά στη μνήμη κάθε κόμβου) με ένα μεγάλο (περίπτωση για την οποία συστήνεται η στρατηγική **Broadcast**).
- Στις περιπτώσεις ένωσης **Shuffle Hash**, **Broadcast** και **Merge**, η μέγιστη διαφορά χρόνου εκτέλεσης είναι περίπου 24 s (μεταξύ στρατηγικών Shuffle Hash και Broadcast). Οι μικροί σε διάρκεια χρόνοι, δηλώνουν μια κατ' αρχήν καταλληλότητα και των

	<p><b>ΔΙΑΧΕΙΡΙΣΗ ΔΕΔΟΜΕΝΩΝ ΜΕΓΑΛΗΣ ΚΛΙΜΑΚΑΣ</b></p> <p><b>Εξαμηνιαία Εργασία</b></p> <p>ΣΥΝΟΔΕΥΤΙΚΗ ΑΝΑΦΟΡΑ</p>	<p>ΙΟΥΝΙΟΣ 2024</p>
		<p>Σελίδα 17 / 20</p>

τριών στρατηγικών για το συγκεκριμένο πρόβλημα. Στην αναζήτηση της πιο κατάλληλης, η αξιολόγηση μόνο του (μικρού) χρόνου μεμονωμένων εκτελέσεων δεν είναι επαρκής. Για τον λόγο, αυτό παρακάτω εξηγείται πως λειτουργεί καθεμιά από τις δύο στρατηγικές που δεν έχουν αναλυθεί μέχρι τώρα:

- **Merge:** Σε αυτή την περίπτωση και τα δύο dataframes ταξινομούνται κατά το κλειδί συνένωσής τους και στη συνέχεια τα ταξινομημένα δεδομένα συγχωνεύονται. Είναι χρήσιμη για περιπτώσεις μεγάλων συνόλων δεδομένων τα οποία είναι ήδη ταξινομημένα ή μπορούν να ταξινομηθούν αποδοτικά. Στην περίπτωση του ερωτήματος, ο μικρός χρόνος εκτέλεσης θεωρείται ότι οφείλεται σε αποδοτική ταξινόμηση (εφόσον δεν ήταν εξαρχής ταξινομημένα).
- **Shuffle Hash:** Αυτή είναι η στρατηγική που απαίτησε τον λιγότερο χρόνο ανάμεσα σε όλες του Query 3. Με αυτόν τον τύπο join, τα δύο dataframes καταμερίζονται (shuffle) με βάση το κλειδί συνένωσης και στη συνέχεια εκτελείται hash join ανεξάρτητα σε κάθε καταμερισμό. Η συγκεκριμένη στρατηγική συστήνεται για μεσαίου μεγέθους σύνολα δεδομένων, των οποίων οι πίνακες κατακερματισμού μπορούν να χωρέσουν στη μνήμη. Ο μικρός χρόνος εκτέλεσης στην περίπτωση της άσκησης καταδεικνύει ότι για το μέγεθος του συγκεκριμένου (κύριου) συνόλου δεδομένων, το shuffling δεν κοστίζει σημαντικά.
- Συνοψίζοντας, καταλληλότερες στρατηγικές join για το Query 3 κρίνονται τόσο οι **Broadcast** και **Merge** (που χρησιμοποιούνται και από τον Catalyst Optimizer) όσο και η **Shuffle Hash** (λόγω του μικρότερου χρόνου εκτέλεσης). Στη γενική περίπτωση ωστόσο, όταν συνενώνονται ένα μικρό με ένα μεγάλο σύνολο δεδομένων, καταλληλότερη θεωρείται η στρατηγική Broadcast.

	<p><b>ΔΙΑΧΕΙΡΙΣΗ ΔΕΔΟΜΕΝΩΝ ΜΕΓΑΛΗΣ ΚΛΙΜΑΚΑΣ</b></p> <p><b>Εξαμηνιαία Εργασία</b></p> <p>ΣΥΝΟΔΕΥΤΙΚΗ ΑΝΑΦΟΡΑ</p>	<p>ΙΟΥΝΙΟΣ 2024</p>
		<p>Σελίδα 18 / 20</p>

## ΖΗΤΟΥΜΕΝΟ 6

Για το Query 4 υλοποιήθηκαν δύο (2) join αλγόριθμοι (broadcast και repartition) κάνοντας χρήση του RDD API. Τα join περιλάμβαναν το βασικό σύνολο δεδομένων (Los Angeles Crime Data) και το LA Police Stations.

Για τον αλγόριθμο broadcast, δημιουργήθηκε ένα αντίγραφο του LA Police Stations σε όλους του κόμβους, μέσω της συνάρτησης broadcast. Το μέγεθος του συγκεκριμένου RDD είναι αρκετά μικρό κι έτσι χωράει ολόκληρο στη μνήμη του κάθε κόμβου. Στη συνέχεια, κάθε tuple του RDD των εγκλημάτων μετασχηματίστηκε κατάλληλα ώστε να συμπεριληφθούν οι αντίστοιχες πληροφορίες από το αντίγραφο του LA Police Stations. Η προγραμματιστική υλοποίηση όσων περιγράφηκαν φαίνεται στο Απόσπασμα Κώδικα 1.

Για τον αλγόριθμο repartition, προστέθηκε μία ετικέτα (crime και LAPD) σε κάθε ένα tuple των RDDs και συγχωνευτήκαν σε ένα RDD. Ομαδοποιήθηκαν βάσει των κλειδιών και διαχωρίστηκαν τα tuples βάσει της ετικέτας τους. Στη συνέχεια, δημιουργήθηκε μία λίστα που περιείχε όλους τους συνδυασμούς tuples με διαφορετική ετικέτα. Έτσι, το τελικό RDD αποτελούταν από tuples που περιείχαν τα εγκλήματα και τα αντίστοιχα αστυνομικά τμήματα. Η προγραμματιστική υλοποίηση όσων περιγράφηκαν φαίνεται στο Απόσπασμα Κώδικα 2.

Μετά την εκτέλεση των δύο αλγορίθμων, υλοποιήθηκε το Query 4 (Πίνακας 8).

```

broadcasted_LAPD_rdd = spark.broadcast(LAPD_rdd_formatted \
    .keyBy(lambda x:x[0]).collectAsMap())

broadcast_value = broadcasted_LAPD_rdd.value

joined_rdd = crimes_rdd_formatted \
    .map(lambda x: [x[0], [x[1], broadcast_value.get(x[0])]])

```

### Απόσπασμα Κώδικα 1: Broadcast join

```

crimes_rdd_formatted = crimes_rdd \
    .map(lambda x: [int(x[4]), ["crime", [x[16], x[26], x[27]]]])

LAPD_rdd_formatted = LAPD_rdd \
    .map(lambda x: [int(x[3]), ["LAPD", [x[1], x[4], x[5]]]])

def arrange(seq):
    crime_origin = []

```



## ΔΙΑΧΕΙΡΙΣΗ ΔΕΔΟΜΕΝΩΝ ΜΕΓΑΛΗΣ ΚΛΙΜΑΚΑΣ

Εξαμηνιαία Εργασία

ΣΥΝΟΔΕΥΤΙΚΗ ΑΝΑΦΟΡΑ

ΙΟΥΝΙΟΣ 2024

Σελίδα 19 / 20

```
LAPD_origin = []
for (n, v) in seq:
    if n == "crime":
        crime_origin.append(v)
    elif n == "LAPD":
        LAPD_origin.append(v)
return [(v, w) for v in crime_origin for w in LAPD_origin]

dataset = crimes_rdd_formatted.union(LAPD_rdd_formatted) \
    .groupByKey().flatMapValues(lambda x: arrange(x))
```

**Απόσπασμα Κώδικα 2: Repartition join**

**ΖΗΤΟΥΜΕΝΟ 7**

Τέλος, υλοποιήθηκε το Query 4 χρησιμοποιώντας το DataFrame API. Τα αποτελέσματα φαίνονται στον ακόλουθο πίνακα (Πίνακας 8).

division	average_distance	incidents total
77TH STREET	2.688	17019
SOUTHEAST	2.105	12942
NEWTON	2.019	9846
SOUTHWEST	2.700	8912
HOLLENBECK	2.652	6202
HARBOR	4.082	5621
RAMPART	1.575	5115
MISSION	4.716	4504
OLYMPIC	1.822	4424
NORTHEAST	3.904	3920
FOOTHILL	3.803	3774
HOLLYWOOD	1.460	3641
CENTRAL	1.138	3614
WILSHIRE	2.314	3525
NORTH HOLLYWOOD	2.719	3466
WEST VALLEY	3.530	2902
VAN NUYS	2.222	2733
PACIFIC	3.729	2708
DEVONSHIRE	4.010	2471
TOPANGA	3.486	2283
WEST LOS ANGELES	4.244	1541

**Πίνακας 8. Αποτέλεσμα Query 4**

**Σημείωση:** Η τιμή της απόστασης δύναται να διαφέρει με τα αποτελέσματα σύγκρισης στο τρίτο (3ο) δεκαδικό ψηφίο, λόγω της διαφοράς στη γεωγραφική προβολή του τύπου που επιλέχθηκε για τον υπολογισμό της απόστασης (δεν έγινε χρήση βιβλιοθήκης georgy).